POLITECNICO DI TORINO

Laurea Magistrale in Ingegneria Informatica

Politecnico
di Torino

Tesi di Laurea Magistrale

L'impatto degli strumenti GenAI sullo sviluppo software:

evoluzione, rischi ed opportunità

Relatori

Prof. Luca ARDITO

Prof. Riccardo COPPOLA

Candidato

Raffaele Pane

Dicembre 2025

Abstract

Negli ultimi anni, l'Intelligenza Artificiale Generativa (GenAI) ha profondamente trasformato il ciclo di vita dello sviluppo software (Software Development Life Cycle, SDLC), offrendo nuove opportunità di automazione e ottimizzazione, ma introducendo al contempo sfide di rilievo di natura tecnica, organizzativa ed etica. La presente tesi analizza in modo sistematico l'impatto della GenAI sullo sviluppo software, con l'obiettivo di valutare in che misura gli strumenti basati su Large Language Models (LLM) possano migliorare i processi, mettendone in evidenza sia i benefici operativi sia le principali limitazioni tecnologiche e metodologiche.

Il lavoro propone un approccio sperimentale basato sulla progettazione e realizzazione di un sistema in grado di fornire una visione multilivello dell'evoluzione di un progetto software. L'architettura è stata implementata impiegando modelli linguistici leggeri, ottimizzati per l'esecuzione su singola GPU, in modo da garantire la riproducibilità e la scalabilità del sistema anche in contesti con risorse computazionali limitate. L'architettura sviluppata combina una pipeline offline, dedicata alla sintesi automatica dei commit e alla strutturazione delle informazioni di progetto, e una pipeline online, che consente ad un chatbot agentico di rispondere in linguaggio naturale a domande complesse, coordinando dinamicamente diversi tool specializzati per il recupero, la contestualizzazione e la generazione delle risposte. L'obiettivo è analizzare come la GenAI possa fungere da strumento di supporto cognitivo per sviluppatori e revisori, migliorando la comprensione dell'evoluzione del codice e la comunicazione all'interno dei team di sviluppo.

La validazione è stata condotta attraverso la creazione di due Golden Standard di riferimento e l'utilizzo congiunto di metriche quantitative (ROUGE, BLEU, METEOR, BERTScore) e qualitative, valutate sia automaticamente (tramite G-Eval) sia tramite giudizio umano. Per la pipeline offline, i risultati lessicali mostrano valori medi di BERT compresi tra 0.85 e 0.89, a conferma di una solida corrispondenza semantica tra i riassunti generati e quelli di riferimento. Le valutazioni qualitative ribadiscono la qualità dei testi prodotti, con punteggi medi complessivi compresi tra 3.5 e 4 su 5. La pipeline online ha evidenziato una qualità complessiva delle risposte in linea con quella osservata nella pipeline offline, sia sul piano lessicale che su quello percettivo, registrando prestazioni elevate nelle attività di recupero e generazione, con NDCG@10 = 0.85, MAP = 0.81 e un basso tasso di allucinazioni ($< 10\%$).

Il lavoro conferma il potenziale della GenAI come strumento di supporto avanzato per l'ingegneria del software, capace di migliorare la tracciabilità, la comunicazione e la qualità dei processi di sviluppo. Le evidenze sperimentali suggeriscono come l'adozione di architetture agentiche basate su LLM possa rappresentare un passo concreto verso sistemi sempre più intelligenti, trasparenti e integrati nel ciclo di vita del software.

Ringraziamenti

Non sono mai stato bravo a esprimere i miei sentimenti.

Di solito preferisco lasciare che a parlare siano le cose che faccio, i risultati che inseguo, o — quando proprio serve — una battuta sarcastica.
Eppure, arrivato a questo punto del percorso, mi sono accorto che certe parole vanno dette, e che alcune persone meritano molto più di un silenzioso grazie pensato e mai espresso.

Questa laurea non è nata solo tra righe di codice, paper scientifici e file rinominati con creatività discutibile. È cresciuta tra giornate piene, pause sottili come carta velina e orari che non sempre avevano senso. Studiare mentre si lavora significa convivere con un equilibrio instabile: un piede nella vita professionale, uno negli esami, e le braccia a tentare di tenere insieme tutto il resto — tempo, relazioni, energie.

Non è stato semplice. Ci sono state sere in cui la stanchezza batteva forte, e mattine in cui avrei voluto più tempo, più spazio o semplicemente più caffè. Ma passo dopo passo, senza rumore, si è giunti al traguardo.

E in questo cammino non sono stato solo.
Ci sono state persone — e non solo — che hanno reso possibile arrivare fin qui.

E allora, è giusto ringraziare.

Prima di tutto, Chiara: grazie per esserci stata in ogni fase di questo percorso — sia nei festeggiamenti che nei giorni complicati, nelle serate passate a studiare invece che uscire, nelle pause improvvisate.
Grazie per il tuo sostegno, per la pazienza e per l'amore che hai saputo dare anche quando io ero troppo stanco per accorgermene.
Grazie per la capacità di riportarmi alla realtà quando finivo troppo nello studio, nel lavoro o nelle preoccupazioni.

A volte bastava un tuo abbraccio, altre volte uno sguardo che diceva chiaramente: "Ok, però adesso riposa anche un po'."
Senza di te tutto questo sarebbe stato possibile... ma infinitamente meno bello e meno sensato.

Un ringraziamento speciale va a Clem, che purtroppo non c'è più, ma che continua a contribuire alla mia vita con la stessa energia con cui contribuiva alle mie call più importanti: abbaiando. Grazie per avermi insegnato che la priorità nella vita non è finire una tesi, ma lanciare la pallina.

E grazie anche a Giorgio, che ha seguito questa laurea con la stessa passione con cui ha seguito quella di Chiara: con impegno, costanza e tanti sbadigli.

Grazie ai miei amici Anna, Federica e Alessia, testimoni involontarie del mio lento declino mentale durante il percorso. Le loro battute, i loro messaggi e la loro capacità di deviare conversazioni serie nel caos più totale sono stati fondamentale carburante emotivo.

Un ringraziamento va anche a Massimiliano, compagno di discussioni infinite, parentesi filosofiche, rant tecnologici e domande che non avevano una vera risposta — ma che, nel frattempo, ci facevano sentire più intelligenti. Alcune delle nostre conversazioni non hanno portato a nulla di concreto... altre invece hanno lasciato idee, prospettive e qualche sorriso in più. E già questo, credo, valga molto.

Grazie alla mia famiglia. Non sempre è stato semplice trovare le parole giuste, i tempi giusti o la sintonia giusta — ma, in modi a volte diretti e altre volte silenziosi, avete comunque contribuito a farmi arrivare fin qui. Anche se non tutto è stato perfetto, questa meta porta con sé anche una parte del vostro contributo, e per questo ve ne sono grato.

Un grazie anche alla famiglia di Chiara, che in questi anni mi ha accolto con naturalezza, supporto e tanta pazienza — soprattutto nei periodi in cui ero più impegnato a parlare con i libri che con le persone. Mi avete sostenuto anche nella scelta della magistrale, quando ancora non era del tutto chiaro se fosse una buona idea o un atto di incoscienza. Sentirmi parte della vostra quotidianità ha reso questo percorso più leggero.

Un grazie anche ai miei colleghi universitari, in particolare Simran e Dimitri, con cui ho condiviso una buona parte di questo percorso tra esami, progetti e tanto stress. Il vostro supporto è stato molto importante.

Grazie ai professori che hanno contribuito a questo percorso con spiegazioni, indicazioni, tempo e fiducia. Alcuni non lo sanno, ma una frase detta al momento giusto può valere quanto un intero corso: alcune delle mie scelte, oggi, portano anche la loro impronta.

E infine, grazie a tutti gli altri — quelli che hanno avuto un ruolo evidente e quelli che magari non sanno neanche di averlo avuto.
Se leggi questo e ti stai chiedendo se parlo anche di te, la risposta è: sì.
O almeno... molto probabilmente.

A questo punto non credo ci sia molto altro da aggiungere.
Posso solo dire che questo percorso mi ha insegnato qualcosa che, forse, vale più di tutto il resto:
che nella vita non si avanza perché tutto è facile, ma perché si sceglie di continuare.

"Tutto ciò che possiamo decidere è come disporre del tempo che ci è dato"
— da un certo viaggio nella Terra di Mezzo

Raffaele Pane

Indice

Elenco delle tabelle

Elenco delle figure

Capitolo 1

Introduzione

1.1 Scopo della tesi

Nel panorama tecnologico contemporaneo, l'Intelligenza Artificiale Generativa (GenAI) sta emergendo con forza dirompente, trasformando e ridefinendo con forza i paradigmi fondamentali sia dei prodotti che dello sviluppo software. Questa evoluzione non rappresenta solo l'introduzione di nuovi strumenti, ma segna potenzialmente l'inizio di una nuova era nella progettazione, implementazione e manutenzione dei sistemi. Secondo Gartner, *"entro il 2026 oltre l'80% delle imprese utilizzerà API o modelli di IA Generativa e/o implementerà applicazioni basate su di essa in ambienti di produzione"*[1].

In un contesto in cui queste tecnologie assumono un ruolo sempre più centrale, diventa essenziale comprenderne non solo i vantaggi, ma anche le sfide e i rischi. La presente tesi si propone di analizzare in modo approfondito e sistematico l'impatto degli strumenti GenAI sull'intero ciclo di vita dello sviluppo software (SDLC – Software Development Lifecycle), esplorando le molteplici dimensioni di questa rivoluzione tecnologica. In particolare, si intende investigare come tali strumenti stiano trasformando le pratiche tradizionali, rendendo possibili approcci più efficienti, automatizzati e innovativi. Verranno analizzati i vantaggi in termini di produttività, qualità del codice e ottimizzazione dei tempi, ma anche le criticità emergenti, come le problematiche legate alla sicurezza, alla qualità del software, ai diritti d'autore e ai bias insiti nei modelli generativi. Un ulteriore elemento di analisi riguarda l'impatto organizzativo e culturale dell'introduzione della GenAI nei processi aziendali: l'adozione di questi strumenti richiede infatti non solo competenze tecniche adeguate, ma anche una revisione dei processi, dei ruoli e delle dinamiche di lavoro all'interno dei team di sviluppo.

Per affiancare alla riflessione teorica un riscontro pratico, la tesi include un caso di studio sperimentale incentrato sull'uso dei Large Language Models (LLM) per l'analisi di dati estratti da repository Git, come i messaggi di commit e i diff di codice. L'obiettivo è esplorare come la GenAI possa supportare e migliorare le attività di code review e l'analisi dell'evoluzione di un progetto software. I modelli vengono impiegati per sintetizzare l'intento delle modifiche, identificare pattern di sviluppo, raggruppare commit significativi e generare report contestualizzati utili alla comunicazione tra membri del team. Questo esperimento consente di valutare concretamente l'efficacia della GenAI nel migliorare trasparenza, tracciabilità e qualità nei processi di sviluppo.

La rilevanza dello studio è duplice. Da un lato, offre agli sviluppatori strumenti per un uso più consapevole ed efficace della GenAI; dall'altro, fornisce alle aziende indicazioni utili per integrare queste tecnologie nei propri processi produttivi, massimizzandone i benefici e mitigando i rischi. Infine, la tesi ambisce a contribuire al dibattito accademico, proponendo un'analisi critica e applicata utile sia ai ricercatori che ai professionisti del settore.

1.2 Contesto: il ruolo crescente della GenAI nell'industria del software

Negli ultimi anni, la GenAI ha guadagnato una posizione di primo piano tra le innovazioni tecnologiche più promettenti. Strumenti come ChatGPT, Gemini e Copilot hanno dimostrato la loro efficacia nel supportare i professionisti di ogni settore in molteplici attività, tra cui la scrittura automatica di codice, la generazione di test case e la creazione di documentazione. Questi progressi sono stati resi possibili grazie ai significativi avanzamenti nel campo dell'Intelligenza Artificiale, in particolare con l'avvento dei Large Language Models (LLM), capaci di generare contenuti con un livello di sofisticazione senza precedenti.

In un contesto industriale sempre più competitivo, l'integrazione di strumenti GenAI sta contribuendo in maniera profonda a trasformare le metodologie di lavoro nelle varie fasi del ciclo di vita dello sviluppo software. Dalla raccolta dei requisiti al testing, fino al rilascio e alla manutenzione, ci si può avvalere di nuovi strumenti in grado di supportare e ottimizzare numerosi compiti, migliorando l'efficienza e aumentando la qualità dei risultati finali. Questo scenario è ulteriormente dinamizzato dalla rapida evoluzione dei modelli stessi, che diventano sempre più performanti e in grado di comprendere il contesto d'uso con maggiore precisione. Di conseguenza, le organizzazioni devono adottare un approccio flessibile, capace di adattarsi continuamente per sfruttare appieno il potenziale offerto da tali tecnologie.

Accanto alle opportunità, emergono tuttavia sfide complesse. Le imprese si trovano a dover affrontare questioni legate alla proprietà intellettuale, alla sicurezza del software generato, alla privacy dei dati e alla crescente dipendenza da strumenti esterni. Inoltre, l'introduzione della GenAI implica spesso cambiamenti culturali e organizzativi significativi, non sempre privi di resistenze o difficoltà operative. Infine, si aprono interrogativi cruciali sul futuro delle professioni e sull'evoluzione dei ruoli in un ambiente sempre più automatizzato.

1.3 Struttura del lavoro

Per affrontare in modo sistematico e approfondito la tematica in oggetto, il lavoro è stato strutturato in cinque capitoli principali, al fine di garantire una trattazione organica e coerente.

Il primo capitolo introduce gli obiettivi della tesi, il contesto tecnologico e le motivazioni alla base dello studio, fornendo una base concettuale solida per la lettura dei capitoli successivi. Nel secondo capitolo si analizza nel dettaglio come la GenAI stia trasformando le diverse fasi del ciclo di vita dello sviluppo software, con l'ausilio di esempi pratici, casi d'uso ed evidenze tratte dalla letteratura. Si discutono inoltre gli strumenti più diffusi, le tecniche di prompt engineering e le principali sfide etiche, tecniche e organizzative. Il terzo capitolo descrive il progetto sperimentale, illustrando l'applicazione dei LLM per l'analisi dei dati Git e il supporto alla code review. Viene valutata la capacità dei modelli di generare insight strutturati, facilitare la comprensione delle modifiche e migliorare la comunicazione nei team. Il quarto capitolo presenta una discussione critica dei risultati ottenuti dal caso di studio, mettendoli in relazione con quanto emerso dalla letteratura esistente, e delineando le principali implicazioni per lo sviluppo software. Infine, il quinto capitolo sintetizza i risultati della tesi, offrendo spunti di riflessione per sviluppatori e aziende e suggerendo possibili direzioni per future ricerche e applicazioni della GenAI nel software engineering.

Nel complesso, la struttura proposta mira a mantenere un equilibrio tra riflessione teorica e sperimentazione pratica, fornendo una visione completa, integrata e critica del fenomeno analizzato.

Capitolo 2

GenAI nel ciclo di vita del software

La crescente integrazione degli strumenti GenAI nel ciclo di vita dello sviluppo software sta modificando profondamente processi, ruoli e pratiche consolidate. In questo capitolo analizzeremo come tecnologie quali Copilot e ChatGPT, e più in generali i prodotti basati su LLM, stanno intervenendo concretamente nelle diverse fasi del SDLC, ridefinendone operatività ed efficienza.

Si partirà da una breve panoramica sulle fasi tradizionali del ciclo di vita, per poi esaminare gli impatti più rilevanti dell'adozione della GenAI: dall'automazione nella scrittura del codice alla generazione assistita di test e documentazione, fino al supporto nelle attività di pianificazione e monitoraggio dei progetti. Verrà inoltre approfondita la disciplina del Prompt Engineering, leva fondamentale per un utilizzo efficace e controllato dei modelli generativi.

Verranno discussi i principali rischi e le criticità emerse, con particolare attenzione agli aspetti tecnici, etici e organizzativi. L'obiettivo è tracciare un quadro chiaro delle potenzialità e dei limiti attuali, per comprendere appieno come queste tecnologie stiano progressivamente ridefinendo l'intero sviluppo software.

2.1 Panoramica sul ciclo di vita del software tradizionale

Il ciclo di vita del software (Software Development Life Cycle, SDLC) è un modello organizzativo che definisce le fasi sequenziali (o iterative) per lo sviluppo, il collaudo e la manutenzione di un sistema software. Fornisce un processo standard per creare, testare e distribuire software di alta qualità [2]. Questo processo, pur avendo subito numerosi adattamenti nel tempo, segue uno schema di base che si articola in una sequenza di fasi interdipendenti. Ogni fase è progettata per garantire la qualità e l'affidabilità del prodotto finale, minimizzando al contempo i rischi associati a eventuali errori o ritardi.

Al fine di garantire una trattazione chiara e lineare, in questa sezione si è scelto di adottare una suddivisione arbitraria e granulare delle possibili fasi fondamentali: pianificazione, analisi dei requisiti, progettazione, implementazione, testing, rilascio e manutenzione. Questa struttura, pur semplificando alcune complessità, offre una visione efficace delle caratteristiche e delle criticità che si possono riscontrare lungo l'intero ciclo di vita del software. In questo modo, funge da base utile per poter poi analizzare e comprendere come la Generative AI possa essere iniettata nelle attività delle varie fasi.

Pianificazione

La pianificazione rappresenta la fase iniziale del SDLC, dedicata alla definizione degli obiettivi, dell'ambito, delle risorse e delle tempistiche di progetto. In questa fase si svolgono attività di analisi di fattibilità, stima dei costi e dei tempi, identificazione dei rischi e dei vincoli di progetto. Si redige un piano di progetto formale che include, tra l'altro, la descrizione del contesto, l'elenco dei deliverable attesi, i criteri di successo, le milestone, l'organigramma di progetto e il piano di gestione dei rischi. Una pianificazione accurata è fondamentale per individuare precocemente rischi e vincoli e per garantire il controllo di tempi e costi.

Analisi dei requisiti

L'analisi dei requisiti ha l'obiettivo di identificare, documentare e validare le esigenze funzionali e non funzionali del software. In questa fase gli analisti interagiscono con gli stakeholder per raccogliere i requisiti mediante tecniche di elicitation (interviste, questionari, prototipi). L'obiettivo è ottenere una descrizione precisa e condivisa delle aspettative del progetto, evitando ambiguità che potrebbero compromettere enormemente le fasi successive. Tali informazioni vengono spesso formalizzate in uno o più documenti di specifica dei requisiti, così da fornire una base chiara, completa e verificabile per le attività successive. È importante sottolineare come la

qualità della definizione dei requisiti sia cruciale per il successo del progetto; errori o ambiguità in questa fase possono compromettere l'intero ciclo di sviluppo.

Progettazione

La fase di progettazione traduce i requisiti funzionali in una struttura architetturale del sistema e in specifiche di dettaglio. Vengono definiti i componenti principali del software, le loro interfacce e l'architettura complessiva, spesso mediante diagrammi UML, prototipi e documenti di design. L'obiettivo è garantire che la soluzione progettata soddisfi i requisiti di funzionalità, prestazioni e qualità espressi. Una progettazione accurata è una base fondamentale per favorire la manutenibilità e l'estensibilità del software. Questa fase include, inoltre, la definizione di standard di codifica, linee guida progettuali e la valutazione dei trade-off (prestazioni, sicurezza, scalabilità) necessari per ottenere il miglior compromesso possibile a partire dai requisiti ricevuti.

Implementazione

Durante l'implementazione, i programmatori sviluppano il codice necessario per costruire il software, seguendo le specifiche delineate nella progettazione. Questa fase rappresenta il momento operativo in cui il progetto prende vita, con il codice che viene scritto, testato e migliorato. Sebbene l'implementazione segua le linee guida definite, è comunque un processo estremamente dinamico dove spesso emergono sfide tecniche o nuove esigenze che richiedono possibili affinamenti. La collaborazione tra i membri del team è quindi fondamentale per garantire che tutte le parti del sistema siano integrate correttamente. Il risultato finale è un insieme di componenti software integrati e funzionanti, pronti per essere sottoposti al testing.

Testing

La fase di testing mira a identificare e correggere difetti nel software sviluppato prima che finisca in ambiente produttivo. Viene condotto per identificare bug, valutare le prestazioni e garantire che il prodotto soddisfi i requisiti stabiliti. Tutti i difetti riscontrati vengono tracciati, corretti e ritestati. Ci sono varie tipologie di test che possono essere condotti. Tra i più importanti abbiamo quelli che includono controlli su singoli moduli (unit testing), verifiche sull'integrazione tra i componenti (integration testing) e test complessivi sul sistema (system testing). A questi si aggiunge il test di accettazione (acceptance testing), che rappresenta la fase finale del processo di verifica, dove l'obiettivo è quello di confermare che il software soddisfi i requisiti funzionali e non funzionali concordati, nonché le aspettative degli utenti finali o dei committenti (spesso coinvolge direttamente gli utenti finali, i quali valutano se il sistema è pronto per l'uso in un ambiente reale). Questa fase

assicura che il software sia pronto per essere utilizzato senza problemi significativi. Solo dopo il superamento con successo di tutti i test critici il software può essere considerato pronto per il rilascio.

Rilascio

La fase di rilascio consiste nella consegna del software agli utenti finali o nell'ambiente di produzione. In questa fase si preparano i pacchetti di installazione o le immagini di sistema, si aggiorna la documentazione tecnica e utente, e si formano gli operatori di sistema o gli utenti finali. Inoltre, può prevedere una fase di deployment controllato con la verifica che il sistema operi correttamente.

Una release di successo si conclude con verifiche post-deployment finalizzate a confermare l'effettivo funzionamento del software in produzione e l'acquisizione del feedback iniziale dagli utenti.

Manutenzione

La manutenzione è l'ultima fase del ciclo di vita, ma non meno importante. Una volta che il software è stato rilasciato, inizia un processo continuo di monitoraggio e aggiornamento. Questo include la risoluzione di bug rilevati dopo il rilascio, l'implementazione di nuove funzionalità e l'adattamento del software ai cambiamenti tecnologici o alle esigenze degli utenti. La manutenzione assicura che il prodotto rimanga funzionale, sicuro e rilevante nel tempo, prolungandone la durata e massimizzando il valore per gli utenti.

2.2 GenAI e LLM: definizioni e concetti generali

Prima di analizzare nel dettaglio l'impatto della GenAI sulle diverse fasi del SDLC, è fondamentale comprendere i concetti e le terminologie chiave alla base dei principali modelli generativi sviluppati negli ultimi anni. Questi strumenti rappresentano un'evoluzione significativa nel campo dell'intelligenza artificiale, distinguendosi per caratteristiche uniche rispetto ai modelli precedentemente disponibili. Approfondire il loro funzionamento consente non solo di valutarne i vantaggi e le limitazioni, ma anche di comprenderne al meglio le potenzialità e le modalità di utilizzo ottimali.

2.2.1 Generative Artificial Intelligence (GenAI)

L'intelligenza artificiale generativa (GenAI) [3] è un ramo dell'intelligenza artificiale che sfrutta modelli generativi per produrre risultati sotto forma di testo, immagini, video o altre forme di dati. Il termine "generativo" indica la capacità di questi modelli di apprendere gli schemi e le strutture sottostanti ai dati di addestramento, utilizzandoli poi per generare nuovi contenuti in risposta agli input ricevuti, spesso espressi in linguaggio naturale (prompt). Questi modelli si basano su strutture statistiche che descrivono la distribuzione congiunta tra una variabile osservabile e una variabile dipendente, detta anche target. I dati prodotti sono, quindi, generati a partire dai pattern appresi e, implicitamente, dalla distribuzione sottostante. In altri termini, i modelli generativi sono degli strumenti che, analizzando grandi quantità di dati, imparano a prevedere o a creare nuovi dati (variabile target) basandosi su quelli che hanno già visto (variabile osservabile). Per fare un'analogia, è un processo simile a quello che potrebbe avere un pittore che vuole dipingere, ad esempio, dei gatti. Osservando tante immagini di gatti diversi, il pittore sarà in grado di capire quali caratteristiche identificano il concetto di gatto e sarà in grado di riprodurle su tela. L'intelligenza artificiale generativa funziona in modo simile: osserva tanti dati (come il pittore guarda le immagini) e ne apprende le caratteristiche principali. Poi usa ciò che ha imparato per creare nuovi contenuti che sono simili a quelli di partenza, proprio come i gatti dipinti dal pittore.

Tali concetti, nonostante possono sembrare ad una prima lettura superflui, pongono invece le basi ad importanti questioni etiche e sociali legate al diritto d'autore e alla proprietà del contenuto generato. Inoltre, la natura intrinsecamente statistica di questi modelli porta alla generazione di output non deterministici, che impongono un approccio ingegneristico per il loro utilizzo. Entrambe le tematiche verranno approfondite più in avanti nella trattazione.

Dal punto di vista tecnico, l'evoluzione della GenAI è stata resa possibile grazie alla crescente disponibilità di dati e risorse computazionali, oltre che dai progressi nel campo delle Deep Neural Network. In particolare, un contributo determinante è

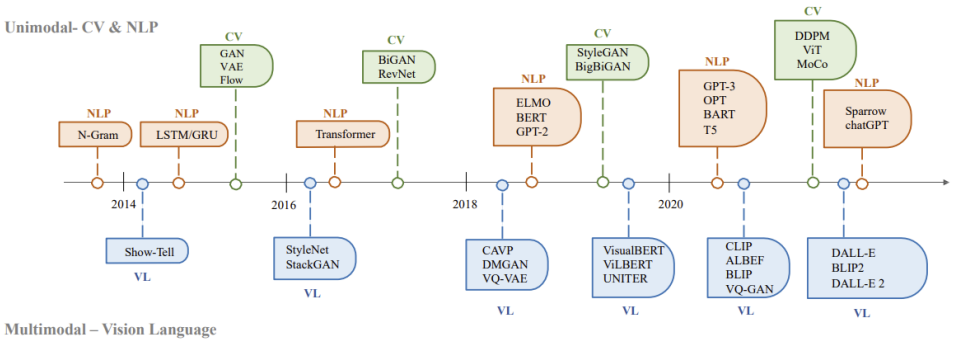

Figura 2.1: Storia della Generative AI. Immagine tratta da [4]

stato l'avvento dei modelli basati su Transformer [5], che hanno farvorito lo sviluppo di Large Language Model (LLM) come GPT (Generative Pre-trained Transformer) [6] di OpenAI [7]. Tali innovazioni hanno costituito un decisivo avanzamento [4] tecnologico rispetto alle opzioni precedentemente disponibili, quali Long Short-Term Memory (LSTM) [8], Variational Autoencoder (VAE) [9] e Generative Adversarial Network (GAN) [10].

Negli ultimi cinque anni, le capacità di questi modelli hanno registrato una crescita esponenziale, dimostrando una sempre maggiore capacità di generare contenuti estremamente realistici e contestualmente coerenti. Questi progressi hanno trovato applicazione in ambiti quali chatbot avanzati, assistenti virtuali e strumenti di generazione automatizzata di contenuti. Oltre ai modelli basati sul linguaggio naturale, le tecnologie di generazione di immagini come DALL-E [7], Midjourney [11] e Stable Diffusion [12] hanno dimostrato notevoli capacità nel trasformare descrizioni testuali in immagini altamente dettagliate e realistiche, mentre Sora ha rappresentato un passo decisivo nella creazione di contenuti video.

Ed è qui che spunta un altro aspetto chiave dei prodotti GenAI contemporanei: ovvero la capacità di lavorare con dati unimodali e multimodali [13]. I sistemi unimodali si concentrano su un'unica tipologia di dato, come il testo o le immagini, mentre i sistemi multimodali sono in grado di integrare e processare informazioni proveniente da tipologie di fonti diverse, combinando, ad esempio, testo e immagini per ottenere risposte più contestualizzate e ricche. Un esempio significativo è GPT-4 [7], che supporta input testuali, audio e visivi, consentendo nuove forme di interazione utente-modello.

2.2.2 Large Language Model

Un altro concetto chiave che deve essere dettagliato in questa trattazione è quello dei Large Language Models (LLM) [14]. Si tratta di una classe avanzata di modelli di intelligenza artificiale che rientra nella categoria della GenAI, progettati specificatamente per processare e generare linguaggio naturale (NLP), includendo attività come la comprensione, la generazione e la traduzione del testo. Questi modelli sono addestrati su enormi quantità di dati testuali, come libri, articoli e pagine web, e sono caratterizzati da un numero estremamente elevato di parametri, che possono raggiungere le decine o le centinaia di miliardi (da qui l'aggettivo "Large"). Tali caratteristiche consentono agli LLM di catturare le sfumature linguistiche e generare testo con una convincente precisione sintattica, semantica ed ontologica. Tuttavia, questa forte dipendenza dalla qualità dei dati di addestramento comporta l'ereditarietà di eventuali inaccuratezze e bias presenti nei dati stessi.

Alla base del funzionamento degli LLM troviamo un'architettura chiamata Transformer [5]. A differenza dei modelli precedenti, come le RNN, che elaboravano il testo in modo sequenziale, i Transformer riescono ad elaborare l'intero contesto contemporaneamente, individuando le connessioni tra le parole in modo molto più efficace ed efficiente.

Per capire al meglio come utilizzare questi strumenti è utile passare in rassegna alcuni dei termini e delle caratteristiche più importanti che li caratterizzano.

Uno degli elementi chiave che influenzano le prestazioni degli LLM è la Context Window [14][15], ossia la quantità di testo che il modello può considerare in un'unica elaborazione. Ogni parola o frammento di parola viene suddiviso in unità chiamate token, e il modello ha un limite massimo di token che può elaborare alla volta. Ad esempio, un modello come GPT-3 ha una finestra di contesto di circa 2048 token [6], mentre modelli più avanzati, come GPT-4.1, possono elaborare fino a 1.047.576 token [16]. Questo limite influisce direttamente sulle capacità del modello di mantenere il "filo logico" in testi lunghi o conversazioni estese: se la lunghezza del testo supera la finestra di contesto, le informazioni più vecchie rischiano di essere dimenticate, a meno che non vengano riassunte o richiamate opportunamente.

Quando interagiamo con un LLM, l'efficacia delle risposte dipende in larga parte dalla qualità dell'input che forniamo. Qui entra in gioco il concetto di Prompt Engineering [17][18], ovvero l'arte di scrivere comandi testuali (prompt) in modo chiaro e strutturato per ottenere risposte pertinenti. Un buon prompt fornisce al modello tutte le informazioni necessarie per contestualizzare la richiesta e guidarlo verso la risposta desiderata. Ad esempio, chiedere "Spiegami la relatività ristretta come se avessi 10 anni" produrrà un risultato molto diverso rispetto a una richiesta

più generica come "Parlami della relatività ristretta". Su questo tema verrà dedicato un capitolo dedicato nel quale verranno mostrate tutte le tecniche più importanti.

Un altro aspetto fondamentale per chi utilizza gli LLM è la possibilità di effettuare il Fine-Tuning [19] del modello, ovvero di adattarlo a domini specifici. Sebbene questi modelli siano addestrati su dataset vastissimi e molto eterogenei, spesso si rende necessario affinare il loro comportamento per settori particolari, come la medicina, il diritto o la finanza. Il fine-tuning permette, infatti, di proseguire l'addestramento su un corpus di dati mirato, in modo da migliorare la precisione e la pertinenza delle risposte, riducendo al contempo il rischio che il modello generi contenuti fuori contesto o perpetui bias presenti nei dati di partenza.

Un altro elemento importante da considerare è il parametro della Temperatura [20], che influenza il grado di casualità nella generazione del testo. In pratica, la temperatura modula la distribuzione di probabilità utilizzata dal modello per scegliere il token successivo durante la generazione. Impostazioni con temperatura bassa tendono a rendere le risposte più determinate e coerenti, favorendo precisione e affidabilità, mentre temperature più elevate introducono una maggiore varietà e creatività nei risultati, sebbene a volte a discapito della coerenza. La scelta del valore ideale dipende dall'obiettivo specifico: per applicazioni in cui è essenziale mantenere un alto grado di accuratezza e coerenza si preferiscono temperature basse, mentre per attività che richiedono esplorazione creativa o generazione di idee originali è possibile sperimentare con valori più elevati.

Un ultimo concetto interessante nell'uso degli LLM è rappresentato dal paradigma della Retrieval-Augmented Generation (RAG) [21][22][23]. Questa tecnica combina le capacità di generazione del linguaggio dei modelli con l'accesso a una banca dati esterna, consentendo al sistema di integrare informazioni aggiornate o specifiche durante la fase di risposta. In altre parole, invece di affidarsi esclusivamente alla conoscenza appresa durante il l'addestramento, il modello, in presenza di una richiesta, può eseguire una ricerca su un corpus esterno e "recuperare" dati pertinenti. Queste informazioni vengono poi fuse con il testo generato, contribuendo a produrre risposte non solo più ricche e contestualizzate, ma anche più accurate rispetto a quelle basate su dati statici. Questo approccio risulta particolarmente utile in scenari in cui le informazioni evolvono rapidamente, o quando è necessario rispondere a domande che richiedono conoscenze molto verticali o estremamente aggiornate.

Tra i modelli più noti abbiamo GPT di OpenAI [7], Claude di Anthropic [24], Gemini di Google [25] ed Llama di Meta [26].

2.2.3 Transformer

L'architettura Transformer, introdotta nel 2017 con il celebre articolo Attention Is All You Need [5], ha rivoluzionato il campo dell' elaborazione del linguaggio naturale (NLP) e rappresentato la base per lo sviluppo di modelli generativi avanzati come Generative Pre-trained Transformer (GPT) [6].

Il cuore dell'architettura è il meccanismo di Self-Attention, che consente al modello di pesare in modo dinamico l'importanza di ciascun token rispetto agli altri presenti nella stessa sequenza. In particolare, il meccanismo di Scaled Dot-Product Attention opera su tre insiemi di vettori derivati dalla rappresentazione degli input: query (Q), key (K) e value (V). Il calcolo è strutturato come in figura 2.2 ed è sintetizzabile attraverso la seguente formula:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V$$

dove d_k rappresenta la dimensione delle chiavi. Questa normalizzazione tramite $\frac{1}{\sqrt{d_k}}$ è fondamentale per evitare che i prodotti scalari assumano valori troppo elevati, il che potrebbe portare la funzione Softmax in regioni con gradiente estremamente piccolo e, di conseguenza, diminuire la stabilità dell'addestramento.

Per catturare diversi aspetti relazionali e semantici della sequenza, i Transformer adottano il concetto di Multi-Head Attention (figura 2.3. In pratica, invece di eseguire una singola operazione, il modello esegue h operazioni parallele, ognuna

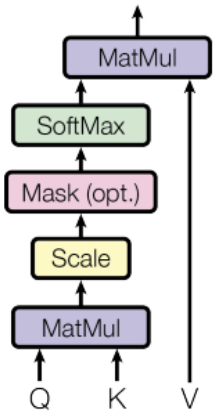

Figura 2.2: Scaled Dot-Product Attention [5]

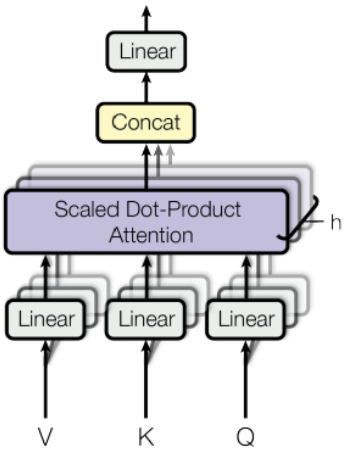

Figura 2.3: Multi-Head Attention [5]

con propri insiemi di proiezioni lineari per Q, K e V. Ogni "testa" elabora una rappresentazione differente del contesto, e i risultati vengono successivamente concatenati e proiettati in uno spazio finale. Formalmente, se indichiamo con Q_i, K_i e V_i le proiezioni relative alla i-esima testa, l'operazione complessiva è:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)\, W^O$$

dove ogni $\text{head}_i = \text{Attention}(QW_i^Q,\, KW_i^K,\, VW_i^V)$ e W_i^Q, W_i^K, W_i^V e W^O sono matrici di parametri. Questo approccio permette di modellare in maniera più ricca le interdipendenze tra i token, considerando diverse "prospettive" del contesto simultaneamente.

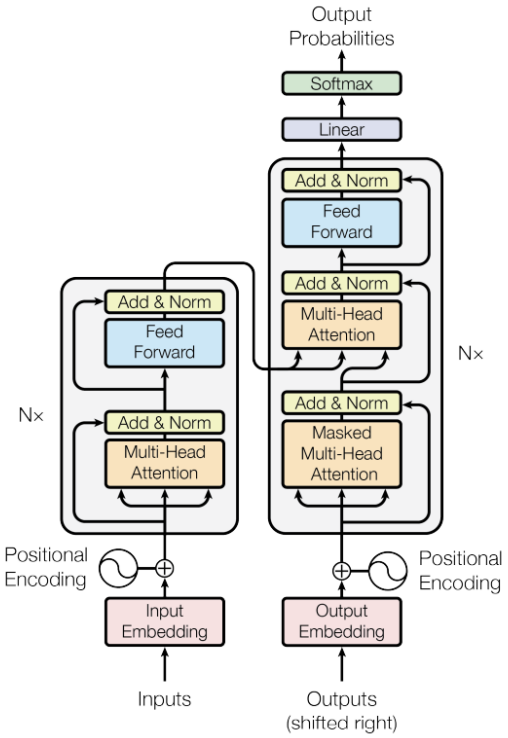

Figura 2.4: Architettura Transformer [5]

L'architettura Transformer classica, in figura 2.4 è suddivisa in due parti principali:

- **Encoder:** Composto da una serie di blocchi identici, ciascuno dei quali integra un modulo Multi-Head Attention e una rete Feed-Forward. L'Encoder trasforma la sequenza simbolica di input $x = (x_1, \ldots, x_n)$ in una rappresentazione continua $z = (z_1, \ldots, z_n)$ che cattura le informazioni contestuali.

- **Decoder:** In maniera simile, anche il decoder è organizzato in n blocchi uguali tra loro, ma include un ulteriore meccanismo di attenzione che permette di "guardare" la rappresentazione prodotta dall'Encoder. Inoltre, nella parte bassa, viene impiegata una versione Masked della Multi-Head Attention che assicura che ogni token generato possa basarsi soltanto sui token precedenti, preservando l'ordine temporale durante la generazione del testo (nella pratica si crea una matrice triangolare superiore, ponendo i valori sotto la diagonale a $-\infty$, così che con la softmax diventino zeri).

Questi meccanismi permettono di calcolare le dipendenze tra le parole di una sequenza in parallelo, superando i limiti di Vanishing Gradient e di inefficienza computazionale su sequenze lunghe di cui soffrivano le RNN [5].

2.2.4 GPT

Il modello GPT (Generative Pre-trained Transformer) è stato introdotto da OpenAI [7] nel paper "Improving Language Understanding by Generative Pre-Training" [6] del 2018 e rappresenta una ulteriore svolta decisiva nel campo, grazie all'introduzione di un approccio innovativo atto a migliorare le prestazioni nei compiti di comprensione del linguaggio naturale.

Fino a quel momento i modelli tradizionali di NLP richiedevano enormi quantità di dati etichettati per effettuare gli addestramenti necessari. Tuttavia, tali dati sono spesso scarsi e costosi da ottenere, soprattutto per compiti complessi. La capacità di apprendere in modo efficace da testi grezzi, ovvero non strutturati né classificati, è quindi un elemento cruciale per ridurre la dipendenza dall'addestramento supervisionato. Il lavoro di GPT ha mostrato che è possibile utilizzare enormi quantità di dati non etichettati per pre-allenare un modello e, successivamente, adattarlo (fine-tuning) a compiti specifici tramite l'ausilio di pochi dati etichettati.

Partendo dai vantaggi rilevati nel modello Transformer, i ricercatori di OpenAI hanno rimosso la parte di Encoder, puntando ad una architettura formata solo dal solo Decoder. Questa scelta architetturale è dettata dalla necessità di generare testo in maniera autoregressiva: il modello predice ogni token in sequenza, condizionato esclusivamente sui token precedenti, grazie all'applicazione di una mascheratura che impedisce al modello di basarsi anche sulle informazioni future. Entrando nel dettaglio delle due fasi, si ha che nella prima il modello viene addestrato in maniera non supervisionata su un ampio corpus di testo, senza l'utilizzo di etichette. L'obiettivo è apprendere una rappresentazione generale del linguaggio e delle sue strutture. Dato un insieme non supervisionato di token $U = \{u_1, \ldots, u_n\}$, il training punta a massimizzare la seguente funzione obiettivo:

$$L_1(\theta) = \sum_i \log P(u_i \mid u_{i-k}, \ldots, u_{i-1}; \theta)$$

dove k è la grandezza della Context Window e la probabilità condizionale P è modellata usando una rete neurale con parametri θ. Questi parametri sono addestrati usando la Stochastic Gradient Descent.

Nella seconda fase, il modello pre-addestrato viene adattato a compiti specifici utilizzando un insieme di dati etichettati. L'idea è quella di sfruttare la conoscenza acquisita durante il pre-addestramento per migliorare le prestazioni su task particolari, come la classificazione o il Question Answering.

Si assume un dataset etichettato C, in cui ogni istanza consiste in una sequenza di token di input $x^1, \ldots, x^m$, insieme a un'etichetta y. Gli input vengono passati

attraverso il modello pre-addestrato per ottenere l'attivazione finale del blocco transformer h_l^m, che viene poi inserita in un livello lineare di output con parametri W_y per predire y:

$$P(y \mid x^1, \ldots, x^m) = \text{softmax}(h_l^m W_y)$$

Questo porta alla seguente funzione obiettivo da massimizzare:

$$L_2(C) = \sum_{(x,y)} \log P(y \mid x^1, \ldots, x^m)$$

Nei modelli più recenti (come GPT-3, GPT-4 e GPT-5) questo approccio ha dimostrato una scalabilità eccezionale: aumentando il numero di parametri (nell'ordine di centinaia di miliardi) e la quantità di dati, le prestazioni migliorano notevolmente [4]. Ed è proprio grazie al successivo paper del 2020 "Language Models are Few-Shot Learners" [27], sempre di OpenAI, che si sono poste le basi per l'utilizzo moderno degli LLM.

Come accennato precedentemente, prima dell'avvento di GPT-3 gli LLM richiedevano un processo di fine tuning specifico per il task target, indispensabile per garantire performance ottimali. Tale procedura necessitava di dataset specifici e comportava la spesa di ingenti risorse economiche, tecnologiche e temporali per effettuare l'addestramento.

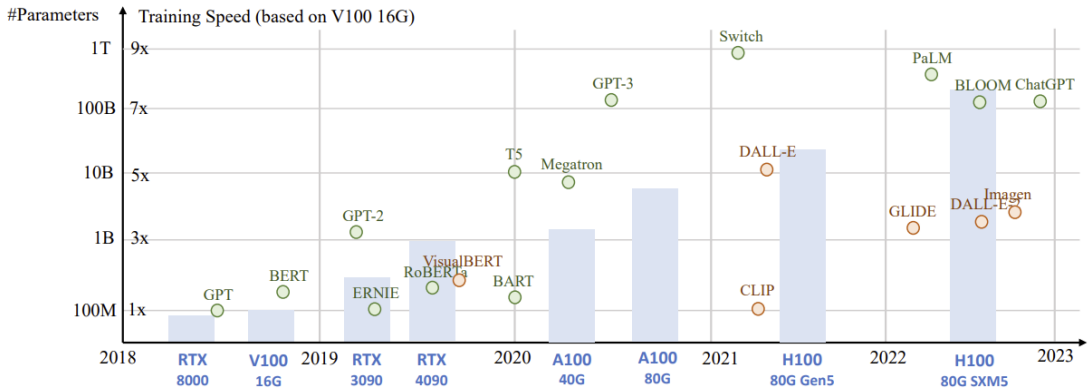

Figura 2.5: Statistiche sulle dimensioni del modello e sulla velocità di addestramento tra diversi modelli e dispositivi di calcolo [4]

Il paper in questione ha invece evidenziato come l'aumento di scala possa eliminare la dipendenza dal fine tuning, trasformando i modelli in veri "Few-Shot Learners", capaci di adattarsi efficacemente anche fornendo pochi, o addirittura nessun, esempio o istruzione.

Ciò ha permesso di cambiare totalmente l'approccio necessario verso di essi, rendendo di fatto la tecnologia estremamente più accessibile e stimolando in maniera sostanziale l'interesse e i progressi successivi nel settore.

Dal punto di vista ingegneristico, è però bene sottolineare che l'implementazione di tali modelli richiede l'uso di infrastrutture hardware specializzate, come GPU e TPU ad alte prestazioni, e framework di machine learning ottimizzati. Inoltre, aspetti come l'efficienza computazionale, la gestione della memoria e l'ottimizzazione degli iperparametri sono cruciali per garantire prestazioni adeguate in ambienti di produzione.

2.3 La GenAI e la trasformazione del Ciclo di Vita del Software

Questo paragrafo si pone l'obiettivo di offrire una panoramica generale sulle opportunità che offrono gli strumenti di GenAI per ognuna delle fasi del ciclo di vita del software. Benché si punti ad una trattazione esaustiva, è un argomento così in fervente evoluzione che resta estremamente complesso riuscire ad avere una visione aggiornata di ogni singola nuova opportunità. Proprio per questo motivo, in ognuna delle fasi verranno esposti gli esempi reputati più rappresentativi ed interessanti, consapevole del fatto che sicuramente verrà tralasciato qualcosa.

Pianificazione

La fase di pianificazione rappresenta uno dei momenti più delicati e strategici di qualsiasi progetto. Un errore di valutazione in questa fase (come anche per quelle di Analisi e di Progettazione) può tradursi in costi elevati, ritardi significativi e, nei casi peggiori, nel fallimento dell'intera iniziativa. A differenza di altri ambiti meno critichi in cui la GenAI può automatizzare compiti ripetitivi con notevoli vantaggi e pochi rischi, la pianificazione resta ancora profondamente ancorata all'esperienza, all'intuito e alle relazioni umane. Il coinvolgimento di stakeholder, la capacità di negoziare priorità, di interpretare correttamente le esigenze esplicite e implicite, così come di adattarsi alle variabili impreviste, sono elementi che difficilmente possono essere sostituiti.

Tuttavia, esistono delle aree in cui è possibile un supporto fruttuoso di questi strumenti, come anche di altre forme di AI più classiche.

Un primo esempio è l'assistenza nei meeting: a partire dall'audio o dalla trascrizione della riunione (ormai disponibile nella maggior parte degli strumenti in commercio), la GenAI può essere utilizzata per estrarre informazioni rilevanti dalle conversazioni. Tali informazioni, integrate con dati sul dominio applicativo, possono poi essere rielaborate per generare documenti più strutturati, come i requisiti di business. Questa elaborazione può includere anche la produzione automatizzata di report, verbali di riunione, compilazioni di template standard o suddivisioni in parti. In generale, è importante osservare sin da subito che, ovunque sia presente linguaggio naturale, esiste un potenziale ambito di applicazione per gli LLM.

Un altro esempio concreto, benché trasversale alle varie fasi del progetto ma sempre ad appannaggio di PM, riguarda la generazione automatizzata di testi più operativi. La GenAI può essere impiegata per generare automaticamente ticket per strumenti di gestione del lavoro, come Jira o Trello, categorizzandoli in base a priorità, assegnando loro etichette pertinenti e suggerendo descrizioni basate su

best practice consolidate. Questo non solo accelera il processo di schedulazione, ma in potenza può ridurre anche il rischio di omissioni o interpretazioni errate, garantendo maggiore coerenza nelle specifiche di progetto.

Oltre questi casi, è interessante notare come restino ancora valide forme di AI più tradizionali, come gli algoritmi di Machine Learning, che possono dare supporto attraverso l'analisi dei requisiti iniziali e dei dati storici di progetti simili per fornire stime di tempo e budget più accurate rispetto ai metodi manuali. Oppure attraverso l'analisi di pattern ricorrenti e dati progettuali, identificando potenziali rischi tecnici o organizzativi fin dalle fasi iniziali, consentendo di adottare misure preventive in modo più tempestivo [28]. La GenAI ci permette, quindi di avere nuovi strumenti e di sfruttare nuove opportunità, senza per forza invalidare quelle precedentemente esistenti.

Analisi dei Requisiti

Come illustrato nella sezione precedente, è possibile estrarre in modo efficace i requisiti da documentazione testuale o persino da trascrizioni di meeting, identificando automaticamente elementi chiave come funzionalità richieste, attori coinvolti e vincoli. Questa capacità risulta particolarmente utile per supportare la generazione di documenti funzionali più strutturati, come gli AFU, o per trasformare i requisiti raccolti in User Story ben definite, complete di criteri di accettazione e dettagli contestuali.

Oltre alla generazione di contenuti, la GenAI può essere impiegata anche per la validazione dei requisiti e, più in generale, per il miglioramento della qualità della documentazione. Ad esempio, è possibile effettuare un'analisi automatizzata per individuare eventuali incongruenze, ambiguità o omissioni tra i requisiti, segnalando potenziali conflitti prima che diventino problematici. Questo approccio consentirebbe di ottenere documenti più coerenti, leggibili e affidabili, riducendo il rischio di interpretazioni errate e facilitando il lavoro.

Un ulteriore ambito di applicazione, più trasversale, riguarda la creazione di chatbot basati su LLM, su cui è stato effettuato Fine-Tuning e/o RAG (Retrieval-Augmented Generation) con informazioni specifiche del dominio di riferimento. Questi assistenti intelligenti possono fungere da esperti virtuali, rispondendo con precisione a domande spcifiche del dominio e delle progettualità passate, fornendo supporto continuo su tematiche complesse, rendendo più semplice l'accesso alle informazioni e migliorando l'efficienza operativa.

Studi recenti confermano che la GenAI viene già applicata con successo in una vasta gamma di compiti [29], inclusi quelli sopra menzionati. Tuttavia, sebbene il suo utilizzo sia promettente, permane la necessità di una supervisione umana.

Ricerche dedicate evidenziano che, accanto ai notevoli vantaggi offerti, è essenziale validare gli output generati per evitare bias, incongruenze o errori che potrebbero compromettere l'affidabilità delle informazioni prodotte [30]. Questo tema, benché vero in generale, è particolarmente importante in questo contesto, in quanto i risultati qui generati vanno poi a costruire i documenti formali su cui saranno basate le fasi successive dello sviluppo.

Progettazione

Durante la fase di progettazione software, la GenAI sta aprendo nuove possibilità nel supportare gli architetti nella creazione di soluzioni. Come già osservato in altri ambiti, il suo impiego si configura principalmente come un supporto e un potenziamento del lavoro umano. Questo perché i risultati prodotti dai modelli richiedono sempre un certo grado di supervisione, specialmente in contesti critici come quello della progettazione, dove eventuali errori tendono a propagarsi nelle fasi successive dello sviluppo.

Un primo esempio è dato dalla possibilità di istruire il modello con le specifiche dei requisiti, ottenendo in output possibili strutture architetturali o schemi di design. Esperimenti di progettazione collaborativa con ChatGPT, ad esempio, hanno dimostrato che il modello è in grado di articolare e raffinare artefatti architetturali a partire da input in linguaggio naturale, arrivando in parte a simulare il ruolo di un architetto software umano [31]. Questa collaborazione può accelerare e migliorare il lavoro di definizione dell'architettura di alto livello, proponendo bozze che i progettisti possono poi perfezionare o, viceversa, proponendo affinamenti alle soluzioni già progettate o in progettazione.

Un altro impatto significativo è presente nella generazione di diagrammi UML e altri modelli grafici. Tradizionalmente la creazione di diagrammi - come quelli di caso d'uso, di classe o di sequenza - richiede tempo e competenza. Oggi la GenAI può assistere anche in questo. Uno studio esplorativo condotto con studenti ha evidenziato che strumenti basati su LLM permettono anche a persone non ancora esperte di produrre diagrammi a partire da requisiti testuali [32], precisando che, benché possa essere di aiuto, non viene garantita accuratezza. In un altro studio viene anche evidenziato come tenda ad amplificare gli errori se i requisiti non sono scritti chiaramente [33]. È quindi richiesta una attenta supervisione anche in questi casi.

Un ultimo esempio può essere dato come sempre dal supporto alla generazione della documentazione, in questo caso di natura tecnica. La fase di design prevede infatti la produzione di una serie di documenti fondamentali, che fungono sia da validazione formale della soluzione proposta, sia da guida per gli sviluppatori incaricati dell'implementazione.

Implementazione

La fase di implementazione è forse quella che più sta venendo rivoluzionata dalla GenAI. Le molteplici aree di applicazione e le competenze avanzate degli utilizzatori sono state il volano verso una trasformazione sostanziale riguardante come si progetta, scrive, documenta e testa il codice.

Strumenti avanzati come GitHub Copilot [34] o Machinet [35] puntano ad assistere gli sviluppatori come se fossero veri e propri Pair Programmer, suggerendo frammenti di codice in tempo reale e offrendo al contempo una chat contestuale interrogabile direttamente da dentro l'IDE. Queste opportunità non solo aumentano potenzialmente la produttività dello sviluppatore, ma consentono anche di concentrarsi su attività più strategiche e creative, favorendo l'innovazione e migliorando potenzialmente la qualità complessiva del software prodotto. Tra le capacità offerte da questi tool vi è la possibilità di produrre porzioni di codice, scrivere la documentazione accessoria, generare unit test, valutare possibili miglioramenti e refactor a soluzioni già esistenti, offrire supporto per eventuali errori e molto altro.

Come si può notare, si va a colpire una gran parte delle attività ordinarie di un programmatore medio, portando con sé un sicuro vantaggio allo sviluppo. Ad esempio, uno studio condotto presso Microsoft, Accenture e una multinazionale manifatturiera ha coinvolto quasi 5.000 sviluppatori, mostrando un aumento del 26% nel numero di task completati tra coloro che utilizzavano Copilot. Questo incremento è stato particolarmente significativo tra i programmatori meno esperti, che hanno registrato guadagni di produttività superiori rispetto ai colleghi con più esperienza [36].

Tuttavia, rimane cruciale l'attenzione alla qualità del codice generato. Diversi studi hanno evidenziato che una percentuale significativa del codice prodotto può contenere vulnerabilità. Ad esempio, una ricerca ha rilevato che il 29.8% del codice generato da Copilot presentava vulnerabilità di sicurezza [37]. Queste possono derivare da difetti nel codice o da pratiche di programmazione non sicure, rendendo il software suscettibile ad attacchi. Pertanto, è essenziale che gli sviluppatori revisionino attentamente il codice suggerito dall'AI, implementando le migliori pratiche disponibili in termini di sicurezza. La revisione del codice generato da un'AI non è un'opzione ma un atto dovuto per garantire l'affidabilità, la sicurezza e la correttezza formale del software prodotto.

Testing

La fase di testing è tradizionalmente una delle più dispendiose in termini di tempo e risorse richieste. I modelli generativi offrono oggi nuove opportunità in questo ambito, a partire dalla generazione automatica di test unitari. Come accennato nel

paragrafo precedente, un LLM può generare suite di test – complete di asserzioni e input di prova – partendo dal codice sorgente di una funzione o dalla descrizione della sua funzionalità. Questo approccio è in grado di ampliare la copertura di test in modo rapido, riducendo drasticamente il carico manuale, pur mantenendo (e talvolta migliorando) la capacità di scovare errori. Tuttavia, è emerso da studi accademici che la complessità del problema impatta significativamente le performance dei modelli: al crescere della difficoltà del codice o dello scenario da testare, anche i modelli più avanzati faticano a generare test case corretti e significativi. Ciò è dovuto principalmente a limiti nel ragionamento computazionale dei modelli attualmente esistenti – ad esempio determinare l'output atteso di funzioni molto complesse può essere arduo per un LLM puro. Per alleviare questi problemi, sono in sviluppo approcci ibridi come l'integrazione di LLM con strumenti esecutivi: ad esempio, un framework multi-agente realizzato attraverso la tecnica ReAct ha permesso all'AI di interagire con un interprete Python per verificare in tempo reale gli output, migliorando sensibilmente l'accuratezza dei test generati [38].

Un altro ambito in cui la GenAI può offrire un contributo significativo è la scrittura dei casi di test destinati alla validazione utente. Partendo dai requisiti funzionali e dalle informazioni sul sistema esistente, i modelli generativi possono supportare la produzione o il miglioramento di test funzionali, di performance e di UI/UX. A questo si affianca la possibilità di automatizzare anche la stesura dei report che seguono l'esecuzione dei test, snellendo ulteriormente il processo.

In un'ottica di automazione più estesa, è possibile prevedere anche l'esecuzione orchestrata delle suite di test e un'analisi intelligente dei risultati. Strumenti specializzati, alimentati da AI, possono gestire l'esecuzione su diversi ambienti, analizzare i log generati e identificare rapidamente le cause dei fallimenti. Un assistente AI, ad esempio, potrebbe leggere i messaggi di errore ed effettuare una prima diagnosi ("Test fallito per NullPointerException nel modulo X: aggiungere il seguente codice alla riga Y. . . "), accelerando così la localizzazione e la risoluzione dei problemi da parte dello sviluppatore.

Infine, un'ulteriore frontiera è rappresentata dall'identificazione predittiva dei bug. Oltre alla classica analisi statica del codice, si possono introdurre strumenti basati su LLM capaci di segnalare porzioni di codice potenzialmente problematiche o ricorrenti pattern di errore, prima ancora dell'esecuzione. In questo modo, l'AI – attingendo alla conoscenza appresa da milioni di frammenti di codice – è in grado di riconoscere costrutti sospetti e contribuire a una sorta di debug anticipato, riducendo il rischio di errori a valle.

Rilascio

La GenAI sta trovando applicazione anche nella fase di rilascio del SDLC, potenziando le pipeline di distribuzione e contribuendo a migliorare sia la velocità di delivery sia la qualità complessiva del codice [39]. L'integrazione all'interno dei flussi di Continuous Integration/Continuous Delivery (CI/CD), ha dato avvio a sperimentazioni orientate all'automazione di attività come la generazione automatica delle note di rilascio a partire dal changelog, traducendo le modifiche in linguaggio naturale per una comunicazione più efficace con gli stakeholder. Altre applicazioni includono la gestione automatizzata delle attività post-commit — dall'analisi del codice all'integrazione nei branch di sviluppo, dall'esecuzione dei test alla preparazione dei pacchetti di rilascio — oltre al supporto nella progettazione e configurazione delle pipeline stesse.

Questa evoluzione ha portato all'emergere di nuovi paradigmi, come DevGenOps [40]: un approccio che fonde i principi del DevOps tradizionale con le potenzialità della GenAI, con l'obiettivo di innalzare ulteriormente il livello di automazione nei processi di sviluppo e distribuzione. Uno degli elementi centrali di questa filosofia è la Continuous Generation, ovvero l'impiego di modelli generativi per la creazione automatica di artefatti direttamente all'interno delle pipeline di build e deployment.

In una prospettiva più ampia, considerando l'intero spettro delle tecnologie AI, si aprono scenari che includono l'uso di strumenti predittivi e decisionali per individuare le finestre temporali ottimali di rilascio, riducendo l'impatto sugli utenti grazie all'analisi dei dati di utilizzo. Parallelamente, l'analisi dei pattern di consumo delle risorse permette di individuare possibili ottimizzazioni lungo le diverse fasi del processo, con benefici in termini di minore utilizzo delle risorse, prevenzione dei colli di bottiglia e accelerazione dei cicli di consegna.

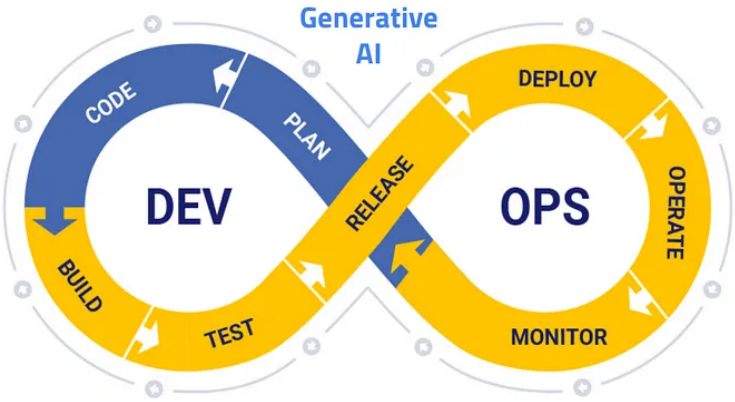

Figura 2.6: DevGenOps [40]

Manutenzione

La fase di manutenzione beneficia in modo significativo dell'AI generativa, sia in modalità reattiva che proattiva.
Sul fronte predittivo, la GenAI può analizzare i flussi di monitoraggio (log applicativi, metriche di performance, tracce di errori) per individuare pattern anomali o segnali precoci di possibili problemi, allertando il team prima che si verifichino malfunzionamenti critici. Ad esempio, un modello addestrato sui log di un sistema può riconoscere che una certa sequenza di eventi porta tipicamente a un crash, generando un avviso e suggerendo azioni correttive, come il riavvio di un servizio o la pulizia di una coda. Questo approccio aumenta l'affidabilità del software in esercizio e riduce il rischio di downtime imprevisti.

Un altro ambito riguarda le attività di miglioramento continuo del codice (ottimizzazioni, refactoring tardivi, aggiornamento di librerie) e la correzione di bug o vulnerabilità scoperte dopo il rilascio. Se, ad esempio, viene divulgata una nuova vulnerabilità in una libreria, un sistema AI può scandire il codebase per individuare i punti in cui essa viene utilizzata e proporre una patch o un aggiornamento di versione. In questo modo la gestione delle vulnerabilità diventa più rapida ed efficace: l'AI non solo individua i problemi, ma suggerisce anche soluzioni, mantenendo il software sicuro e conforme nel tempo.

In parallelo, la GenAI può contribuire all'efficientamento del codice. Parti dell'applicazione poco performanti (ad esempio funzioni con elevato consumo di CPU o memoria) possono essere riscritte in versioni più efficienti. La ricerca si sta muovendo in questa direzione, con LLM specializzati nell'ottimizzazione che trattano il miglioramento del codice come un problema di generazione: data una funzione, il modello ne propone una variante più veloce e leggera. [41].

Guardando oltre, verso scenari più avanzati, si possono immaginare sistemi agentici in grado di intervenire sia sul piano operativo che su quello di monitoraggio. Sul primo fronte, un sistema AI potrebbe automatizzare (anche solo in parte) la gestione dei ticket, classificandoli e proponendo soluzioni. Sul secondo, un esempio è un tool di Root Cause Analysis capace di analizzare i dati provenienti da strumenti di osservabilità (come Splunk per i log o Dynatrace per le performance) e da sistemi di versionamento (come i commit Git), individuando il componente più probabilmente responsabile di un malfunzionamento e suggerendo le azioni correttive più adatte.

Infine, una forma di AI più generale può essere impiegata per l'ottimizzazione delle risorse in produzione, grazie a meccanismi come l'auto-scaling intelligente o il tuning dinamico dei parametri di sistema, garantendo performance ottimali anche al variare del carico.

2.4 Prompt Engineering: la chiave per sfruttare gli strumenti GenAI

La natura intrinsecamente non deterministica degli LLM porta come inevitabile conseguenza che essi non generino mai risposte fisse: anche a fronte dello stesso input, l'output può variare. Senza una formulazione accurata e strategica del prompt, il modello potrebbe produrre risposte incoerenti o divergenti dall'obiettivo prefissato.

Il Prompt Engineering rappresenta una disciplina emergente nel campo dell'intelligenza artificiale dedicata alla progettazione, ottimizzazione e raffinamento degli input (o "prompt") forniti agli LLM al fine di ottenere output il più possibile accurati, coerenti e pertinenti. In sostanza, un Prompt è un input in linguaggio naturale che guida il modello nell'esecuzione di un compito specifico, e la qualità di tale input è determinante per l'efficacia della risposta generata. La formulazione di un prompt efficace non si riduce alla mera scelta di parole, ma implica l'organizzazione del contenuto in modo tale da fornire contesto, specificare il tono e delineare eventuali vincoli stilistici o narrativi, elementi che possono influenzare significativamente il risultato finale. Questa ingegnerizzazione del testo in ingresso non solo permette di sfruttare al massimo le capacità generative dei modelli, ma contribuisce anche a ridurre i fenomeni indesiderati, quali allucinazioni e i bias ereditati dai dati di addestramento, garantendo così risultati più robusti ed affidabili.

L'importanza del Contesto

Un elemento essenziale nel Prompt Engineering [18] è la capacità di fornire un contesto dettagliato e mirato per l'attività richiesta. Il contesto funge da cornice interpretativa che orienta l'LLM nel reperire e selezionare dai propri dati di addestramento le informazioni più rilevanti ai fini del compito. In pratica, un prompt accompagnato dal giusto contesto riduce l'ambiguità e aiuta a delimitare l'ambito della risposta, rendendola più coerente con gli obiettivi prefissati. Ad esempio, specificare nel prompt il dominio applicativo, il tono comunicativo o il ruolo da assumere può fare una grande differenza: un'istruzione come *"Sei un insegnante che spiega a uno studente delle elementari il concetto di relatività"* guiderà il modello verso un registro semplice e intuitivo, diverso da un generico *"Parlami del teorema della relatività"*. Includere questi dettagli contestuali stabilisce sin dall'inizio chi parla, a quale scopo e in che contesto, permettendo al modello di adattare di conseguenza stile e contenuto della risposta.

Per formalizzare tali indicazioni, è stato proposto il framework RGC (Role, Goal, Context). Esso suggerisce di strutturare il prompt suddividendolo in sezioni ben definite — tre obbligatorie (ruolo, obiettivo e contesto) e due opzionali (risultato

atteso e vincoli). In altre parole, nel prompt andrebbero specificati: Role, il ruolo che l'LLM deve impersonare; Goal, l'obiettivo o compito specifico da svolgere; Context, il contesto situazionale o applicativo; ed eventualmente anche il Result atteso (il formato o prodotto desiderato) e i Constraint da rispettare (limitazioni stilistiche, lunghezza, tono, ecc.). Ad esempio, seguendo RGC si potrebbe scrivere:

In qualità di ingegnere informatico [role], spiega il concetto di ricorsione [goal] nel contesto dei linguaggi di programmazione [context]. Usa esempi semplici per illustrare il processo [constraint].

Questo approccio strutturato fornisce al modello una guida chiara su chi deve essere, cosa deve fare, di cosa deve parlare e come farlo, aumentando così le probabilità di ottenere un output pertinente e mirato. Tali accorgimenti rientrano tra le pratiche di base del prompt engineering e risultano efficaci nel migliorare la coerenza e la specificità delle risposte generative.

Inoltre, fornire il contesto adeguato aiuta anche a mitigare fenomeni indesiderati come le allucinazioni. Contestualizzare in modo preciso la domanda significa delimitare il campo delle possibili risposte alla conoscenza pertinente, riducendo la probabilità che il modello "inventi" fatti non accurati.

2.4.1 Tecniche

Di seguito verranno presentate tutte le tecniche più rilevanti disponibili al momento della stesura di questo documento. Benché ognuna abbia le proprie caratteristiche peculiari, è possibile e consigliabile combinarle per ottenere risultati ancora migliori.

Per ulteriori dettagli e per tecniche più avanzate, si può fare riferimento ad [17] e [42].

Zero-Shot Prompting

Lo Zero-Shot Prompting consiste nel fornire al modello una richiesta diretta, senza alcun esempio esplicativo. Il modello si affida interamente alla sua conoscenza pre-addestrata per interpretare l'istruzione e generare il testo. È particolarmente utile quando il compito richiesto è abbastanza generico o quando il modello ha già acquisito sufficienti competenze sul genere in questione durante il suo training. Tuttavia, senza ulteriori indicazioni, il rischio è che l'output possa essere troppo generico o non allineato perfettamente alle esigenze specifiche del compito.

Esempio:
"Scrivi un haiku il cui tema è la pioggia."

Few-Shot Prompting

Il Few-Shot Prompting arricchisce l'istruzione fornita al modello includendo uno o pochi esempi esplicativi che illustrano il formato, lo stile e il tono desiderati. Questi esempi servono come guida, mostrando al modello come strutturare il risultato finale. Come dimostrato da Language Models are Few-Shot Learners [27], negli LLM moderni bastano un numero basso di esempi per raggiungere risultati molto migliori rispetto alla variante Zero-Shot. Come si può evincere dall'immagine, l'accuratezza migliora all'aumentare del numero di esempi forniti, rallentando la propria effiacia, però, una volta che si raggiungono numeriche abbastanza basse. L'utilizzo di esempi ben scelti può ridurre l'ambiguità e aumentare la coerenza dell'output, sebbene la qualità degli esempi stessi sia fondamentale per il successo del prompt.

Esempio:
"Scrivi un haiku il cui tema è la pioggia.
Segui lo stile e la struttura degli esempi qui sotto:

Es. 1:

> *Lacrime di pioggia*
> *bagnano l'anima stanca*

di vecchi addii.

Es. 2:

Gocce leggere
bagnano il vecchio ponte
sotto il cielo grigio."

Chain-of-Thought (CoT)

Il Chain-of-Thought, introdotto da Chain-of-Thought Prompting Elicits Reasoning in Large Language Models [43], è una tecnica che incoraggia il modello a esplicitare una sequenza di passaggi intermedi — una sorta di "pensare ad alta voce" — prima di fornire la risposta finale.
Invece di limitarsi al solo output finale, il modello scompone il problema in sottopassi coerenti e verificabili, rendendo il processo più trasparente e, spesso, più accurato nei compiti che richiedono ragionamento aritmetico o simbolico. Questa esplicitazione dei passaggi consente anche a chi usa il sistema di individuare e correggere eventuali errori intermedi, migliorando la controllabilità del risultato.

La letteratura mostra che il CoT funziona sia in modalità Few-Shot, fornendo nel prompt alcuni esempi di ragionamento passo-passo, sia anche in ZeroShot, aggiungendo soltanto un trigger linguistico come "Rifletti passo dopo passo".

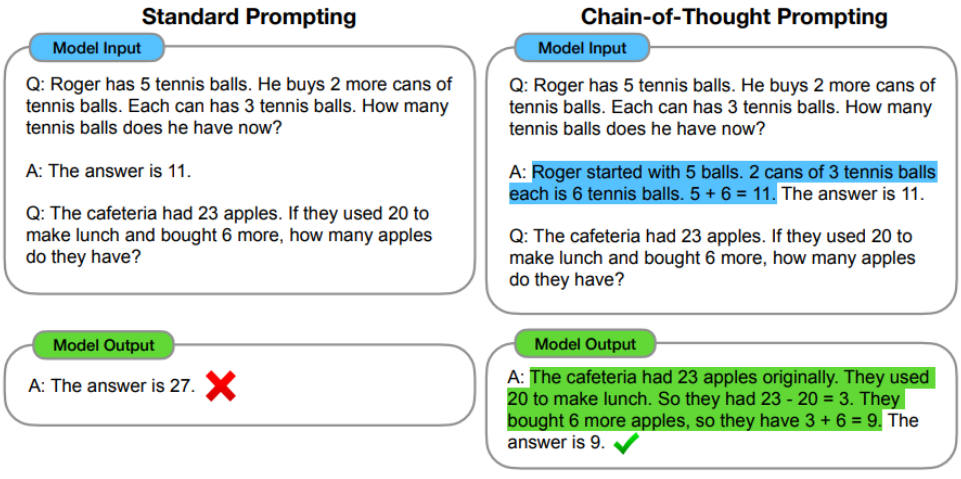

Figura 2.7: Chain-of-Thought [43]

Self Consistency

La Self Consistency [44] applicata al CoT si fonda sull'osservazione che, nei compiti complessi, possono coesistere molteplici percorsi di ragionamento ugualmente validi in grado di condurre alla stessa risposta corretta. Anziché affidarsi a un'unica traiettoria, il metodo esplora più catene di ragionamento e seleziona la risposta finale più frequente tramite voto di maggioranza (Majority Vote). In questo modo, eventuali errori isolati vengono compensati e si ottiene una risposta finale più robusta e accurata.

Operativamente, la procedura si articola in tre passaggi, come da figura 2.8:

1. fornire al modello un prompt che induca il CoT;
2. sostituire il Greedy Decoding con un campionamento dal decodificatore del modello, così da generare un insieme diversificato di percorsi di ragionamento;
3. marginalizzare i percorsi e aggregare gli esiti selezionando, tra le risposte finali, quella più coerente/frequente.

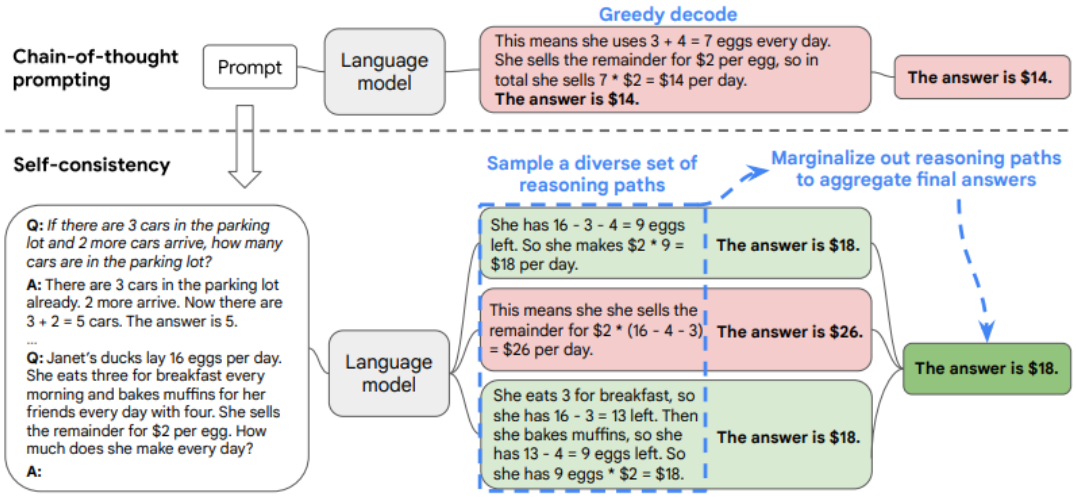

Figura 2.8: Self Consistency [44]

Tree of Thoughts (ToT)

Il Tree of Thoughts (ToT) [45] rappresenta una generalizzazione del CoT, passando da un ragionamento lineare a uno strutturato ad albero in cui il problema viene suddiviso in piccoli passaggi. Il modello genera molteplici possibilità ad ogni fase e valuta, tramite criteri come il punteggio o la votazione, quali percorsi sono più promettenti per arrivare a una soluzione. Questa struttura ad albero permette al

modello di esplorare diverse soluzioni in parallelo e di effettuare eventualmente backtracking quando un percorso non conduce alla risposta corretta. Questo approccio permette di migliorare le abilità di risoluzione dei problemi e migliorare la gestione delle incertezze.

Operativamente, il ToT propone un approccio pratico che parte dalla scelta della giusta granularità dei pensieri — che possono essere parole, righe di equazioni o interi paragrafi, a seconda del compito. A questo si affianca la progettazione dei prompt di generazione e di valutazione, che possono includere anche meccanismi di voto tra diversi stati, e la selezione dell'algoritmo di esplorazione più adatto, come una ricerca in ampiezza con beam search limitato o una ricerca in profondità con soglie di potatura. In questo quadro, tecniche come CoT e Self-Consistency possono essere interpretate come varianti semplificate di alberi con ampiezza o profondità limitata, mentre ToT esplicita in modo più sistematico sia il processo di esplorazione sia la selezione informata dei rami (si veda figura 2.9).

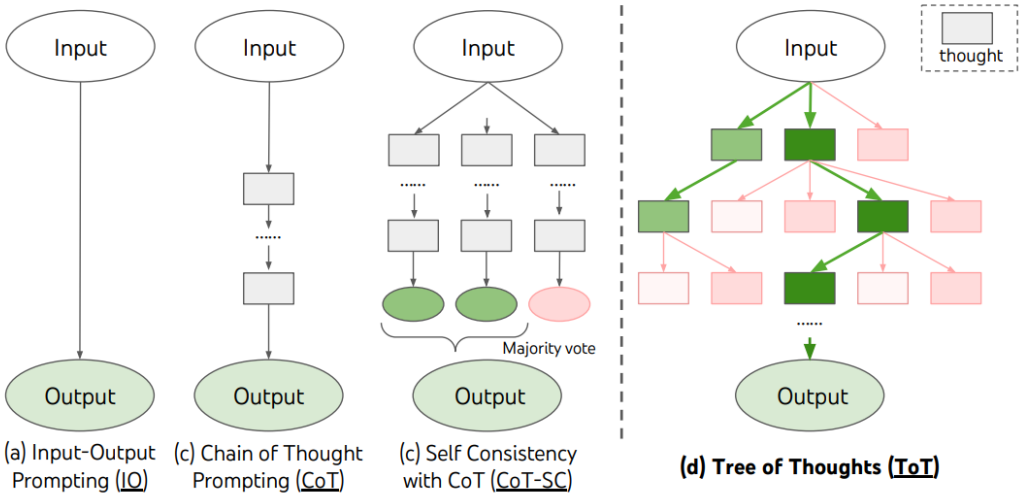

Figura 2.9: Schema illustrativo dei vari approcci alla risoluzione dei problemi con gli LLM. Ogni riquadro rettangolare rappresenta un pensiero (thought), ovvero una sequenza linguistica coerente che funge da passaggio intermedio verso la risoluzione del problema. [45]

Reasoning and Acting (ReAct)

ReAct [46] è un approccio che integra in modo sinergico la componente di ragionamento con l'esecuzione di azioni. Analogamente al CoT puro, il modello viene guidato a produrre una sequenza di passaggi intermedi che spiegano il proprio

ragionamento (Reason). La differenza sostanziale è che, ad ogni passo, il modello può decidere di eseguire una o più azioni (Action), volte a raccogliere nuove informazioni o a compiere operazioni utili alla risoluzione del problema.

Queste azioni possono assumere forme diverse: la produzione di output strutturati, l'invocazione di strumenti esterni (ad esempio il recupero di informazioni da un motore di ricerca o un database), oppure l'esecuzione di comandi specifici in un determinato ambiente. Dopo ogni azione, il sistema riceve un'osservazione (Observation), ossia un feedback che descrive l'esito dell'azione intrapresa. L'osservazione viene quindi integrata nel flusso di ragionamento, aggiornando il piano e consentendo al modello di adattarsi in base alle nuove informazioni.

Il ciclo *Thought* → *Action* → *Observation* si ripete iterativamente, dando vita a un processo di ragionamento dinamico ed interattivo. In questo modo, il sistema non si limita a seguire un piano predefinito, ma può formulare ipotesi, verificarle con azioni mirate, correggere eventuali errori e arricchire la propria base informativa man mano che procede. Questo approccio riduce la probabilità di allucinazioni e la propagazione di errori, poiché il modello non si affida unicamente alla conoscenza interna, ma si confronta con risorse esterne e osservazioni reali.

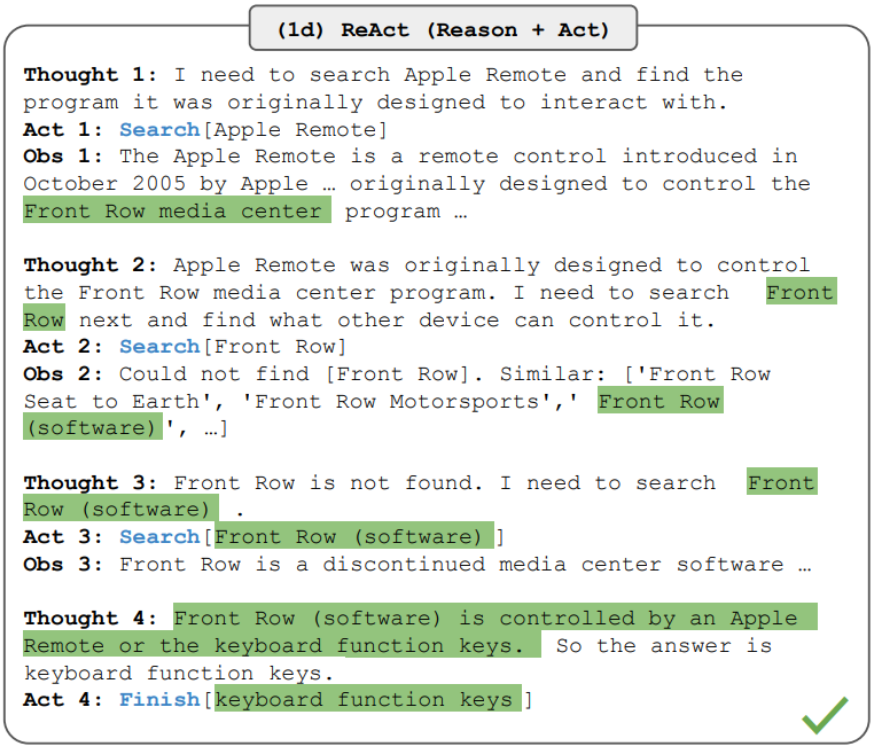

Figura 2.10: ReAct (Reason+Act). [46]

Come si può intuire, ReAct non è soltanto un'estensione del CoT, ma una modalità che trasforma il ragionamento in un ciclo interattivo di esplorazione e verifica: il modello ragiona, agisce, osserva i risultati e aggiorna di conseguenza la propria strategia fino a raggiungere la soluzione.

Prompt Chaining

Per migliorare l'affidabilità e le prestazioni degli LLM, una delle principali tecniche disponibili consiste nella decomposizione dei compiti complessi in sotto-compiti più gestibili. In questo contesto si colloca il paradigma del Prompt Chaining, che prevede la suddivisione di un problema in una sequenza di fasi, in cui l'output di ciascuna diventa l'input della successiva.

Questo approccio modulare consente di affrontare il problema in maniera incrementale: ogni fase si concentra su un aspetto specifico, permettendo di ridurre la complessità e di mantenere il controllo sull'evoluzione della soluzione. Il risultato finale non è quindi generato in un unico passaggio, ma costruito progressivamente attraverso una catena di trasformazioni guidate.

Il Prompt Chaining si rivela particolarmente utile nei casi in cui un LLM avrebbe difficoltà a gestire un compito se sollecitato tramite un singolo prompt molto lungo e articolato. Oltre a migliorare le prestazioni complessive, questo metodo contribuisce anche ad aumentare la trasparenza e la controllabilità del processo: la presenza di output intermedi rende possibile analizzare il comportamento del modello passo dopo passo, facilitando attività di debugging e verifica della correttezza logica delle singole fasi.

Retrieval-Augmented Generation (RAG)

Gli LLM incorporano una vasta quantità di conoscenza nei propri parametri, ma tale conoscenza è difficile da aggiornare e tende a diventare rapidamente obsoleta. Inoltre, l'accesso puntuale a contenuti specialistici rimane complesso, soprattutto quando si tratta di informazioni di nicchia o di dominio ristretto. Queste limitazioni derivano anche dal fatto che l'addestramento degli LLM richiede tempi e risorse considerevoli, rendendo impraticabile un aggiornamento frequente dei dati di training.

Il Retrieval-Augmented Generation (RAG) [47] [48] è una tecnica che affronta in modo diretto queste criticità collegando la generazione del modello a una memoria non parametrica esterna. In pratica, a ogni query il sistema effettua un'operazione di retrieval, recuperando evidenze da un archivio di conoscenza — ad esempio un database documentale, fonti aziendali o risorse aperte come Wikipedia — e

condiziona la generazione sulle informazioni reperite. Questo arricchimento consente di migliorare la specificità, la fattualità e la tracciabilità delle risposte.

L'idea è stata formalizzata principalmente nel paper Retrieval-augmented generation for knowledge-intensive NLP tasks [47], dove è stata proposta la combinazione di un modello seq2seq con un indice denso di passaggi testuali, dimostrando progressi significativi in task knowledge-intensive. Operativamente, l'implementazione tipica del RAG prevede l'uso di un database vettoriale contenente gli embedding dei documenti esterni. Quando l'utente pone una domanda, il sistema confronta la query con lo spazio vettoriale, individua i documenti più pertinenti e li utilizza per arricchire il prompt inviato al modello, che genera così una risposta fondata sia sul contesto originale sia sull'evidenza recuperata.

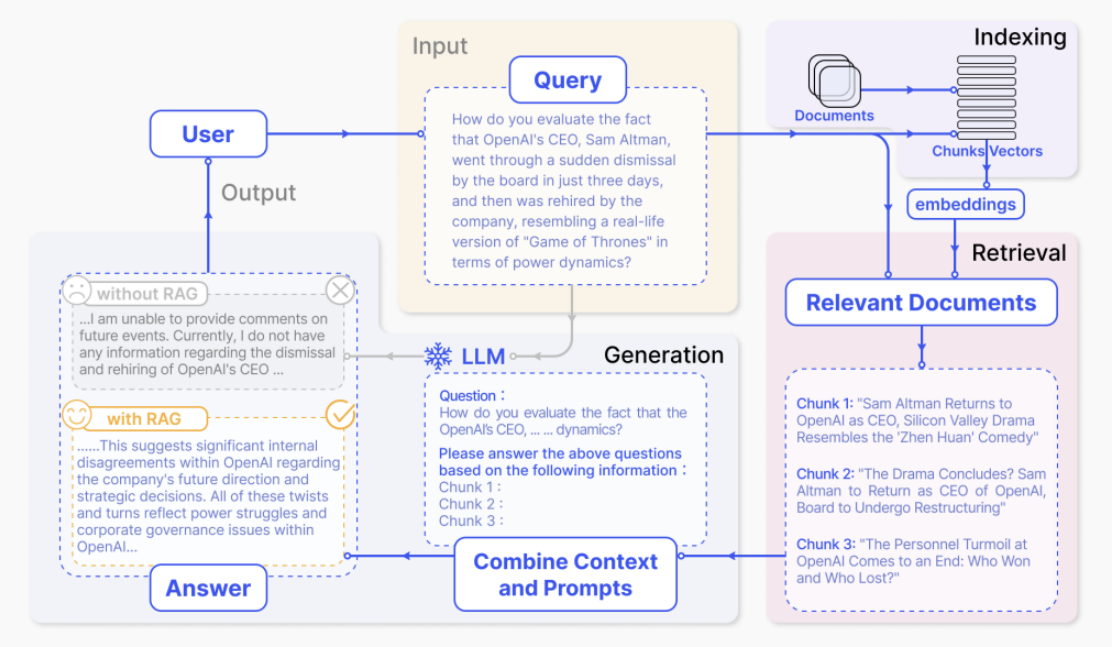

Figura 2.11: Un esempio rappresentativo del processo RAG applicato alla risposta a domande [49]. Consiste principalmente di 3 fasi:
1) Indicizzazione. I documenti vengono suddivisi in blocchi, codificati in vettori e memorizzati in un database vettoriale.
2) Recupero. Recuperare i k blocchi più rilevanti per la domanda in base alla somiglianza semantica.
3) Generazione. Inserire la domanda originale e i blocchi recuperati insieme nell'LLM per generare la risposta finale

Rispetto alla formulazione originaria, oggi il paradigma RAG può essere esteso in chiave più dinamica grazie all'integrazione con tool esterni e sistemi agentici

[49]. L'LLM non si limita a consultare una memoria statica, ma può interagire attivamente con strumenti di ricerca, API, basi dati in tempo reale o pipeline di analisi, aggiornando continuamente il proprio contesto informativo. In questo modo, il retrieval non è più un'operazione isolata, ma diventa parte di un ciclo iterativo in cui il modello recupera, interpreta, agisce e raffina progressivamente la risposta.

Questo approccio si dimostra particolarmente utile in contesti aziendali o settoriali, dove l'accesso a informazioni aggiornate e affidabili è cruciale. Grazie all'integrazione di fonti autorevoli, il RAG riduce la probabilità che il modello produca contenuti inesatti o fuorvianti, garantendo risposte più accurate, pertinenti e verificabili.

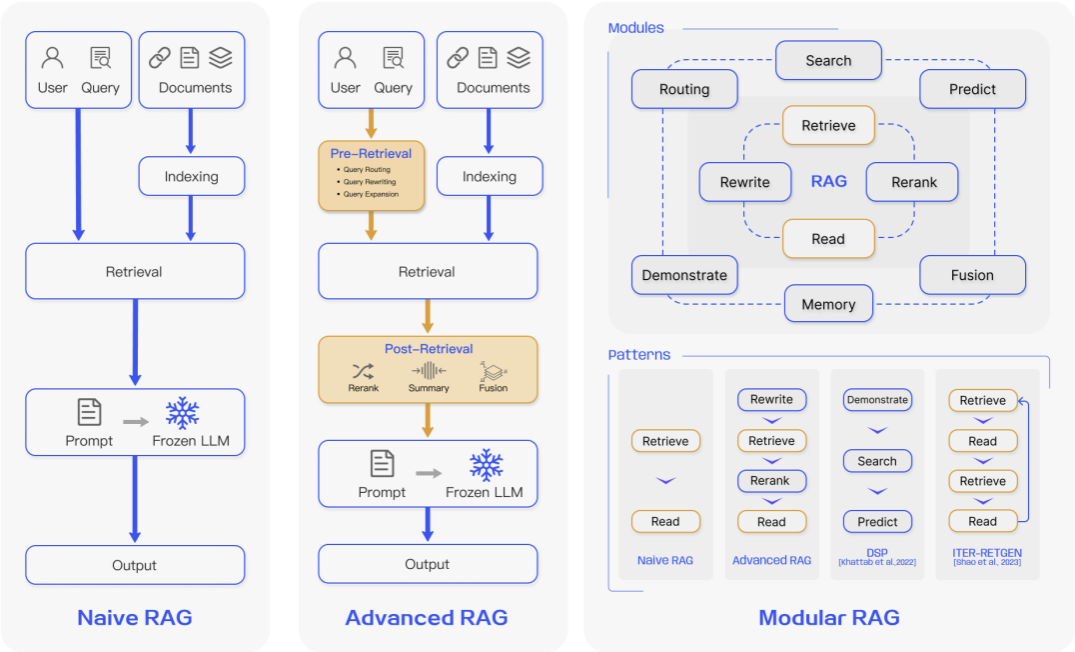

Figura 2.12: Confronto tra i tre paradigmi di RAG [49].
(Sinistra) Il RAG naive consiste principalmente di tre parti: indicizzazione, recupero e generazione.
(Centro) Il RAG avanzato propone diverse strategie di ottimizzazione relative al pre-recupero e al post-recupero, con un processo che segue comunque una struttura a catena, come per la versione naive.
(Destra) Il RAG modulare, invece, eredita e sviluppa il paradigma precedente, dimostrando una maggiore flessibilità complessiva. Ciò è evidente nell'introduzione di più moduli funzionali specifici e nella sostituzione dei moduli esistenti. Il processo complessivo non si limita al recupero e alla generazione sequenziali, ma include metodi come il recupero iterativo e adattivo.

2.5 Rischi e sfide della GenAI

L'impiego di strumenti di Generative AI nel SDLC offre numerosi vantaggi immediatamente percepibili dagli utilizzatori, migliorando produttività, velocità e qualità complessiva del processo. Tuttavia, insieme a questi benefici emergono anche rischi e criticità, talvolta meno evidenti nel breve e medio periodo, che richiedono un'attenta valutazione e strategie mirate di mitigazione.

In particolare, l'adozione di tali strumenti solleva questioni di natura tecnica, legale, organizzativa ed etica, che devono essere analizzate in modo sistematico per garantire un utilizzo consapevole e sostenibile. Inoltre, l'integrazione sempre più stretta con tool dinamici e sistemi agentici rende il quadro ancora più complesso: se da un lato aumenta la flessibilità e la capacità di adattamento dei modelli, dall'altro amplifica la necessità di controlli accurati, governance e monitoraggio continuo.

Alla luce di ciò, nei paragrafi successivi verranno esaminati in maniera strutturata i principali aspetti critici connessi all'uso della GenAI nel ciclo di vita del software, con l'obiettivo di delineare un approccio più maturo e consapevole alla loro adozione.

2.5.1 Tecniche

Un primo insieme di problematiche riguarda gli aspetti più strettamente tecnici del SDLC. Si tratta di rischi che possono manifestarsi in qualunque fase del processo, dalla raccolta dei requisiti fino al testing e alla manutenzione, e che incidono in maniera diretta sulla qualità complessiva del prodotto. Tali criticità possono compromettere la correttezza del codice, la sua robustezza e manutenibilità, nonché la capacità del software di soddisfare requisiti funzionali e non funzionali.

In particolare, le sfide tecniche derivanti dall'uso di strumenti di AI generativa non si limitano al mero aspetto implementativo: esse possono influenzare l'architettura complessiva, l'integrazione con sistemi preesistenti, l'affidabilità del ciclo di build e di deploy, fino ad arrivare alla gestione della sicurezza e alla conformità a standard industriali. Un codice generato rapidamente ma non verificato può introdurre errori nascosti che si propagano a valle, aumentando i costi di correzione in fasi più avanzate del progetto. Inoltre, la difficoltà nel garantire trasparenza e tracciabilità delle decisioni prese dal modello rende più complessa la revisione del software e la sua validazione rispetto a requisiti stringenti, come quelli tipici di settori regolamentati (finanza, sanità, pubblica amministrazione).

In altre parole, i rischi tecnici connessi alla GenAI non devono essere visti come problematiche isolate, ma come fattori che si intrecciano con le pratiche consolidate di ingegneria del software. Se non gestiti con strumenti e processi adeguati (code review, testing automatico, analisi statica, linee guida di sviluppo), tali rischi possono ridurre drasticamente i benefici attesi dall'adozione dell'IA, trasformando un potenziale acceleratore in una fonte di complessità aggiuntiva.

Bias nei modelli

Uno dei problemi più discussi nell'ambito della GenAI è la presenza di bias nei modelli generativi, una criticità che si colloca all'incrocio tra aspetti tecnici, metodologici ed etici. I modelli apprendono schemi e soluzioni da grandi insiemi di dati eterogenei, raccolti spesso da fonti pubbliche o da repository online. Se tali dati contengono distorsioni o squilibri – ad esempio, il predominio di un certo paradigma di programmazione, la sovra-rappresentazione di specifiche tipologie di soluzioni o la scarsità di contributi provenienti da comunità diverse – il modello tende a riprodurre automaticamente quelle stesse asimmetrie.

Il risultato sono output che riflettono pregiudizi impliciti, ipotesi non valide o soluzioni tecniche sistematicamente orientate verso approcci non ottimali. Bias meno evidenti possono tradursi in una sorta di "invisibile conformismo algoritmico": ad esempio, un modello che privilegia costantemente implementazioni tradizionali anche laddove esistono alternative più moderne, efficienti o più eque per il contesto.

Questi bias non abbiano soltanto una valenza etica (discriminazioni implicite, esclusione di approcci minoritari), ma abbiano anche ricadute dirette sulla qualità del software: un modello che incorpora assunzioni errate rischia di propagare errori logici nei sistemi generati, con effetti a cascata sull'affidabilità complessiva.

Allucinazioni e affidabilità delle risposte

Un secondo rischio tecnico di grande rilievo è rappresentato dalla tendenza dei modelli generativi a produrre contenuti inesatti o del tutto fittizi, fenomeno comunemente noto come Allucinazione (hallucination). Questo limite affonda le sue radici nel funzionamento intrinseco dei modelli di intelligenza artificiale: essi non possiedono una vera comprensione semantica del codice o del testo che producono, ma operano sulla base di predizioni statistiche, selezionando la sequenza di token più probabile a partire dal contesto fornito. Tale approccio, pur risultando estremamente efficace nel generare output plausibili e coerenti a livello superficiale, espone inevitabilmente al rischio che vengano proposte soluzioni formalmente corrette ma sostanzialmente errate.

Il pericolo principale risiede proprio nella plausibilità di questi output. A differenza degli errori evidenti, che vengono individuati rapidamente e scartati senza esitazione, le allucinazioni hanno l'aspetto di codice legittimo e funzionante, inducendo così lo sviluppatore ad accettarle come valide. Non è raro, ad esempio, che la GenAI generi funzioni in grado di compilare senza errori, ma che restituiscano risultati scorretti in particolari condizioni limite. In altri casi, vengono introdotte chiamate a procedure che non esistono, ma che appaiono del tutto credibili a chi legge, magari perché richiamano convenzioni di denominazione diffuse o librerie realmente esistenti. Ancora, si riscontrano test case apparentemente ben strutturati che però fraintendono aspetti fondamentali del dominio applicativo, con il rischio di validare erroneamente comportamenti non conformi ai requisiti di business.

Il carattere insidioso delle allucinazioni risiede dunque nella loro capacità di creare un falso senso di affidabilità. Uno sviluppatore, soprattutto se alle prime armi, può essere portato a integrare tali soluzioni senza verificarle con attenzione, convinto dalla loro forma sintatticamente impeccabile. Questo atteggiamento non solo facilita l'introduzione di bug logici difficili da rilevare in fase di testing, ma può anche compromettere la fiducia che il team ripone nel processo di sviluppo stesso: se il codice generato dall'IA si rivela inaffidabile, ogni fase successiva (dalla code review fino alla messa in produzione) risulta appesantita dalla necessità di ulteriori controlli.

Vulnerabilità

Un ulteriore aspetto tecnico di primaria importanza riguarda la sicurezza del codice generato. La natura stessa dei modelli generativi spiega il motivo di tale criticità: essi vengono addestrati su enormi corpora di codice reperibile in rete, che includono progetti open source di varia qualità, frammenti di documentazione tecnica, esempi da forum di programmazione e repository spesso non sottoposti ad alcun processo di validazione. In questo insieme di dati convivono tanto soluzioni eleganti e consolidate quanto pratiche obsolete, pattern deprecati e, non di rado, vulnerabilità note. Poiché il modello non possiede una comprensione intrinseca del concetto di "sicurezza", ma si limita a riprodurre correlazioni statistiche, non è in grado di distinguere autonomamente tra codice sicuro e codice pericoloso: ciò significa che debolezze e cattive pratiche presenti nei dati di training possono venire inconsapevolmente perpetuate negli output generati.

Le conseguenze non sono puramente teoriche. Esempi concreti documentati mostrano come i modelli suggeriscano con frequenza soluzioni vulnerabili. Un caso classico riguarda la concatenazione diretta di stringhe all'interno di query SQL, una pratica diffusa in molti vecchi progetti e che espone immediatamente al rischio di SQL injection.

La letteratura scientifica conferma la gravità del problema. Una ricerca condotta dalla Cornell University ha evidenziato che circa il 40% dei programmi completati con l'ausilio di GitHub Copilot presentava vulnerabilità di sicurezza [50]. Questo dato non sorprende se si considera che la probabilità di incontrare e riprodurre cattive pratiche aumenta proporzionalmente all'ampiezza del dataset di addestramento e al suo carattere eterogeneo. La capacità predittiva del modello, pur sofisticata, non implica in alcun modo la garanzia che la soluzione proposta sia sicura, robusta o conforme a standard di settore.

Di fronte a tale scenario, è evidente che un approccio di cieca fiducia verso le soluzioni generate dalla GenAI risulta intrinsecamente rischioso. Al contrario, l'uso di questi strumenti deve essere sempre incardinato all'interno di un contesto di pratiche consolidate di sicurezza del software. Tra queste, assumono un ruolo cruciale strumenti come le code review, l'impiego di strumenti di analisi statica e di analisi dinamica, l'integrazione di test di sicurezza specifici. A questi strumenti tecnici deve affiancarsi una consapevolezza organizzativa: le aziende che introducono la GenAI nei propri flussi di sviluppo hanno la responsabilità di formare adeguatamente il personale, sensibilizzando sul fatto che il codice generato non è mai "sicuro di default". Solo combinando la potenza generativa dei modelli con solide pratiche di ingegneria del software e con una cultura condivisa della sicurezza sarà possibile sfruttare appieno i benefici che essa può offrire.

Soluzioni generiche e mancata aderenza ai requisiti di business

Un altro limite sostanziale delle soluzioni presenti attualmente sul mercato è la scarsa comprensione del contesto progettuale e dei requisiti di business specifici. Gli output della GenAI, pur essendo sintatticamente corretti, spesso rappresentano soluzioni generiche, valide in media ma non ottimizzate per il particolare dominio o scenario d'uso dell'applicazione target.

Questo accade perché il modello non dispone di una visione globale degli obiettivi di progetto: manca della capacità di cogliere la finalità ultima del software, le priorità di business, le aspettative degli stakeholder e le nuance dei requisiti funzionali e non funzionali. Ad esempio, un modello potrebbe generare una funzione per calcolare prezzi senza tenere conto di regole aziendali peculiari (scontistiche, arrotondamenti normativi, ecc.). Allo stesso modo, vincoli non funzionali come performance, scalabilità, usabilità o conformità normative possono essere ignorati: studi evidenziano che l'IA tende a focalizzarsi sull'implementazione funzionale di base, trascurando considerazioni architetturali e di qualità del software.

Il risultato sono soluzioni distanti dalle aspettative, che richiedono interventi correttivi o addirittura reingegnerizzazioni, riducendo i benefici di produttività. Inoltre, poiché i modelli si basano su pattern consolidati, le soluzioni tendono a essere poco innovative: mancano cioè della capacità di introdurre idee progettuali originali o vantaggi competitivi, che invece un team umano può sviluppare ragionando in modo creativo sui requisiti.

Accumulo di debito tecnico

L'ultimo rischio tecnico riguarda l'accumulo di debito tecnico, fenomeno ben noto nello sviluppo software tradizionale ma potenzialmente amplificato in modo significativo dall'adozione della GenAI. Con il termine debito tecnico si intende il costo implicito derivante da scelte di sviluppo rapide e non ottimali, che consentono di ottenere un risultato immediato ma generano lavoro aggiuntivo futuro in termini di manutenzione, correzione o refactoring. Come ogni debito, esso può apparire vantaggioso nel breve periodo – permettendo di raggiungere obiettivi con maggiore velocità – ma comporta interessi che si accumulano progressivamente nel tempo, fino a gravare pesantemente sulla sostenibilità del progetto.

La GenAI, con la sua promessa di accelerare drasticamente la produzione di software in ogni fase del ciclo di vita, rischia di rafforzare dinamiche già presenti ma di solito più contenute. La disponibilità di strumenti in grado di generare codice in pochi secondi può indurre sviluppatori e team a privilegiare soluzioni rapide e superficiali, trascurando pratiche fondamentali di ingegneria del software come la progettazione modulare, la stesura di documentazione accurata, l'esecuzione di test sistematici o

la revisione del codice da parte di pari. In altre parole, la tentazione di "prendere scorciatoie" aumenta proporzionalmente alla percezione di poter contare su un alleato potente e veloce.

Benché il debito tecnico sia un rischio strutturale da sempre presente nello sviluppo software, l'introduzione della GenAI può portarlo a nuovi livelli di pericolosità. Se non si lavora in maniera sistematica per diffondere una cultura della qualità del codice e per creare, dove necessario, gli "anticorpi" organizzativi e metodologici adeguati, il rischio è quello di accumulare in tempi rapidissimi grandi quantità di codice apparentemente funzionante ma strutturalmente fragile. Tale codice, nel lungo termine, rallenta l'evoluzione del sistema, poiché ogni nuova funzionalità deve fare i conti con una base instabile, e aumenta sensibilmente i costi di manutenzione e gestione.

Le osservazioni provenienti dal mondo industriale e accademico confermano questa tendenza. Molti addetti ai lavori sottolineano che, senza un'attenta governance, la GenAI rischia di aumentare il carico tecnico complessivo anziché ridurlo, contribuendo ad accrescere quella "montagna di debito tecnico" che già caratterizza numerose codebase legacy. Indagini recenti rafforzano questa preoccupazione: sebbene il 92% degli sviluppatori dichiari un incremento del volume di codice prodotto grazie all'IA, una quota significativa segnala anche un ampliamento del "raggio di impatto" di codice di bassa qualità, ossia di porzioni che richiedono correzioni e rifattorizzazioni successive [51].

2.5.2 Organizzative

Oltre alle implicazioni di natura prettamente tecnica, l'introduzione della GenAI nei processi di sviluppo software porta con sé un insieme di rischi organizzativi che meritano un'attenta considerazione. Se, infatti, le criticità tecniche incidono direttamente sulla qualità e sull'affidabilità del codice prodotto, i rischi organizzativi si manifestano più sottilmente, influenzando il modo in cui i team operano, collaborano e si evolvono nel tempo.

Queste problematiche non riguardano soltanto la dimensione operativa quotidiana, ma toccano aspetti più profondi della vita aziendale: la distribuzione delle competenze tra professionisti, la capacità di mantenere un adeguato livello di autonomia tecnica, la definizione di policy interne coerenti e il delicato equilibrio tra produttività individuale e collaborazione collettiva. In altre parole, l'adozione della GenAI non rappresenta solo una questione di strumenti, ma anche e soprattutto una sfida culturale e organizzativa.

Le implicazioni possono assumere diverse forme. Da un lato, vi è il rischio di dipendenza eccessiva dagli strumenti di AI, che può condurre a un impoverimento progressivo delle competenze umane e alla formazione di quello che alcuni autori definiscono "debito di competenze". Dall'altro, emerge la necessità di definire politiche di governance chiare e condivise, senza le quali l'uso dell'IA rischia di diventare anarchico, esponendo l'organizzazione a problemi di sicurezza, conformità o gestione dei flussi di lavoro. Infine, occorre considerare l'impatto sulle dinamiche collaborative: la GenAI può alterare i processi consolidati di comunicazione e revisione, generando sia nuove opportunità sia potenziali attriti tra sviluppatori.

In questo senso, i rischi organizzativi non vanno interpretati come semplici corollari dei rischi tecnici, ma come fattori che ne moltiplicano o ne attenuano gli effetti. Un'organizzazione che non gestisce adeguatamente le proprie politiche interne, che non forma il personale all'uso critico degli strumenti o che non integra l'IA in una cornice di collaborazione strutturata, si espone infatti al pericolo di vedere compromessi non solo i risultati di singoli progetti, ma anche la propria capacità di innovazione e di crescita nel lungo periodo.

Dipendenza dagli strumenti AI e "debito di competenze"

L'adozione estesa di tool di AI generativa nel SDLC introduce inevitabilmente anche il rischio di una dipendenza eccessiva da tali strumenti, con conseguente impoverimento progressivo delle competenze umane. Questo fenomeno, che si manifesta in maniera sottile ma profonda, riguarda soprattutto i programmatori meno esperti: questi, potendo contare su un assistente capace di generare codice in tempi rapidissimi, possono sviluppare l'abitudine ad affidarsi totalmente al

sistema per la risoluzione dei problemi, limitandosi ad accettarne le soluzioni senza comprenderle a fondo.

Se tale tendenza si consolida, il rischio è quello di generare quello che possiamo definire un vero e proprio "debito di competenze". A breve termine, l'organizzazione guadagna in velocità e produttività apparente, beneficiando della capacità dell'IA di fornire risposte immediate a domande anche complesse. Tuttavia, sul lungo periodo, la perdita di conoscenze critiche e abilità fondamentali diventa inevitabile: competenze come il problem solving, il ragionamento algoritmico, la capacità di debugging o la padronanza di architetture complesse rischiano di atrofizzarsi, lasciando spazio a professionalità che sanno "chiedere all'AI" ma non più "fare da sé". In questo senso, la GenAI può trasformarsi da strumento abilitante a fattore di regressione formativa, soprattutto per le nuove generazioni di sviluppatori.

Alcuni articoli preliminari suggeriscono già questa deriva: si prospetta infatti un peggioramento delle capacità di coding di base tra i programmatori che fanno uso intensivo di assistenti di intelligenza artificiale [52]. Il rischio non è quindi soltanto individuale, ma collettivo: interi team potrebbero diventare meno autonomi, meno creativi e meno resilienti di fronte a problemi non standard, proprio perché hanno perso l'esercizio quotidiano del pensiero critico e del ragionamento logico.

La dipendenza, tuttavia, non si manifesta soltanto sul piano delle competenze. Un ulteriore elemento di vulnerabilità è rappresentato dal vendor lock-in: quando un'organizzazione costruisce i propri workflow intorno a un modello o a un servizio specifico, il passaggio ad alternative si fa complesso e oneroso. Ciò lega a doppio filo l'azienda alle scelte tecnologiche, economiche e politiche del fornitore, riducendo la flessibilità strategica e aumentando il rischio di dipendenza infrastrutturale.

Senza adeguate contromisure, l'organizzazione rischia di trovarsi con professionisti abili a interagire con l'IA, ma sempre meno preparati a operare in autonomia. Per evitare che questo debito di competenze diventi un fardello strutturale, è indispensabile investire in formazione continua, incoraggiando pratiche che mantengano vive le capacità umane fondamentali, e promuovendo un uso dell'AI che sia realmente complementare e non sostitutivo dell'ingegno umano. Solo in questo modo la GenAI può essere integrata nei processi di sviluppo come risorsa potenziante, anziché come fattore di dipendenza che impoverisce progressivamente il capitale di conoscenza dell'organizzazione.

Governance e policy di utilizzo

Un ulteriore aspetto cruciale riguarda la definizione di una chiara governance interna e di politiche di utilizzo appropriate per la GenAI. L'assenza di regole condivise e di meccanismi di controllo rappresenta infatti un rischio organizzativo significativo:

in mancanza di linee guida precise, ciascun sviluppatore potrebbe decidere in autonomia come e quando ricorrere all'IA, con conseguente esposizione dell'azienda a vulnerabilità di sicurezza, violazioni di conformità normativa o compromissioni della qualità del prodotto. In altri termini, senza una governance strutturata, l'adozione della GenAI rischia di evolvere in modo anarchico, generando benefici immediati ma anche effetti collaterali potenzialmente gravi.

Dati recenti confermano la portata del problema. Un'indagine del 2023 ha rivelato che soltanto il 21% delle aziende aveva già istituito policy formali sull'uso dell'IA da parte dei dipendenti, nonostante la diffusione esplosiva di queste tecnologie nei flussi di lavoro [53]. Questo divario evidenzia un disallineamento tra l'entusiasmo per l'adozione e la maturità organizzativa nella gestione degli strumenti: se da un lato cresce rapidamente l'uso di assistenti generativi, dall'altro mancano spesso regole e processi in grado di incanalare tale utilizzo in un quadro sicuro e sostenibile.

Una governance solida dovrebbe partire dall'elaborazione di una politica aziendale che indichi con chiarezza gli usi consentiti e quelli vietati, stabilendo ad esempio che non sia possibile inserire nei prompt dell'IA codice proprietario o dati sensibili, al fine di evitare fughe di informazioni riservate. La stessa policy dovrebbe chiarire che i risultati prodotti dalla GenAI non possono essere accettati senza un'adeguata revisione umana, e che l'integrazione del codice generato deve avvenire attraverso processi di verifica e validazione consolidati. Parallelamente, occorre che l'organizzazione definisca quali strumenti siano effettivamente autorizzati, e quali invece siano da escludere per ragioni di sicurezza, licenza o conformità con le normative vigenti, come l'AI Act europeo.

A questi principi deve affiancarsi una chiara definizione di ruoli e responsabilità. Le organizzazioni più mature potrebbero istituire comitati interni o figure di riferimento incaricate di valutare periodicamente l'impatto della GenAI nei progetti, di monitorare la conformità alle regole e di condurre audit sul codice prodotto con l'ausilio dell'IA. In assenza di tali meccanismi, il rischio è che lo strumento venga utilizzato in maniera inconsapevole o persino impropria: sviluppatori che condividono porzioni di codice proprietario senza rendersene conto, o che integrano soluzioni generate senza adeguata verifica, con conseguenze potenzialmente gravi nel medio-lungo periodo [53].

La cultura organizzativa, poi, gioca un ruolo altrettanto decisivo. Una policy, per quanto ben scritta, non è sufficiente se non è accompagnata da una comunicazione chiara e da un senso di responsabilità condivisa. I team devono essere istruiti non solo su cosa è consentito o vietato, ma anche sul perché: comprendere i rischi legati alla gestione dei dati, alla proprietà intellettuale e alla qualità del software è essenziale per favorire un uso consapevole e maturo dell'IA. Solo attraverso un

impegno culturale di questo tipo è possibile trasformare le regole formali in prassi effettivamente rispettate nella vita quotidiana dei progetti.

Impatto sulla collaborazione tra sviluppatori e sul flusso di lavoro

L'integrazione degli strumenti di questi strumenti all'interno dei team di sviluppo non ha solo conseguenze tecniche, ma può modificare in maniera significativa anche le dinamiche collaborative e i flussi di lavoro consolidati. Il loro impatto, infatti, non si limita alla fase di scrittura del codice, ma si estende all'intero ecosistema di interazioni che caratterizza lo sviluppo software moderno, in cui la condivisione di conoscenze, il confronto tra pari e la revisione collettiva sono elementi centrali.

Da un lato, la GenAI tende a ridurre la necessità di interazioni umane per alcune attività di routine. In passato, ad esempio, un programmatore junior che incontrava una difficoltà avrebbe cercato il supporto di un collega più esperto, generando un'occasione di confronto, di scambio di esperienze e di crescita formativa reciproca. Oggi, invece, lo stesso problema può essere affrontato semplicemente chiedendo suggerimento all'IA, ottenendo una soluzione immediata ma isolata. Se questo comportamento diventa sistematico, il rischio è che lo scambio di conoscenza all'interno del gruppo si impoverisca: errori comuni che prima avrebbero alimentato discussioni e portato a un apprendimento collettivo diventano occasioni mancate, perché la risposta arriva dal modello ma non viene necessariamente compresa, interiorizzata o condivisa con il resto del team. La crescita del know-how collettivo, elemento fondamentale per la maturazione dei gruppi di lavoro, può così risentirne sensibilmente.

Dall'altro lato, l'uso diffuso della GenAI introduce nuove complessità nei processi collaborativi già consolidati. La fase di code review, ad esempio, assume un'importanza ancora maggiore, perché i revisori non sono più chiamati soltanto a valutare la correttezza e l'aderenza del codice ai requisiti, ma devono anche essere in grado di riconoscere e analizzare eventuali frammenti generati dall'IA che, pur essendo plausibili a livello sintattico, possono nascondere vulnerabilità, errori logici o assunzioni non corrette. Se lo sviluppatore che ha integrato tali porzioni di codice non ne ha piena padronanza, il revisore rischia di trovarsi di fronte a un compito più complesso e dispendioso in termini di tempo, con inevitabili rallentamenti nel processo di integrazione. Questo scenario può anche intaccare la fiducia reciproca tra i membri del team: la revisione rischia di trasformarsi in un'attività di "controllo" piuttosto che di collaborazione costruttiva, alimentando frizioni e riducendo la fluidità del workflow.

Le evidenze empiriche confermano questa dinamica. Studi condotti su progetti open-source che hanno introdotto GitHub Copilot mostrano che, a fronte di un incremento misurabile della produttività individuale (nell'ordine del 5–6%), si è registrato un

aumento significativo del tempo di integrazione del codice prodotto, stimato intorno al 41% [54]. Tale differenza sembra imputabile ai maggiori costi di coordinamento e alle verifiche aggiuntive richieste per il codice generato dall'IA. Ciò indica che l'uso della GenAI può determinare un vero e proprio overhead collaborativo, fatto di più iterazioni, discussioni più lunghe e necessità di aggiustamenti frequenti prima di arrivare all'accettazione di una modifica. In altre parole, il guadagno di velocità individuale rischia di tradursi in una perdita di efficienza a livello di team.

Infine, non bisogna dimenticare che la GenAI, per quanto sofisticata, non può sostituire la comunicazione diretta con gli stakeholder e con gli utenti finali. Attività come la raccolta e la chiarificazione dei requisiti, le sessioni di brainstorming per la progettazione o la negoziazione delle priorità rimangono fortemente ancorate all'interazione umana. Una delega eccessiva di tali momenti all'IA rischierebbe di impoverire il dialogo interno ed esterno, compromettendo quella dimensione relazionale che rappresenta la linfa vitale di un progetto software sano. In definitiva, la GenAI può potenziare il lavoro dei team, ma non sostituire la ricchezza cognitiva e collaborativa che nasce dall'incontro tra persone.

2.5.3 Etiche, Ambientali e Legali

Accanto ai rischi tecnici e organizzativi, l'adozione della GenAI nel ciclo di vita del software solleva questioni di carattere etico, giuridico e ambientale che trascendono la dimensione puramente tecnologica ma che, al tempo stesso, condizionano in modo determinante le possibilità di utilizzo sostenibile e responsabile di queste tecnologie. Si tratta di rischi che investono ambiti tradizionalmente regolati da norme di diritto, da principi etici e da considerazioni di sostenibilità, e che pertanto non possono essere ignorati da chi intende integrare la GenAI nei processi aziendali o istituzionali.

Proprietà intellettuale e licenze del risultato generato

L'utilizzo di sistemi di intelligenza artificiale generativa per produrre testi, documenti, immagini o codice sorgente solleva questioni inedite e tuttora dibattute in merito alla proprietà intellettuale e al rispetto delle licenze applicabili ai contenuti. I modelli linguistici vengono infatti addestrati su vastissimi insiemi di dati, che includono libri, articoli, materiale proveniente dal web, documentazione tecnica, repository di codice e una molteplicità di altre fonti, molte delle quali protette da copyright o distribuite con licenze specifiche.

Il fatto che i modelli possano generare testi o contenuti che riproducono, anche solo parzialmente, strutture, stili o soluzioni provenienti da opere preesistenti, apre scenari giuridici complessi. Vi è il timore che l'output di un sistema di GenAI possa configurarsi, in alcuni casi, come un'opera derivata, con conseguente rischio di violazione involontaria della proprietà intellettuale. La giurisprudenza su questi temi è ancora in evoluzione, ma diversi esperti hanno già sottolineato come il rischio di infrazione, pur non sempre immediatamente evidente, sia tutt'altro che trascurabile.

Un esempio particolarmente critico riguarda le licenze copyleft, come la GPL nel campo del software. Se un modello generativo, avendo appreso durante l'addestramento da materiale coperto da tale licenza, produce un contenuto che ne incorpora elementi sostanziali, chi utilizza quell'output in un contesto proprietario potrebbe, almeno teoricamente, essere obbligato a rilasciare l'intera opera sotto la stessa licenza, con conseguenze rilevanti. Nel caso del codice sorgente, ciò significherebbe dover distribuire parti di software proprietario come open source, ma scenari analoghi si possono ipotizzare anche per altre tipologie di contenuti testuali.

A tutto ciò va poi aggiunto il tema che riguarda la mancata attribuzione: anche quando l'output generato non coincide testualmente con una fonte specifica, può esserne fortemente ispirato. Si pensi, ad esempio, a quando si chiede a un modello di scrivere un racconto breve: il risultato può rivelarsi una variazione più o meno

evidente di opere letterarie già esistenti. Ciò solleva questioni etiche, legate al mancato riconoscimento degli autori originari, e legali, legate al diritto morale d'autore, che in molti ordinamenti tutela la paternità e l'integrità dell'opera.

Vi è poi la problematica dell'apolidia delle opere generate dall'IA. In alcuni sistemi giuridici, un contenuto prodotto interamente da una macchina senza contributo creativo umano non può essere protetto da copyright. Questo significa che i testi, le immagini o i codici generati potrebbero ricadere in una sorta di "zona grigia", privi di protezione legale e quindi liberamente riutilizzabili da chiunque. Per le aziende, ciò implica l'impossibilità di rivendicare diritti esclusivi sui contenuti creati dall'IA, con conseguenze potenzialmente gravi in termini di tutela e valorizzazione degli asset immateriali [55].

A questi va poi aggiunto un rischio trasversale che riguarda l'esposizione involontaria di proprietà intellettuali aziendali. Molti fornitori di GenAI, specialmente nelle versioni gratuite, dichiarano di poter raccogliere i prompt e i dati inseriti dagli utenti per migliorare i modelli. Qualora uno sviluppatore o un dipendente introducesse all'interno di un prompt parti di codice proprietario o documentazione interna riservata, tali informazioni potrebbero essere conservate e, in seguito, riemergere in output generati per terzi. Questo scenario non solo mina la riservatezza e la sicurezza aziendale, ma comporta anche una perdita di controllo su conoscenze strategiche e su elementi di proprietà intellettuale.

Responsabilità legale e compliance normativa

Un'organizzazione che adotta la GenAI nel proprio SDLC deve considerare attentamente le responsabilità legali derivanti dall'uso di codice generato automaticamente, e più in generale dei dati forniti in ingresso e ricevuti in uscita dai modelli. Dal punto di vista giuridico, infatti, permane in capo all'azienda (o agli sviluppatori) la responsabilità per il corretto funzionamento e la liceità del software prodotto, anche se parti di esso sono state scritte dall'IA. L'uso di un assistente AI, infatti, non rappresenta in alcun modo un'esenzione di responsabilità. Se un software contenente porzioni generate da un modello dovesse provocare danni a terzi – ad esempio per un bug critico, un malfunzionamento in un sistema mission critical o una violazione normativa – il fatto che quel codice o quel contenuto sia stato prodotto da una macchina non costituirebbe una difesa sufficiente in sede legale.

La prassi corrente dei fornitori di GenAI conferma questa impostazione. I principali attori del settore, consapevoli delle implicazioni potenzialmente rilevanti, tendono a declinare esplicitamente ogni forma di responsabilità sui risultati prodotti dai propri modelli, imponendo all'utente finale l'onere di verifica e validazione. I termini di servizio di GitHub Copilot [56], ad esempio, specificano chiaramente che lo sviluppatore è tenuto a controllare e testare ogni output prima dell'utilizzo, sollevando di

fatto Microsoft e GitHub da ogni responsabilità per eventuali conseguenze negative legate al codice suggerito. In altre parole, i team di sviluppo devono trattare i contenuti generati dall'IA come se fossero contributi di un collaboratore umano non esperto, sottoponendoli a processi rigorosi di code review, testing e validazione prima della distribuzione. In caso contrario, un eventuale contenzioso per danni, negligenza o violazioni contrattuali ricadrebbe interamente sull'azienda produttrice del software e non sul fornitore dell'IA.

Accanto alla responsabilità civile e contrattuale, l'uso della GenAI porta con sé anche significative sfide di compliance normativa. In Europa, ad esempio, il Regolamento Generale sulla Protezione dei Dati (GDPR) [57] impone vincoli stringenti sul trattamento delle informazioni personali. Laddove per generare codice o configurazioni si forniscano all'IA dati reali di utenti – come esempi di record, credenziali o informazioni anagrafiche – si rischia di compiere un trattamento non autorizzato, soprattutto qualora il fornitore del servizio conservi tali dati per finalità di addestramento. Non meno problematico è lo scenario in cui il modello, per effetto dei dati presenti nel training, produca stringhe contenenti informazioni sensibili che vengano poi integrate inconsapevolmente nel software o nella documentazione: anche in questo caso l'azienda sarebbe responsabile di una violazione della privacy o di obblighi di riservatezza.

Il tema della conformità normativa si fa ancora più stringente in settori ad alta criticità, come quello sanitario, finanziario o automotive, nei quali le leggi richiedono non solo correttezza funzionale ma anche trasparenza, tracciabilità e validazione formale del software. In tali contesti, l'utilizzo di componenti generate dall'IA, per loro natura difficili da spiegare e da certificare a causa del carattere di "black-box" dei modelli, rischia di entrare in conflitto diretto con requisiti di audit e certificazione. Le organizzazioni che operano in questi ambiti devono quindi valutare attentamente l'impatto dell'introduzione della GenAI, integrando misure di controllo aggiuntive e predisponendo strategie di mitigazione per garantire che il rispetto degli standard normativi non venga compromesso.

Impatti occupazionali e trasformazione del ruolo dello sviluppatore

L'adozione crescente di strumenti di intelligenza artificiale generativa nel settore del software apre interrogativi profondi sugli impatti occupazionali e sulla possibile trasformazione dei ruoli professionali. La questione non riguarda soltanto la quantità di posti di lavoro disponibili, ma anche la natura stessa delle competenze richieste e l'evoluzione del profilo dello sviluppatore.

Da un lato, esiste il timore che l'automazione di una parte delle attività di programmazione possa ridurre la domanda complessiva di programmatori [58]. Se un modello come ChatGPT o sistemi analoghi sono in grado di generare in modo

autonomo porzioni significative di codice funzionante, molte aziende potrebbero non avere più necessità di mantenere lo stesso numero di addetti dedicati alla scrittura manuale di software. Questo scenario appare particolarmente critico per le posizioni junior o per i ruoli caratterizzati da mansioni ripetitive e a basso valore aggiunto, che rischiano di essere le prime a subire la pressione dell'automazione.

Sul medio-lungo periodo, tuttavia, l'impatto potrebbe assumere una forma diversa e più articolata. Se l'innovazione tecnologica continuerà a progredire con i ritmi osservati negli ultimi anni, è plausibile che la figura dello sviluppatore tradizionale subisca una trasformazione radicale. Il programmatore potrebbe progressivamente evolvere in una sorta di "supervisore dell'IA", maggiormente focalizzato sulla definizione dei requisiti, sulla progettazione architetturale e sulla validazione critica delle soluzioni generate, piuttosto che sulla scrittura puntuale di ogni riga di codice. In questo senso, il lavoro di sviluppo diverrebbe più vicino al design concettuale e strategico, lasciando all'AI la parte più esecutiva e ripetitiva delle attività.

Un simile cambiamento comporterebbe inevitabilmente una riconfigurazione delle competenze richieste. La capacità di memorizzare dettagli sintattici o di riprodurre algoritmi standard potrebbe diventare meno centrale, mentre assumerebbero maggiore importanza abilità quali il ragionamento analitico, la progettazione ad alto livello, la revisione critica del codice generato e il cosiddetto prompt engineering, ossia la capacità di interagire efficacemente con i modelli per ottenere risultati pertinenti e di qualità. Non sorprende che alcune aziende stiano già avviando programmi di formazione interna per preparare i propri dipendenti a questa transizione, consapevoli che il futuro del lavoro dello sviluppatore richiederà un insieme di competenze nuove e complementari.

Storicamente, l'automazione non ha quasi mai eliminato del tutto il lavoro umano, ma ne ha modificato le modalità e i contenuti. È quindi plausibile che anche la GenAI finisca per spostare il baricentro delle mansioni piuttosto che sopprimerle completamente. In questa prospettiva, l'intelligenza artificiale potrebbe liberare i programmatori dalle attività più ripetitive e standardizzate, consentendo loro di dedicare più tempo a compiti di maggiore valore strategico, come il design innovativo delle soluzioni, l'ottimizzazione delle performance, la collaborazione con i clienti o la sperimentazione di nuove idee. Paradossalmente, la figura del software engineer potrebbe così diventare ancora più cruciale: meno "artigiano del codice" e più architetto di soluzioni, guida della strategia digitale e attore chiave dei processi di innovazione.

Resta tuttavia il fatto che, nel breve termine, le sfide occupazionali siano già tangibili. Le dinamiche del mercato del lavoro negli Stati Uniti mostrano segnali di ristrutturazione in alcuni segmenti del settore IT [59], e non mancano dichiarazioni

di CEO di importanti aziende – come il caso di Klarna [60] – che hanno esplicitamente sottolineato l'impatto della GenAI sulle strategie occupazionali delle proprie organizzazioni.

Per affrontare questa transizione in maniera equilibrata, appare fondamentale adottare un approccio etico e proattivo, investendo in programmi di riqualificazione professionale e in percorsi di formazione continua. La sfida non è soltanto tecnica, ma anche culturale e organizzativa: occorre garantire che i lavoratori non subiscano passivamente la trasformazione, ma siano messi nelle condizioni di sfruttare le potenzialità della GenAI per accrescere la propria produttività e il proprio valore professionale. Solo così l'introduzione dell'intelligenza artificiale potrà trasformarsi da minaccia occupazionale a opportunità di crescita, in grado di ridefinire i confini della professione senza eroderne la centralità.

Sostenibilità ambientale

Un aspetto spesso trascurato nelle discussioni sull'adozione della GenAI riguarda il suo impatto ambientale, che si sta rivelando sempre più significativo. Se molto si discute degli aspetti tecnici, organizzativi o giuridici, minore attenzione è stata finora dedicata ai costi ecologici di questi strumenti, nonostante essi rappresentino un fattore cruciale per la sostenibilità futura della tecnologia. Gli LLM sono infatti estremamente energivori, sia nella fase di addestramento sia in quella di inferenza, con conseguenze tangibili in termini di consumo di risorse ed emissioni di anidride carbonica.

La fase di training è quella più intensiva dal punto di vista energetico. Per addestrare un modello sono necessari calcoli su scala massiva, distribuiti su migliaia di unità di elaborazione specializzate. Studi recenti stimano che l'energia richiesta possa variare da alcune decine a oltre un migliaio di megawattora, a seconda della dimensione del modello e della durata dell'addestramento. Per avere un riferimento concreto, lo sviluppo di GPT-3 è stato associato a circa 552 tonnellate di CO_2 emesse in atmosfera durante la sola fase di training, un dato che rende evidente la portata ambientale di tali operazioni [61].

Anche la fase di inferencza, cioè l'utilizzo del modello già addestrato per rispondere a domande o generare testi, non è priva di conseguenze. Ogni singola query rivolta a un modello come ChatGPT ha un costo energetico che si traduce in emissioni di alcuni grammi di CO_2 [62]. Tale cifra, apparentemente irrilevante se considerata in isolamento, diventa significativa se moltiplicata per le centinaia di milioni di richieste che vengono processate quotidianamente a livello globale. L'accumulo su larga scala di questi consumi genera un impatto ambientale non trascurabile, che tende ad amplificarsi con la diffusione sempre più capillare della tecnologia.

Oltre al consumo elettrico, un'altra variabile critica è rappresentata dalle risorse idriche necessarie al raffreddamento dei data center. Gli impianti che ospitano le infrastrutture di calcolo utilizzate per l'addestramento e l'inferenza dei modelli richiedono infatti enormi quantità di acqua per mantenere stabili le temperature operative dell'hardware. Secondo stime del professor Shaolei Ren (University of California, Riverside), una singola sessione di interazioni con GPT-3, equivalente a 10–50 query consecutive, può comportare indirettamente un consumo di circa mezzo litro di acqua dolce per il raffreddamento [63]. Se questo valore viene proiettato su scala annuale e moltiplicato per milioni di utenti, l'impatto sul fabbisogno idrico diventa colossale, con potenziali conseguenze per i territori in cui sono collocati i data center, spesso già caratterizzati da stress idrico.

La crescente domanda di modelli sempre più grandi e performanti rende inevitabile un aumento proporzionale di tali consumi, con un conseguente aggravio in termini di emissioni, energia e risorse naturali. In questo scenario, la questione ambientale non può essere considerata secondaria né demandata al futuro, ma richiede fin da subito un impegno congiunto di aziende e governi. È necessario sviluppare strategie volte a mitigare l'impatto ecologico dell'IA, investendo in soluzioni tecnologiche più efficienti dal punto di vista energetico, favorendo l'uso di fonti rinnovabili per alimentare i data center e introducendo regolamentazioni specifiche che pongano limiti e vincoli all'impronta ambientale dei modelli di intelligenza artificiale.

Il successo della GenAI non può essere misurato soltanto in termini di produttività, innovazione o competitività economica, ma deve essere valutato anche rispetto alla sua sostenibilità complessiva. Solo affrontando in maniera proattiva le implicazioni ambientali sarà possibile garantire che questa tecnologia contribuisca realmente al progresso, senza scaricare costi occulti sul pianeta e sulle generazioni future.

Capitolo 3

I Dati Git e gli LLM per l'Analisi e il Miglioramento dei Workflow e delle Code Review

Dopo aver analizzato nei capitoli precedenti il ruolo della GenAI nello sviluppo software e le principali aree in cui gli LLM stanno trovando applicazione, questo capitolo introduce la parte sperimentale della tesi. L'obiettivo è esplorare come gli LLM possano essere impiegati per estrarre conoscenza dai dati dei repository Git, con l'intento di supportare sia le attività di code review sia la comprensione del workflow di progetto.

Il capitolo si articola in più sezioni complementari. Nella prima parte vengono delineati il contesto e le domande di ricerca che orientano lo studio. Successivamente viene presentato il progetto originario a cui questo lavoro si ispira, illustrandone l'architettura e le tecniche adottate. Segue quindi la descrizione del caso di studio selezionato – il progetto open-source MuJS – scelto come banco di prova per valutare l'efficacia dell'approccio. Infine, viene introdotta la soluzione proposta, che rappresenta il contributo originale di questa tesi e costituisce il cuore della sperimentazione.

3.1 Contesto e Obiettivi

Nel contesto dello sviluppo software moderno, tracciare e comprendere l'evoluzione di un progetto complesso è essenziale per garantire la qualità del codice e supportare decisioni informate. Tuttavia, i tradizionali processi di comunicazione tra le diverse figure di un team di sviluppo spesso faticano a reggere di fronte all'enorme mole di commit e modifiche, soprattutto in progetti caratterizzati da storie evolutive lunghe, team numerosi e molteplici interdipendenze. A ciò si aggiunge la naturale dinamicità del settore, che comporta un frequente ricambio di ruoli e persone, aumentando il rischio di perdita di informazioni cruciali e di una visione complessiva del progetto e del dominio applicativo.

In questo scenario, come discusso nei capitoli precedenti, i LLM offrono un approccio promettente. Grazie alle loro capacità di analisi del linguaggio naturale e di riconoscimento di pattern, possono elaborare i dati provenienti dai repository Git – ad esempio messaggi di commit e diff – consentendo di riassumere, categorizzare e contestualizzare le modifiche nel tempo. In questo modo, il team di sviluppo può ottenere insight strutturati riguardo allo scopo dei commit, alle motivazioni sottostanti e al loro impatto complessivo sul progetto. Un LLM, ad esempio, può raggruppare i commit relativi a una specifica funzionalità, mettere in evidenza le aree del codice soggette a modifiche frequenti o individuare pattern ricorrenti di refactoring e bug fix, offrendo così una visione chiara dell'evoluzione del progetto.

All'interno di questo quadro emergono due principali domande di ricerca che guidano l'indagine:

- **RQ1**: Quanto efficacemente gli LLM possono riassumere e interpretare i messaggi di commit Git per esprimere l'intento alla base delle modifiche al codice?
- **RQ2**: In che modo gli LLM possono facilitare una migliore comunicazione tra i membri del team generando report contestualizzati sull'evoluzione del progetto?

In altri termini, la RQ1 si concentra sulla capacità degli LLM di interpretare e sintetizzare i singoli commit, cogliendone motivazioni e obiettivi. La RQ2, invece, esplora come strumenti basati su LLM possano supportare la condivisione delle conoscenze all'interno del team, ad esempio attraverso report narrativi che descrivano l'evoluzione del codice e le decisioni prese nel tempo. Il problema affrontato è quindi duplice: da un lato, sintetizzare in modo efficace le informazioni provenienti dai commit per comprenderne l'intento (RQ1); dall'altro, investigare se gli LLM siano in grado di migliorare la comunicazione e la comprensione condivisa dello stato del progetto tra tutti gli attori coinvolti nello sviluppo (RQ2).

3.2 Progetto originario

Per rispondere alle domande di ricerca sopra poste, si è preso come punto di partenza il lavoro di Alzate, Francios e Monaco (Politecnico di Torino, 2025) intitolato *"Code Review and Project Workflow Analysis for Git Data"* [64]. In tale studio viene proposta un'architettura per l'analisi dei dati Git con l'obiettivo di supportare la code review e lo studio del workflow di progetto. In particolare, si ispira a recenti studi basati su LLM in ambito software: il framework multi agente di Tao et al. in *"MAGIS: LLM-Based Multi-Agent Framework for GitHub Issue Resolution"* [65], il generatore di test di regressione per commit di Liu et al. [66], e la metodologia di analisi di large codebases di Zheng et al. [67].

L'architettura descritta in [64], adottata come base per questo studio, è di tipo modulare e strutturata come una pipeline multi-stadio (figura 3.1). A partire dai dati estratti da un repository Git, essa produce automaticamente categorie di commit, riassunti testuali delle modifiche e persino narrazioni (*"stories"*) che descrivono l'evoluzione del progetto.

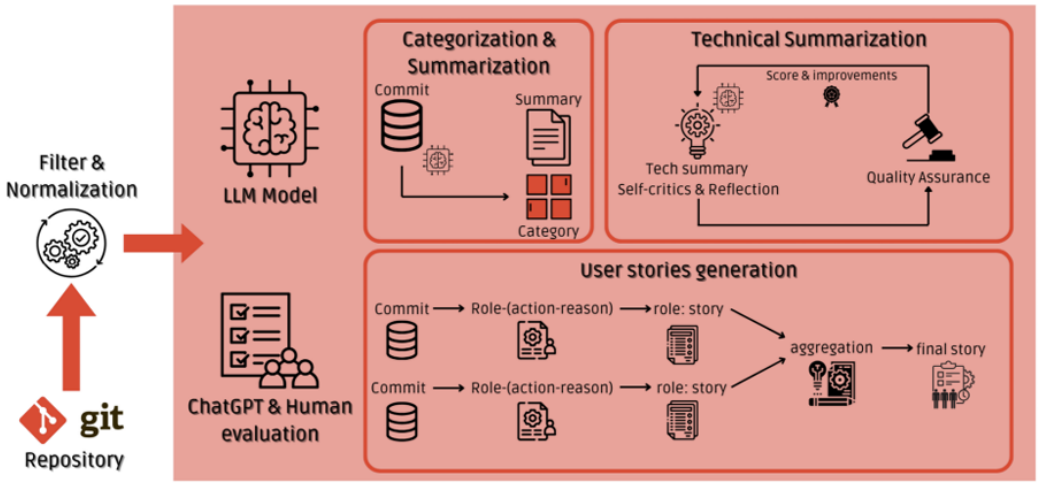

Figura 3.1: Architettura ad alto livello del lavoro svolto in *"Code Review and Project Workflow Analysis for Git Data"* [64]

3.2.1 Architettura

Entrando nel dettaglio, l'architettura comprende un primo modulo di estrazione, che raccoglie i dati grezzi dal repository Git e filtra i commit non significativi o "banali". Le informazioni rilevanti (messaggi, diff, autore, data, ecc.) vengono normalizzate in un formato uniforme per le fasi successive.

Ogni commit viene quindi classificato in una delle categorie funzionali predefinite (ad es. feature, bug fix). L'LLM riceve come input le informazioni associate al commit e sceglie la categoria più appropriata a partire da un elenco fisso. Per questo compito sono stati confrontati l'appoccio zero-shot con quello few-shot.

A valle di questo componente c'è una doppia catena di *summarization*, che genera per ogni commit due riassunti di carattere:

- **Generale (summary)**: in linguaggio naturale comune, spiega ad alto livello cosa è stato fatto nel commit e perché, in modo comprensibile a sviluppatori e stakeholder non tecnici;
- **Tecnico (technical summary)**: più dettagliato, evidenzia specificamente le modifiche al codice (quali file o componenti sono stati toccati, quali funzionalità sono state aggiunte/modificate, ecc.), fornendo una visione più approfondita per gli sviluppatori.

Anche qui, è l'LLM che elabora tutte le informazioni rilevanti del commit per produrre i due tipi di riassunto. In questo caso si è scelto di usare il solo few-shot, in quanto il setup zero-shot si è rivelato poco adeguato. L'output di questa fase sono due testi per commit (riassunto "user-friendly" e riassunto tecnico) che descrivono la stessa modifica a livelli differenti di astrazione.

Per aumentare l'affidabilità delle sintesi tecniche, il sistema integra un meccanismo iterativo di controllo qualità ispirato al paradigma multi-agente di MAGIS [65]. In questo schema, un primo agente genera il riassunto tecnico, mentre un secondo ne valuta la qualità assegnando un punteggio da 0 a 10. Se la valutazione risulta inferiore a una soglia prefissata (pari a 8), il riassunto viene rigettato e il primo agente deve produrne uno nuovo. In questo modo il sistema incorpora meccanismi di riflessione e autocritica, mantenendo solo le sintesi considerate sufficientemente valide.

L'ultimo componente dell'architettura è dedicato a creare una narrazione complessiva dell'evoluzione del progetto a partire dai singoli commit analizzati. Mentre le sintesi descritte sopra trattano i commit individualmente, questa fase mira a integrare le informazioni per raccontare come il progetto si sviluppa nel tempo. In pratica, il sistema elabora i riassunti generati per ciascun commit e li trasforma in "storie" strutturate che descrivono l'avanzamento del progetto dal punto di vista degli stakeholder. In particolare, dal riassunto (tecnico e/o generale) di ogni commit vengono individuati tre elementi fondamentali: il soggetto/attore coinvolto ("chi" è protagonista del cambiamento, ad esempio uno sviluppatore o componente software), l'obiettivo o azione compiuta ("cosa" è stato fatto/modificato) e la motivazione o beneficio che ne deriva ("perché" questa modifica è importante). Il prompt di generazione delle story prende questi elementi e li mappa su specifici ruoli

di stakeholder rilevanti nel contesto software, come ad esempio sviluppatore, tester, ecc. Per ogni commit, il modello costruisce quindi una frase in formato user story tipico dello sviluppo agile: ***"As a [role], I want [goal] so that [reason/benefit]."*** Questa rappresentazione trasforma la modifica tecnica in termini di valore per un particolare ruolo, rendendo esplicito chi beneficia del cambiamento e per quale scopo. Vengono generate due versioni: una basata sul riassunto tecnico e una basata sul riassunto generale, per mantenere sia la prospettiva dettagliata sia quella più astratta/user-friendly. Successivamente interviene una fase di aggregazione delle story prodotte in base al singolo ruolo. Lo scopo è concatenare queste frasi in una narrazione continua che segua l'ordine temporale dei commit e dia il senso dello sviluppo progettuale.

Ciascun componente dell'architettura contribuisce alle domande di ricerca iniziali: i primi moduli (estrazione, classificazione, sintesi) rispondono alla RQ1, mentre le fasi di narrazione affrontano la RQ2, trasformando i dati in un racconto accessibile a tutti gli stakeholder.

3.2.2 Modelli

Il modello impiegato è Llama 3.2-1B Instruct (Meta) [68], un LLM da circa 1 miliardo di parametri disponibile su HuggingFace [69]. La scelta di un modello compatto è motivata dalla volontà di testare la fattibilità di utilizzare LLM di dimensioni ridotte, privilegiando efficienza computazionale e rapidità di esecuzione. In parallelo, per accelerare la valutazione dei risultati, è stato utilizzato anche OpenAI ChatGPT [16] in modo supervisionato: le sue uscite hanno fornito un riferimento con cui confrontare e affinare quelle di Llama, con il supporto di un controllo umano per correggere eventuali inesattezze.

3.2.3 Tecniche

Dal punto di vista delle tecniche di prompting, sono stati sperimentati approcci di prompting zero-shot (senza esempi) e few-shot (con esempi input-output forniti nel prompt). Nel primo caso, la comprensione del compito è affidata interamente al modello pre-addestrato; nel secondo, il modello è guidato da esempi che ne orientano la produzione. La classificazione dei commit ha visto il confronto tra i due approcci, mentre nella generazione di riassunti si è preferito il few-shot per ottenere risultati più mirati. Inoltre, ispirandosi a lavori recenti sui sistemi multi-agente (es. MAGIS), il progetto adotta un paradigma in cui diversi LLM collaborano o si validano a vicenda in compiti specifici, introducendo meccanismi di auto-riflessione e verifica incrociata dei contenuti generati, con l'obiettivo di aumentarne la robustezza.

3.3 Caso di studio: il progetto MuJS

Per validare e sperimentare l'architettura descritta, è stato scelto come caso di studio il repository open-source MuJS [70]. MuJS è un interprete JavaScript leggero, conforme allo standard ES5 e interamente implementato in linguaggio C. Si tratta di un progetto di media dimensione (circa 15.000 linee di codice) concepito per essere integrato in altre applicazioni come motore di scripting. Il repository originale è pubblico su GitHub ed è mantenuto da Artifex Software con il contributo della comunità open source.

La scelta di MuJS è motivata da diversi fattori. Innanzitutto, il progetto è sufficientemente esteso e maturo: la sua lunga storia di sviluppo fornisce un ampio insieme di commit, distribuiti su periodi e contesti diversi, che consente di testare l'efficacia della pipeline su commit eterogenei (bug fix, nuove funzionalità, ottimizzazioni, ecc.). In secondo luogo, il dominio applicativo è ben definito, il che rende i risultati della narrazione più facilmente verificabili e interpretabili. È infatti possibile confrontare la sintesi prodotta con eventi concreti dell'evoluzione del progetto, come l'introduzione di nuove feature del linguaggio, la risoluzione di bug critici o interventi di refactoring per migliorare le performance. Infine, l'impiego di un repository open-source reale assicura che l'approccio sia validato su dati concreti e non su esempi artificiali, confermando la solidità dell'architettura proposta in uno scenario realistico.

A partire da agosto 2025, il progetto è stato migrato da GitHub a Codeberg [71].

3.4 Soluzione Proposta

Il progetto di partenza ha dimostrato il potenziale degli LLM nell'analisi dei dati Git, in particolare nella creazione di riassunti dei commit e nella categorizzazione delle modifiche, mettendo però in luce anche alcune limitazioni. Tra queste, l'incapacità di generare una narrazione fluida e coerente dell'intera evoluzione del progetto e la tendenza a produrre output ridondanti o non sempre pertinenti quando vengono combinati numerosi commit. Nel report originale vengono suggerite alcune possibili direzioni di miglioramento: l'adozione di modelli LLM più grandi (che potrebbero garantire maggiore coerenza e accuratezza nella sintesi globale), l'impiego di approcci multi-agente in cui più modelli collaborano per affinare la narrazione (sulla scia di MAGIS, citato in precedenza), e l'integrazione di strumenti di RAG per superare i limiti di contesto, consentendo al modello di recuperare dinamicamente informazioni rilevanti durante la generazione della storia.

Alla luce di queste osservazioni, l'estensione sperimentale qui proposta si concentra in particolare sulla RQ2, con l'obiettivo di migliorare la comunicazione e la comprensione condivisa dell'evoluzione del codice. Invece di forzare il sistema a produrre un unico resoconto globale di tutti i commit — approccio che si è dimostrato fragile in termini di coerenza — si è scelto di sviluppare un sistema interattivo di Q&A: un chatbot intelligente capace di rispondere in linguaggio naturale a domande sul repository e sulla sua storia. La conoscenza di base è costituita dal codice sorgente, dalla documentazione di progetto, dai dati dei commit e dai relativi riassunti generati nella pipeline offline. Il LLM utilizza questi dati come base di conoscenza per fornire risposte mirate e contestualizzate.

In questo scenario, un membro del team può porre domande specifiche (ad esempio: *"Quali commit hanno introdotto la funzionalità X e chi li ha effettuati?"* oppure *"Quando è stato risolto il bug relativo al modulo Y?"*) e ricevere una risposta sintetica, supportata dalle informazioni raccolte dalle varie fonti. L'approccio on-demand migliora la fruibilità delle informazioni storiche: ciascun utente ottiene soltanto le parti rilevanti della storia, evitando sia la ridondanza di un report statico sia la scarsa coesione di una narrazione unica. Inoltre, un simile chatbot può fungere da assistente virtuale della conoscenza centralizzata del progetto, facilitando la comunicazione interna e riducendo il carico cognitivo necessario a consultare manualmente grandi quantità di informazioni sparse su più fonti.

L'integrazione del RAG non si limita alla fase di Q&A, ma apre anche nuove prospettive per la RQ1. Contestualizzare la generazione dei riassunti dei commit con informazioni storiche e di dominio permette infatti di ottenere testi più completi, coerenti e informativi. Invece di basarsi esclusivamente sui dati del singolo commit, il modello può attingere a un contesto più ampio, con il potenziale di migliorare

sensibilmente la qualità e l'accuratezza delle sintesi prodotte.

3.4.1 Architettura

Partendo dalle intenzioni sopra descritte, è stato progettato un sistema in grado di rispondere efficacemente alle domande di ricerca RQ1 e RQ2. Dal punto di vista architetturale, il sistema implementato può essere scomposto in due pipeline. La prima, "Offline" 3.2, si occupa dell'elaborazione dei commit, della loro storicizzazione, classificazione e sintesi. La seconda, "Online", rappresenta il chatbot vero e proprio, che recuperando le informazioni di contesto riesce a dare riposte precise e mirate alle richieste utente.

Entrando nel dettaglio, la pipeline "Offline" (figura 3.2) si occupa di elaborare i dati relativi ai commit, generando dei riassunti sintetici che vengono successivamente caricati in un database vettoriale. La generazione dei riassunti avviene in maniera sequenziale, ossia ciascun commit è elaborato individualmente e non in parallelo. Questa scelta consente di aggiornare il database vettoriale al termine di ogni elaborazione, rendendo immediatamente disponibili i riassunti generati come contesto potenziale per i commit successivi. In altre parole, come si può vedere in figura 3.3,durante l'elaborazione del commit n-esimo, i riassunti degli n-1 commit precedenti sono già disponibili nel database vettoriale e utilizzabili tramite RAG, fornendo così un contesto utile alle sintesi del commit corrente. Una volta generato, anche il riassunto del commit n-esimo viene caricato nel database, diventando a sua volta disponibile per l'elaborazione del commit successivo. Tale scelta architetturale nasce dalla convinzione che informazioni contenute in un commit possano essere rilevanti o giustificare quelli successivi.

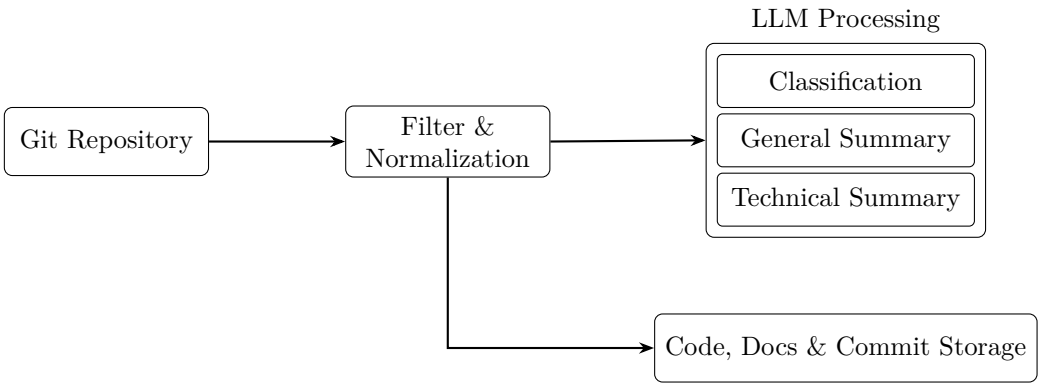

Figura 3.2: Dettaglio della pipeline Offline proposta

La pipeline "Online", visibile in figura 3.4, è dedicata, invece, all'interazione utente-sistema tramite un chatbot agentico basato su LLM. Qui, le informazioni di contesto

sono recuperate mediante l'utilizzo di tool a disposizione dell'agente. Ognuno di essi è verticalizzato nel recupero di una specifica tipologia di informazione da collezioni precedentemente alimentate dalla pipeline "Offline". Entrando più nel dettaglio nel processo, l'utente inserisce una domanda in linguaggio naturale attraverso l'interfaccia del chatbot. Questa viene utilizzata dall'agente per trovare, se necessario, il tool (o anche più di uno) che reputa più adatto per recuperare le informazioni utili per generare la risposta contestualizzata. Una volta individuato, effettua la chiamata passandogli una query che è una rielaborazione della domanda iniziale. A questo punto, il tool recupera le informazioni e le restituirà all'agente, che potrà quindi decidere di elaborare la risposta da restituire o di effettuare qualche altra chiamata ad altri tool.

Per migliorare la qualità di elaborazione dell'agente, è stato combinato il pattern ReAct, disponibile tramite il React Agent del framework LangGraph, con l'utilizzo del modello Qwen 3 da 8B, che presenta nativamente caratteristiche di CoT tramite la funzionalità di "Thinking". Il framework ha permesso inoltre di aggiungere molto facilmente anche la cronologia delle interazioni, così da mantenere lo stato conversazionale nel caso di più interazioni consecutive di Q&A. Grazie a questa possibilità, l'agente ha a disposizione anche le informazioni storiche dell'interazione, così da poter evitare eventuali chiamate a tool se nello storico ha già a disposizioni le informazioni necessarie per generare la risposta e, in generale, fornire un flusso conversazionale più omogeneo e coerente.

Un aspetto architetturale significativo riguarda l'aggiornamento della knowledge base. Con l'evolversi continuo del repository Git tramite nuovi commit, il database vettoriale necessita di aggiornamenti periodici che richiedono di eseguire la pipeline "Offline" per i nuovi dati. Tale processo può essere potenzialmente automatizzato attraverso workflow di tipo CI/CD, che ad ogni push sul branch principale eseguono

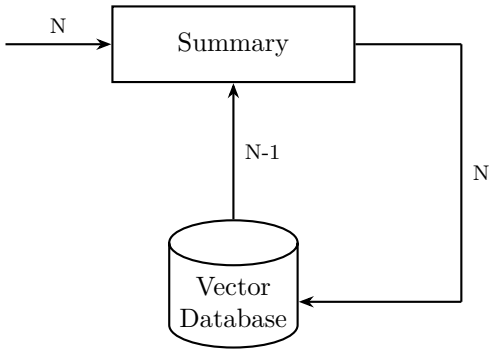

Figura 3.3: Dettaglio dell'elaborazione della sintesi e del recupero del singolo commit nella pipeline Offline

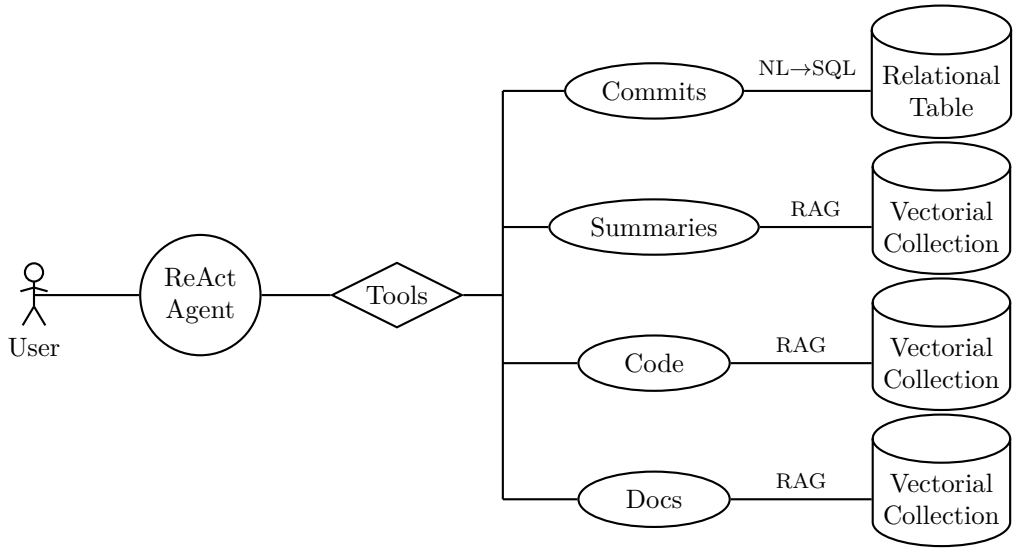

Figura 3.4: Dettaglio della pipeline Online proposta

automaticamente l'estrazione e la sintesi dei nuovi commit, aggiornando le collezioni. In contesti reali, l'LLM impiegato per la generazione dei riassunti potrebbe essere ottimizzato per tale task (tramite fin-tuning) e risiedere su un servizio dedicato.

Come visibile nell'architettura in figura 2, alcuni componenti presentati in [64] sono stati riutilizzati e potenziati. In particolare, è stato mantenuto il modulo iniziale che estrae i dati grezzi dal repository del progetto MuJS, filtrando i commit non significativi e normalizzandoli in un formato uniforme per le fasi successive. Anche la struttura di base per la generazione dei riassunti è stata mantenuta, ma potenziata dall'integrazione del RAG e da prompt ed esempi migliori, che dovrebbe migliorare significativamente la qualità delle sintesi prodotte. Anche il classificatore è stato potenziato da esempi più aderenti.

Relativamente al modello adottato, si è scelto di mantenere la continuità con l'approccio di [64], privilegiando modelli leggeri in grado di operare su una singola GPU domestica, al fine di minimizzare i requisiti computazionali e favorire l'accessibilità del sistema anche a privati o aziende di piccole e medie dimensioni. Tuttavia, l'introduzione del RAG e la necessità di supportare un chatbot evoluto, capace di gestire conversazioni estese e l'utilizzo avanzato di tool, ha portato alla scelta di modelli più potenti e performanti rispetto al lavoro originario. In particolare, è stato selezionato il modello Llama 3.1 8B-Instruct [72] per la pipeline "Offline", per il suo equilibrio tra prestazioni avanzate, capacità di comprensione contestuale e requisiti hardware ancora estremamente contenuti. Invece, per quella "Online" si è optato per Qwen 3 8B [73] per il supporto nativo al "Thinking" e per la capacità di

gestire in maniera efficace più tool.

3.4.2 Tecnologie

In questa sezione verranno descritte le principali tecnologie adottate nella soluzione proposta.

Per ulteriori dettagli implementativi riguardo al loro utilizzo all'interno del progetto, si rimanda al codice disponibile al link [74].

Llama 3.1 - 8B - Instruct

Llama 3.1 8B-Instruct [72] è un LLM basato sull'architettura Transformer auto-regressiva, sviluppato da Meta [26]. Appartiene alla famiglia Llama 3.1, disponibile in diverse versioni con 8, 70 e 405 miliardi di parametri. La variante Instruct è ottimizzata attraverso fine-tuning supervisionato (SFT) e apprendimento per rinforzo con feedback umano (RLHF), al fine di eseguire istruzioni e interagire in modo naturale in contesti conversazionali.

I modelli Llama 3.1 sono multilingue (supportano, tra le altre, inglese, tedesco, francese, italiano e spagnolo) e implementano tecniche avanzate come la Grouped Query Attention, che consente di gestire contesti estesi fino a circa 128.000 token. La versione utilizzata supporta inoltre l'integrazione nativa con tool esterni ed è progettata per compiti complessi quali generazione di testi, sintesi automatica, dialoghi multi-turno e traduzioni.

Qwen 3 - 8B

Qwen3-8B [73] è un LLM leggero, parte della terza generazione della serie Qwen sviluppata da Alibaba Cloud [75] [76]. Si tratta di un modello denso con circa 8,2 miliardi di parametri totali (di cui 6,95 miliardi esclusi gli embedding), distribuiti su 36 livelli e basato su Grouped Query Attention (GQA).

Una caratteristica distintiva di Qwen 3-8B è il meccanismo di ragionamento ibrido: il modello può passare da una modalità "thinking", adatta a compiti complessi come logica, matematica e programmazione, a una modalità "non-thinking", più efficiente per conversazioni rapide e generiche.

Il modello gestisce contesti fino a 32.768 token in modo nativo, estendibili a 131.072 token tramite la tecnica YaRN (Yet Another RoPE extension). È stato addestrato su circa 36 trilioni di token e supporta oltre 100 lingue e dialetti, garantendo buone capacità multilingue, creatività testuale, dialoghi multi-turno e integrazione con strumenti esterni.

Ollama

Ollama [77] è un framework leggero ed estensibile progettato per eseguire LLM in locale, senza dipendere da servizi cloud. Consente di scaricare, gestire e avviare modelli linguistici tramite una CLI o API REST locali. Ad esempio, con `ollama pull <modello>` è possibile scaricare un modello specifico (come Llama 3.x o DeepSeek), mentre con `ollama run <modello>` si avvia l'inferenza direttamente sul dispositivo locale.

Ollama offre integrazioni client in diversi linguaggi (tra cui Python) e mette a disposizione un ampio catalogo di modelli pre-addestrati, rendendo semplice l'esecuzione e la sperimentazione di LLM in ambienti locali.

LangGraph

LangGraph [78] è un framework open-source, sviluppato dal team di LangChain, per costruire applicazioni basate su LLM come grafi di stato e di esecuzione. A differenza delle semplici "catene" sequenziali, LangGraph modella i flussi come nodi (funzioni/attori) e archi (transizioni guidate dallo stato), consentendo controllo esplicito sul ciclo di esecuzione, gestione fine degli errori e percorsi condizionali. Il cuore del modello è lo StateGraph, una struttura che trasporta lo stato condiviso (messaggi, variabili, risultati intermedi) tra i nodi; le transizioni possono essere deterministiche o basate su regole/euristiche apprese.

Il framework offre funzionalità essenziali per applicazioni reali: persistenza dello stato e checkpointing (per riprese e roll-back), interrupt e passaggi human-in-the-loop, orchestrazione di strumenti esterni (tool invocation) e supporto naturale a scenari multi-agente. Ciò rende LangGraph adatto a costruire agenti reattivi, pipeline RAG complesse e workflow event-driven.

Per accelerare lo sviluppo, LangGraph fornisce componenti predefiniti come create_react_agent [79], un costruttore di agenti in stile ReAct che combina pianificazione e ragionamento con chiamate a strumenti (tool calling). Questo componente incapsula un ciclo "pianifica → agisci → osserva", integrandosi con lo StateGraph e semplificando l'implementazione di agenti che risolvono compiti multi-step tramite strumenti esterni, mantenendo al contempo tracciabilità e controllo del flusso.

ChromaDB

ChromaDB [80] è un database vettoriale open-source progettato per archiviare e recuperare embeddings, ovvero rappresentazioni numeriche di testi, immagini o audio. Funziona come un indice vettoriale che permette ricerche semantiche basate sulla similarità tra vettori.

Le query vengono eseguite trasformando il testo in un embedding e recuperando i documenti più simili presenti nel database. Oltre alla ricerca semantica pura, ChromaDB supporta modalità ibride che combinano ricerca vettoriale, filtri sui metadati e filtri testuali.

Il sistema è pensato per scalare facilmente da notebook sperimentali a cluster di produzione. Nel contesto di applicazioni RAG, ChromaDB funge da archivio vettoriale per un recupero efficiente e preciso del contesto necessario alla generazione delle risposte da parte degli LLM.

SQLite

SQLite [81] è un sistema di gestione di basi di dati relazionali embedded, serverless e a configurazione nulla. I dati vengono memorizzati in un singolo file portabile, caratteristica che ne facilita la distribuzione, la replicazione e l'integrazione in applicazioni desktop, mobile ed edge. Nonostante la sua leggerezza, SQLite garantisce proprietà ACID tramite journaling e modalità Write-Ahead Logging (WAL), offrendo transazioni atomiche e durabilità.

Nel contesto della soluzione proposta, SQLite funge da archivio leggero per i log applicativi, i dati dei commit e le informazioni relative ai test di validazione, integrandosi agevolmente con i componenti Python e con gli agenti LLM.

Python 3.13

Python 3.13 [82] è il linguaggio di programmazione scelto per l'implementazione della soluzione. È ampiamente apprezzato per la sintassi semplice e leggibile, l'ampia disponibilità di librerie e framework, l'essere un linguaggio interpretato e l'esteso supporto della comunità.

La versione 3.13 introduce miglioramenti di performance al runtime e nuove ottimizzazioni del compilatore, mantenendo la piena compatibilità con l'ecosistema esistente. Un aspetto rilevante è l'avanzamento verso l'eliminazione del Global Interpreter Lock (GIL), reso opzionale in build dedicate. Questa evoluzione apre la strada a un migliore supporto per il multithreading nativo, rendendo Python più adatto a carichi di lavoro intensivi e concorrenti.

3.5 Implementazione

Come già descritto nella sezione 3.4.1, la soluzione si articola in due componenti complementari: una pipeline Offline, che si occupa di elaborare e storicizzare i dati provenienti dal repository Git, e una pipeline Online, che espone tali conoscenze attraverso un agente conversazionale. Le due parti sono strettamente collegate: la prima costruisce e aggiorna la base informativa, la seconda la utilizza in tempo reale per fornire risposte mirate agli utenti.

In generale, verranno trattate solo le parti più interessanti dal punto di vista della trattazione, concentrandosi sugli aspetti di progettazione. Verranno, quindi, esclusi i dettagli più di basso livello e tutte quelle classi e funzioni a supporto della soluzione, che resta visionabile al seguente link [74].

3.5.1 Pipeline Offline

La pipeline Offline, implementata nel file `offline_pipeline.py`, rappresenta il cuore della fase di pre-elaborazione. Qui ogni commit del progetto MuJS viene analizzato in sequenza: ne vengono estratti i dati essenziali e rimossi i casi banali o ridondanti. Ogni commit è quindi classificato in una tipologia funzionale (ad esempio feature, bug fix, refactoring) tramite prompting few-shot, gestito nel file `summary_categorization/categorization.py`.

Segue la fase di generazione dei riassunti, distinta in due varianti:

- una versione **generale**, pensata per figure non tecniche, definita in `summary_categorization/general_summarization.py`;
- una versione **tecnica**,`summary_categorization/technical_summarization.py`, orientata agli sviluppatori e arricchita da dettagli implementativi.

Per quest'ultima è previsto un meccanismo iterativo di controllo qualità: una seconda istanza del modello valuta la sintesi generata e, se ritenuta insufficiente, ne richiede la rigenerazione, introducendo così una forma di autocorrezione. Anche in questo caso si è prediletto un approccio few-shot.

Un aspetto centrale della pipeline è l'integrazione del RAG. I riassunti prodotti vengono progressivamente caricati in un database vettoriale (ChromaDB), che funge da memoria storica interrogabile in fase di generazione. In questo modo, quando il sistema si trova a sintetizzare un nuovo commit, può recuperare i riassunti precedenti più simili e utilizzarli come contesto, migliorando coerenza e continuità narrativa. Parallelamente, i dati strutturati vengono memorizzati in SQLite, così da consentire interrogazioni puntuali e deterministicamente accurate.

La scelta di processare i commit in ordine temporale e aggiornare immediatamente il database vettoriale riflette l'idea che molte modifiche diventino pienamente comprensibili solo alla luce di quelle che le hanno precedute. In scenari reali, questa pipeline potrebbe essere facilmente integrata in flussi CI/CD, così da aggiornare la base di conoscenza in maniera incrementale a ogni nuovo push.

3.5.2 Pipeline Online

La pipeline Online, descritta nel file `online_pipeline.py`, è invece dedicata all'interazione con l'utente. Qui entra in gioco l'agente `graph_react_agent` (definito in `online_pipeline_models/models/graph_agent_react.py`), costruito con il framework LangGraph secondo il paradigma ReAct. L'agente riceve in input la domanda dell'utente e, prima di formulare la risposta, riflette sul percorso da seguire: valuta quali informazioni siano necessarie, decide se consultare la base di conoscenza e sceglie quali strumenti (*tool*) utilizzare. Questo processo, reso possibile dalle capacità di ragionamento del modello Qwen 3 8B, è supportato da una memoria conversazionale che conserva lo storico delle interazioni, garantendo continuità nei dialoghi multi-turno e permettendo di evitare richieste ridondanti.

I tool a disposizione dell'agente rappresentano veri e propri canali specializzati di accesso alla conoscenza accumulata nella pipeline Offline. Alcuni operano su base semantica, come il *Commit Code*, che recupera i riassunti dei commit più pertinenti rispetto a una query, o il *Semantic Code*, che consente di cercare direttamente nel codice sorgente. Altri si fondano su interrogazioni strutturate, come il *NL→SQL Commit Query*, che traduce domande in linguaggio naturale in query SQL eseguite su SQLite, restituendo risultati deterministici. Vi è infine il *General Project Info*, dedicato alla documentazione e alle informazioni di alto livello sul progetto.

L'agente è in grado di usare questi strumenti singolarmente o in combinazione: ad esempio, può usare *NL→SQL Commit Query* per identificare un commit specifico e successivamente un recupero semantico per contestualizzarlo con commit correlati, oppure chiamare entrambi in parallelo.

L'integrazione di queste componenti realizza un sistema altamente dinamico, capace di rispondere a domande molto diverse: dalle più generali, sul funzionamento e sugli obiettivi del progetto, alle più puntuali, relative a chi ha introdotto una certa modifica o a come è implementata una determinata funzione. Le risposte non sono generate "a vuoto", ma sempre ancorate a fonti concrete — commit, documentazione o codice — recuperate dinamicamente attraverso i tool. In questo modo, l'agente combina la flessibilità espressiva del linguaggio naturale con l'affidabilità dei dati effettivamente presenti nella knowledge base, offrendo un supporto interattivo e verificabile alle attività di analisi e comprensione del workflow.

Modelli scartati

Durante lo sviluppo della pipeline Online sono stati sperimentati diversi approcci. Ognuno ha aiutato a chiarire limiti e potenzialità, fino a portare alla scelta del modello finale `online_pipeline_models/models/graph_agent_react.py`.

Il primo esperimento, `online_pipeline_models/models/chain_simple.py`, è stato un prototipo minimale di recupero tramite RAG: la domanda dell'utente veniva trasformata in query, i documenti più rilevanti recuperati, e il modello generava direttamente la risposta. Questa soluzione si è rivelata efficace solo su richieste molto semplici, ma mostrava limiti evidenti su domande articolate o multi-turno, con un'alta tendenza ad allucinazioni in assenza di evidenza testuale precisa.

Per superare la rigidità della catena singola è stato sviluppato `online_pipeline_models/models/chain_multi_query.py`, che produceva più varianti della stessa domanda e ne fondeva i risultati. In questo modo si recuperava più contesto utile, migliorando il richiamo, ma a discapito della precisione ed efficienza: le risposte risultavano spesso verbose, a volte contraddittorie, e i tempi di latenza crescevano in modo significativo.

Un passo avanti è stato compiuto con `online_pipeline_models/models/chain_agent_react.py`, il primo prototipo agentico in stile ReAct. L'agente era in grado di decidere se e quando attivare i tool, migliorando la pertinenza delle risposte rispetto alle catene puramente sequenziali. Tuttavia, l'architettura restava lineare e priva di uno stato conversazionale robusto: la gestione dei follow-up e delle domande che richiedevano più fasi rimaneva fragile, così come l'orchestrazione in caso di errori o percorsi alternativi.

Queste criticità hanno portato a `online_pipeline_models/models/graph_agent_react.py`, che ha consolidato i vantaggi dell'approccio ReAct integrandoli con le capacità di orchestrazione di LangGraph. La modellazione a grafo ha introdotto persistenza dello stato, checkpoint e transizioni condizionali, consentendo all'agente di pianificare strategie multi-tool in maniera affidabile (ad esempio combinando query SQL e recupero semantico). Questo modello finale si è dimostrato il più adatto a gestire scenari reali: capace di sfruttare la knowledge base aggiornata dalla pipeline Offline, di ridurre allucinazioni grazie all'ancoraggio a fonti strutturate e semantiche, e di mantenere coerenza nei dialoghi multi-turno grazie alla memoria integrata.

Capitolo 4

Validazione e Risultati

Dopo aver descritto il progetto sperimentale, l'architettura del sistema proposto e le tecniche adottate, è fondamentale verificarne l'efficacia rispetto agli obiettivi della ricerca. La validazione ha lo scopo di misurare, in maniera sistematica e rigorosa, quanto bene il sistema soddisfi le due domande di ricerca (RQ) su cui si fonda il progetto:

- **(RQ1)** Riassumere e interpretare correttamente i messaggi di commit Git, restituendo in forma sintetica l'intento alla base delle modifiche al codice;
- **(RQ2)** Supportare una comunicazione più efficace all'interno del team di sviluppo, generando report o risposte contestualizzate che facilitino la comprensione condivisa dell'evoluzione del progetto.

Questo capitolo si propone di illustrare in dettaglio i metodi di validazione adottati, motivandone la scelta in relazione agli obiettivi della ricerca, e di presentare i risultati emersi dalle diverse analisi. Tali risultati verranno successivamente interpretati e discussi in chiave critica, al fine di evidenziare i punti di forza del sistema, le eventuali limitazioni riscontrate e le implicazioni per un possibile impiego pratico.

L'analisi complessiva mira inoltre a delineare prospettive future di miglioramento e a verificare in che misura il sistema contribuisca in modo concreto al raggiungimento degli obiettivi definiti dalle domande di ricerca.

4.1 Approccio Generale

Entrando nel dettaglio del lavoro svolto, il processo di validazione è stato affrontato analizzando separatamente le due pipeline principali del sistema — e, di conseguenza, le due domande di ricerca — attraverso metodologie quantitative e qualitative, opportunamente adattate alla specificità del tool. Questo approccio integrato permette di rispondere alle due domande di ricerca in modo completo: combinando misure oggettive con analisi qualitative, ed evidenziando sia i risultati medi sia i casi anomali, si ottiene un quadro chiaro dei punti di forza e dei limiti del sistema proposto.

4.1.1 Pipeline Offline

Per la pipeline offline, data la natura del problema, è stato selezionato un sottoinsieme di 100 commit, per i quali è stato costruito un Golden Standard manuale: a ciascun commit sono stati associati due riassunti di riferimento redatti manualmente, uno generale e uno tecnico, utilizzati come base di confronto con i riassunti generati dal modello. Ciò ha permesso di condurre una valutazione quantitativa tramite metriche di similarità tra testo generato e riferimento.

In parallelo, è stata svolta una valutazione qualitativa basata sia da una componente umana che da una LLM (G-Eval), focalizzata su criteri quali accuratezza, completezza, leggibilità, utilità e profondità tecnica del riassunto, oltre che una generale. L'obiettivo è quello di avere un'analisi più interpretativa della qualità dei riassunti generati.

In ultima battuta, è stata condotta anche una valutazione mirata della qualità della classificazione, per verificare se e come i miglioramenti introdotti abbiano influenzato le prestazioni rispetto al lavoro di riferimento. In questo caso sono state utilizzate metriche classiche come Accuratezza, Richiamo e Precisione.

4.1.2 Pipeline Online

Per la parte online si è valutato come il chatbot agentico, potenziato da LLM e dalla knowledge base di progetto, sia in grado di rispondere a domande in linguaggio naturale sui commit e sulla la storia del repository. È stato definito un insieme di 50 domande tipiche che un utente potrebbe porre al sistema (ad esempio, *"Recuperami l'ultimo commit"*, *"Quali commit hanno introdotto la funzionalità X e chi li ha effettuati?"*, *"Quando è stato risolto il bug relativo al modulo Y?"*). Questo ha consentito di valutare sia le capacità di recupero delle informazioni sia quelle di sintesi ed esposizione del chatbot.

Anche in questo caso, la validazione è stata condotta seguendo un approccio duplice, che combina analisi quantitative e qualitative. Da un lato, è stato effettuato un confronto sistematico tra le risposte generate dal chatbot e un Golden Standard costruito manualmente, utilizzando metriche automatiche per misurarne la coerenza e la correttezza. Questo ha permesso di valutare in modo oggettivo la capacità del sistema di fornire risposte precise e pertinenti rispetto alle informazioni presenti nel repository.

Dall'altro lato, la componente qualitativa ha previsto un giudizio espresso sia da una componente umana che da LLM (G-Eval), applicando gli stessi criteri impiegati per la pipeline offline. L'obiettivo di questa seconda analisi è stato comprendere non solo quanto il chatbot fosse corretto dal punto di vista fattuale, ma anche quanto risultasse effettivamente chiaro, utile e informativo per un potenziale utente che desideri interrogare la storia del progetto in linguaggio naturale.

Accanto a queste due dimensioni principali, sono stati presi in esame anche alcuni indicatori specifici del comportamento del sistema, che consentono di approfondire la qualità dell'interazione tra il modello e i suoi strumenti di supporto. In particolare, è stata analizzata l'efficacia del modulo di retrieval nel reperire le informazioni più rilevanti, attraverso metriche come precisione, richiamo e NDCG; si è inoltre osservato l'uso dei tool interni da parte del chatbot — ad esempio, la capacità di accedere correttamente al codice dei commit quando necessario — e si è verificata la presenza di eventuali allucinazioni, ossia risposte contenenti informazioni non supportate dai dati reali.

Questo insieme articolato di analisi ha permesso di ottenere una visione completa delle prestazioni del sistema, evidenziando non solo la qualità delle risposte generate, ma anche il grado di affidabilità, consistenza e trasparenza del comportamento del chatbot nel contesto d'uso previsto.

4.2 Metriche e Indicatori

Per valutare quantitativamente la qualità dei testi generati — sia i riassunti dei commit (RQ1) sia le risposte del chatbot (RQ2) — sono state utilizzate sia metriche classiche di similarità con un testo di riferimento che metriche più innovative basate su LLM. A queste si affiancano indicatori qualitativi, derivanti da giudizio umano e G-Eval, che permettono di misurare aspetti più soggettivi, come utilità e leggibilità. L'integrazione dei due approcci consente dunque di ottenere un quadro più completo e bilanciato della qualità complessiva del sistema, combinando rigore numerico e sensibilità interpretativa.

A completamento dell'analisi, sono state considerate anche metriche specifiche su alcuni degli stumenti utilizzati, con l'obiettivo di misurare in modo più puntuale le loro prestazioni e il contributo alla qualità complessiva del sistema.

Nelle sezioni successive vengono presentate nel dettaglio tutte le metriche e gli indicatori selezionati, illustrandone le finalità specifiche e le motivazioni che ne hanno guidato la scelta.

4.2.1 Metriche Quantitative

Le metriche quantitative si basano su misurazioni numeriche e replicabili, che consentono di confrontare oggettivamente le prestazioni del sistema rispetto a un riferimento predefinito (ad esempio un riassunto umano o una risposta attesa). Esse mirano a catturare in forma sintetica la somiglianza lessicale o semantica tra il testo generato e quello di riferimento, fornendo così un'indicazione oggettiva della correttezza e della copertura informativa delle risposte prodotte.

ROUGE

ROUGE (Recall-Oriented Understudy for Gisting Evaluation) [83] è una metrica classica basata sul recall n-gram, ampiamente utilizzata per valutare riassunti. Il calcolo di ROUGE-N avviene confrontando gli N-grammi del testo generato con quelli del riferimento: il punteggio risultante riflette la frazione di contenuto del riferimento che è stato "coperto" dal riassunto generato. In altri termini, ROUGE misura quanto del contenuto informativo originale (riassunto di riferimento) viene recuperato dal modello. Punteggi ROUGE elevati (vicini a 1) indicano che molte parole o frasi chiave presenti nel riferimento compaiono anche nel generato, suggerendo quindi un'elevata aderenza informativa. In questo caso, si è calcolato ROUGE-1 (unigrammi), ROUGE-2 (bigrammi) e ROUGE-L (basato sulle sottosequenze comuni più lunghe) per avere un quadro sia di corrispondenza lessicale elementare sia di coerenza su frasi più lunghe. ROUGE è stato scelto perché fornisce una

prima indicazione quantitativa sulla completezza informativa dei riassunti/risposte generati rispetto ai contenuti attesi.

BLEU

BLEU (Bilingual Evaluation Understudy) [84] è una metrica di riferimento nella valutazione di sistemi di traduzione automatica, spesso adottata anche per il testo generato. Contrariamente a ROUGE, il BLEU è orientato alla precisione degli n-grammi: conta la quota di n-grammi del testo generato che trovano corrispondenza nel riferimento. Intuitivamente, BLEU premia riassunti che utilizzano parole ed espressioni simili al riferimento, penalizzando aggiunte non pertinenti. In questo lavoro, BLEU (in particolare nella variante BLEU-4 cumulativa) contribuisce a misurare la fedeltà lessicale delle generazioni: un punteggio BLEU alto indica che il modello non solo recupera il contenuto corretto, ma lo esprime con termini molto simili a quelli umani di riferimento (alta precisione). D'altra parte, BLEU tende a penalizzare la varietà lessicale e sinonomica, dunque valori bassi di BLEU potrebbero non indicare necessariamente scarsa qualità, ma piuttosto l'uso di una formulazione diversa (cosa comune nei testi generati da LLM). Si è incluso comunque BLEU per confrontare i risultati con lavori precedenti e avere un indicatore complementare a ROUGE.

METEOR

METEOR (Metric for Evaluation of Translation with Explicit ORdering) [85] è una metrica più avanzata progettata per correlare meglio con i giudizi umani rispetto a BLEU. Essa considera non solo corrispondenze esatte di parole, ma anche corrispondenze a livello di stem, sinonimi e riorganizzazioni parziali. In METEOR, precisione e recall unigram vengono combinati in un singolo punteggio, con un sistema di pesi e penalità che tiene conto di allineamenti allentati tra generato e riferimento. Si è scelto METEOR perché nel contesto dei riassunti di commit e delle risposte discorsive ci si aspetta che il modello possa esprimere gli stessi concetti con parole diverse; METEOR è più tollerante a tali variazioni lessicali ed è quindi un indicatore di somiglianza semantica più robusto del BLEU. Un punteggio METEOR elevato suggerisce che, anche se le parole esatte differiscono, il modello ha catturato gran parte delle informazioni chiave del riferimento (grazie a match sinonimici, ecc.). Nel presentare i risultati, verranno affiancati METEOR, ROUGE e BLEU per fornire una visione bilanciata tra aderenza letterale e aderenza concettuale.

BERTScore

Si tratta di una metrica basata su modelli di linguaggio pre-addestrati (ad es. BERT) che confronta il significato dei testi tramite le loro rappresentazioni in uno spazio vettoriale [86]. In pratica, BERTScore calcola, per ciascuna parola del testo generato, la similarità coseno con le rappresentazioni delle parole nel riferimento, stabilendo corrispondenze semantiche a livello di embedding. Vengono poi aggregati i risultati in termini di Precisione, Recall e F1-score. In questo lavoro, si è usato BERTScore per ottenere una misura di allineamento semantico globale tra il riassunto/risposta generato e il riferimento. Ad esempio, un BERTScore F1 alto (>0.85) significa che, considerate tutte le sfumature semantiche, il contenuto generato è molto vicino al contenuto atteso, anche se potrebbe usare parole differenti. BERTScore è stato scelto perché gli LLM potrebbero generare testi parafrastici di alta qualità difficili da valutare con semplici match lessicali: questa metrica permette di catturare somiglianze a livello di idea e contenuto sottostante, completando il quadro fornito da ROUGE, BLEU e METEOR. In letteratura recente sull'analisi automatica di testo, BERTScore ha mostrato una buona correlazione con i giudizi umani di qualità, il che giustifica il suo impiego come indicatore della bontà semantica delle generazioni.

4.2.2 Metriche Qualitative

Un' analisi puramente quantitativa non è sufficiente per valutare pienamente la qualità dei testi generati. Alcuni aspetti fondamentali — come la chiarezza espositiva, la leggibilità, la coerenza logica o l'utilità percepita — non possono essere catturati da un semplice confronto numerico. Per questo motivo, alle metriche automatiche è stata affiancata una valutazione qualitativa, basata sia su un giudizio umano che tramite G-Eval (si è usata la versione 2.5 flash di Google Gemini), che permette di analizzare la qualità del testo da una prospettiva più interpretativa.

In altre parole, mentre le metriche quantitative misurano quanto il testo generato si avvicina a quello atteso, la valutazione qualitativa indaga come e quanto bene il sistema riesca a comunicare in modo efficace, utile e comprensibile. L'integrazione dei due approcci consente dunque di ottenere un quadro più completo e bilanciato della qualità complessiva del sistema, combinando rigore numerico e sensibilità interpretativa.

I criteri considerati riflettono vari aspetti della qualità e utilità dei riassunti/risposte nel contesto d'uso previsto e hanno permesso poi di avere dei valori specifici utili a creare delle statistiche sulla bontà della soluzione. Nei vari test sperimentali, ognuna delle componenti è stata misurata su una scala da 1 a 5, dove 1 è Completamente Insoddisfacente e 5 è Completamente Soddisfacente.

Accuratezza

Indica quanto il contenuto generato sia corretto rispetto ai dati originali. Un riassunto accurato cattura l'intento reale del commit senza introdurre distorsioni o errori fattuali; una risposta accurata del chatbot fornisce informazioni veritiere rispetto alla cronologia del progetto. Questo criterio è cruciale per entrambe le RQ, poiché un riassunto inaccurato può fuorviare lo sviluppatore sull'intento delle modifiche, e una risposta inaccurata del chatbot può diffondere informazioni sbagliate nel team.

Completezza

Valuta la copertura informativa. Si verifica se il riassunto (o la risposta) include tutti gli elementi salienti che ci si aspetterebbe. Ad esempio, per un commit, un riassunto completo menziona tutte le modifiche di rilievo e il loro scopo; per una risposta del chatbot, completezza significa che tutti gli aspetti della domanda posta dall'utente trovano riscontro nella risposta. Questo criterio integra l'accuratezza: anche un riassunto accurato potrebbe essere incompleto se omette dettagli importanti.

Utilità

È un criterio più soggettivo che risponde alla domanda: quanto il riassunto/risposta risulta utile all'utente? Un riassunto utile dovrebbe aiutare un membro del team a capire rapidamente il perché e il cosa di un commit, possibilmente facilitando attività come code review, debugging o documentazione. Analogamente, una risposta del chatbot è utile se effettivamente chiarisce il dubbio dell'utente e gli risparmia tempo nella ricerca manuale di informazioni. Questo indicatore sintetizza un po' l'impatto pratico: testi magari accurati ma troppo vaghi o troppo dettagliati potrebbero essere valutati meno utili.

Leggibilità

Riguarda la forma espositiva del testo generato – grammatica, chiarezza, scorrevolezza e appropriatezza del linguaggio. Poiché i riassunti devono poter essere letti velocemente dai vari membri del gruppo di lavoro, è fondamentale che siano ben formulati, senza ambiguità o costrutti contorti. Lo stesso vale per le risposte del chatbot, le quali dovrebbero idealmente essere chiare e comprensibili al primo colpo. Questo fattore è importante perché un riassunto tecnicamente corretto ma poco leggibile vanificherebbe in parte il suo scopo.

Profondità Tecnica

Questo criterio è stato applicato in particolare ai riassunti tecnici dei commit (previsti dall'architettura in aggiunta al riassunto generale). Misura il grado di dettaglio tecnico fornito: ad esempio menzione specifica di file o componenti modificati, dettagli su implementazioni, riferimenti a codice, ecc. Un punteggio alto (5) indica un riassunto molto approfondito tecnicamente, utile per uno sviluppatore che voglia capire esattamente cosa è cambiato nel codice; un punteggio basso (1) indica un riassunto privo di dettagli tecnici (magari troppo generico). Questo indicatore è stato introdotto perché RQ1 prevede esplicitamente anche riassunti mirati agli sviluppatori, e si voleva misurare se il modello riuscisse ad aggiungere tali dettagli. Per i riassunti generali questo criterio non è applicabile, in quanto per definizione un riassunto generale non entra nei dettagli tecnici; infatti nella valutazione tali casi sono marcati come N/A.

Giudizio complessivo

Oltre ai singoli criteri, è stato dato un punteggio complessivo 1-5 che sintetizzi la qualità generale del riassunto/risposta. Questo overall è un'impressione generale che tiene conto di tutti gli aspetti (es. un testo potrebbe avere piccole imprecisioni ma essere comunque estremamente utile, e viceversa). Avere un giudizio complessivo aiuta a capire come gli eventuali trade-off fra criteri si riflettano sulla percezione generale dell'output.

4.2.3 Metriche Specifiche per il Retrieval

Nel contesto della pipeline online, la capacità del sistema di recuperare le informazioni corrette rappresenta un aspetto cruciale per il buon funzionamento dell'intera architettura. Poiché il chatbot si basa su un meccanismo di recupero informativo, la qualità del materiale che esso recupera dalle varie fonti a sua disposizione influenza direttamente la pertinenza e la correttezza delle risposte generate. Per valutare questa componente, è stata effettuata una misurazione mirata delle prestazioni del modulo di retrieval, inteso come il sistema incaricato di selezionare i documenti o frammenti informativi più rilevanti in risposta a una query in linguaggio naturale.

A tal fine, sono state adottate metriche consolidate nell'ambito dell'Information Retrieval, capaci di catturare sia la precisione dei risultati restituiti sia la qualità del loro ordinamento. La Precision@K (P@K) misura la proporzione di elementi rilevanti tra i primi K risultati, evidenziando quanto il sistema tenda a collocare in testa al ranking le informazioni effettivamente utili. La NDCG@K (Normalized Discounted Cumulative Gain) tiene conto non solo della rilevanza dei risultati ma anche della loro posizione, penalizzando quelli corretti che compaiono troppo

in basso nella lista. La Mean Average Precision (MAP) fornisce una visione complessiva della precisione mantenuta lungo l'intero ranking, mentre la Mean Reciprocal Rank (MRR) valuta quanto rapidamente il sistema riesca a fornire almeno una risposta corretta.

L'insieme di queste metriche consente di misurare la qualità complessiva del retrieval da più prospettive: accuratezza, stabilità e rapidità nel restituire contenuti pertinenti. Inoltre, esse forniscono un quadro quantitativo utile a interpretare anche i risultati qualitativi: un buon sistema di retrieval, infatti, tende a ridurre gli errori nella fase generativa successiva, migliorando la coerenza e la fedeltà semantica delle risposte fornite dal chatbot.

Per garantire un'analisi bilanciata, la valutazione è stata condotta a diversi livelli di profondità (ad esempio per $K = 5$, $K = 10$ e $K = 15$), osservando come la precisione vari al crescere della quantità di informazioni recuperate.

Oltre alla valutazione numerica del retrieval, è stato introdotto un controllo specifico sul corretto utilizzo dei tool da parte dell'agente, con l'obiettivo di verificare la coerenza tra le chiamate effettivamente effettuate e quelle attese per ciascuna tipologia di query. Questo controllo ha permesso di distinguere i casi in cui il chatbot abbia utilizzato in modo completo gli strumenti necessari, da quelli in cui l'esecuzione sia stata parziale o incompleta. Tale analisi fornisce un riscontro diretto sulla capacità dell'agente di orchestrare correttamente le proprie risorse interne — come i moduli di retrieval SQL o di accesso al codice — e contribuisce a valutare la robustezza del comportamento operativo del sistema.

È stato, inoltre, effettuato un controllo sul tasso di allucinazione delle risposte generate, inteso come la presenza di informazioni non supportate dai dati effettivamente recuperati. Il fenomeno è stato analizzato sia tramite valutazione umana sia tramite G-Eval, considerando tre livelli di classificazione (assenza, presenza parziale o presenza totale di allucinazione). Questa analisi qualitativa consente di stimare in modo più diretto la fedeltà semantica delle risposte rispetto alle fonti, integrando le metriche precedente esposte, offrendo una misura più realistica dell'affidabilità complessiva del sistema.

4.2.4 Metriche Specifiche per la Classificazione

Un ulteriore aspetto oggetto di validazione riguarda la componente di classificazione dei commit, che costituisce una fase preliminare fondamentale nella pipeline offline. Questa parte del sistema ha lo scopo di assegnare automaticamente a ciascun commit una categoria semantica — ad esempio feature, bug fix, refactoring — fornendo così una prima interpretazione funzionale delle modifiche al codice. La

qualità di questa classificazione è cruciale, poiché da essa dipende in larga misura la correttezza e la contestualizzazione dei riassunti generati in seguito.

Per valutarne l'efficacia sono state utilizzate tre metriche principali, ampiamente riconosciute nella letteratura sul machine learning: Accuratezza, Precisione e Richiamo.

L'accuratezza rappresenta la percentuale complessiva di commit classificati correttamente sul totale e fornisce una prima misura sintetica della prestazione generale. Tuttavia, quando le classi non sono perfettamente bilanciate, tale valore può risultare fuorviante. Per questo motivo, viene affiancata la precisione, che misura la proporzione di commit correttamente assegnati a una categoria rispetto a tutti quelli etichettati come appartenenti a quella categoria, e il richiamo, che indica invece la capacità del sistema di individuare tutti i commit effettivamente appartenenti a una determinata classe. Il bilanciamento tra queste due ultime metriche fornisce una misura più stabile delle prestazioni effettive del modello, evitando che il sistema venga premiato per un comportamento sbilanciato verso una sola direzione (ad esempio alta precisione ma basso richiamo).

Le metriche riportate sono calcolate a partire da una funzione di valutazione che, seguendo l' implementazione originale, tratta la classificazione come un problema binario. In particolare, tutti i commit appartenenti alle categorie diverse da *Other* vengono considerati come un'unica classe positiva, mentre la categoria *Other* rappresenta la classe negativa, in quanto definita come residuale (o di fallback). Di conseguenza, le metriche di precisione e richiamo si riferiscono alla capacità del modello di distinguere i commit "rilevanti" (cioè appartenenti a categorie specifiche) da quelli generici o non informativi.

Questa semplificazione permette di ottenere una misura sintetica delle prestazioni, ma non riflette pienamente la complessità del problema multi-classe, in cui ciascuna categoria tematica dovrebbe essere valutata separatamente. In una futura estensione, sarebbe opportuno calcolare le metriche per ciascuna classe e riportare valori medi per ottenere una stima più accurata della qualità complessiva del classificatore.

La valutazione è stata condotta su un sottoinsieme rappresentativo di commit per i quali era disponibile un'etichetta di riferimento, costruita manualmente già nel lavoro di partenza, e qui riutilizzata per un confronto diretto.

4.3 Limiti degli Approcci Utilizzati

Sebbene le metodologie di validazione adottate — comprendenti sia analisi quantitative basate su metriche automatiche sia valutazioni qualitative di tipo umano e tramite LLM — offrano una visione complessiva e coerente del comportamento del sistema, è necessario riconoscere alcuni limiti intrinsechi al disegno sperimentale. Le metriche automatiche forniscono una misura oggettiva e replicabile della qualità dei risultati, ma non riescono a cogliere appieno aspetti più sottili come la chiarezza comunicativa, la leggibilità o l'effettiva utilità per l'utente finale. D'altro canto, la componente qualitativa — pur permettendo di esplorare tali dimensioni — resta basata su un campione limitato e su giudizi soggettivi, difficilmente generalizzabili. Inoltre, le metriche di retrieval, pur efficaci nel misurare la precisione del recupero informativo, si fondano su un insieme di query predefinite e non riflettono necessariamente la varietà e la complessità delle richieste reali che un utente potrebbe formulare in un contesto operativo.

Un ulteriore limite riguarda la definizione stessa di "buon" riassunto o "buona" risposta. In letteratura non esiste ancora un consenso univoco su quali caratteristiche definiscano in modo oggettivo la qualità di un testo generato in ambito software, né metodi di valutazione pienamente consolidati. Le metriche automatiche correnti si basano prevalentemente sulla similarità testuale e tendono a trascurare la dimensione semantica o l'utilità pratica del contenuto, mentre i giudizi umani, seppur più aderenti all'esperienza d'uso, restano inevitabilmente soggettivi. Di conseguenza, la qualità percepita di un riassunto o di una risposta dipende fortemente dal contesto e dal profilo dell'utente, così come dallo scopo specifico della consultazione. Inoltre, il dataset di test — pur accuratamente selezionato — non può rappresentare l'intera eterogeneità dei repository software reali, nei quali i messaggi di commit possono variare notevolmente per lunghezza, stile, grado di dettaglio e qualità descrittiva.

È, inoltre, opportuno sottolineare che la validazione condotta si concentra prevalentemente su aspetti tecnici e di correttezza formale, includendo valutazioni di natura personale, ma non misura ancora l'impatto reale del sistema all'interno di un contesto operativo di team di sviluppo. In altre parole, la sperimentazione finora svolta consente di stimare la qualità del sistema da un punto di vista funzionale, ma non fornisce ancora evidenze dirette sul suo contributo effettivo ai processi collaborativi di lavoro. Sarebbe pertanto auspicabile che le analisi qualitative e i testi di riferimento venissero, in futuro, estesi e validati anche dal gruppo di sviluppo del progetto MuJS, al fine di ottenere un riscontro più oggettivo e aderente alla realtà produttiva. In questa direzione, una possibile evoluzione del lavoro potrà prevedere una validazione in the loop, in cui il sistema venga effettivamente impiegato in scenari di sviluppo collaborativo, consentendo di osservare metriche legate alla produttività, alla chiarezza comunicativa e alla qualità percepita delle

interazioni.

Allo stesso modo, potranno essere esplorate tecniche di valutazione più avanzate, come quelle proposte dal framework RAGAS, che permettono di stimare in modo più diretto la coerenza tra fonti recuperate e contenuto generato, individuando in maniera automatica fenomeni di allucinazione o di mancanza di fedeltà alle fonti.

4.4 Risultati Sperimentali

In questa sezione vengono presentati e commentati i risultati delle valutazioni condotte, organizzati per ciascuna delle due pipeline del progetto (Offline e Online). I risultati includono sia metriche quantitative oggettive, sia valutazioni qualitative soggettive, allo scopo di rispondere ai due quesiti di ricerca nel contesto dell'impatto degli strumenti GenAI sullo sviluppo software. Di seguito, per ciascuna pipeline, si illustrano i principali indicatori numerici e qualitativi riscontrati, seguite da un'analisi delle osservazioni più significative emerse.

4.4.1 Pipeline Offline

Risultati Quantitativi

Nella pipeline offline, la qualità dei riassunti generati è stata valutata tramite metriche di similarità rispetto ai testi di riferimento, Golden Standard formato dai riassunti tecnici e generali di 100 commit. La tabella e la figura che seguono riportano i punteggi medi ottenuti sia per i riassunti di tipo generale sia per quelli tecnici.

Metrica	Generale	Tecnico
ROUGE-1	0.4229	0.4539
ROUGE-2	0.1215	0.1368
ROUGE-L	0.2458	0.2233
BLEU	0.0890	0.1226
METEOR	0.3337	0.3145
BERT Precision	0.8892	0.8756
BERT Recall	0.8877	0.8564
BERT F1	0.8884	0.8658

Tabella 4.1: Risultati medi per i test quantitativi dei riassunti

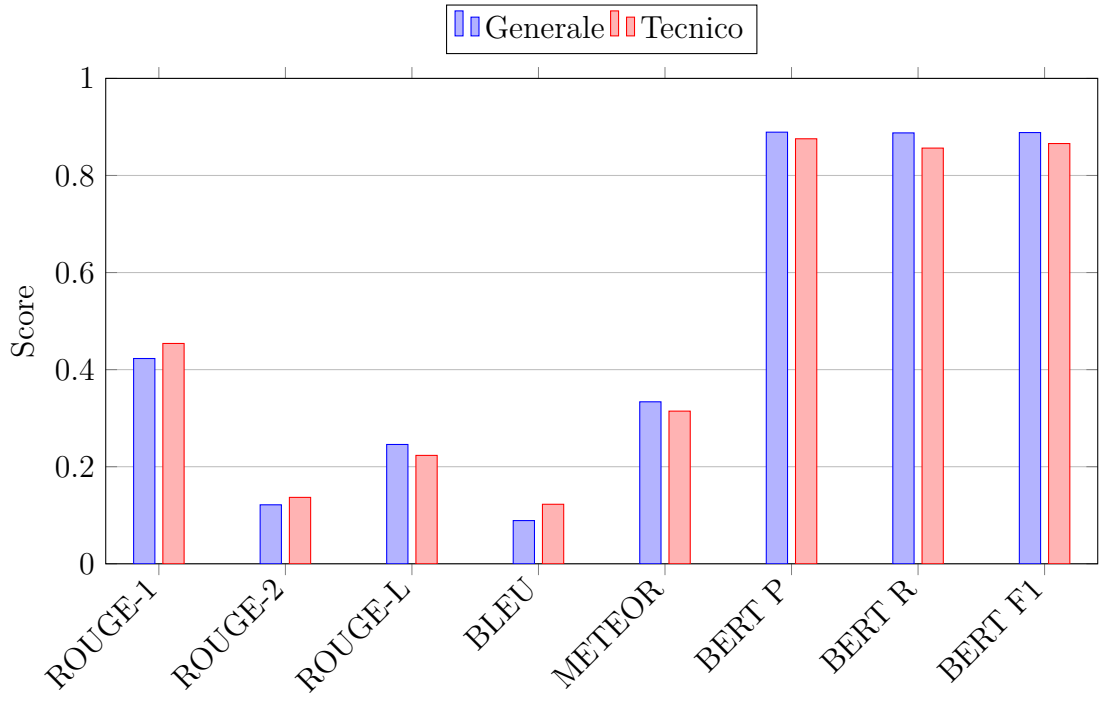

Figura 4.1: Risultati medi per metrica

Come si può osservare, i riassunti mostrano un buon livello di sovrapposizione con i testi di riferimento. In media, i riassunti generali ottengono valori di ROUGE-1 intorno a 0.42 e di ROUGE-2 pari a 0.12, mentre i riassunti tecnici raggiungono punteggi lievemente superiori (ROUGE-1 ~ 0.45, ROUGE-2 ~ 0.14), a indicare una copertura marginalmente migliore dei contenuti originali. Anche il punteggio BLEU risulta leggermente più alto per i riassunti tecnici (0.12 contro 0.09), segnalando una maggiore aderenza lessicale e strutturale rispetto ai testi di riferimento. Al contrario, la metrica METEOR mostra valori simili per entrambe le tipologie ($\sim$ 0.33–0.31), suggerendo una qualità complessiva comparabile dal punto di vista lessicale e semantico.

Passando alle metriche basate su embedding (BERTScore), i valori risultano elevati per entrambi i tipi di riassunto: il punteggio medio di BERT F1 si colloca tra 0.87 e 0.89, con una varianza minima (la quasi totalità dei riassunti presenta punteggi compresi tra 0.85 e 0.90). Questo indica che, nonostante le differenze lessicali, i riassunti generati riescono a catturare gran parte del contenuto semantico dei commit. Si tratta probabilmente dell'aspetto più rilevante, poiché — al di là dell'allineamento letterale — è la coerenza semantica a determinare la qualità effettiva del riassunto.

Metrica	Generale		Tecnico	
	Min	Max	Min	Max
ROUGE-1	0.1852	0.6061	0.2849	0.5847
ROUGE-2	0.0075	0.2769	0.0271	0.2393
ROUGE-L	0.1185	0.3896	0.1259	0.3099
BLEU	0.0000	0.2020	0.0000	0.2879
METEOR	0.1785	0.4766	0.1963	0.5093
BERT Precision	0.8325	0.9269	0.8302	0.9165
BERT Recall	0.8249	0.9184	0.8232	0.8974
BERT F1	0.8249	0.9175	0.8313	0.9062

Tabella 4.2: Valori minimi e massimi per i test quantitativi dei riassunti

Analizzando in modo più approfondito la distribuzione dei punteggi ottenuti, emergono alcune differenze e tendenze interessanti tra le due tipologie di riassunti.

Per quanto riguarda i riassunti generali - figure 4.2 e 4.3 -, le metriche ROUGE mostrano una variabilità moderata. La maggior parte dei riassunti si colloca nella fascia 0.4–0.6 per ROUGE-1 (65 casi su 100), mentre pochi esempi si trovano in fasce estreme, con un minimo di circa 0.18 e un massimo di 0.61. I valori di ROUGE-2 risultano generalmente più bassi, concentrandosi prevalentemente tra 0.0 e 0.2 (91% dei casi), a indicare una limitata sovrapposizione a livello di bigrammi. Anche ROUGE-L si distribuisce in modo simile, con la maggioranza dei punteggi compresi tra 0.2 e 0.4. Le metriche BLEU e METEOR confermano questa tendenza: BLEU risulta inferiore a 0.2 per quasi tutti i riassunti (valore

Metrica	Generale			Tecnico		
	25°	Mediana	75°	25°	Mediana	75°
ROUGE-1	0.3854	0.4366	0.4679	0.4139	0.4640	0.4932
ROUGE-2	0.0824	0.1247	0.1603	0.1062	0.1382	0.1655
ROUGE-L	0.2069	0.2442	0.2861	0.1954	0.2226	0.2488
BLEU	0.0000	0.0614	0.0974	0.0813	0.1195	0.1624
METEOR	0.2807	0.3337	0.3839	0.2723	0.3097	0.3485
BERT Precision	0.8773	0.8902	0.9020	0.8646	0.8759	0.8864
BERT Recall	0.8773	0.8909	0.9003	0.8462	0.8557	0.8678
BERT F1	0.8778	0.8895	0.8993	0.8563	0.8664	0.8759

Tabella 4.3: Quartili (25°, Mediana, 75°) per i riassunti generali e tecnici

medio pari a 0.09), mentre METEOR presenta una distribuzione leggermente più bilanciata, con la maggioranza dei valori nella fascia 0.2–0.4 e alcuni casi fino a 0.47. Al contrario, le metriche basate su embedding semantici (BERTScore) mostrano valori sistematicamente elevati e stabili, con F1 medio pari a 0.89 e tutti i punteggi compresi tra 0.82 e 0.92, indicando una forte coerenza semantica tra i riassunti e i testi di riferimento, anche in presenza di variazioni lessicali.

I riassunti tecnici - figure 4.4 e 4.5 - mostrano un quadro analogo, ma con prestazioni leggermente superiori. I punteggi medi di ROUGE-1 (0.45) e ROUGE-2 (0.14) risultano più alti rispetto a quelli dei riassunti generali, con una concentrazione della maggioranza dei valori nella fascia 0.4–0.6 per ROUGE-1 e 0.0–0.2 per ROUGE-2. Anche BLEU evidenzia un miglioramento, con il 12% dei casi nella fascia 0.2–0.4 e un valore medio di 0.12. La metrica METEOR mantiene una distribuzione simile a quella precedente (prevalentemente tra 0.2 e 0.4), mentre i punteggi BERT si attestano su valori lievemente inferiori ma comunque elevati (F1 medio $\sim$ 0.87, range 0.83–0.91).

Nel complesso, i risultati suggeriscono che il modello riesce a catturare efficacemente le informazioni salienti dei commit, mantenendo un buon equilibrio tra accuratezza lessicale e coerenza semantica. Le lievi differenze tra riassunti generali e tecnici evidenziano come il contenuto più strutturato dei commit tecnici favorisca una generazione leggermente più aderente ai testi di riferimento.

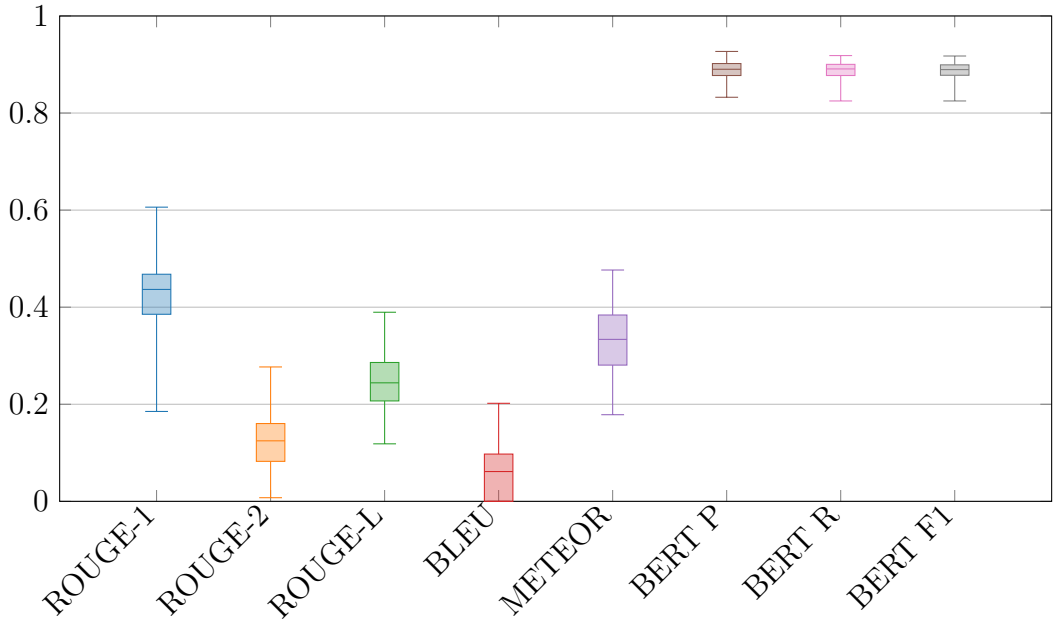

Figura 4.2: Riassunti generali — boxplot

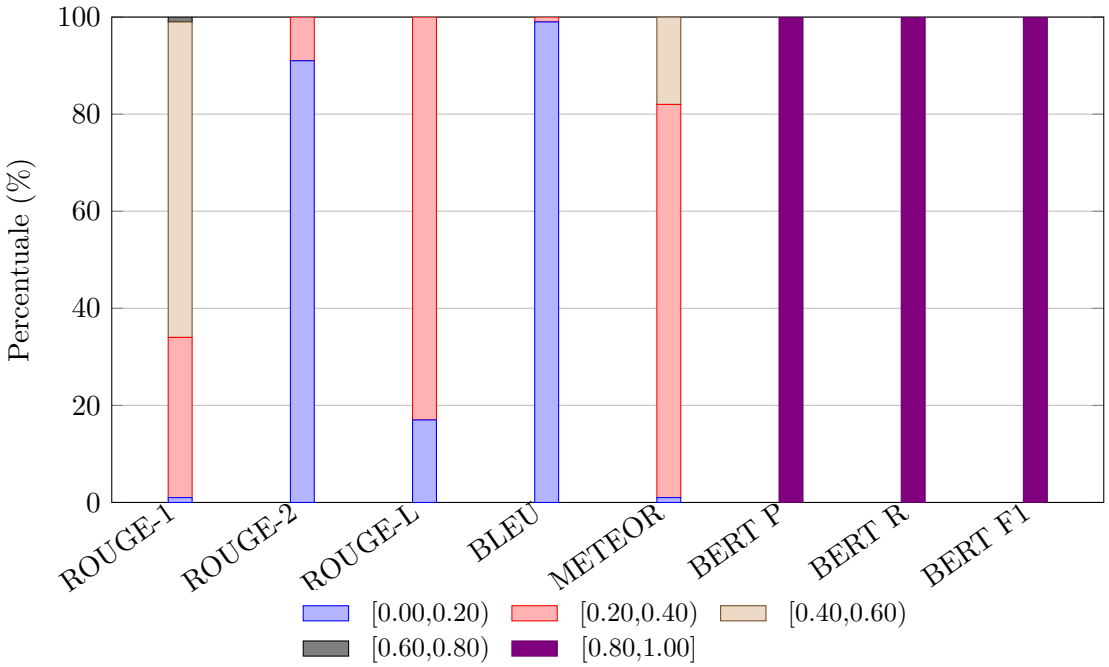

Figura 4.3: Riassunti generali — distribuzione per fasce (stacked 100%)

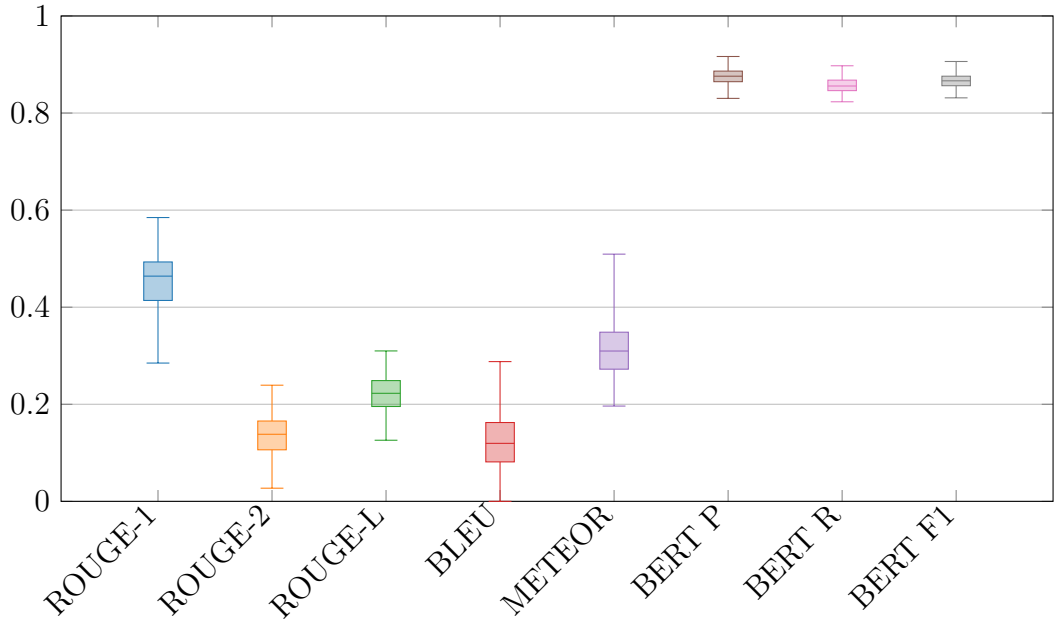

Figura 4.4: Riassunti tecnici - boxplot

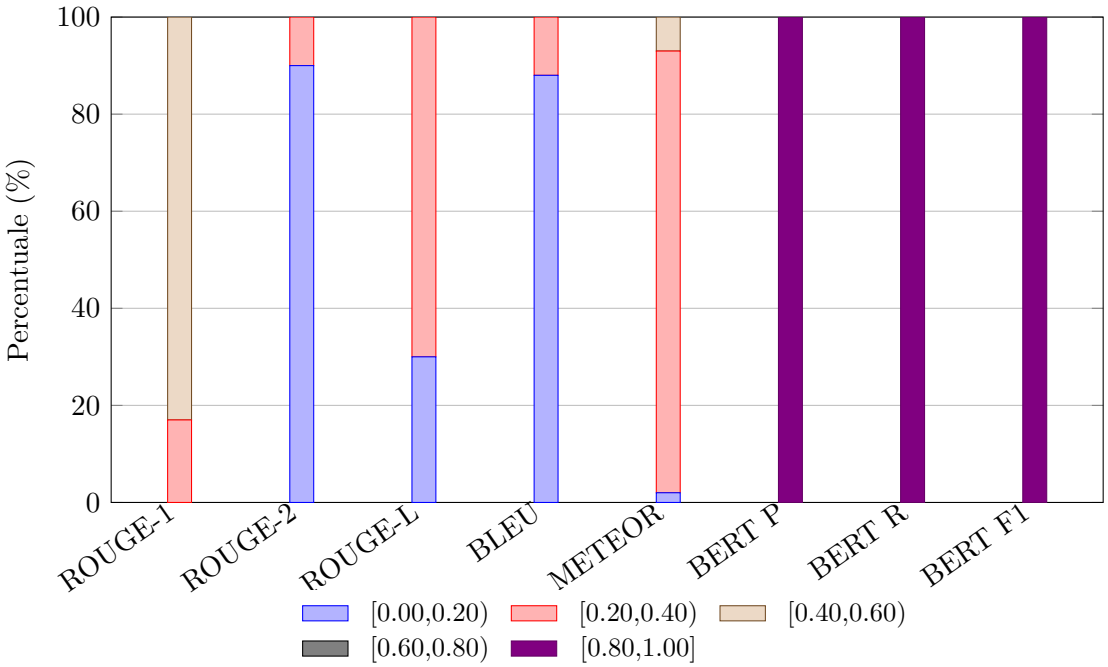

Figura 4.5: Riassunti tecnici — distribuzione per fasce (stacked 100%)

Risultati Qualitativi

La valutazione qualitativa dei riassunti generati ha previsto l'analisi dei giudizi espressi, su scala da 1 a 5, per ciascuno dei criteri definiti in precedenza: accuratezza, completezza, utilità, leggibilità e profondità tecnica (solo per i riassunti tecnici). I punteggi sono stati raccolti sia per i riassunti generali sia per quelli tecnici, al fine di comprendere le differenze in termini di chiarezza, correttezza e valore informativo percepito.

Metrica	Generale		Tecnico	
	Umano	G-Eval	Umano	G-Eval
accuracy	3.53	3.97	2.83	3.18
completeness	3.04	3.64	2.53	3.09
usefulness	3.10	3.97	2.59	3.23
readability	4.38	4.39	3.97	3.67
technological depth	N/A	N/A	2.33	2.85
overall	3.51	3.99	2.83	3.20

Tabella 4.4: Valori medi complessivi, sia umani che g-eval, per riassunti generali e tecnici

Come si può vedere dalla tabella 4.4, G-Eval tende ad assegnare in media punteggi superiori rispetto alla valutazione umana su accuracy, completeness e usefulness, sia per i riassunti generali sia per quelli tecnici; la readability risulta allineata (generale) o leggermente inferiore (tecnico) nelle valutazioni G-Eval. Sui riassunti tecnici, technological_depth emerge come dimensione più sfidante per entrambi i canali, con valori medi inferiori alle altre metriche.

Considerando l'intero quadro, i risultati qualitativi possono essere valutati come complessivamente positivi. I punteggi medi superiori a 3.5 nelle principali dimensioni per i riassunti generali e intorno a 3 per quelli tecnici indicano che il sistema è in grado di produrre testi generalmente corretti, chiari e utili, in linea con gli obiettivi della RQ1. La leggibilità rappresenta uno dei punti di forza più costanti, con valori prossimi al massimo in entrambe le tipologie di riassunto, a conferma della qualità linguistica e della scorrevolezza del testo generato. Le differenze più marcate emergono invece nei riassunti tecnici, dove l'accuratezza e la profondità tecnica risultano più difficili da mantenere costantemente elevate, segno che la componente semantica legata al codice richiede ancora una rappresentazione più precisa del contesto.

Riassunti generali Entrando nel dettaglio dei risultati qualitativi per i riassunti generali, è possibile notare che G-Eval tende a collocare la maggioranza dei campioni nella fascia alta per accuracy, usefulness e readability, con mediane pari a 5. I giudizi umani confermano l'eccellente leggibilità (mediana 4 e 75° quantile 5) e mostrano una distribuzione più conservativa su completeness e usefulness, concentrate tra 2 e 4. L'overall segue l'andamento dei singoli criteri: più elevato in G-Eval (mediana 4.50) e comunque positivo per la parte umana (mediana 3.75). Questi risultati indicano che la forma espositiva è costantemente chiara, mentre la copertura informativa e l'impatto pratico sono percepiti come buoni.

Metrica	Umano		G-Eval	
	Min	Max	Min	Max
accuracy	1.0	5.0	1.0	5.0
completeness	1.0	5.0	1.0	5.0
usefulness	1.0	5.0	1.0	5.0
readability	3.0	5.0	1.0	5.0
overall	1.5	5.0	1.0	5.0

Tabella 4.5: Minimi e massimi per i riassunti generali (Umano vs G-Eval)

Metrica	Umano			G-Eval		
	25°	Mediana	75°	25°	Mediana	75°
accuracy	3.00	4.00	5.00	3.00	5.00	5.00
completeness	2.00	3.00	4.00	3.00	4.00	5.00
usefulness	2.00	3.00	4.00	3.00	5.00	5.00
readability	4.00	4.00	5.00	4.00	5.00	5.00
overall	2.75	3.75	4.25	3.44	4.50	4.75

Tabella 4.6: Quartili per i riassunti generali (Umano vs G-Eval)

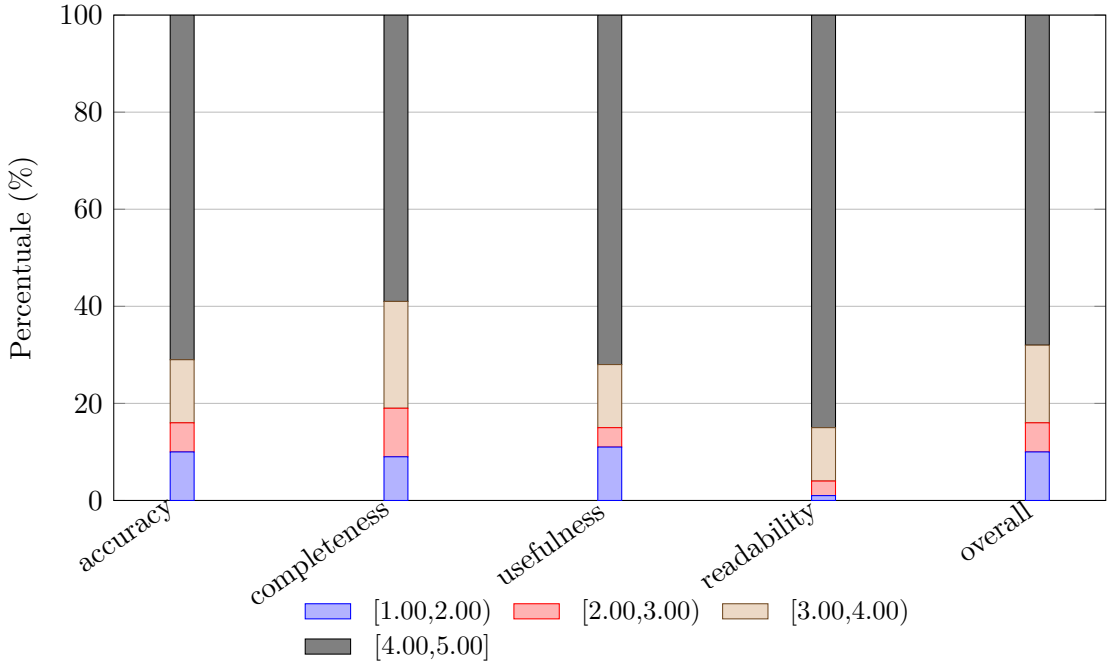

Figura 4.6: Riassunti generali — distribuzione per fasce (G-Eval)

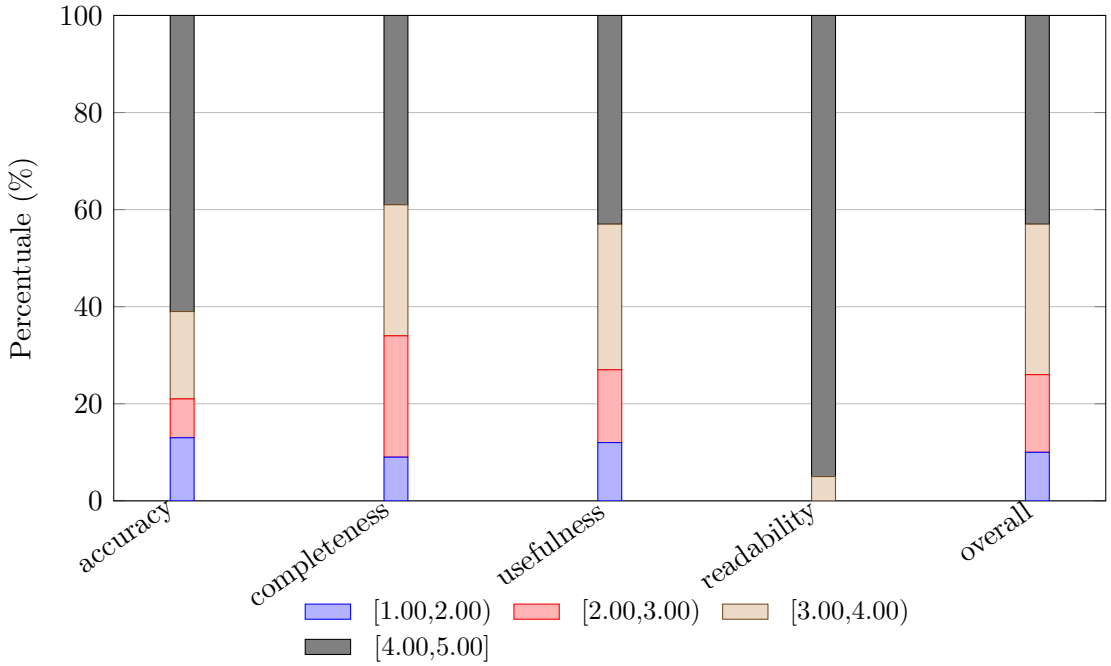

Figura 4.7: Riassunti generali — distribuzione per fasce (umano)

Riassunti tecnici Per i riassunti tecnici si osserva, invece, una maggiore dispersione rispetto ai generali, specie su accuracy, completeness e usefulness. In G-Eval le mediane sono pari a 3 per accuracy, completeness e usefulness, con il 34–49% dei casi nelle fasce alte; la readability resta solida (mediana 4). La dimensione technological depth, qui presente, si colloca mediamente più in basso rispetto alle altre (mediana 2 e 75° quantile 3), segnalando spazio di miglioramento nel dettaglio implementativo esplicito. I giudizi umani confermano la buona leggibilità (mediana 4, distribuzione fortemente sbilanciata verso la fascia 4–5), mentre risultano più stringenti su accuracy, completeness e usefulness, per le quali la massa si concentra tra 2 e 3. L'overall riflette questo quadro più prudente, con mediane inferiori rispetto rispetto ai riassunti generali.

Metrica	Umano		G-Eval	
	Min	Max	Min	Max
accuracy	1.0	5.0	1.0	5.0
completeness	1.0	5.0	1.0	5.0
usefulness	1.0	5.0	1.0	5.0
readability	2.0	5.0	1.0	5.0
technological_depth	1.0	4.0	1.0	5.0
overall	1.2	4.4	1.0	5.0

Tabella 4.7: Minimi e massimi per i riassunti tecnici (Umano vs G-Eval)

Metrica	Umano			G-Eval		
	25°	Mediana	75°	25°	Mediana	75°
accuracy	2.00	3.00	4.00	2.00	3.00	5.00
completeness	2.00	3.00	3.00	2.00	3.00	4.00
usefulness	2.00	3.00	3.00	2.00	3.00	4.00
readability	4.00	4.00	4.00	3.00	4.00	4.00
technological_depth	2.00	2.00	3.00	2.00	2.00	3.00
overall	2.40	2.90	3.40	2.20	3.20	4.20

Tabella 4.8: Quartili per i riassunti tecnici (Umano vs G-Eval)

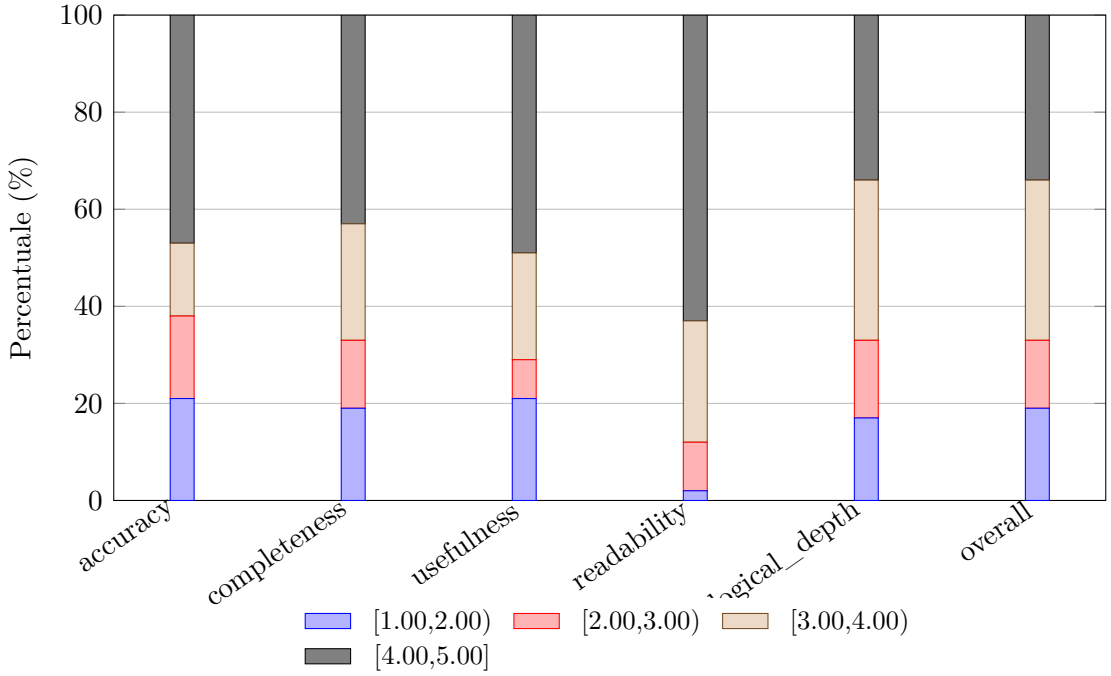

Figura 4.8: Riassunti tecnici — distribuzione per fasce (G-Eval)

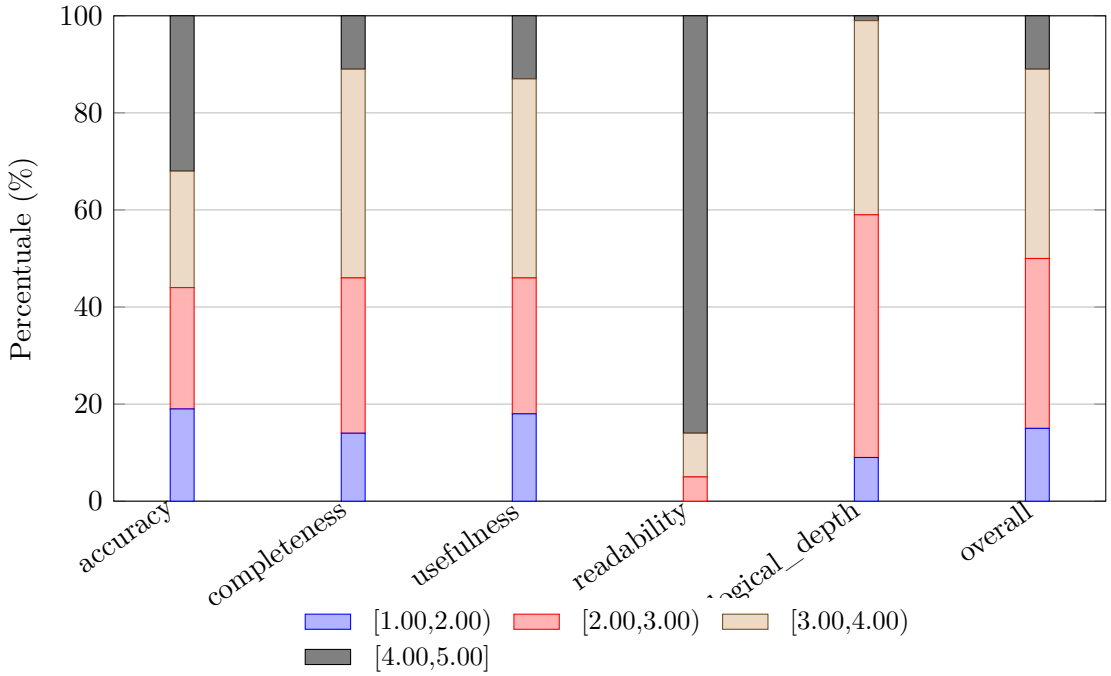

Figura 4.9: Riassunti tecnici — distribuzione per fasce (umano)

Risultati della Classificazione

Un componente fondamentale della pipeline offline è la classificazione automatica dei commit in categorie tematiche (es. *feature*, *bug fix*, *refactoring*, ecc.). Si ricorda che le metriche riportate in tabella 4.9 sono calcolate a partire da una funzione di valutazione che, seguendo l' implementazione originale, tratta la classificazione come un problema binario.

Di seguito le prestazioni ottenute nei diversi setting sperimentali:

Esperimento	**Precisione**	**Richiamo**	**Accuratezza**
Originale	0.59	0.46	–
Esempi originali con LLaMA 3.1 8B	0.67	1.00	0.67
Nuovi esempi con LLaMA 3.1 8B	0.73	0.99	0.72

Tabella 4.9: Prestazioni del modulo di classificazione dei commit

Nel progetto originario, il classificatore ottiene una precisione di 0.59 a fronte di un richiamo di 0.46. Ciò indica che meno della metà dei commit sono stati assegnati alla categoria corretta (richiamo 46%), e tra quelli etichettati come appartenenti a una data categoria, solo il 59% erano effettivamente corretti (precisione 59%).

Introducendo un modello LLM più potente (LLaMA 3.1 da 8 miliardi di parametri), le prestazioni migliorano sensibilmente. In particolare, il richiamo raggiunge il 100% mentre la precisione sale a 0.67. Questo si traduce in un'accuratezza globale del 67%: il modello classifica correttamente circa due terzi dei commit. Usando invece esempi ex-novo (rielaborati o aggiuntivi) per il few-shot, si ottiene un ulteriore incremento di performance: la precisione arriva a 0.73, mantenendo un richiamo molto alto. L'accuratezza complessiva migliora fino al 72%. Questo risultato suggerisce che la qualità e la varietà degli esempi forniti al modello hanno un impatto positivo, permettendo al classificatore di essere più selettivo (precisione più alta, meno falsi positivi) pur continuando a individuare quasi tutti i casi rilevanti (richiamo prossimo a 1).

4.4.2 Pipeline Online

Risultati Quantitativi

Nella pipeline online, la qualità delle risposte del chatbot è stata valutata rispetto a un Golden Standard costruito manualmente su 50 possibili query che vanno ad esplorare casistiche tipiche per cui potrebbe essere utilizzato, i casi limite e gli aspetti tecnici. Le metriche riportate sono le medesime già presenti per la pipeline offline, ovvero ROUGE-1/2/L, BLEU, METEOR e BERTScore (Precision, Recall, F1). Di seguito si presentano i valori medi, i range min–max, i quartili e le distribuzioni per fasce.

Metrica	Media	Min	Max	25°	Mediana	75°
ROUGE-1	0.4131	0.0488	1.0000	0.3066	0.4035	0.5137
ROUGE-2	0.2144	0.0000	1.0000	0.0707	0.1389	0.3154
ROUGE-L	0.3114	0.0488	1.0000	0.1692	0.2336	0.4443
BLEU	0.0759	0.0000	1.0000	0.0093	0.0632	0.1630
METEOR	0.2805	0.0113	0.9977	0.1879	0.2670	0.3401
BERT Precision	0.8614	0.7505	1.0000	0.8307	0.8513	0.8946
BERT Recall	0.8629	0.7688	1.0000	0.8261	0.8552	0.8950
BERT F1	0.8615	0.7919	1.0000	0.8297	0.8517	0.8873

Tabella 4.10: Statistiche riassuntive per le metriche quantitative del chatbot

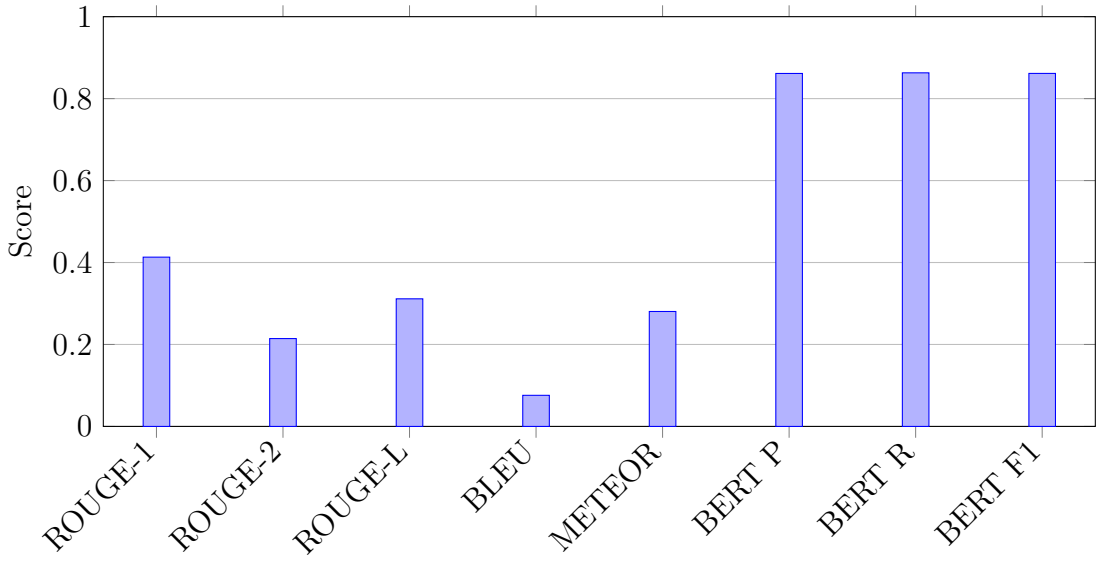

Figura 4.10: Risultati medi per metrica

I risultati mostrano come le distribuzioni tendono a concentrarsi nella parte bassa per le metriche più lessicali, in particolare ROUGE-2 e BLEU, mentre le metriche più semantiche, Bertscore, si confermano stabilmente alte (oltre il 90% dei casi in [0.80,1.00] per F1), a indicare coerenza semantica con il Golden Standard. I quartili mostrano mediane solide (es. ROUGE-1 ~ 0.40), ma con coda lunga positiva (max ≈ 1.0) dovuta alla presenza di una query con risposta perfettamente allineata al riferimento. In generale, i valori sono in linea con ciò che si era già ottenuto con la pipeline offline.

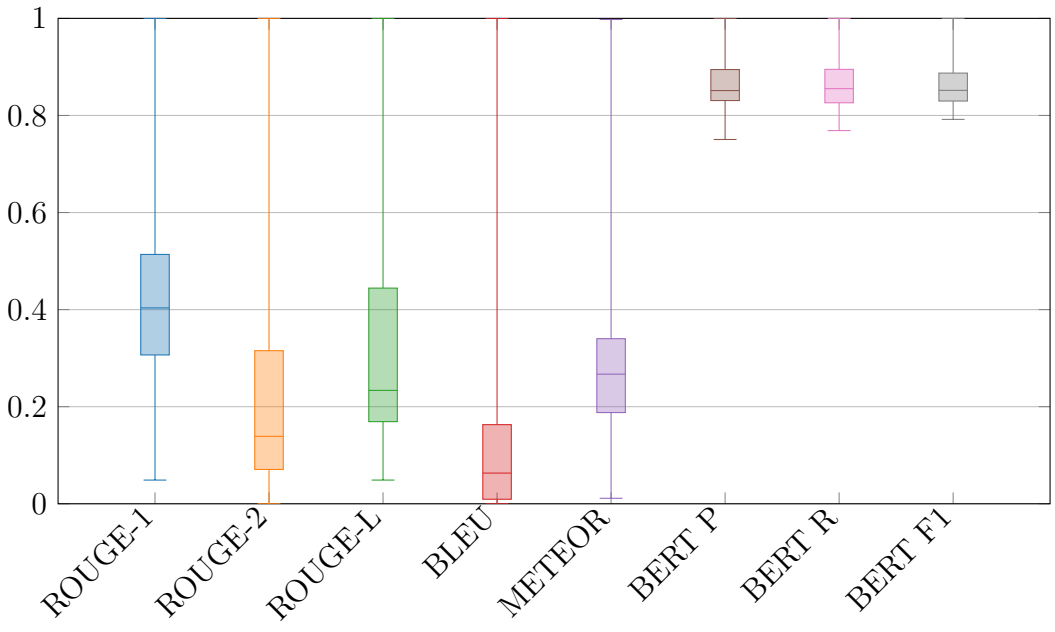

Figura 4.11: Metriche quantitative chatbot - boxplot

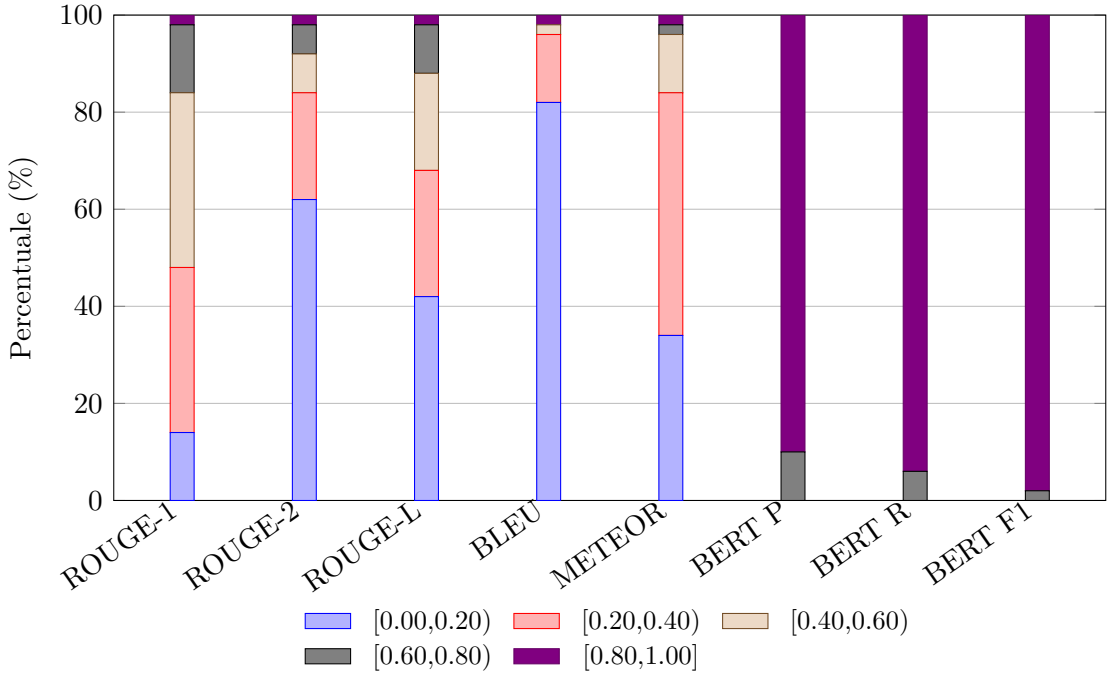

Figura 4.12: Metriche quantitative chatbot - Distribuzione per fasce (stacked 100%)

Risultati Qualitativi

Le valutazioni qualitative mostrano valori medi complessivamente elevati in tutte le metriche, con un andamento stabile tra giudizi umani e G-Eval. Le dimensioni *accuracy* e *usefulness* mantengono valori medi superiori a 4.2 in entrambi i casi, a indicare che le risposte fornite risultano generalmente corrette e rilevanti rispetto alle richieste dell'utente. La *readability* presenta i punteggi più alti, con medie prossime al valore massimo (4.78 per la valutazione umana e 4.86 per G-Eval) e il 100% dei giudizi collocati nella fascia più alta per il modello automatico. Ciò conferma che il linguaggio utilizzato dal chatbot è chiaro, coerente e facilmente leggibile anche in contesti tecnici.

La metrica *completeness* rappresenta invece l'aspetto più variabile: la valutazione umana ha attribuito una media pari a 4.32, mentre G-Eval si mostra più conservativo (3.74). Questo divario suggerisce che, sebbene le risposte siano ritenute accurate e comprensibili, il modello automatico tende a penalizzare lievi omissioni o mancanza di dettagli accessori, pur non compromettendo la qualità complessiva della risposta.

I giudizi "*overall*" rimangono alti (4.44 per la parte umana e 4.34 per G-Eval), con

oltre l'85% delle valutazioni concentrate nella fascia [4,5]. La distribuzione dei punteggi conferma una generale coerenza tra le due modalità di valutazione: entrambe evidenziano una forte concentrazione nelle fasce alte e una bassa dispersione, come indicato dai quartili superiori (mediane ≥ 4.5).

I risultati indicano che la pipeline online mantiene livelli qualitativi analoghi a quelli della pipeline offline, garantendo risposte consistenti, corrette e leggibili anche in contesti d'uso interattivi. L'unica differenza di rilievo riguarda la *completeness*, che rimane il parametro più sensibile al tipo di valutazione, ma senza impatto significativo sulla percezione complessiva della qualità.

Metrica	Umano	G-Eval
accuracy	4.64	4.60
completeness	4.32	3.74
usefulness	4.48	4.20
readability	4.78	4.86
overall	4.44	4.34

Tabella 4.11: Valori medi delle metriche qualitative.

Metrica	Umano	G-Eval
	Min – Max	Min – Max
accuracy	1 – 5	2 – 5
completeness	1 – 5	1 – 5
usefulness	1 – 5	2 – 5
readability	1 – 5	4 – 5
overall	1 – 5	2 – 5

Tabella 4.12: Valori minimi e massimi delle valutazioni qualitative.

Metrica	Umano			G-Eval		
	25°	Mediana	75°	25°	Mediana	75°
accuracy	4.00	5.00	5.00	4.00	5.00	5.00
completeness	4.00	4.50	5.00	3.00	4.00	5.00
usefulness	4.00	5.00	5.00	4.00	5.00	5.00
readability	5.00	5.00	5.00	5.00	5.00	5.00
overall	4.00	5.00	5.00	4.00	4.50	5.00

Tabella 4.13: Quartili per le valutazioni qualitative.

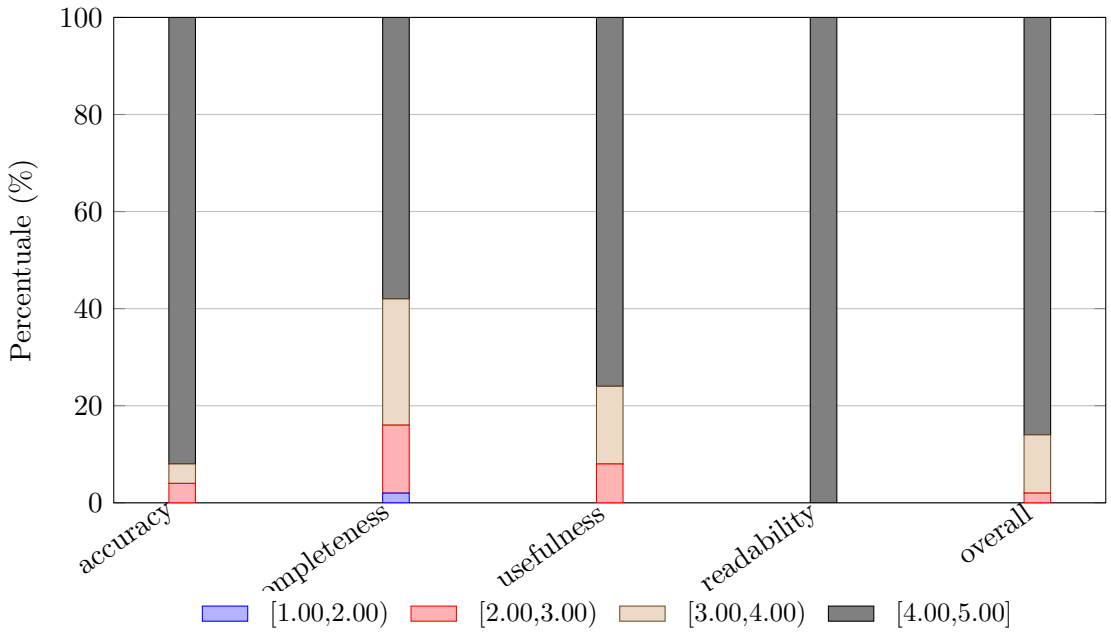

Figura 4.13: Metriche qualitative chatbot G-Eval - Distribuzione per fasce (stacked 100%)

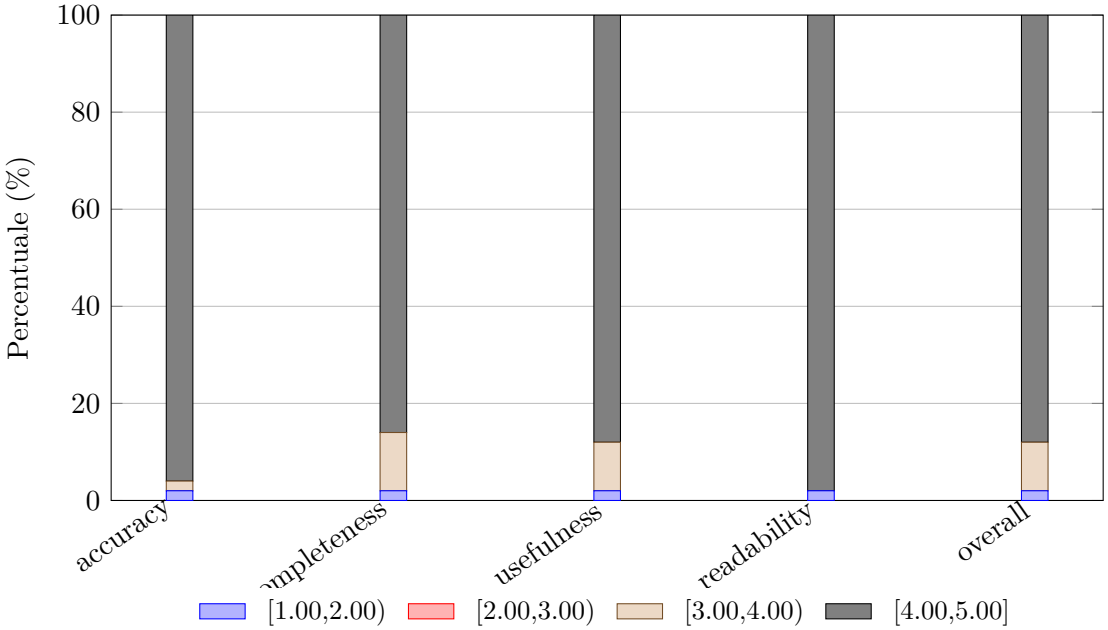

Figura 4.14: Metriche quantitative chatbot Umano - Distribuzione per fasce (stacked 100%)

Risultati del Retrival

Il comportamento del sistema di retrieval nella pipeline online, basato sulla presenza di quattro tool a disposizione dell'agente, è stato valutato considerando tre componenti principali: metriche standard di information retrival, invocazione dei tool e hallucination rate.

Metriche di Information Retrival Per le metriche standard si è utilizzate le classiche Precision@K (P@K), Normalized Discounted Cumulative Gain (NDCG@K), Mean Average Precision (MAP) e Mean Reciprocal Rank (MRR). Le misurazioni sono state effettuate per differenti valori di K (5, 10 e 15).

I risultati evidenziano prestazioni complessivamente elevate. Il valore di NDCG@K risulta stabile e superiore a 0.84 per tutti i livelli di K, indicando che i documenti più rilevanti vengono costantemente posizionati ai primi posti del ranking. Il valore di MAP (0.8130) e MRR (0.8847) conferma un buon comportamento medio e una risposta efficace già ai primi risultati. La precision@K decresce come previsto all'aumentare di K, passando da 0.38 a 0.17, comportamento tipico dei modelli di retrieval ben calibrati.

Metrica	K=5	K=10	K=15
P@K	0.3810	0.2476	0.1730
NDCG@K	0.8436	0.8518	0.8509
MAP	0.8130	0.8130	0.8130
MRR	0.8847	0.8847	0.8847

Tabella 4.14: Metriche di retrieval per diversi valori di K.

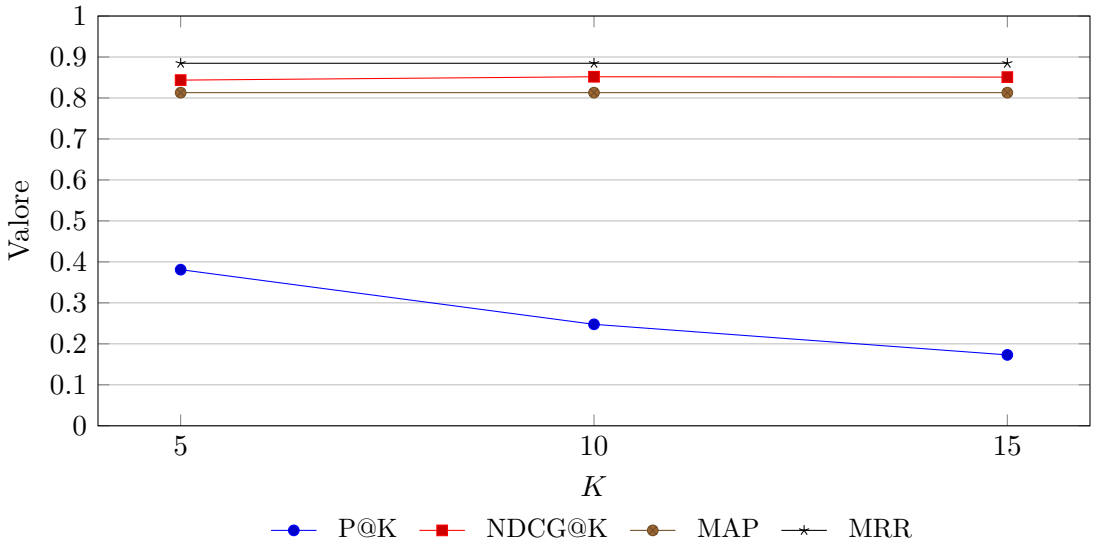

Figura 4.15: Andamento delle metriche di retrieval per differenti valori di K.

Correttezza delle chiamate ai tool La sezione di verifica delle chiamate ai tool mostra una corretta orchestrazione nel 90% dei casi (45 su 50 query), con solo 5 casi di chiamate parziali e nessun fallimento completo. Le analisi di debug confermano che i casi parziali riguardano l'omissione del tool *commit_code*, senza impatti rilevanti sulla correttezza complessiva del risultato. Un modello più potente, unito a tecniche più avanzate, avrebbe sicuramente aiutato nel migliorare ulteriormente i risultati.

Stato	Conteggio
OK	45
PARTIAL	5
KO	0

Tabella 4.15: Correttezza delle chiamate ai tool.

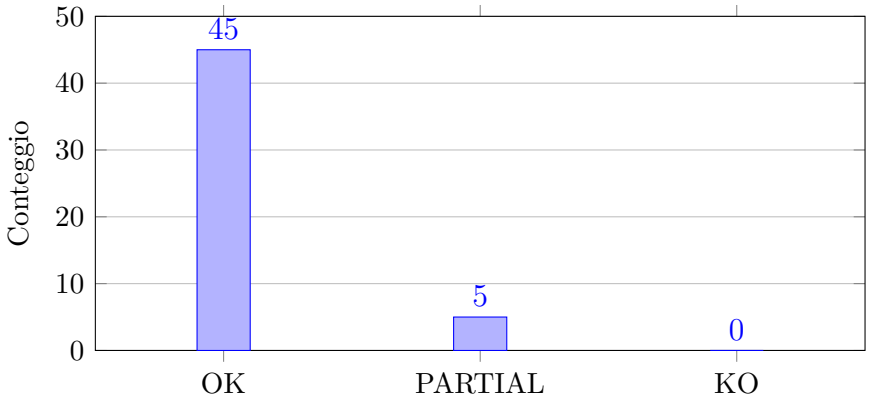

Figura 4.16: Distribuzione degli stati di esecuzione dei tool.

Tasso di allucinazione Il tasso di allucinazione è contenuto: nelle valutazioni umane solo il 10% delle risposte mostra elementi parzialmente inventati, mentre per G-Eval il valore è leggermente più alto (12%). Entrambi i risultati evidenziano una buona aderenza semantica delle risposte ai contenuti effettivamente recuperati. Solo in un caso di validazione umana si è riscontrata una risposta totalmente allucinata. Nel complesso, i valori raggiunti confermano la solidità del processo di retrieval nella pipeline online, che mantiene elevata precisione, consistenza e affidabilità anche in contesti interattivi.

Categoria	Valutazione	
	Umano (%)	G-Eval (%)
YES	2.00	0.00
PARTIALLY	8.00	12.00
NO	90.00	88.00
Totale allucinato (YES+PARTIALLY)	10.00	12.00

Tabella 4.16: Tasso di allucinazione (Human vs G-Eval).

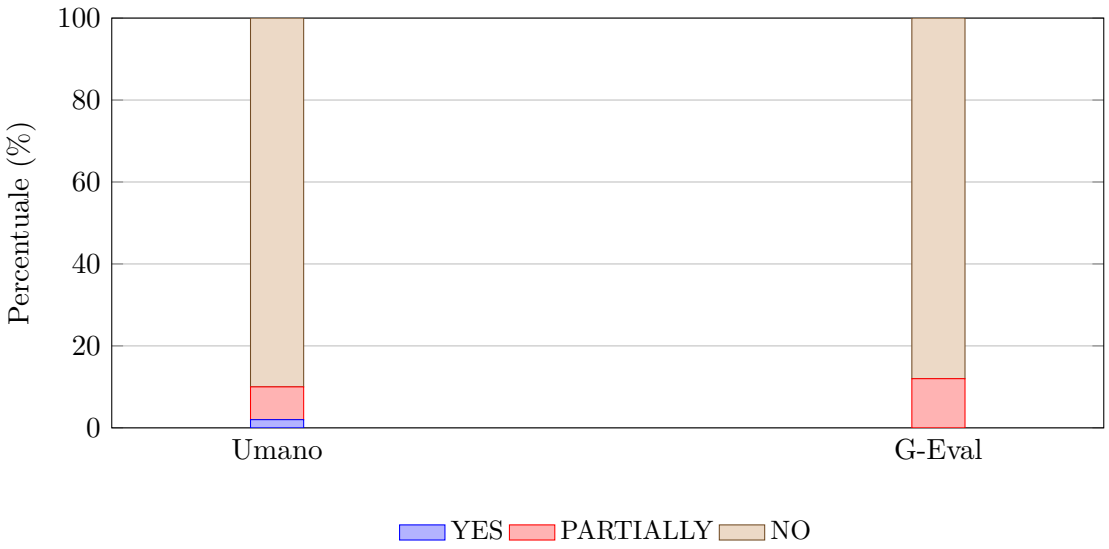

Figura 4.17: Distribuzione delle valutazioni di allucinazione.

Capitolo 5

Conclusioni

Questo capitolo raccoglie le riflessioni conclusive sul lavoro svolto, sintetizzando i risultati ottenuti, le implicazioni emerse e le prospettive future del progetto. L'obiettivo è discutere in che misura le soluzioni sviluppate siano riuscite a rispondere alle domande di ricerca iniziali, valutando al tempo stesso il contributo scientifico e applicativo della proposta. Dopo una prima sezione dedicata alla discussione dei risultati rispetto alle due domande di ricerca, il capitolo analizza il comportamento complessivo del sistema, le sue potenzialità di integrazione nel ciclo di vita del software. Infine, vengono tratte alcune considerazioni generali sulle prospettive evolutive legate alla diffusione della GenAI nel SDLC.

5.1 Discussione rispetto le due domande di ricerca

In questa sezione si riflette sull'efficacia complessiva del sistema nel rispondere alle due domande di ricerca che hanno guidato lo sviluppo del progetto. Le analisi condotte sulle due pipeline — offline e online — mostrano un quadro coerente e nel complesso positivo, sia dal punto di vista delle prestazioni quantitative, sia per la qualità percepita e la rilevanza dei risultati ottenuti.

La prima domanda di ricerca riguardava la capacità del sistema di analizzare e riassumere in modo accurato i messaggi di commit, mantenendo la coerenza con l'intento originale delle modifiche.

Nella pipeline offline, i riassunti generati offrono una buona base narrativa per descrivere i commit, come confermato anche dagli alti valori di BERTScore. I generali risultano leggibili, coerenti e mediamente informativi, mentre quelli tecnici presentano una maggiore fragilità, con valutazioni che scendono fino a 2,3-2,8/5 per la profondità tecnica. Questo suggerisce che il sistema riesce a cogliere il senso delle modifiche, ma incontra maggiori difficoltà nel rappresentarne i dettagli di implementazione, probabilmente a causa della complessità dei diff e della dimensione molto ridotta del modello adottato.

Anche l'analisi della pipeline online fornisce ulteriori elementi positivi in risposta a RQ1. Le risposte generate alle domande degli utenti sono convincenti e riescono a sintetizzare in modo preciso gli intenti alla base delle modifiche del codice. I punteggi assegnati risultano mediamente più alti, con evidenze di maggiore accuratezza e completezza. Questo suggerisce che, quando la generazione è guidata da un contesto informativo più ricco e mirato, il sistema riesce a produrre spiegazioni più dettagliate e pertinenti. In tale prospettiva, l'interazione online, quindi, compensa parzialmente le limitazioni della sintesi automatica offline, grazie alla possibilità di adattare la risposta alle esigenze e al livello di dettaglio richiesto dall'utente, valorizzando così meglio il contenuto del commit e del codice sottostante.

La seconda domanda, invece, riguarda come gli LLM siano in grado di migliorare la comunicazione tra i membri del team, generando report contestualizzati e favorendo una comprensione condivisa dell'evoluzione del progetto. Questa è stata interpretatata attraverso un chatbot agentico che sia in grado, tramite interazioni in linguaggio naturale, di rispondere a domande sul progetto e sulla sua evoluzione.

In questo ambito, la pipeline online ha mostrato prestazioni complessivamente molto solide, sia sotto il profilo tecnico, sia da quello dell'esperienza d'uso. Il chatbot, basato su un'architettura agentica ReAct e potenziato da un modulo di retrieval semantico, si è dimostrato capace di comprendere domande complesse,

scomporle in sotto-task e coordinare autonomamente i tool disponibili per recuperare le informazioni necessarie. Durante i test, il sistema ha mostrato una notevole capacità nel contestualizzare le richieste dell'utente: ad esempio, nelle domande relative all'evoluzione di un determinato file o alla natura di un cambiamento, l'agente è stato in grado di combinare più fonti (riassunti di commit, diff, metadati) e produrre risposte articolate ma sintetiche, in cui la componente descrittiva è accompagnata da una chiara interpretazione del processo di sviluppo. Questo comportamento emerge anche nella qualità linguistica delle risposte, giudicate nel complesso scorrevoli, precise e coerenti con la domanda di partenza.
Dal punto di vista del recupero delle informazioni, le metriche di retrieval confermano una buona capacità nell' individuare i documenti rilevanti. Il tasso di allucinazione basso dimostra la solidità dell'approccio RAG nel limitare errori semantici o contenuti non fondati. Inoltre, l'agente ha mostrato un uso corretto e coerente dei tool interni, con circa il 90% delle esecuzioni completate senza errori, a indicare una buona orchestrazione tra ragionamento, ricerca e generazione.

Un aspetto critico emerso riguarda tuttavia l'effettiva utilità dei riassunti generati dalla pipeline offline all'interno del processo di retrieval online. In diversi casi, infatti, le risposte basate direttamente su commit e porzioni di codice hanno mostrato una maggiore precisione e rilevanza rispetto a quelle che si appoggiavano ai riassunti sintetici. Questo evidenzia come, nella configurazione attuale, l'informazione derivata dai riassunti non apporti ancora un vantaggio significativo in fase di interrogazione, e che l'accesso diretto ai dati originali resti spesso più efficace per domande di natura tecnica o specifica.

Nel complesso, i risultati ottenuti permettono di affermare che entrambe le domande di ricerca hanno trovato una risposta positiva. La pipeline offline ha dimostrato la capacità di costruire in modo affidabile una base informativa sintetica e coerente, mentre la pipeline online ha mostrato come tale conoscenza possa essere resa resa fruibile e adattabile al contesto dell'interazione, consentendo all'utente di esplorare l'evoluzione del progetto in maniera naturale e dinamica. L'integrazione tra le due componenti costituisce il punto di forza principale del sistema: la prima struttura e organizza l'informazione, la seconda la trasforma in uno strumento interattivo di comprensione e analisi. Nel suo insieme, il sistema proposto non si limita ad automatizzare attività di sintesi o ricerca, ma si configura come un vero e proprio supporto cognitivo per sviluppatori e revisori, migliorando la tracciabilità delle modifiche, la lettura del codice e la comunicazione all'interno dei team di sviluppo.

5.2 Discussione complessiva del progetto

In questa sezione si discutono in modo più ampio i risultati del progetto, evidenziandone i punti di forza, le criticità e le possibili direzioni di sviluppo. Dopo aver analizzato separatamente le due pipeline, è possibile riflettere sul comportamento complessivo del sistema, sulla coerenza dell'architettura e sulle prospettive che essa apre per l'integrazione della GenAI nel ciclo di vita del software.

Il progetto ha raggiunto risultati significativi, dimostrando che l'integrazione dei LLM nel SDLC può generare valore reale e misurabile. Le due pipeline si completano a vicenda: la componente offline estrae e struttura la conoscenza dai dati Git, mentre la componente online ne consente un'esplorazione interattiva e personalizzata. Questo approccio ibrido di analisi, sintesi e interazione agentica ha effettivamente migliorato la comprensione dei processi di sviluppo, rendendo più accessibile la conoscenza nascosta nei repository e favorendo una forma di documentazione dinamica.

Dal punto di vista tecnico, il sistema ha dimostrato una stabilità complessiva e un grado di maturità adeguato agli obiettivi di ricerca. L'adozione di un'architettura ReAct integrata tramite LangGraph ha garantito un controllo dei flussi operativi e un'elevata trasparenza nelle interazioni tra i moduli. L'utilizzo del RAG ha ridotto le allucinazioni e migliorato la coerenza semantica, mentre la modularità del design ha permesso una sperimentazione agile e iterativa. L'esperienza d'uso del chatbot è risultata fluida e naturale, con risposte coerenti e linguisticamente solide, confermando la validità dell'impostazione architetturale adottata. Pur essendo concepito come prototipo di ricerca, il sistema mostra già caratteristiche che ne rendono plausibile un'applicazione in contesti reali.

Accanto ai risultati positivi, l'analisi ha evidenziato anche margini di miglioramento. In prima battuta, è emersa — come già osservato in precedenza — una criticità legata all'effettiva utilità dei riassunti generati dalla pipeline offline all'interno del processo di retrieval online. Nella maggior parte dei casi, infatti, le risposte basate direttamente su commit e codice si sono rivelate più efficaci e pertinenti rispetto a quelle che facevano uso dei riassunti sintetici. Ciò suggerisce che una diversa gestione o integrazione dell'informazione potrebbe migliorare ulteriormente l'efficacia complessiva del sistema. Sul piano modellistico, il sistema potrebbe beneficiare dell'impiego di tecniche avanzate di ottimizzazione, come il Fine-Tuning [19] o strategie di Reinforcement Learning from Human Feedback (RLHF) [87][88], volte a raffinare la precisione e l'aderenza semantica delle risposte. Un ulteriore passo evolutivo potrebbe consistere nella scelta di LLM più potenti, in grado di gestire con maggiore precisione contesti più complessi. Dal punto di vista architetturale, l'introduzione di un approccio multi-agente rappresenterebbe un'evoluzione naturale

del sistema, permettendo la cooperazione tra agenti specializzati (analisi, retrieval, validazione, generazione linguistica) e migliorando scalabilità e precisione. In parallelo, una versione personalizzata dell'agente LangGraph, progettata per gestire in modo più dinamico i flussi tra i tool e la memoria, potrebbe rendere il sistema più adattivo e controllabile. Anche l'integrazione con protocolli come il Model Context Protocol (MCP) [89] aprirebbe la possibilità di una comunicazione più strutturata e standardizzata tra il sistema e altri strumenti software, favorendo un'integrazione più flessibile e robusta in ambienti di sviluppo reali.

Nel complesso, il lavoro può considerarsi soddisfacente già nella sua forma attuale: ha raggiunto gli obiettivi scientifici e tecnici stabiliti, validando la fattibilità di un sistema in grado di analizzare, sintetizzare e comunicare informazioni sul codice in modo coerente e adattivo. Al di là dei risultati sperimentali, il progetto offre anche una prospettiva applicativa concreta: mostra come i LLM possano diventare strumenti di supporto intelligenti e integrabili nel ciclo di vita del software, contribuendo a migliorare la tracciabilità, la collaborazione e la consapevolezza collettiva all'interno dei team di sviluppo.

5.3 Discussione generale della GenAI nel SDLC

La progressiva integrazione della GenAI nel ciclo di vita del software (SDLC) sta modificando in modo profondo il modo in cui i team sviluppano, comprendono e mantengono il codice. Gli strumenti basati su LLM stanno evolvendo da semplici assistenti testuali a vere e proprie piattaforme di supporto cognitivo, in grado di analizzare repository, proporre modifiche e contestualizzare decisioni progettuali. In questo scenario, i risultati di questo lavoro confermano il potenziale della GenAI come strumento in grado di potenziare lo sviluppo software.

Al tempo stesso, l'esperienza maturata mostra come tali sistemi debbano essere impiegati con una chiara consapevolezza dei loro limiti. L'efficacia della generazione automatica di contenuti tecnici dipende fortemente dalla qualità dei dati di input, dalla trasparenza del ragionamento del modello e dalla sua capacità di mantenere coerenza semantica nel tempo. La GenAI non sostituisce le competenze umane nello sviluppo, ma può integrarle, automatizzando compiti ripetitivi e rendendo più accessibile la conoscenza distribuita all'interno dei team. In questo senso, il valore più rilevante non risiede nella sostituzione, ma nella collaborazione uomo-macchina, dove il modello diventa un amplificatore della comprensione collettiva.

Guardando alle prospettive attuali e future, la direzione della ricerca e dell'industria si sta orientando verso architetture più modulari, efficienti e flessibili. L'emergere del MCP [89] sta favorendo la creazione di ecosistemi interoperabili, nei quali i modelli possono comunicare con strumenti esterni — ambienti di sviluppo, IDE, database o altri agenti — attraverso canali standardizzati. Parallelamente, gli approcci multi-agente stanno guadagnando terreno come paradigma per orchestrare competenze specializzate: diversi agenti possono collaborare per analizzare, generare, validare e spiegare, rendendo i sistemi più adattivi e affidabili. Sul fronte modellistico, si osserva una duplice tendenza: da un lato lo sviluppo di LLM sempre più grandi e generalisti, capaci di un ragionamento profondo e multidominio; dall'altro la nascita di Small Language Models (SLM) [90] e modelli verticalizzati, più efficienti, interpretabili e adatti a scenari aziendali ristretti o a dispositivi edge. Il futuro prossimo sembra dunque orientato verso sistemi ibridi, nei quali modelli di dimensioni differenti cooperano secondo un principio di specializzazione funzionale, bilanciando capacità di ragionamento e sostenibilità computazionale.

In questa prospettiva, la GenAI nel SDLC potrà assumere un ruolo sempre più integrato: non solo come assistente, ma come componente attiva nel monitoraggio della qualità, nella pianificazione evolutiva e nella gestione della conoscenza tecnica. Il passo successivo sarà l'emergere di ecosistemi intelligenti in cui agenti, strumenti e persone coesistono in modo sinergico, condividendo contesto e obiettivi.

5.4 Conclusioni

Il lavoro svolto ha mostrato come la combinazione di analisi automatica, sintesi informativa e interazione agentica possa costituire una base solida per nuove forme di comprensione e collaborazione nello sviluppo del software. La ricerca condotta non si limita a dimostrare la fattibilità tecnica di un sistema basato su LLM, ma propone un modello di interazione che riflette un cambiamento più ampio nel modo di concepire la produttività e la conoscenza nei processi di ingegneria del software.

La sfida principale non sarà soltanto rendere i modelli più potenti, ma integrarli in modo sicuro, trasparente e realmente utile per le persone che li utilizzano. Se il codice rappresenta la logica delle macchine, la GenAI ne diventa oggi il linguaggio di mediazione, capace di tradurre l'intento umano in conoscenza operativa. In questo senso, l'obiettivo finale non è costruire sistemi che pensino al posto nostro, ma strumenti che ci aiutino a pensare meglio insieme alle macchine.

Bibliografia

[1] Gartner. *What's Driving the Hype Cycle for Generative AI, 2024*. Accesso: Gennaio 2025. 2024 (cit. a p. 1).

[2] Roger S. Pressman. *Software Engineering: A Practitioner's Approach*. 9th. New York: McGraw-Hill, 2020 (cit. a p. 6).

[3] Wikipedia contributors. *Generative artificial intelligence*. Accesso: Gennaio 2025. 2025 (cit. a p. 9).

[4] Yihan Cao, Siyu Li, Yixin Liu, Zhiling Yan, Yutong Dai, Philip S. Yu e Lichao Sun. *A Comprehensive Survey of AI-Generated Content (AIGC): A History of Generative AI from GAN to ChatGPT*. 2023. DOI: 10.48550/arXiv.2303.04226. URL: https://arxiv.org/abs/2303.04226 (cit. alle pp. 10, 17).

[5] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser e Illia Polosukhin. «Attention Is All You Need». In: *Advances in Neural Information Processing Systems* (2017). URL: https://arxiv.org/abs/1706.03762 (cit. alle pp. 10, 11, 13–15).

[6] Alec Radford, Karthik Narasimhan, Tim Salimans e Ilya Sutskever. *Improving Language Understanding by Generative Pre-Training*. Technical Report. OpenAI, 2018. URL: https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf (cit. alle pp. 10, 11, 13, 16).

[7] OpenAI. *OpenAI*. Accesso: Maggio 2025. 2025 (cit. alle pp. 10, 12, 16).

[8] Sepp Hochreiter e Jürgen Schmidhuber. «Long Short-Term Memory». In: *Neural Computation* 9 (nov. 1997), pp. 1735–1780. DOI: 10.1162/neco.1997.9.8.1735 (cit. a p. 10).

[9] Diederik Kingma e Max Welling. «Auto-Encoding Variational Bayes». In: *ICLR* (dic. 2013) (cit. a p. 10).

[10] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville e Y. Bengio. «Generative Adversarial Networks». In: *Advances in Neural Information Processing Systems* 3 (giu. 2014). DOI: 10.1145/3422622 (cit. a p. 10).

[11] Midjourney. *Midjourney*. Accesso: Maggio 2025. 2025 (cit. a p. 10).

[12] Stability AI. *Stable Diffusion*. Accesso: Maggio 2025. 2025 (cit. a p. 10).
[13] Cole Stryker. *Che cos'è l'AI multimodale?* Accesso: Maggio 2025. 2024 (cit. a p. 10).
[14] Wikipedia contributors. *Large language model*. Accesso: Maggio 2025. 2025 (cit. a p. 11).
[15] Dave Bergmann. *What is a context window?* Accesso: Maggio 2025. 2024 (cit. a p. 11).
[16] OpenAI. *GPT-4.1*. Accesso: Maggio 2025. 2025 (cit. alle pp. 11, 57).
[17] Prompt Engineering Guide. *Prompt Engineering Guide*. Accesso: Maggio 2025. 2025 (cit. alle pp. 11, 28).
[18] Wikipedia contributors. *Prompt engineering*. Accesso: Maggio 2025. 2025 (cit. alle pp. 11, 26).
[19] Wikipedia contributors. *Fine-tuning (deep learning)*. Accesso: Maggio 2025. 2025 (cit. alle pp. 12, 106).
[20] Jacob Ph.D. Murel e Joshua Noble. *What is LLM Temperature?* Accesso: Maggio 2025. 2024 (cit. a p. 12).
[21] Luis Lastras. *What is retrieval-augmented generation (RAG)?* Accesso: Maggio 2025. 2023 (cit. a p. 12).
[22] Wikipedia contributors. *Retrieval-augmented generation*. Accesso: Maggio 2025. 2025 (cit. a p. 12).
[23] Prompt Engineering Guide. *Retrieval Augmented Generation (RAG)*. Accesso: Maggio 2025. 2025 (cit. a p. 12).
[24] Anthropic. *Claude*. Accesso: Maggio 2025. 2025 (cit. a p. 12).
[25] Google. *Gemini*. Accesso: Maggio 2025. 2025 (cit. a p. 12).
[26] Meta. *LLaMA*. Accesso: Maggio 2025. 2025 (cit. alle pp. 12, 63).
[27] Tom B. Brown et al. *Language Models are Few-Shot Learners*. 2020. arXiv: 2005.14165 [cs.CL]. URL: https://arxiv.org/abs/2005.14165 (cit. alle pp. 17, 28).
[28] Dorothea S. Adamantiadou e Loukas Tsironis. «Leveraging Artificial Intelligence in Project Management: A Systematic Review of Applications, Challenges, and Future Directions». In: *Computers* 14.2 (2025). ISSN: 2073-431X. DOI: 10.3390/computers14020066. URL: https://www.mdpi.com/2073-431X/14/2/66 (cit. a p. 20).
[29] Haowei Cheng, Jati H. Husen, Yijun Lu, Teeradaj Racharak, Nobukazu Yoshioka, Naoyasu Ubayashi e Hironori Washizaki. *Generative AI for Requirements Engineering: A Systematic Literature Review*. 2025. arXiv: 2409.06741 [cs.SE]. URL: https://arxiv.org/abs/2409.06741 (cit. a p. 20).
[30] Nuno Marques, Rodrigo Rocha Silva e Jorge Bernardino. «Using ChatGPT in Software Requirements Engineering: A Comprehensive Review». In: *Future Internet* 16.6 (2024). ISSN: 1999-5903. DOI: 10.3390/fi16060180. URL: https://www.mdpi.com/1999-5903/16/6/180 (cit. a p. 21).

[31] Aakash Ahmad, Muhammad Waseem, Peng Liang, Mahdi Fehmideh, Mst Shamima Aktar e Tommi Mikkonen. *Towards Human-Bot Collaborative Software Architecting with ChatGPT*. 2023. arXiv: 2302.14600 [cs.SE]. URL: https://arxiv.org/abs/2302.14600 (cit. a p. 21).

[32] Beian Wang, Chong Wang, Peng Liang, Bing Li e Cheng Zeng. *How LLMs Aid in UML Modeling: An Exploratory Study with Novice Analysts*. 2024. arXiv: 2404.17739 [cs.SE]. URL: https://arxiv.org/abs/2404.17739 (cit. a p. 21).

[33] Alessio Ferrari, Sallam Abualhaija e Chetan Arora. *Model Generation with LLMs: From Requirements to UML Sequence Diagrams*. 2024. arXiv: 2404.06371 [cs.SE]. URL: https://arxiv.org/abs/2404.06371 (cit. a p. 21).

[34] GitHub and OpenAI. *GitHub Copilot*. https://github.com/features/copilot/. 2025 (cit. a p. 22).

[35] Machinet. *Machinet AI*. https://www.machinet.net/. Plugin JetBrains per coding contestuale e generazione di unit test. 2024 (cit. a p. 22).

[36] Zheyuan Cui, Mert Demirer, Sonia Jaffe, Leon Musolff, Sida Peng e Tobias Salz. *The Effects of Generative AI on High-Skilled Work: Evidence from Three Field Experiments with Software Developers*. SSRN eLibrary. Available at https://ssrn.com/abstract=4945566 or http://dx.doi.org/10.2139/ssrn.4945566. Ago. 2025 (cit. a p. 22).

[37] Yujia Fu, Peng Liang, Amjed Tahir, Zengyang Li, Mojtaba Shahin, Jiaxin Yu e Jinfu Chen. *Security Weaknesses of Copilot-Generated Code in GitHub Projects: An Empirical Study*. 2025. arXiv: 2310.02059 [cs.SE]. URL: https://arxiv.org/abs/2310.02059 (cit. a p. 22).

[38] Kefan Li e Yuan Yuan. *Large Language Models as Test Case Generators: Performance Evaluation and Enhancement*. 2024. arXiv: 2404.13340 [cs.SE]. URL: https://arxiv.org/abs/2404.13340 (cit. a p. 23).

[39] Suprit Pattanayak, Pranav Murthy e Aditya Mehra. «Integrating AI into DevOps pipelines: Continuous integration, continuous delivery, and automation in infrastructural management: Projections for future». In: *International Journal of Science and Research Archive (IJSRA)* 13.1 (2024) (cit. a p. 24).

[40] Marco Biscardi. *DevGenOps — Unlocking the Power of Generative AI for DevOps*. Set. 2023. URL: https://medium.com/go-reply-tech/devgenops-57f8e4b7ff22 (cit. a p. 24).

[41] Shuzheng Gao, Cuiyun Gao, Wenchao Gu e Michael Lyu. *Search-Based LLMs for Code Optimization*. 2024. arXiv: 2408.12159 [cs.SE]. URL: https://arxiv.org/abs/2408.12159 (cit. a p. 25).

[42] Pranab Sahoo, Ayush Kumar Singh, Sriparna Saha, Vinija Jain, Samrat Mondal e Aman Chadha. *A Systematic Survey of Prompt Engineering in Large Language Models: Techniques and Applications*. 2025. arXiv: 2402.07927 [cs.AI]. URL: https://arxiv.org/abs/2402.07927 (cit. a p. 28).

[43] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le e Denny Zhou. *Chain-of-Thought Prompting Elicits Reasoning in Large Language Models*. 2023. arXiv: 2201.11903 [cs.CL]. URL: https://arxiv.org/abs/2201.11903 (cit. a p. 29).

[44] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery e Denny Zhou. *Self-Consistency Improves Chain of Thought Reasoning in Language Models*. 2023. arXiv: 2203.11171 [cs.CL]. URL: https://arxiv.org/abs/2203.11171 (cit. a p. 30).

[45] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao e Karthik Narasimhan. *Tree of Thoughts: Deliberate Problem Solving with Large Language Models*. 2023. arXiv: 2305.10601 [cs.CL]. URL: https://arxiv.org/abs/2305.10601 (cit. alle pp. 30, 31).

[46] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan e Yuan Cao. *ReAct: Synergizing Reasoning and Acting in Language Models*. 2023. arXiv: 2210.03629 [cs.CL]. URL: https://arxiv.org/abs/2210.03629 (cit. alle pp. 31, 32).

[47] Patrick Lewis et al. *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks*. 2021. arXiv: 2005.11401 [cs.CL]. URL: https://arxiv.org/abs/2005.11401 (cit. alle pp. 33, 34).

[48] Meta AI. *Retrieval Augmented Generation: Streamlining the creation of intelligent natural language processing models*. https://ai.meta.com/blog/retrieval-augmented-generation-streamlining-the-creation-of-intelligent-natural-language-processing-models/. Blog post; accessed 2025-08-31. Set. 2020 (cit. a p. 33).

[49] Yunfan Gao et al. *Retrieval-Augmented Generation for Large Language Models: A Survey*. 2024. arXiv: 2312.10997 [cs.CL]. URL: https://arxiv.org/abs/2312.10997 (cit. alle pp. 34, 35).

[50] Hammond Pearce, Baleegh Ahmad, Benjamin Tan, Brendan Dolan-Gavitt e Ramesh Karri. *Asleep at the Keyboard? Assessing the Security of GitHub Copilot's Code Contributions*. 2021. arXiv: 2108.09293 [cs.CR]. URL: https://arxiv.org/abs/2108.09293 (cit. a p. 39).

[51] Harness, Inc. *The State of Software Delivery Report 2025: Beyond CodeGen: The Role of AI in the SDLC*. https://www.harness.io/state-of-software-delivery. Accessed 2025-08-31. 2025 (cit. a p. 41).

[52] Rafael Quintanilha. *Are AI Assistants Making Us Worse Programmers?* https://rafaelquintanilha.com/are-ai-assistants-making-us-worse-programmers/. Blog post; accessed 2025-08-31. Nov. 2024 (cit. a p. 43).

[53] Michael Chui, Lareina Yee, Bryce Hall, Alex Singla e Alexander Sukharevsky. *The State of AI in 2023: Generative AI's Breakout Year*. https://www.mckinsey.com/capabilities/quantumblack/our-insights/the-state-

of-ai-in-2023-generative-ais-breakout-year. Survey; accessed 2025-08-31. Ago. 2023 (cit. a p. 44).

[54] Fangchen Song, Ashish Agarwal e Wen Wen. *The Impact of Generative AI on Collaborative Open-Source Software Development: Evidence from GitHub Copilot*. 2025. arXiv: 2410.02091 [cs.SE]. URL: https://arxiv.org/abs/2410.02091 (cit. a p. 46).

[55] Ellen Glover. *AI-Generated Content and Copyright Law: What We Know*. https://builtin.com/artificial-intelligence/ai-copyright. Built In; accessed 2025-08-31. Ago. 2025 (cit. a p. 48).

[56] GitHub, Inc. *GitHub Copilot Product Specific Terms*. https://assets.ctfassets.net/8aevphvgewt8/1Y0gmEkMnAs8W6N4ai2R1g/89d4b89b3c016eb2b9925785c2aa113e/GitHub_Copilot_Product_Specific_Terms_-_2024_10_24_-_FINAL.pdf. PDF; Version: October 2024; accessed 2025-08-31. Ott. 2024 (cit. a p. 48).

[57] European Commission. *Data protection under GDPR*. https://europa.eu/youreurope/business/dealing-with-customers/data-protection/data-protection-gdpr/index_en.htm. Your Europe (EU); last checked 03 Mar 2025; accessed 2025-08-31. Mar. 2025 (cit. a p. 49).

[58] David Licursi. *AI: L'impatto sul lavoro e sull'occupazione*. https://www.digital4pro.com/2025/02/04/ai-limpatto-sul-lavoro-e-sulloccupazione/. Digital4Pro; accessed 2025-08-31. Feb. 2025 (cit. a p. 49).

[59] The Wall Street Journal. *IT Unemployment Rises to 5.7 as AI Hits Tech Jobs*. https://www.wsj.com/articles/it-unemployment-rises-to-5-7-as-ai-hits-tech-jobs-7726bb1b. Accessed 2025-08-31. Feb. 2025 (cit. a p. 50).

[60] Sky TG24. *Klarna vuole sostituire metà dei dipendenti con l'intelligenza artificiale*. https://tg24.sky.it/economia/2024/08/29/klarna-dipendenti-intelligenza-artificiale. Accessed 2025-08-31. Ago. 2024 (cit. a p. 51).

[61] Tina Vartziotis, Ippolyti Dellatolas, George Dasoulas, Maximilian Schmidt, Florian Schneider, Tim Hoffmann, Sotirios Kotsopoulos e Michael Keckeisen. *Learn to Code Sustainably: An Empirical Study on LLM-based Green Code Generation*. 2024. arXiv: 2403.03344 [cs.SE]. URL: https://arxiv.org/abs/2403.03344 (cit. a p. 51).

[62] Smartly.ai. *The Carbon Footprint of ChatGPT: How Much CO_2 Does a Query Generate*. https://smartly.ai/blog/the-carbon-footprint-of-chatgpt-how-much-co2-does-a-query-generate. Accessed 2025-08-31. Giu. 2024 (cit. a p. 51).

[63] Green Building Advisor. *As Use of A.I. Soars, So Does the Energy and Water It Requires*. https://www.greenbuildingadvisor.com/article/as-use-

of-a-i-soars-so-does-the-energy-and-water-it-requires. Accessed 2025-08-31. Feb. 2024 (cit. a p. 52).

[64] F. Alzate, D. Monaco e M. Francios. *Code Review and Project Workflow Analysis for Git Data*. https://github.com/maxfra01/code-review-and-project-workflow-analysis-for-git-data. [Online]. 2025 (cit. alle pp. 55, 62).

[65] Wei Tao, Yucheng Zhou, Yanlin Wang, Wenqiang Zhang, Hongyu Zhang e Yu Cheng. *MAGIS: LLM-Based Multi-Agent Framework for GitHub Issue Resolution*. 2024. arXiv: 2403.17927 [cs.SE]. URL: https://arxiv.org/abs/2403.17927 (cit. alle pp. 55, 56).

[66] Jing Liu, Seongmin Lee, Eleonora Losiouk e Marcel Böhme. *Can LLM Generate Regression Tests for Software Commits?* 2025. arXiv: 2501.11086 [cs.SE]. URL: https://arxiv.org/abs/2501.11086 (cit. a p. 55).

[67] Yusheng Zheng, Yiwei Yang, Haoqin Tu e Yuxi Huang. *Code-Survey: An LLM-Driven Methodology for Analyzing Large-Scale Codebases*. 2024. arXiv: 2410.01837 [cs.SE]. URL: https://arxiv.org/abs/2410.01837 (cit. a p. 55).

[68] Meta AI. *Llama 3.2 3B Instruct*. https://huggingface.co/meta-llama/Llama-3.2-3B-Instruct. [Online]. 2025 (cit. a p. 57).

[69] Hugging Face. *Hugging Face*. https://huggingface.co/. [Online] (cit. a p. 57).

[70] T. Andersson e contributors. *MuJS: An Embeddable JavaScript Interpreter in C*. https://github.com/ccxvii/mujs. [Online]. 2025 (cit. a p. 58).

[71] T. Andersson e contributors. *MuJS: An Embeddable JavaScript Interpreter in C*. https://codeberg.org/ccxvii/mujs. [Online], migrated from GitHub. 2025 (cit. a p. 58).

[72] Meta AI. *Llama 3.1 8B Instruct*. https://huggingface.co/meta-llama/Llama-3.1-8B-Instruct. [Online]. 2024 (cit. alle pp. 62, 63).

[73] Ollama. *qwen3:8b*. Ollama model library. Updated 3 months ago (da documento consultato il 10 settembre 2025). 2025. URL: https://ollama.com/library/qwen3%3A8b (cit. alle pp. 62, 63).

[74] Pane Raffaele. *Repository del progetto della tesi*. https://github.com/bred91/git-data-llm-workflow-code-review-analysis. 2025 (cit. alle pp. 63, 66).

[75] Qwen Team. *Qwen3: Think Deeper, Act Faster*. Accessed on 10 September 2025. 2025. URL: https://qwenlm.github.io/blog/qwen3/ (cit. a p. 63).

[76] An Yang et al. *Qwen3 Technical Report*. 2025. arXiv: 2505.09388 [cs.CL]. URL: https://arxiv.org/abs/2505.09388 (cit. a p. 63).

[77] Ollama Team. *Ollama*. GitHub repository. Accessed on 10 September 2025. 2025. URL: https://github.com/ollama/ollama (cit. a p. 64).

[78] LangChain. *LangGraph*. https://www.langchain.com/langgraph. 2025 (cit. a p. 64).

[79] LangChain. *create_react_agent*. https://langchain-ai.github.io/langgraph/reference/agents/#langgraph.prebuilt.chat_agent_executor.create_react_agent. Funzione che crea un agente ReAct nella libreria LangChain / LangGraph. 2025 (cit. a p. 64).

[80] Chroma. *ChromaDB: A fast, open-source vector database*. https://github.com/chroma-core/chroma. Open-source vector database for embeddings and similarity search, Apache 2.0 license. 2022 (cit. a p. 64).

[81] SQLite Consortium. *SQLite Documentation*. https://sqlite.org/docs.html. Documentazione ufficiale per il motore di database SQLite; visitata il 25 novembre 2025. 2025 (cit. a p. 65).

[82] Python Software Foundation. *What's New in Python 3.13*. https://docs.python.org/3.13/whatsnew/3.13.html. Documenta le principali novità di Python 3.13. Ott. 2024 (cit. a p. 65).

[83] Chin-Yew Lin. «ROUGE: A Package for Automatic Evaluation of Summaries». In: *Text Summarization Branches Out*. Barcelona, Spain: Association for Computational Linguistics, lug. 2004, pp. 74–81. URL: https://aclanthology.org/W04-1013/ (cit. a p. 72).

[84] Kishore Papineni, Salim Roukos, Todd Ward e Wei-Jing Zhu. «Bleu: a Method for Automatic Evaluation of Machine Translation». In: *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*. A cura di Pierre Isabelle, Eugene Charniak e Dekang Lin. Philadelphia, Pennsylvania, USA: Association for Computational Linguistics, lug. 2002, pp. 311–318. DOI: 10.3115/1073083.1073135. URL: https://aclanthology.org/P02-1040/ (cit. a p. 73).

[85] Satanjeev Banerjee e Alon Lavie. «METEOR: An Automatic Metric for MT Evaluation with Improved Correlation with Human Judgments». In: *Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*. A cura di Jade Goldstein, Alon Lavie, Chin-Yew Lin e Clare Voss. Ann Arbor, Michigan: Association for Computational Linguistics, giu. 2005, pp. 65–72. URL: https://aclanthology.org/W05-0909/ (cit. a p. 73).

[86] Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q. Weinberger e Yoav Artzi. *BERTScore: Evaluating Text Generation with BERT*. 2020. arXiv: 1904.09675 [cs.CL]. URL: https://arxiv.org/abs/1904.09675 (cit. a p. 74).

[87] Paul Christiano, Jan Leike, Tom B. Brown, Miljan Martic, Shane Legg e Dario Amodei. *Deep reinforcement learning from human preferences*. 2023. arXiv: 1706.03741 [stat.ML]. URL: https://arxiv.org/abs/1706.03741 (cit. a p. 106).

[88] Long Ouyang et al. *Training language models to follow instructions with human feedback*. 2022. arXiv: 2203.02155 [cs.CL]. URL: https://arxiv.org/abs/2203.02155 (cit. a p. 106).

[89] Anthropic. *Model Context Protocol (MCP)*. https://modelcontextprotocol.io/. Open-source standard for AI applications to connect with external data sources and tools. 2024 (cit. alle pp. 107, 108).

[90] Peter Belcak, Greg Heinrich, Shizhe Diao, Yonggan Fu, Xin Dong, Saurav Muralidharan, Yingyan Celine Lin e Pavlo Molchanov. *Small Language Models are the Future of Agentic AI*. 2025. arXiv: 2506.02153 [cs.AI]. URL: https://arxiv.org/abs/2506.02153 (cit. a p. 108).