POLITECNICO DI TORINO

SEDE DI TORINO

Corso di Laurea in Ingegneria Informatica

Tesi di Laurea

Automatic Documentation Drafting

Relatore
prof. Paolo Garza

Candidato
Andrea Pio Scaturro
matricola: 319480

Tutor Aziendale DATA Reply Srl
dott. Luca Montinari

Anno accademico 2024-2025

Alla mia famiglia

Sommario

Abstract

La scrittura e il mantenimento della documentazione rappresentano attività essenziali ma spesso trascurate nei progetti di sviluppo software, in particolare in ambito ETL, dove le pipeline evolvono rapidamente. La documentazione manuale risulta non solo time-consuming, ma anche soggetta a inconsistenze dovute alla molteplicità di template e pratiche adottate dai vari sviluppatori.

In questa tesi si propone un tool innovativo in grado di generare automaticamente la documentazione a partire dal codice sorgente di pipeline ETL realizzate con Apache Airflow e PySpark. Il sistema sfrutta modelli di Generative AI e tecniche di prompt engineering per estrarre in modo intelligente le informazioni rilevanti dal codice, organizzandole in una struttura standardizzata e pubblicandole su piattaforme collaborative come Confluence.

Il progetto, sviluppato in collaborazione con Data Reply, integra servizi cloud di AWS (in particolare SageMaker Studio) e si pone l'obiettivo di ridurre significativamente il carico di lavoro legato alla manutenzione della documentazione, migliorandone l'accuratezza e la coerenza.

I risultati mostrano come l'adozione di un approccio basato su GenAI possa rappresentare un valido supporto per la gestione della documentazione, garantendo aggiornamenti tempestivi e una standardizzazione che favorisce la collaborazione e la condivisione delle conoscenze all'interno dei team di sviluppo.

Ringraziamenti

Desidero dedicare questa tesi a tutte le persone che hanno reso possibile questo percorso, sostenendomi con affetto, pazienza e incoraggiamento lungo il cammino.

Un ringraziamento speciale va ai miei genitori, che mi hanno sempre sostenuto con amore incondizionato. Grazie per aver creduto in me anche nei momenti più difficili, per avermi spronato a dare il massimo e per aver condiviso con me gioie e sacrifici: senza di voi, questo traguardo non sarebbe stato possibile.

Un particolare ringraziamento va a mio fratello, la cui presenza ha reso questo viaggio meno faticoso e più ricco di momenti preziosi. Grazie per le parole di conforto e per essere sempre stato un punto di riferimento per me.

Ringrazio inoltre tutta la mia famiglia, i miei nonni per il conforto prima di ogni esame, i miei zii per i consigli e i miei cugini per il loro affetto e la vicinanza.

Un sentito ringraziamento va al mio relatore, per la disponibilità, la guida e i preziosi consigli ricevuti durante la stesura della tesi. Grazie per avermi seguito con attenzione e per avermi trasmesso conoscenze e spunti che hanno arricchito il mio percorso accademico.

Devo un grande grazie a Leonardo, che in questi anni è stato per me un vero mentore. Mi ha sempre indicato la strada giusta e mi ha aiutato in molte scelte accademiche, e di questo non posso che essergli profondamente grato.

Un ringraziamento speciale va a Giada, che mi ha supportato più di tutti in questo percorso: senza di lei, probabilmente non avrei portato a termine questa tesi, e non sarei quello che sono ora. So che non basta, ma grazie per tutto ciò che fai per me.

Un doveroso ringraziamento va ai miei più cari amici, Filippo e Alessandro, con i quali ho condiviso sia successi che difficoltà. Grazie per il vostro supporto incondizionato, per la vostra presenza e per aver reso questo percorso ancora più speciale.

Infine, un grazie speciale a quella che è diventata la mia nuova famiglia, i miei coinquilini. Grazie ad Alekos per la sua costante disponibilità e per la capacità di alleggerire ogni giornata, ad Andrea per la sua amicizia e i consigli preziosi, a Matteo per le risate condivise, a Gabriele per la sua energia contagiosa e il suo supporto in ogni situazione, e a Francesco per la sua compagnia e il suo sostegno. Con voi, ogni giorno è stato più bello.

Indice

Elenco delle tabelle 9

Elenco delle figure 10

I Introduzione **11**

1 Introduzione 13
1.1 Contesto e motivazione 13
1.2 Problematica della documentazione nei processi ETL 13
1.3 Obiettivi e contributi della tesi 14
1.4 Struttura della tesi 15

II Background **17**

2 Background e Stato dell'Arte 19
2.1 La documentazione nel ciclo di vita del software: sfide e best practice 19
2.2 Tecnologie per i processi ETL: Apache Airflow e PySpark . . . 22
2.3 Generative AI e Prompt Engineering: concetti e applicazioni . 35
2.4 Integrazione con strumenti collaborativi: Confluence e GitHub 46
2.5 Panoramica degli strumenti AWS: SageMaker Studio e altri servizi correlati 47

III Specifiche **49**

3 Specifiche 51
3.1 Definizione dei casi d'uso 51

3.2 Scelta delle tecnologie e motivazioni progettuali 54
3.3 Architettura del sistema e flusso di dati 57

IV Implementazione **63**

4 Metodologia e Implementazione 65
4.1 Approccio di prompt engineering: definizione e ottimizzazione dei prompt 65
4.2 Pipeline di estrazione delle informazioni dal codice 68
4.3 Generazione automatica della documentazione 71
4.4 Integrazione con AWS SageMaker Studio e Confluence 73
4.5 Descrizione CI/CD per l'implementazione 74

V Valutazione **77**

5 Casi di Studio e Valutazione 79
5.1 Descrizione dei casi d'uso implementati 79
5.2 Analisi qualitativa e quantitativa dei risultati 82
5.3 Benchmark 87

VI Conclusione **91**

6 Discussione, Conclusioni e Sviluppi Futuri 93
6.1 Sintesi dei risultati e contributi della tesi 93
6.2 Limiti e criticità riscontrate 95
6.3 Proposte di miglioramento e possibili sviluppi futuri 96
6.4 Conclusioni finali 97

Bibliografia 99

Elenco delle tabelle

4.1 Descrizione delle operazioni di join. 72
5.1 Risultati Operatori. 84
5.2 Risultati ETL: Input/Output Tables 85
5.3 Risultati ETL: Join. 85

Elenco delle figure

2.1 ETL Process [38] 22
2.2 Apache Airflow [39] 28
2.3 Apache Spark [40] 32
2.4 Prompt di esempio nella guida Antropic [33] 45
3.1 Architettura Automatic Documentation Drafting 58
4.1 Definizione della struttura dati 66
4.2 Istruzioni di un prompt per le descrizioni ETL 67
4.3 Risultati unparsed 70
4.4 Risultati parsed 70
4.5 Risultati descrizione ETL 73

Parte I

Introduzione

Capitolo 1

Introduzione

1.1 Contesto e motivazione

Nell'era attuale, caratterizzata da una rapida evoluzione tecnologica e dall'importanza crescente dei dati come asset strategico, le aziende si affidano sempre più a sistemi complessi per l'estrazione, la trasformazione e il caricamento dei dati (ETL). Le pipeline ETL rappresentano il cuore pulsante di molte infrastrutture di business intelligence e data warehousing, garantendo il flusso continuo di informazioni indispensabili per decisioni operative e strategiche. Tuttavia, nonostante il loro ruolo critico, la documentazione che ne descrive il funzionamento e le logiche implementate è spesso trascurata o realizzata in maniera frammentaria. In un contesto in cui la velocità di sviluppo e l'aggiornamento continuo del codice sono fondamentali, la produzione manuale di documentazione si rivela un'attività onerosa e soggetta a errori. Questo scenario ha spinto la comunità tecnologica a cercare soluzioni innovative che possano automatizzare e standardizzare il processo, riducendo il carico di lavoro dei team di sviluppo e migliorando al contempo la qualità e l'accessibilità delle informazioni.

1.2 Problematica della documentazione nei processi ETL

La difficoltà di mantenere una documentazione accurata e aggiornata è particolarmente evidente nei processi ETL. Le pipeline, spesso complesse e soggette a frequenti modifiche, richiedono un monitoraggio costante per evitare che la documentazione diventi rapidamente obsoleta. In molti progetti, la

responsabilità della documentazione ricade su sviluppatori già impegnati in attività critiche, portando a una mancanza di uniformità e a una dispersione delle informazioni. Inoltre, l'assenza di standard condivisi e l'utilizzo di template differenti per la documentazione complicano ulteriormente il compito, generando inconsistenze che possono tradursi in errori di interpretazione e difficoltà di manutenzione. Questo gap tra il codice in continua evoluzione e la documentazione statica rappresenta una problematica non solo tecnica, ma anche organizzativa, poiché incide direttamente sull'efficienza operativa e sulla capacità dei team di collaborare in maniera efficace.

1.3 Obiettivi e contributi della tesi

La presente tesi si propone di affrontare queste sfide attraverso lo sviluppo di un tool innovativo che automatizza la generazione della documentazione a partire dal codice delle pipeline ETL. Il sistema proposto sfrutta le potenzialità della Generative AI, integrando tecniche di prompt engineering per estrarre in maniera intelligente le informazioni rilevanti dal codice scritto, basato su Apache Airflow e PySpark.

Un elemento centrale di questa soluzione è l'utilizzo del modello *Claude Sonnet*, scelto per la sua capacità di comprendere e sintetizzare il linguaggio tecnico in una forma chiara e strutturata. Gli obiettivi principali del lavoro sono i seguenti:

- **Riduzione del carico manuale**: Automatizzare il processo di documentazione per permettere agli sviluppatori di concentrarsi su attività a maggior valore aggiunto.
- **Standardizzazione e coerenza**: Garantire una documentazione uniforme che segua template e standard condivisi, facilitando la collaborazione e la manutenzione.
- **Integrazione tecnologica**: Unire in un'unica soluzione avanzate tecnologie come *AWS SageMaker Studio* per il deployment del tool e *Confluence* per la gestione centralizzata della documentazione.

Il contributo della tesi risiede non solo nella proposta teorica, ma anche nella dimostrazione pratica del tool, realizzato in collaborazione con *Data Reply*, che integra diverse tecnologie all'avanguardia per risolvere una problematica quotidiana ma critica nel ciclo di vita dei sistemi ETL.

1.4 Struttura della tesi

La tesi è organizzata in modo da condurre il lettore attraverso un percorso che va dalla definizione del problema alle soluzioni tecnologiche adottate, fino alla validazione pratica del tool sviluppato:

- **Capitolo 1 – Introduzione**: Viene presentato il contesto, le motivazioni e gli obiettivi della ricerca, evidenziando la rilevanza del problema della documentazione automatica nei progetti ETL.
- **Capitolo 2 – Background**: Viene fornita una panoramica delle tecnologie e metodologie esistenti, analizzando le best practice nella documentazione tecnica, le peculiarità di Apache Airflow e PySpark, e le applicazioni della Generative AI e del prompt engineering.
- **Capitolo 3 – Specifiche**: Vengono descritti i requisiti funzionali e non funzionali del sistema, definendo i casi d'uso e illustrando l'architettura complessiva del tool.
- **Capitolo 4 – Metodologia e Implementazione**: Viene approfondito il processo di sviluppo, dalla definizione dei prompt alla pipeline di estrazione e generazione della documentazione, evidenziando il ruolo di *Claude Sonnet* e le integrazioni con *AWS SageMaker Studio* e *Confluence*.
- **Capitolo 5 – Casi di Studio e Valutazione**: Vengono presentati i risultati ottenuti attraverso l'applicazione pratica del tool, analizzando sia gli aspetti qualitativi sia quelli quantitativi, e confrontando la soluzione proposta con metodologie tradizionali.
- **Capitolo 6 – Discussione, Conclusioni e Sviluppi Futuri**: Vengono riassunti i contributi della ricerca, discusse le limitazioni emerse e proposte possibili evoluzioni del sistema per ulteriori miglioramenti.

Questa introduzione intende fornire una visione d'insieme del problema affrontato e della soluzione proposta, evidenziando come l'integrazione di tecnologie innovative come *Claude Sonnet* e le tecniche di prompt engineering possa rappresentare un significativo passo avanti nella gestione automatizzata della documentazione tecnica nei progetti ETL.

In sintesi, la tesi si pone l'obiettivo di dimostrare che la combinazione di strumenti avanzati di intelligenza artificiale e metodologie di automazione

può contribuire a colmare il gap esistente tra sviluppo e documentazione, garantendo così una maggiore efficienza e coerenza nei processi di manutenzione e aggiornamento del software.

Parte II

Background

Capitolo 2

Background e Stato dell'Arte

2.1 La documentazione nel ciclo di vita del software: sfide e best practice

Nel contesto dello sviluppo software, la documentazione rappresenta un asset strategico essenziale per garantire la qualità, la manutenibilità e l'evoluzione continua dei sistemi. Essa non è soltanto un mezzo per registrare le specifiche tecniche, ma svolge anche il ruolo di strumento di comunicazione tra i vari team – sviluppatori, tester, project manager e stakeholder non tecnici – e funge da guida per l'implementazione, il testing, la distribuzione e la manutenzione del software. In un ambiente in cui i requisiti e le tecnologie evolvono rapidamente, mantenere una documentazione aggiornata e coerente si configura come una sfida non indifferente, ma allo stesso tempo come un elemento chiave per il successo del ciclo di vita del software. A tal proposito, studi precedenti (Chen e Huang, 2009; Garousi et al., 2015) [1] [2] evidenziano come una buona documentazione migliori significativamente la comprensione del codice e riduca i costi di manutenzione, supportando l'importanza di sviluppare strumenti automatizzati che generino documentazioni esplicative dettagliate. Inoltre, Hu et al. (2022) [3] sottolineano l'esigenza crescente tra gli sviluppatori di strumenti capaci di produrre documentazione che non si limiti a spiegare cosa fa il codice, ma che chiarisca anche il "perché" e il "come", ossia il ragionamento dietro alle scelte implementative.

Sfide nella documentazione del software

Complessità e dinamiche evolutive
Uno dei principali ostacoli è rappresentato dalla natura dinamica dello sviluppo software. Durante l'intero ciclo di vita, dal concepimento fino alla fase di manutenzione, il codice subisce continui aggiornamenti e modifiche. Questo rende difficile mantenere una documentazione allineata in tempo reale. Spesso, la documentazione viene aggiornata solo in maniera sporadica, creando discrepanze tra il codice effettivo e la documentazione disponibile.

Mancanza di standardizzazione
Un'altra sfida significativa riguarda la mancanza di standard condivisi. Nei team in cui ogni sviluppatore o gruppo utilizza template, strumenti e convenzioni diverse, la documentazione risulta frammentata e poco uniforme. Ciò ostacola la comprensione comune del sistema e rende complicata la formazione di nuovi membri o il passaggio di consegne tra team.

Tempo e risorse dedicate
La scrittura e l'aggiornamento della documentazione richiedono tempo e risorse – risorse che spesso vengono orientate prioritariamente allo sviluppo di nuove funzionalità o alla risoluzione dei bug. Il risultato è un overhead significativo, che può portare a una documentazione superficiale o addirittura assente. Inoltre, la difficoltà di integrare la documentazione nei processi di sviluppo (ad esempio, in fase di CI/CD) può rendere più oneroso il compito di mantenerla sincronizzata con il prodotto.

Resistenza culturale
Infine, va considerata la resistenza culturale all'interno dei team di sviluppo. Spesso, la documentazione viene percepita come un compito secondario o addirittura un "male necessario" anziché come uno strumento fondamentale per la qualità del software. Questa percezione può generare un debito di conoscenza, in cui il know-how critico rimane in gran parte non documentato e concentrato nelle menti di pochi esperti.

Best practice per una documentazione efficace

Per superare le sfide sopra descritte, è possibile adottare diverse strategie e best practice che integrino la documentazione come parte integrante del ciclo di vita del software.

Integrazione nel processo di sviluppo
È fondamentale che la documentazione non sia un'attività "a parte", ma venga integrata nel workflow di sviluppo. L'adozione di metodologie DevOps e l'integrazione della documentazione nei processi CI/CD consentono di aggiornare automaticamente la documentazione in concomitanza con il rilascio di nuove modifiche. In questo modo, ogni commit o merge request può essere associato a un aggiornamento documentale, riducendo il gap tra codice e documentazione.

Adozione di standard e template condivisi
Stabilire standard e template comuni per la documentazione è essenziale per garantire la coerenza e la completezza delle informazioni. La definizione di linee guida interne – che specifichino formati, convenzioni di denominazione, strutture di file e strumenti da utilizzare (ad esempio, sistemi di wiki come Confluence) – favorisce una documentazione uniforme e facilmente consultabile da tutti i membri del team.

Automazione e uso di tecnologie emergenti
L'automazione può alleggerire notevolmente il carico di lavoro relativo alla documentazione. Strumenti basati su Generative AI, come il modello Claude Sonnet, possono essere impiegati per estrarre automaticamente informazioni rilevanti dal codice e generarne bozze documentali. In questo modo, si riduce l'intervento manuale, si garantisce un aggiornamento continuo e si consente di focalizzarsi sulle personalizzazioni e sul perfezionamento della documentazione. Ulteriormente, ricerche recenti hanno evidenziato l'efficacia di approcci specifici per il trattamento del codice. Ad esempio, Feng et al. (2020) [4] presentano CodeBERT, un modello pre-addestrato specifico per linguaggi di programmazione, che viene utilizzato per definire il CodeBERTScore, una metrica adattata per valutare la similarità semantica nelle attività che combinano codice e linguaggio naturale. Parallelamente, la tecnica CodeExp: Explanatory Code Document Generation (Cui et al.)[5] si propone di generare documentazione esplicativa a partire dal codice, illustrando non solo cosa faccia il codice ma anche il ragionamento dietro le scelte implementative. L'approccio proposto in questa tesi si discosta dai metodi tradizionali, poiché sfrutta il prompt engineering applicato al modello Claude Sonnet per guidare in modo preciso la generazione automatica di documentazione dettagliata.

Revisione e feedback continui

La documentazione deve essere considerata come un prodotto "vivo" che necessita di revisioni periodiche e di un processo di feedback costante. Organizzare sessioni di revisione documentale in parallelo alle retrospettive del team o integrarle nelle fasi di code review può aiutare a individuare aree di miglioramento, colmare lacune informative e garantire che la documentazione rispecchi fedelmente lo stato attuale del software.

Formazione e sensibilizzazione
Per superare la resistenza culturale, è necessario promuovere all'interno dell'organizzazione una cultura orientata alla condivisione del sapere. La formazione sui benefici della documentazione e la diffusione di casi di successo possono contribuire a valorizzare questo strumento, incoraggiando tutti i membri del team a partecipare attivamente alla sua creazione e al suo aggiornamento.

In sintesi, la documentazione rappresenta un pilastro fondamentale nel ciclo di vita del software, capace di favorire la qualità, la trasparenza e la continuità operativa dei sistemi. Nonostante le sfide – dalla dinamicità del codice alla mancanza di standard, fino alla percezione negativa da parte degli sviluppatori – l'adozione di best practice mirate può trasformare la documentazione da un compito oneroso a un vantaggio competitivo. L'integrazione automatica, l'uso di template standardizzati, l'impiego di tecnologie innovative e la promozione di una cultura della conoscenza costituiscono gli elementi chiave per garantire che la documentazione rimanga un asset strategico e sempre allineato alle evoluzioni del software.

2.2 Tecnologie per i processi ETL: Apache Airflow e PySpark

Figura 2.1. ETL Process [38]

I processi ETL (Extract, Transform, Load) rappresentano l'infrastruttura portante dell'integrazione dati moderna, fungendo da ponte tra sistemi eterogenei e consentendo alle organizzazioni di trasformare dati grezzi in asset strategici. Questi processi, essenziali nell'architettura dei sistemi informativi contemporanei, seguono un flusso ben definito che permette di gestire l'intero ciclo di vita del dato, dalla sua origine fino all'utilizzo finale per scopi analitici e decisionali.

Estrazione (Extract): Acquisizione dei Dati Grezzi

La fase di estrazione costituisce il punto d'ingresso dell'intero processo ETL, caratterizzandosi per la complessità derivante dalla molteplicità delle fonti dati e dei formati disponibili. Questa fase si articola in diverse componenti critiche:

1. **Strategie di Acquisizione Dati**

   - **Estrazione completa:** Preleva l'intero dataset dalla fonte, adatta per dataset di dimensioni contenute o quando è necessaria una ricostruzione totale.
   - **Estrazione incrementale:** Acquisisce solo i dati modificati dall'ultima estrazione, minimizzando il carico sui sistemi sorgente e ottimizzando le risorse di rete.
   - **Estrazione logica:** Utilizza viste o query per estrarre solo i dati pertinenti, riducendo il volume di dati da processare nelle fasi successive.

2. **Gestione delle Fonti Dati**

   - **Database relazionali:** Connessione tramite driver JDBC/ODBC, ottimizzazione delle query di estrazione per ridurre il carico sui sistemi operazionali.
   - **API e servizi web:** Implementazione di meccanismi di autenticazione sicura, gestione delle rate limits e paginazione delle richieste.
   - **File system e oggetti storage:** Supporto per formati strutturati (CSV, JSON, XML), semi-strutturati (logs) e non strutturati (documenti, immagini).
   - **Sistemi legacy:** Utilizzo di adattatori specifici e middleware per accedere a sistemi proprietari o mainframe.

3. **Metadata Management nell'Estrazione**

- **Catalogazione delle fonti:** Creazione e mantenimento di un registro centralizzato delle fonti dati con relative caratteristiche tecniche.
- **Lineage tracking:** Tracciamento dell'origine del dato, fondamentale per la governance e la compliance normativa.
- **Profiling automatico:** Analisi preliminare delle caratteristiche statistiche dei dati estratti per identificare anomalie o pattern ricorrenti.

4. **Resilienza e Affidabilità**

- **Circuit breaking:** Implementazione di meccanismi per prevenire sovraccarichi sui sistemi sorgente.
- **Retry policies:** Strategie di ripetizione dei tentativi in caso di fallimenti temporanei.
- **Checkpointing:** Salvataggio dello stato di avanzamento per consentire riprese in caso di interruzioni.

Trasformazione (Transform): Il Cuore del Processo ETL

La fase di trasformazione rappresenta il nucleo dell'intelligenza del processo ETL, dove i dati grezzi vengono sottoposti a manipolazioni complesse per renderli fruibili, coerenti e conformi alle esigenze aziendali.

1. **Operazioni di Data Cleaning**

- **Deduplicazione avanzata:** Utilizzo di algoritmi fuzzy matching e distance metrics per identificare duplicati non evidenti.
- **Gestione valori anomali:** Identificazione statistica degli outliers e applicazione di tecniche di correzione contestuale.
- **Trattamento valori mancanti:** Strategie di imputazione basate su regole business, modelli statistici o machine learning.
- **Validazione semantica:** Verifica della coerenza dei dati rispetto a vincoli di dominio e regole di business.

2. **Data Normalization e Standardizzazione**

- **Normalizzazione strutturale:** Conversione dei dati in formati canonici predefiniti per garantire coerenza.

- **Armonizzazione semantica:** Riconciliazione di terminologie e tassonomie diverse utilizzate in fonti eterogenee.
- **Standardizzazione geografica:** Uniformazione di indirizzi, coordinate e riferimenti spaziali secondo standard internazionali.
- **Codifica temporale:** Normalizzazione di date e timestamp in formati standard, gestione dei fusi orari e calendari specifici.

3. **Arricchimento e Feature Engineering**

- **Integrazione con dati esterni:** Incorporazione di dataset pubblici o commerciali per arricchire i dati proprietari.
- **Derivazione di metriche complesse:** Calcolo di metriche avanzate attraverso formule e algoritmi specifici per il dominio.
- **Pattern recognition:** Identificazione di trend, stagionalità e correlazioni nei dati attraverso tecniche analitiche.
- **Text mining:** Estrazione di entità, sentiment e temi da contenuti testuali non strutturati.

4. **Architetture di Trasformazione**

- **ETL vs ELT:** Valutazione del paradigma più adatto in base al volume dei dati e alla complessità delle trasformazioni.
- **Batch vs Streaming:** Implementazione di logiche di trasformazione adatte sia a elaborazioni periodiche che in tempo reale.
- **Transformazioni distribuite:** Utilizzo di framework come Apache Spark per elaborazioni parallele su grandi volumi.
- **Microservices per trasformazioni:** Decomposizione di logiche complesse in servizi atomici e riutilizzabili.

5. **Qualità e Governance nella Trasformazione**

- **Data quality gates:** Implementazione di checkpoint con regole di validazione progressive.
- **Versioning delle trasformazioni:** Gestione delle modifiche alle logiche di business con tracciabilità completa.
- **Audit trail:** Registrazione dettagliata di ogni modifica apportata ai dati durante il processo.

- **Impact analysis:** Valutazione preventiva dell'impatto delle modifiche alle logiche di trasformazione.

Caricamento (Load): Destinazione e Persistenza

La fase di caricamento costituisce lo stadio finale del processo ETL, dove i dati trasformati vengono integrati nei sistemi di destinazione in modo ottimale per supportare i casi d'uso analitici e operativi. Questa fase presenta sfide specifiche legate alla performance, all'integrità e alla disponibilità dei dati:

1. **Modalità di Caricamento**
   - **Bulk loading:** Caricamento massivo ottimizzato per grandi volumi di dati, sfruttando utility native dei database.
   - **Micro-batch:** Suddivisione del carico in piccoli lotti per ridurre l'impatto sui sistemi target.
   - **Change Data Capture (CDC):** Propagazione in tempo reale delle modifiche per mantenere sincronizzati i sistemi.
   - **Upsert intelligente:** Algoritmi di merge che minimizzano le operazioni di I/O preservando l'integrità dei dati.
2. **Ottimizzazione delle Destinazioni**
   - **Schema evolution:** Gestione dei cambiamenti strutturali senza interruzioni di servizio.
   - **Partizionamento dinamico:** Strategia di segmentazione dei dati basata su pattern di accesso e volume.
   - **Compressione selettiva:** Applicazione di algoritmi di compressione specifici per tipologia di dato.
   - **Indexing strategy:** Creazione e manutenzione di indici ottimali per supportare i pattern di query più frequenti.
3. **Data Serving Models**
   - **Data warehouse tradizionale:** Organizzazione dei dati secondo modelli dimensionali (star/snowflake schema).
   - **Data lake:** Archiviazione di dati raw e processed in formati aperti con schema-on-read.

- **Lakehouse:** Approccio ibrido che combina la flessibilità dei data lake con la struttura dei data warehouse.
- **Feature store:** Repository specializzato per attributi derivati utilizzabili in modelli analitici e ML.

4. **Governance nel Caricamento**

- **Security by design:** Implementazione di controlli di accesso granulari e crittografia dei dati sensibili.
- **Compliance automation:** Applicazione automatica di politiche di ritenzione e mascheramento dati.
- **Lineage end-to-end:** Completamento della tracciabilità del dato fino alla destinazione finale.
- **Data catalogue integration:** Aggiornamento automatico dei metadati nei cataloghi aziendali.

Orchestrazione e Monitoraggio dei Processi ETL

Un aspetto cruciale, spesso trascurato nella trattazione dei processi ETL, riguarda la gestione del ciclo di vita completo delle pipeline di dati:

1. **Frameworks di Orchestrazione**

- **Workflow management:** Utilizzo di strumenti come Apache Airflow per definire dipendenze e sequenze.
- **Scheduling avanzato:** Configurazione di trigger basati su eventi, temporali o ibridi.
- **Error handling distributito:** Strategie di gestione degli errori con propagazione controllata.
- **Backfilling e reprocessing:** Meccanismi per rielaborare dati storici mantenendo la coerenza.

2. **Observability e Monitoring**

- **Metriche operative:** Tracciamento di throughput, latenza e consumo di risorse.
- **KPI di data quality:** Monitoraggio continuo della completezza, accuratezza e freschezza dei dati.
- **Alerting proattivo:** Implementazione di sistemi di notifica basati su anomaly detection.

- **SLA tracking:** Misurazione delle performance rispetto agli accordi di livello di servizio.

3. **Evoluzione e Manutenzione**
   - **CI/CD per pipeline dati:** Automazione dei test e del deployment delle modifiche.
   - **A/B testing di trasformazioni:** Valutazione dell'impatto di nuove logiche prima della produzione.
   - **Ottimizzazione continua:** Analisi delle performance e refactoring incrementale.
   - **Documentazione dinamica:** Aggiornamento automatizzato della documentazione tecnica e funzionale.

I processi ETL rappresentano l'infrastruttura critica che abilita la trasformazione digitale delle organizzazioni, permettendo di convertire dati grezzi in insight strategici. Una corretta implementazione dei processi ETL non solo garantisce l'affidabilità e la qualità dei dati, ma costituisce anche un vantaggio competitivo significativo.

Apache Airflow

Figura 2.2. Apache Airflow [39]

Apache Airflow è una piattaforma open-source per l'orchestrazione dei workflow, nata inizialmente all'interno di Airbnb nel 2014 e successivamente donata alla Apache Software Foundation nel 2016. Nel tempo, si è affermata come lo standard di riferimento per la gestione di pipeline ETL complesse in ambienti enterprise, grazie alle sue caratteristiche di flessibilità, scalabilità e robustezza.

Architettura e Componenti Principali

DAG (Directed Acyclic Graph)

In Airflow, i workflow vengono rappresentati come DAG, ossia grafi diretti aciclici [8]. Questa struttura consente di:

- Definire in modo chiaro e dichiarativo l'ordine di esecuzione e le dipendenze tra i vari task.
- Visualizzare l'intero workflow come un grafico navigabile, semplificando il monitoraggio e il debugging.
- Garantire un'esecuzione deterministica dei processi ETL, evitando esecuzioni ridondanti o fuori sequenza.
- Implementare pattern avanzati come il fan-in/fan-out, utile per elaborazioni parallele dei dati.

Task e Operator

I task rappresentano le unità atomiche di lavoro all'interno di un DAG, mentre gli operator definiscono il tipo di operazione che un task deve eseguire. Airflow mette a disposizione diversi operator predefiniti, tra cui:

- **BashOperator** per eseguire comandi shell.
- **PythonOperator** per lanciare funzioni Python personalizzate.
- **SQLOperator** per l'esecuzione di query su database relazionali.
- **DockerOperator** per avviare container Docker.
- **KubernetesPodOperator** per orchestrare l'esecuzione di pod Kubernetes.
- **ECSOperator** per l'integrazione con Amazon ECS.
- **SparkSubmitOperator** per lanciare job Spark.

Oltre agli operator predefiniti, Airflow consente anche la creazione di operator personalizzati.

Dipendenze tra Task

Le dipendenze tra task vengono definite in modo intuitivo attraverso operatori relazionali (» e «) oppure tramite metodi come `set_upstream()` e `set_downstream()`.

XCom (Cross-Communication)

Un meccanismo che permette lo scambio di piccole quantità di dati tra task all'interno di un DAG. Grazie a XCom, è possibile implementare pattern avanzati come map-reduce o pipeline di trasformazione incrementale dei dati.

Componenti Infrastrutturali

Scheduler

Il cuore di Airflow, responsabile della gestione e dell'esecuzione dei DAG. Le sue funzioni includono: rilevare i DAG definiti e generare le relative istanze (DAG run), monitorare le dipendenze tra i task e stabilire quali siano pronti per l'esecuzione, assegnare i task agli executor disponibili. Inoltre lo Scheduler si occupa di implementare meccanismi di riavvio automatico in caso di fallimenti e gestire timeout e heartbeat per garantire la stabilità del sistema.

Executor

Determina come i task vengono eseguiti. Airflow supporta diverse modalità di esecuzione [12], tra cui:

- **SequentialExecutor**, che esegue i task uno alla volta, utile solo per test e debugging.
- **LocalExecutor**, che consente la parallelizzazione su thread e processi locali.
- **CeleryExecutor**, che distribuisce i task su più worker attraverso il framework Celery.
- **KubernetesExecutor**, che permette l'esecuzione dinamica su cluster Kubernetes.
- **DaskExecutor**, che sfrutta il framework di calcolo parallelo Dask.

Metastore

Airflow utilizza un database relazionale (come PostgreSQL, MySQL o SQLite) per memorizzare informazioni fondamentali quali:

Il metastore di Airflow svolge un ruolo cruciale nella gestione e nel monitoraggio dei workflow. Al suo interno vengono archiviate tutte le definizioni dei DAG, insieme al loro stato di esecuzione, consentendo così di tenere traccia dei processi in corso e di quelli completati. Inoltre, il database conserva lo storico delle esecuzioni dei singoli task, permettendo analisi retrospettive e debug più efficace.

Un altro aspetto fondamentale riguarda la gestione delle variabili e delle connessioni, che vengono memorizzate in modo sicuro grazie a meccanismi di crittografia, garantendo così la protezione delle credenziali sensibili. Infine, il metastore raccoglie informazioni dettagliate sui log e sul monitoraggio delle operazioni, offrendo visibilità completa sulle attività del sistema e facilitando l'identificazione di eventuali anomalie.

Sicurezza e Multi-Tenancy

Airflow offre diverse funzionalità avanzate per garantire sicurezza e multi-tenancy, assicurando che l'accesso e l'esecuzione dei workflow avvengano in modo controllato e protetto.

L'autenticazione è supportata attraverso diversi provider, come LDAP, OAuth e autenticazione diretta su database, permettendo di integrare il sistema con le infrastrutture esistenti per la gestione degli accessi.

Per quanto riguarda l'autorizzazione, Airflow implementa un modello RBAC (Role-Based Access Control), che consente di assegnare permessi granulari agli utenti in base ai loro ruoli. Questo approccio garantisce che ogni utente abbia accesso solo alle risorse necessarie, riducendo il rischio di operazioni non autorizzate.

Un altro aspetto chiave della sicurezza è l'isolamento dei task, ottenuto tramite l'uso della containerizzazione. Questa tecnica permette di eseguire i processi in ambienti separati, evitando interferenze tra workflow diversi e migliorando la stabilità complessiva del sistema.

Infine, Airflow protegge le credenziali sensibili e le connessioni attraverso meccanismi di crittografia, assicurando che le informazioni riservate siano memorizzate e trasmesse in modo sicuro, riducendo il rischio di compromissioni.

Evoluzione e Tendenze Recenti

- **Scheduler più performante:** Modello di concorrenza ottimizzato.
- **API REST:** Completa e documentata per facilitare l'integrazione con altri strumenti.
- **Supporto per DAG dinamici e task mapping:** Migliora la flessibilità nella definizione dei workflow.
- **Ergonomia migliorata degli operatori:** Standardizzazione delle interfacce.
- **Integrazione avanzata con Kubernetes:** Perfetto per ambienti cloud-native.

Apache Spark

Apache Spark rappresenta una vera e propria rivoluzione nel panorama dell'elaborazione dati distribuita. Nato nei laboratori dell'Università di California a Berkeley nel 2009 e successivamente accolto dalla Apache Software Foundation nel 2013, Spark ha ridefinito le possibilità di analisi dei big data, superando i limiti che caratterizzavano i precedenti framework come Hadoop MapReduce.

Figura 2.3. Apache Spark [40]

Ciò che rende Spark così importante è la sua capacità di mantenere i dati in memoria durante l'elaborazione. Nel 2012 è stato introdotto il concetto di Resilient Distributed Datasets (RDDs) (Zaharia et al.), permettendo di trattenere i dati in RAM anziché scriverli continuamente su disco. I risultati mostrano prestazioni fino a 100 volte superiori rispetto ai tradizionali approcci basati su MapReduce, soprattutto per workload iterativi come quelli tipici del machine learning.

> "L'approccio in-memory di Spark non è solo un miglioramento incrementale, ma un ripensamento fondamentale di come elaborare i dati su larga scala" (Armbrust, 2015) [13].

E non si tratta solo di velocità: l'architettura distribuita di Spark, con il suo modello driver-executor, offre un equilibrio straordinario tra performance e fault tolerance.

Fault tolerance: se un nodo del cluster dovesse fallire, il sistema non crollerebbe. Grazie al tracciamento delle operazioni (lineage) sui RDD, Spark è in grado di ricostruire i dati persi riprocessando selettivamente solo le porzioni necessarie. Questa resilienza, combinata con la velocità dell'elaborazione in-memory, ha reso Spark lo strumento prediletto per applicazioni data-intensive in svariati settori industriali.

Un ecosistema completo per l'analisi dei dati

Spark non è solo un motore di calcolo distribuito, ma un ricco ecosistema di componenti integrati che coprono l'intero spettro dell'elaborazione dati moderna.

Spark SQL ha trasformato l'interazione con i dati strutturati, introducendo i DataFrame e un sofisticato ottimizzatore di query chiamato Catalyst.

> "L'astrazione DataFrame ha reso accessibile la potenza di Spark a data scientist abituati a strumenti come R e pandas" (Armbrust, 2020)[14].

Questa interfaccia dichiarativa nasconde la complessità dell'elaborazione distribuita, permettendo di concentrarsi sulla logica di business anziché sui dettagli implementativi.

Per chi lavora con dati in tempo reale, Spark Streaming offre un modello di micro-batch che garantisce elaborazione affidabile con semantica exactly-once [17]. Inoltre, questa componente è stata ulteriormente migliorata con l'introduzione di Structured Streaming, portando la semplicità del modello DataFrame anche nel dominio dello stream processing.

Il machine learning distribuito trova in MLlib una soluzione elegante per scalare algoritmi complessi su dataset massivi [18]. Questa libreria non solo implementa versioni parallele di algoritmi classici, ma offre un'API per pipeline end-to-end ispirata a scikit-learn, facilitando la transizione dal prototipo alla produzione.

PySpark

Se Spark ha rivoluzionato l'elaborazione distribuita, PySpark ha incrementato l'accesso a questa tecnologia. L'interfaccia Python per Spark ha permesso a milioni di utenti di sfruttare la potenza del calcolo distribuito senza abbandonare il loro linguaggio preferito.

Il meccanismo che rende possibile questa integrazione è Py4J, che crea un canale di comunicazione tra il driver Python e la JVM sottostante, mentre la serializzazione ottimizzata con Apache Arrow minimizza l'overhead di trasferimento dati. In questo modo, gli sviluppatori possono scrivere codice in Python, beneficiando al contempo delle ottimizzazioni del motore Spark.

L'API DataFrame di PySpark, ispirata a pandas ma progettata per l'elaborazione distribuita, offre una sintassi intuitiva per operazioni complesse su dataset di dimensioni arbitrarie [16]. L'introduzione di Koalas ha ulteriormente migliorato questa compatibilità, implementando l'API pandas su architettura distribuita.

Applicazioni reali e tendenze future

La versatilità di Spark e PySpark si riflette nell'ampia gamma di applicazioni industriali. Dalle tradizionali pipeline ETL all'analisi in tempo reale di dati IoT, dall'addestramento distribuito di modelli di machine learning all'elaborazione di grafi sociali complessi, questi strumenti hanno trovato applicazione in praticamente ogni dominio che coinvolge i big data.

Guardando al futuro, l'integrazione con Kubernetes sta emergendo come tendenza dominante, facilitando il deployment di applicazioni Spark in ambienti cloud-native. Parallelamente, l'ottimizzazione per hardware specializzato come GPU sta aprendo nuove possibilità per carichi di lavoro di deep learning.

L'adozione di formati di tabella aperti come Delta Lake, Iceberg e Hudi sta trasformando Spark da semplice motore di elaborazione a componente centrale di moderne architetture lakehouse, che combinano l'affidabilità dei data warehouse con la flessibilità dei data lake [15]. Questa evoluzione permette transazioni ACID su storage oggetti cloud, risolvendo una delle principali limitazioni delle tradizionali architetture big data.

Apache Spark e PySpark hanno ridefinito le possibilità dell'elaborazione dati distribuita, combinando performance straordinarie con API intuitive. Il loro successo non deriva solo dalle innovazioni tecniche, ma anche dalla capacità di costruire ponti tra comunità diverse: sviluppatori Java, data

scientist Python, analisti SQL possono tutti lavorare efficacemente con lo stesso framework.

In un'epoca in cui i volumi di dati continuano a crescere esponenzialmente, strumenti come Spark non sono più un lusso ma una necessità. E in questo scenario, Spark e PySpark continuano a rappresentare lo stato dell'arte, evolvendo costantemente per rispondere alle sfide emergenti del data engineering moderno.

2.3 Generative AI e Prompt Engineering: concetti e applicazioni

La Generative AI rappresenta una frontiera rivoluzionaria nel campo dell'intelligenza artificiale, caratterizzata dalla capacità di creare contenuti originali piuttosto che limitarsi ad analizzare o classificare dati esistenti. Questa classe di tecnologie, basata su architetture avanzate di deep learning, ha conosciuto un'accelerazione senza precedenti negli ultimi anni, trasformando radicalmente numerosi settori industriali e aprendo nuovi orizzonti per l'automazione creativa [21].

Evoluzione storica dei modelli generativi

Il percorso evolutivo dei modelli generativi può essere tracciato attraverso diverse fasi fondamentali:

Da GANs ai transformer

I primi passi significativi nella generazione di contenuti risalgono all'introduzione delle Generative Adversarial Networks (GANs) (Goodfellow, 2014)[22]. Queste architetture innovative hanno stabilito un paradigma competitivo tra un generatore e un discriminatore, consentendo la creazione di immagini realistiche. Tuttavia, la vera rivoluzione è avvenuta nel 2017 con la pubblicazione del paper "Attention Is All You Need" (Vaswani et al., 2017) [23], che ha introdotto l'architettura Transformer. Come sottolinea LeCun (2022) [24]:

> "L'architettura Transformer ha rappresentato un cambio di paradigma fondamentale, spostando l'attenzione dai modelli sequenziali ricorrenti a meccanismi di attenzione parallela, capaci di catturare dipendenze a lungo raggio in modo più efficiente."

Large Language Models (LLMs)

Dal 2018, l'evoluzione si è concentrata sullo scaling dei modelli linguistici. GPT (Generative Pre-trained Transformer) di OpenAI ha segnato l'inizio di questa tendenza, seguita da una rapida progressione dimensionale: da GPT-1 con 117 milioni di parametri a GPT-3 con 175 miliardi, fino ai più recenti modelli come PaLM di Google (540 miliardi) e GPT-4 di OpenAI (presumibilmente oltre 1 trilione) [25].

Multimodalità e modelli specializzati

Il paradigma più recente vede l'emergere di modelli multimodali come DALL-E, Midjourney e Stable Diffusion per la generazione di immagini, e modelli specializzati come Anthropic Claude e Cohere Command, ottimizzati per casi d'uso specifici. Infatti, la tendenza verso la multimodalità rappresenta un passo cruciale verso sistemi AI più versatili e comprensivi, capaci di integrare diverse forme di conoscenza e comunicazione [27].

Architettura e funzionamento dei modelli generativi:

Transformer e multi-head attention

Al cuore dei moderni modelli generativi troviamo l'architettura Transformer, caratterizzata da:

- Meccanismi di self-attention che permettono al modello di pesare differentemente l'importanza di diverse parti dell'input;
- Elaborazione parallela che supera le limitazioni delle architetture ricorrenti;
- Layer di feed-forward per l'elaborazione non lineare delle rappresentazioni;
- Normalizzazione e connessioni residue per facilitare l'addestramento di reti profonde.

Pre-training e fine-tuning L'addestramento di questi modelli segue tipicamente un approccio in due fasi:

- **Pre-training:** addestramento su vasti corpus testuali non supervisionati per apprendere rappresentazioni linguistiche generali, utilizzando obiettivi come la prediction del token successivo.
- **Fine-tuning:** adattamento a task specifici attraverso addestramento supervisionato su dataset più piccoli e mirati.

L'efficacia di questo approccio è stata dimostrata nel progetto T5 (Text-to-Text Transfer Transformer), che unifica diversi task di NLP in un unico framework di trasformazione testo-testo [25].

Valutazione e metriche per modelli generativi

La valutazione dei modelli generativi presenta sfide uniche, richiedendo approcci multidimensionali.

Metriche automatiche:

- **Perplexity:** misura la capacità predittiva del modello su nuovi testi.
- **BLEU, ROUGE, METEOR:** metriche basate sul confronto con riferimenti umani.
- **BERTScore:** valutazioni semantiche basate su embedding contestuali.

Valutazioni umane:

- **Coherence:** valutazione della coerenza logica e narrativa del testo generato.
- **Factual accuracy:** verifica dell'accuratezza fattuale delle informazioni.
- **Helpfulness e harmlessness:** allineamento con intenzioni e valori umani.

Limiti e sfide della Generative AI

Nonostante i progressi significativi, i modelli generativi presentano ancora limitazioni sostanziali:

Hallucinations e accuratezza fattuale

I modelli tendono a generare informazioni plausibili ma false, specialmente quando operano al di fuori dei domini ben rappresentati nei dati di addestramento [26]. Le hallucinations rappresentano una sfida fondamentale per l'applicazione dei modelli generativi in contesti che richiedono alta affidabilità, come la documentazione tecnica e la comunicazione scientifica.

Bias e rappresentazione

I modelli di intelligenza artificiale non sono entità neutre, ma riflettono i dati su cui vengono addestrati. Se questi dati contengono bias, errori sistematici o rappresentazioni squilibrate, i modelli non solo li ereditano, ma

possono anche amplificarli in modo significativo. Questo fenomeno è particolarmente critico in ambiti come il riconoscimento facciale, la selezione automatizzata del personale o la generazione di contenuti, dove le distorsioni nei dati possono portare a discriminazioni involontarie o alla perpetuazione di stereotipi. Ad esempio, se un dataset utilizzato per addestrare un modello di NLP contiene pregiudizi di genere, il modello potrebbe generare risposte che rafforzano ruoli tradizionali o discriminazioni implicite. Allo stesso modo, un sistema di computer vision addestrato su immagini prevalentemente di un determinato gruppo etnico potrebbe mostrare prestazioni inferiori su volti di altre etnie, con conseguenze potenzialmente gravi in contesti di sicurezza o identificazione personale. Per mitigare questi rischi, è essenziale adottare strategie di debiasing, come la diversificazione dei dataset, l'uso di tecniche di bilanciamento statistico e la valutazione continua dei modelli con metriche di equità. Inoltre, un approccio trasparente e responsabile allo sviluppo dell'IA, che coinvolga esperti di etica e rappresentanti delle comunità potenzialmente impattate, è fondamentale per garantire sistemi più equi e inclusivi.

Contestualizzazione e comprensione profonda

Nonostante la loro capacità di produrre output sintatticamente e semanticamente plausibili, i modelli generativi attuali non possiedono una vera comprensione concettuale o contestuale. Il loro funzionamento si basa su architetture che apprendono distribuzioni statistiche di token all'interno di enormi dataset testuali. Tuttavia, la loro capacità di generare testi coerenti non implica la costruzione di una rappresentazione interna del significato, ma piuttosto l'identificazione di pattern ricorrenti nei dati di addestramento. Questi modelli operano attraverso il calcolo di probabilità condizionali su sequenze di token, senza alcuna forma di inferenza simbolica o comprensione strutturata della conoscenza. Ciò li rende vulnerabili a problemi come l'errore di ancoraggio, la mancanza di robustezza nel ragionamento logico e l'incapacità di disambiguare concetti fuori distribuzione. Inoltre, non possedendo una memoria esplicita e strutturata del contesto globale, la loro coerenza discorsiva è limitata da finestre di attenzione finite, portando a incongruenze su scale testuali estese. Sebbene tecniche come il retrieval-augmented generation (RAG) e i modelli multimodali stiano migliorando la capacità di questi sistemi di integrare informazioni più strutturate, il problema della comprensione semantica profonda rimane una sfida aperta nel campo dell'intelligenza artificiale.

Tendenze future

Il campo della Generative AI continua a evolversi rapidamente, con diverse direzioni promettenti:

Modelli più efficienti e sostenibili

La ricerca si sta orientando verso architetture più efficienti che mantengano capacità competitive con minor consumo computazionale. LLaMA e MosaicML con MPT hanno dimostrato che modelli più piccoli, addestrati strategicamente, possono raggiungere performance competitive con quelli significativamente più grandi [27].

Integrazione con strumenti e knowledge bases

L'evoluzione verso LLM-powered agents in grado di interagire con strumenti esterni e accedere dinamicamente a knowledge bases rappresenta un'area di ricerca in forte sviluppo. L'integrazione di modelli linguistici con meccanismi di retrieval-augmented generation (RAG), sistemi di verifica automatizzata e accesso a database strutturati consente di mitigare in modo significativo il problema delle hallucinations, particolarmente critico in contesti tecnici e scientifici [28]. Tecniche come il grounding su fonti affidabili e la combinazione di modelli generativi con sistemi di symbolic reasoning offrono nuove prospettive per migliorare l'affidabilità e la precisione delle risposte.

Personalizzazione domain-specific

Il fine-tuning su domini verticali specifici rappresenta una strategia chiave per incrementare le performance dei modelli in ambiti specialistici. È stato dimostrato che modelli addestrati su dataset altamente curati, come documentazione tecnica, repository di codice e letteratura scientifica, superano in accuratezza e rilevanza i modelli generici di dimensioni maggiori nella generazione di contenuti specifici. Inoltre, approcci few-shot e continual learning permettono di aggiornare i modelli senza necessità di un riaddestramento completo, facilitando l'adattamento a domini in rapida evoluzione come la medicina, l'ingegneria e la finanza.

Claude 3.5 Sonnet: Architettura e Caratteristiche

Claude 3.5 Sonnet rappresenta uno dei modelli più avanzati della famiglia Claude 3, sviluppata da Anthropic. Questo modello linguistico di grandi

dimensioni (LLM) offre un equilibrio ottimale tra potenza, velocità e costo, posizionandosi come soluzione ideale per applicazioni di documentazione automatica, come quella proposta in questo progetto.

Architettura e Capacità Tecniche

Claude 3.5 Sonnet si basa su un'architettura Transformer avanzata, progettata specificamente per eccellere in:

- **Comprensione del codice:** Capacità superiore di analizzare e interpretare codice sorgente complesso, incluse pipeline ETL strutturate in diversi linguaggi di programmazione.
- **Generazione di testo strutturato:** Abilità di produrre documentazione tecnica coerente, mantenendo una struttura logica conforme a template predefiniti.
- **Context window estesa:** Supporto per l'elaborazione di documenti lunghi e complessi, permettendo l'analisi di interi moduli o pipeline ETL in un'unica richiesta.

Vantaggi nel Contesto del Progetto

L'utilizzo di Claude 3.5 Sonnet come motore centrale per il tool di documentazione automatica offre diversi vantaggi specifici:

- **Estrazione semantica avanzata:** Il modello è in grado di identificare non solo la struttura sintattica del codice, ma anche le relazioni funzionali tra i componenti di una pipeline ETL, migliorando la qualità e la rilevanza delle informazioni estratte.
- **Adattabilità ai template:** La capacità di seguire istruzioni precise consente di formattare l'output secondo template Confluence standardizzati, garantendo coerenza e uniformità nella documentazione.
- **Contestualizzazione delle informazioni:** Il modello può collegare le funzionalità tecniche agli obiettivi di business, arricchendo la documentazione con contesto rilevante e rendendo più comprensibile il valore aggiunto delle soluzioni implementate.

Performance e Limitazioni

È importante considerare sia i punti di forza che le limitazioni di Claude 3.5 Sonnet nell'implementazione del progetto:

- **Punti di forza:**
  - Eccellente comprensione delle relazioni tra componenti software.
  - Capacità di generare spiegazioni tecniche accessibili a diversi livelli di competenza.
  - Efficienza nell'elaborazione di grandi volumi di codice, consentendo analisi rapide e documentazione dettagliata.
- **Limitazioni da considerare:**
  - Necessità di un prompt engineering accurato per garantire output consistenti e pertinenti.
  - Possibili imprecisioni nell'interpretazione di pattern di codice altamente specializzati, che potrebbero richiedere interventi manuali per la validazione.
  - Requisiti di risorse computazionali da bilanciare con le esigenze di automazione continua, specialmente in contesti di produzione su larga scala.

Integrazione nell'Architettura di Sistema

Nel contesto del presente progetto, Claude 3.5 Sonnet viene integrato come componente di elaborazione centrale, posizionato strategicamente tra il sistema di estrazione del codice sorgente e il modulo di pubblicazione su Confluence. Il modello opera attraverso la Anthropic API, utilizzando configurazioni ottimizzate per il parsing e la documentazione del codice ETL, garantendo così un flusso di lavoro efficiente e automatizzato.

Prompt Engineering

Il Prompt Engineering rappresenta l'interfaccia cruciale tra intenzioni umane e capacità della genAI. Questa disciplina emergente combina elementi di linguistica computazionale, psicologia cognitiva e computer science, focalizzandosi sulla progettazione ottimale delle istruzioni fornite ai modelli linguistici per ottenere output precisi e allineati agli obiettivi [29].

Fondamenti teorici del prompt engineering

I Large Language Models (LLMs) operano come sistemi di completamento probabilistico, generando la sequenza di token più probabile in base a una

determinata prompt iniziale. Il loro comportamento è guidato da distribuzioni condizionate apprese durante la fase di addestramento su grandi quantità di dati testuali. La mancanza di una reale comprensione semantica implica che la formulazione del prompt influenzi direttamente la qualità e l'accuratezza della risposta. Tecniche come la formulazione di richieste esplicite, la definizione di ruoli e l'uso di esempi (few-shot prompting) possono orientare il modello verso output più controllati e pertinenti.

Context window e information retrieval

La context window di un modello rappresenta sia una limitazione strutturale che un'opportunità per ottimizzare il prompt engineering. Essa determina la quantità massima di testo che il modello può considerare in un'unica inferenza, influenzando la capacità di mantenere coerenza e precisione nei testi generati. Per superare questo limite, tecniche avanzate come il retrieval-augmented generation (RAG) permettono di estendere la capacità informativa del modello attraverso la consultazione di knowledge bases esterne o documenti specifici. Strategie come la compressione dell'informazione rilevante all'interno della context window e l'uso di strutture gerarchiche nel prompt migliorano l'efficacia della generazione testuale in compiti complessi.

Tecniche avanzate di prompt engineering

L'evoluzione del prompt engineering ha portato allo sviluppo di metodologie più sofisticate per ottimizzare l'interazione con gli LLMs:

- **Prompt-Chaining**

  Suddivisione del processo generativo in più passi sequenziali, in cui ogni risposta parziale del modello funge da input per il passaggio successivo. Questo approccio consente di gestire richieste complesse in modo strutturato e incrementale, riducendo il rischio di errori di coerenza.

- **Chain-of-Thought**

  Una delle innovazioni più significative nel prompt engineering è la tecnica Chain-of-Thought (CoT) [30], che incoraggia il modello a scomporre problemi complessi in passaggi intermedi espliciti. L'idea di fornire esempi di ragionamento passo-passo migliora drasticamente le performance dei modelli su task che richiedono ragionamento multi-step, consentendo

loro di emulare processi cognitivi umani. Questa tecnica si è evoluta in varianti come:

  - **Zero-shot CoT:** utilizzando frasi ponte come "ragioniamo passo per passo"
  - **Self-consistency CoT:** generando multiple traiettorie di ragionamento e selezionando risultati consensuali
  - **Tree of Thoughts:** esplorando diversi percorsi di ragionamento in parallelo

- **Few-shot learning**

  L'apprendimento contestuale attraverso pochi esempi (few-shot learning) rappresenta una tecnica fondamentale. Brown et al. (2020) [31] hanno dimostrato nel loro studio su GPT-3 che "l'inclusione di pochi esempi rappresentativi all'interno del prompt può orientare il modello verso il formato e lo stile desiderati, senza richiedere fine-tuning esplicito". Altre ricerche di hanno ulteriormente raffinato questa tecnica, identificando principi ottimali per la selezione degli esempi, la diversità, rappresentatività e gradualità degli esempi impattano significativamente l'efficacia dell'apprendimento in-context, suggerendo che la cura nella selezione degli esempi è paragonabile all'importanza dell'iperparameter tuning nei paradigmi tradizionali. [32]

- **Tecniche di refinement iterativo**

  Il refinement iterativo prevede un processo di affinamento progressivo degli output attraverso:

  - **Self-reflection:** urichiedere al modello di rivedere criticamente il proprio output
  - **Iterative refinement:** raffinare progressivamente un output attraverso feedback specifici
  - **Critique-revision loops:** alternare fasi di critica e revisione

- **Mitigazione dei bias**

I prompt possono amplificare o mitigare bias presenti nei modelli. Tecniche come i "guardrail prompts" che specificano valori e principi da rispettare possono ridurre il rischio di output problematici.

Prompting Specifico per il Modello Claude Sonnet

Il modello Claude Sonnet si distingue per la sua capacità di elaborare input strutturati in formato XML, il che consente una gestione dettagliata e controllata delle istruzioni. Questa sezione approfondisce gli aspetti tecnici e metodologici legati alla progettazione dei prompt per Claude Sonnet.

Struttura dell'Input XML

Il prompt viene realizzato come documento XML, che permette di definire chiaramente le diverse sezioni e metadati necessari per guidare il modello. Una struttura XML tipica comprende:

- **Tag di Contesto:** Forniscono il background dell'argomento, includendo specifiche tecniche e requisiti del progetto.
- **Tag di Istruzione:** Contengono le direttive esplicite riguardo al formato, al tono, al livello di dettaglio e alla logica di ragionamento desiderata.
- **Tag di Esempio:** Includono esempi di output atteso, facilitando un approccio few-shot che orienta il modello verso la struttura e lo stile voluti.

Questa organizzazione consente di segmentare il prompt in sezioni facilmente interpretabili, riducendo ambiguità e migliorando la coerenza dell'output. Inoltre, la natura strutturata dell'XML agevola l'automazione del processo di generazione dei prompt, integrandosi in maniera fluida con le pipeline di documentazione automatizzata.

Best Practices

Definizione Chiara dei Metadati: Includere metadati dettagliati per definire il contesto e le aspettative relative all'output, aiutando il modello a orientarsi nella generazione del testo.

- **Strutturazione Gerarchica:** Organizzare il prompt XML in una gerarchia che rifletta la struttura del documento finale, garantendo un output ordinato e coerente.

- **Utilizzo di Esempi:** Fornire esempi specifici di output atteso all'interno del prompt, in modo da guidare il modello verso il formato desiderato.

- **Iterazione e Refinement:** Affinare progressivamente il prompt sulla base dei feedback, migliorando l'allineamento tra le istruzioni e il risultato generato.

- **Validazione del Prompt:** Utilizzare strumenti di validazione XML per assicurare la correttezza della struttura, evitando errori sintattici che possano compromettere l'interpretazione del modello.

In sintesi, il prompting per Claude Sonnet, grazie all'uso di input XML strutturati, consente di ottenere output altamente precisi e coerenti. La combinazione di una struttura ben definita, metadati accurati, esempi esplicativi e un processo iterativo di refinement permette di sfruttare appieno le capacità del modello, contribuendo in modo determinante alla creazione automatica di documentazione tecnica di alta qualità.

```
You're a financial analyst at AcmeCorp.
Generate a Q2 financial report for our investors.
AcmeCorp is a B2B SaaS company.
Our investors value transparency and actionable insights.
Use this data for your report:

<data>{{SPREADSHEET_DATA}}</data>
<instructions>
    1. Include sections: Revenue Growth, Profit Margins, Cash Flow.
    2. Highlight strengths and areas for improvement.
</instructions>
Make your tone concise and professional. Follow this structure:
<formatting_example>{{Q1_REPORT}}</formatting_example>
```

Figura 2.4. Prompt di esempio nella guida Antropic [33]

2.4 Integrazione con strumenti collaborativi: Confluence e GitHub

L'integrazione con strumenti collaborativi rappresenta un elemento cruciale per garantire la trasparenza, la coerenza e la condivisione della conoscenza all'interno dei team di sviluppo. In particolare, l'uso delle API REST di Confluence [34] e GitHub [35] consente di automatizzare la creazione, l'aggiornamento e il monitoraggio della documentazione tecnica, collegando in maniera diretta il codice sorgente e il relativo contesto documentale.

Confluence

Confluence è una piattaforma di collaborazione e gestione della documentazione che funge da repository centralizzato per le informazioni tecniche e di progetto. Utilizzando le sue API REST, il tool sviluppato in questa tesi è in grado di:

- **Creare e aggiornare pagine:** Automatizzare la generazione di pagine documentali che riflettano le modifiche apportate al codice, garantendo che la documentazione sia sempre allineata allo stato attuale del progetto.
- **Recuperare metadati:** Estrarre informazioni rilevanti (come titoli, descrizioni e tag) per organizzare e strutturare i contenuti in modo coerente.

Questa integrazione permette di centralizzare le informazioni e di renderle accessibili a tutti i membri del team, migliorando la collaborazione e la comunicazione interna.

GitHub

GitHub è una piattaforma leader per il versionamento del codice e la gestione dei repository, che favorisce la collaborazione tra sviluppatori. Attraverso l'uso delle API REST di GitHub, il tool è in grado di Recuperare dati relativi ai commit, alle issue, ai pull request e ad altre attività che riflettono lo stato evolutivo del codice.

L'integrazione con GitHub, unitamente a Confluence, crea una sinergia tra sviluppo e documentazione, riducendo il debito di conoscenza e assicurando che tutte le informazioni critiche siano facilmente reperibili e aggiornate.

2.5 Panoramica degli strumenti AWS: SageMaker Studio e altri servizi correlati

Nel contesto di questa tesi, l'utilizzo di strumenti avanzati per lo sviluppo e la gestione di modelli di machine learning è stato fondamentale. In particolare, Amazon SageMaker Studio [36] e JupyterLab [37] hanno svolto un ruolo chiave nell'implementazione e nell'automazione delle pipeline ETL.

Amazon SageMaker Studio

Amazon SageMaker Studio è un ambiente di sviluppo integrato (IDE) basato sul web, progettato per fornire un'esperienza unificata in tutte le fasi del machine learning. Offre una singola interfaccia visiva attraverso la quale è possibile eseguire operazioni di preparazione dei dati, costruzione, addestramento, debug, distribuzione e monitoraggio dei modelli di machine learning. Questa unificazione facilita la collaborazione tra i team e accelera il ciclo di vita dello sviluppo dei modelli. Le principali funzionalità di SageMaker Studio includono:

- **Notebook Jupyter integrati:** Permettono di scrivere e eseguire codice in un ambiente interattivo, facilitando l'esplorazione dei dati e la prototipazione rapida dei modelli.
- **Gestione dei cluster EMR:** Consente di creare, sfogliare e connettersi a cluster Amazon EMR direttamente dall'interfaccia di SageMaker Studio, semplificando l'elaborazione di grandi volumi di dati.
- **Monitoraggio e debug:** Fornisce strumenti come Spark UI per monitorare e debugare i job Spark, migliorando la trasparenza e la gestione dei processi di elaborazione dati.

JupyterLab

JupyterLab è un ambiente di sviluppo interattivo che consente di lavorare con notebook Jupyter, codice e dati in un'unica interfaccia. Su AWS, JupyterLab può essere utilizzato attraverso diversi servizi, tra cui SageMaker Studio e SageMaker Studio Lab. Quest'ultimo offre un'esperienza JupyterLab gratuita con storage persistente e risorse computazionali, ideale per l'apprendimento e la sperimentazione nel campo del machine learning. L'adozione di JupyterLab su AWS ha contribuito a creare un ambiente flessibile e collaborativo per lo sviluppo e la gestione delle pipeline ETL, integrandosi perfettamente con gli altri strumenti e servizi utilizzati nel progetto.

Parte III

Specifiche

Capitolo 3

Specifiche

3.1 Definizione dei casi d'uso

L'automazione della documentazione rappresenta una soluzione strategica al problema diffuso della documentazione tecnica incompleta o non aggiornata. Il tool di Automatic Documentation Drafting nasce proprio dalla necessità di trasformare questo processo tradizionalmente lungo e soggettivo in uno più efficiente, standardizzato e meno dipendente dalla diligenza dei singoli sviluppatori. Attraverso un'analisi approfondita delle esigenze degli utenti e delle criticità nei flussi di lavoro moderni, sono stati identificati diversi scenari applicativi che il sistema è stato progettato per supportare.

Generazione automatica della documentazione tecnica

In un contesto aziendale dove le pipeline ETL rappresentano componenti critiche dell'infrastruttura dati, mantenere una documentazione aggiornata e comprensibile diventa essenziale. Gli sviluppatori spesso si trovano a dover scegliere tra dedicare tempo alla scrittura di documentazione dettagliata o concentrarsi sullo sviluppo di nuove funzionalità, creando un inevitabile compromesso. Il tool descritto in questa tesi interviene proprio in questo scenario, automatizzando l'estrazione di informazioni significative dal codice sorgente delle pipeline. L'utilizzo di modelli di intelligenza artificiale permette di produrre descrizioni tecniche che non si limitano a elencare le componenti, ma che ne spiegano anche il funzionamento logico e le relazioni reciproche. Un utente può quindi concentrarsi sulla qualità del codice, mentre il sistema si occupa di generare automaticamente documentazione che descrive la struttura del flusso di elaborazione, gli operatori utilizzati, le dipendenze tra

i task e la logica implementata. Questo approccio non solo risparmia tempo prezioso, ma garantisce anche una standardizzazione nella qualità e nello stile della documentazione prodotta.

Analisi di codice da repository distribuiti

Le moderne architetture software raramente esistono come entità isolate. Più frequentemente, il codice è distribuito attraverso repository Git, con diversi team che contribuiscono a componenti differenti del sistema. Comprendere come questi componenti interagiscono diventa fondamentale per documentare correttamente il funzionamento complessivo. Il tool è stato concepito per navigare efficacemente attraverso questa complessità. Partendo da un URL di repository, il sistema è in grado di clonare il codice, identificare automaticamente le pipeline presenti e mappare le relazioni tra i diversi file e moduli. Questa capacità di "comprendere" l'architettura complessiva del sistema permette di produrre documentazione contestualizzata, che non si limita a descrivere il singolo file in isolamento, ma lo colloca all'interno dell'ecosistema di cui fa parte. Per un team di data engineering distribuito geograficamente, questo significa avere finalmente una visione unificata e coerente dell'infrastruttura dati, accessibile a tutti i membri indipendentemente dalla loro familiarità con specifiche parti del sistema.

Documentazione contestualizzata delle dipendenze

Un aspetto particolarmente critico nella documentazione delle pipeline ETL riguarda la comprensione delle dipendenze. Quando un data engineer o un analista si approccia per la prima volta a una pipeline esistente, comprendere non solo il flusso principale ma anche i moduli importati, le librerie utilizzate e le loro interazioni reciproche può rappresentare una sfida significativa. Il sistema affronta questa complessità attraverso un'analisi stratificata che parte dal riconoscimento degli import e si estende fino alla documentazione contestuale dei moduli esterni. Per ogni pipeline analizzata, vengono identificati non solo i componenti diretti, ma anche le dipendenze indirette, creando una mappa completa delle relazioni. Questo approccio risulta particolarmente prezioso in contesti dove le pipeline possono fare riferimento a librerie proprietarie o moduli condivisi tra diversi team. La documentazione generata diventa così un punto di ingresso privilegiato per comprendere il funzionamento complessivo del sistema, facilitando l'onboarding di nuovi membri del team e la manutenzione del codice esistente.

Pubblicazione su piattaforme collaborative

La documentazione, per quanto accurata, ha valore limitato se non è facilmente accessibile al team. Nell'ecosistema moderno delle organizzazioni data-driven, piattaforme come Confluence rappresentano hub centrali per la condivisione di conoscenza tecnica e la collaborazione. Il tool è stato progettato con questa realtà in mente, implementando un flusso di lavoro che culmina nella pubblicazione automatica della documentazione generata su pagine Confluence strutturate. Questo processo non si limita a un semplice trasferimento di contenuto, ma include la formattazione appropriata, l'organizzazione logica delle informazioni e l'integrazione con la struttura esistente dello spazio documentale. Per un team di data engineering, questo significa avere sempre a disposizione documentazione aggiornata, accessibile attraverso l'interfaccia familiare di Confluence, con la possibilità di arricchirla con commenti, modifiche manuali e integrazioni con altri documenti tecnici o di business.

Validazione e miglioramento continuo della qualità documentale

Un aspetto innovativo dell' approccio riguarda l'attenzione alla qualità della documentazione prodotta. Contrariamente ai sistemi tradizionali che generano output senza meccanismi di controllo, il tool implementa un ciclo di feedback interno che valuta la coerenza, la completezza e l'accuratezza delle descrizioni generate. Questo processo di validazione confronta le diverse rappresentazioni prodotte nelle varie fasi dell'elaborazione, identificando potenziali incongruenze o aree che necessitano di approfondimento. Il sistema è inoltre dotato di capacità diagnostiche che permettono di salvare risultati intermedi per analisi successive e perfezionamenti. Per gli utenti, questo si traduce in una maggiore affidabilità della documentazione generata, con la certezza che le informazioni presentate siano state sottoposte a processi di verifica automatizzati. Questa caratteristica risulta particolarmente preziosa in contesti regolamentati, dove l'accuratezza della documentazione tecnica può avere implicazioni di compliance.

3.2 Scelta delle tecnologie e motivazioni progettuali

La progettazione di un sistema di documentazione automatica per pipeline ETL richiede un'attenta valutazione delle tecnologie disponibili, considerando non solo le capacità tecniche ma anche la sostenibilità a lungo termine e l'integrazione con l'ecosistema esistente. Le scelte implementative adottate in questo progetto riflettono un'analisi approfondita delle esigenze specifiche del dominio e delle best practice emergenti nel campo dell'automazione documentale.

Python

La decisione di utilizzare Python come linguaggio principale per l'implementazione del tool si basa su diverse considerazioni strategiche. In primo luogo, Python rappresenta lo standard de facto nell'ambito del data engineering, con una presenza particolarmente forte nell'ecosistema Apache Airflow. Questa affinità naturale consente una migliore comprensione del codice delle pipeline analizzate, sfruttando le stesse strutture sintattiche e paradigmi.

Framework e strumenti specializzati

L'adozione di GitLoader dalla suite LangChain Community rappresenta una scelta significativa, che va oltre la semplice convenienza implementativa. Questo componente non solo facilita l'interazione con repository Git, ma si integra naturalmente nel più ampio ecosistema LangChain, offrendo potenziali percorsi di evoluzione futura verso applicazioni sempre più sofisticate. La capacità di GitLoader di gestire efficacemente repository di grandi dimensioni, mantenendo al contempo una struttura dati coerente per l'analisi successiva, ha rappresentato un fattore determinante nella sua selezione. L'astrazione fornita da questo componente permette inoltre di separare nettamente la logica di acquisizione del codice sorgente dall'elaborazione semantica, facilitando eventuali sostituzioni future con soluzioni alternative.

Claude 3.5 Sonnet come motore di generazione

La selezione di Claude 3.5 Sonnet come modello di intelligenza artificiale generativa merita un'attenzione particolare, essendo uno degli elementi distintivi del progetto. Questa scelta si basa su diverse caratteristiche chiave:

- **Comprensione contestuale avanzata:** A differenza di altri modelli, Claude 3.5 Sonnet dimostra una notevole capacità di comprendere relazioni complesse tra componenti software, particolarmente rilevante nell'analisi di pipeline ETL dove le interazioni tra moduli risultano spesso intricate.
- **Bilanciamento tra prestazioni e costi:** L'architettura del modello offre un compromesso ottimale tra potenza di elaborazione e requisiti computazionali, consentendo l'analisi di repository complessi mantenendo tempi di risposta accettabili.
- **Efficacia nel prompt engineering:** La reattività del modello alle istruzioni strutturate ha permesso l'implementazione di un sistema sofisticato di prompt engineering, fondamentale per ottenere documentazione coerente e di alta qualità.
- **Capacità di sintesi e strutturazione:** La generazione di documentazione tecnica richiede non solo precisione nei dettagli, ma anche abilità nella strutturazione logica delle informazioni, area in cui Claude 3.5 Sonnet ha dimostrato performance particolarmente elevate.

Architettura modulare

Un principio fondamentale che ha guidato la progettazione del sistema è la modularità. L'intero flusso di elaborazione è stato strutturato in fasi distinte e ben definite, ciascuna con responsabilità specifiche:

- **Setup iniziale e configurazione:** Isolamento della logica di inizializzazione, facilitando la personalizzazione dei parametri di esecuzione.
- **Analisi delle importazioni:** Identificazione delle dipendenze e mappatura delle relazioni tra componenti.
- **Generazione di sommari e descrizioni:** Elaborazione semantica del codice sorgente attraverso modelli di GenAI.
- **Validazione dei risultati:** Meccanismi interni di controllo qualità per garantire coerenza e accuratezza.
- **Strutturazione e pubblicazione della documentazione:** Trasformazione delle analisi in documentazione formattata e pubblicazione sulle piattaforme target.

Questa architettura a pipeline non solo migliora la manutenibilità del codice, ma consente anche l'evoluzione incrementale del sistema. Nuove funzionalità o miglioramenti possono essere implementati in moduli specifici senza impattare l'intero flusso di lavoro, riducendo i rischi associati alle modifiche e facilitando il contributo da parte di diversi membri del team.

Integrazione con piattaforme collaborative

La scelta di integrare il sistema con Confluence risponde a una necessità fondamentale: la documentazione, per quanto accurata, ha valore limitato se non è facilmente accessibile e aggiornabile. L'implementazione di connettori dedicati consente la pubblicazione automatizzata dei risultati in spazi strutturati, facilitando la consultazione e la collaborazione.

Motivazioni progettuali strategiche

Le scelte tecnologiche descritte sono state guidate da obiettivi strategici di più ampio respiro, che trascendono le pure considerazioni implementative.

Riduzione del debito tecnico documentale

In contesti organizzativi complessi, la documentazione inadeguata rappresenta una forma particolarmente insidiosa di debito tecnico. L'automazione del processo documentale non è quindi solo una questione di efficienza operativa, ma una strategia deliberata per mitigare rischi associati alla perdita di conoscenza, specialmente in scenari di elevato turnover o rapida evoluzione delle architetture.

Standardizzazione in ambienti eterogenei

La realtà dei moderni team di sviluppo è caratterizzata da diversità di background, preferenze stilistiche e approcci documentali. L'implementazione di un sistema automatizzato introduce un elemento di standardizzazione senza imporre rigidi vincoli procedurali, facilitando la convergenza verso pratiche documentali coerenti e condivise.

Valorizzazione delle tecnologie emergenti

L'adozione di modelli di GenAI avanzati come Claude 3.5 Sonnet rappresenta una scelta consapevole di posizionamento all'avanguardia tecnologica.

Oltre ai benefici immediati in termini di qualità della documentazione generata, questa strategia consente di costruire competenze organizzative in ambiti emergenti come il prompt engineering e l'integrazione di sistemi AI, creando un vantaggio competitivo sostenibile nel tempo.

Sostenibilità del processo documentale

Una considerazione fondamentale nella progettazione del sistema riguarda la sostenibilità a lungo termine. Anziché concepire la documentazione come un'attività una tantum, l'automazione trasforma il processo in un flusso continuo, sincronizzato con l'evoluzione del codice. Questo cambio di paradigma consente di mantenere documentazione aggiornata con un overhead minimo, rendendo finalmente realizzabile l'ideale di "documentazione sempre attuale".

Considerazioni su scalabilità e prestazioni

La progettazione del sistema ha tenuto conto dei requisiti di scalabilità, considerando scenari di utilizzo che spaziano da repository di piccole dimensioni fino a complesse architetture distribuite:

- **Elaborazione incrementale:** L'implementazione supporta l'analisi selettiva di componenti modificati, evitando la rigenerazione completa della documentazione ad ogni esecuzione.
- **Parallelizzazione:** La struttura modulare consente di distribuire l'elaborazione su risorse computazionali multiple, particolarmente rilevante quando si analizzano repository di grandi dimensioni.
- **Gestione dei token:** L'interazione con modelli AI è ottimizzata per bilanciare qualità dei risultati e consumo di risorse, implementando strategie di chunking intelligente e riutilizzo contestuale.

Queste considerazioni garantiscono che il sistema rimanga performante anche in scenari di utilizzo intensivo, mantenendo tempi di risposta ragionevoli e costi operativi prevedibili.

3.3 Architettura del sistema e flusso di dati

L' **Automatic Documentation Drafting tool** costituisce un sistema innovativo che integra tecniche di analisi del codice con Generative AI per

produrre documentazione a partire dalle pipeline ETL. L'architettura è stata progettata seguendo un approccio modulare e scalabile, in modo da affrontare la complessità dell'elaborazione del codice sorgente, la creazione di descrizioni semantiche e la pubblicazione dei risultati in forma di documentazione tecnica.

Architettura ad alto livello

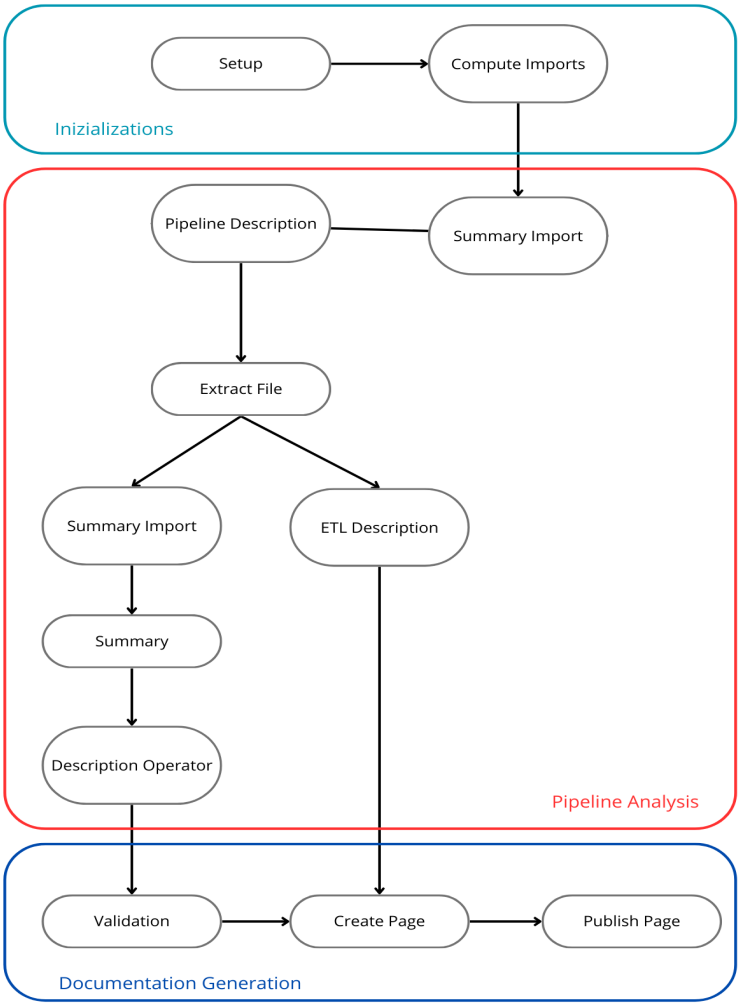

Figura 3.1. Architettura Automatic Documentation Drafting

L'architettura si compone di tre macro-aree, ciascuna dedicata a una fase chiave del processo:

Initialization

In questa fase iniziale vengono eseguite tutte le operazioni di configurazione e preparazione dell'ambiente. Il sistema clona il repository che contiene il codice delle pipeline ETL. Queste operazioni preliminari garantiscono che l'ambiente di lavoro sia correttamente impostato e che i file sorgente siano disponibili per l'analisi.

Pipeline Analysis

Costituisce il nucleo centrale del sistema. Qui avviene l'analisi dettagliata dei file sorgente, finalizzata a estrarre le informazioni semantiche relative alle pipeline (come i DAG, gli operator, le dipendenze e le relazioni tra file). Questa area include diverse operazioni che si susseguono in modo incrementale: generazione di sommari, descrizione approfondita della pipeline, estrazione dei file associati, analisi degli operator e creazione di una rappresentazione semantica ricca e coerente.

Documentation Generation

La fase conclusiva si occupa di validare e organizzare le informazioni raccolte, trasformandole in un documento strutturato pronto per la pubblicazione su Confluence. Vengono applicati template predefiniti e si gestisce la creazione o l'aggiornamento delle pagine di documentazione, garantendo che il risultato finale sia fruibile e di alta qualità.

Componenti principali del sistema

Initialization Area

- **Setup:** Inizializza il sistema, interpretando i parametri di configurazione e predisponendo una cartella di lavoro temporanea. Si occupa di clonare il repository target mediante il GitLoader e di identificare, in modo preliminare, la presenza di pipeline ETL nel codice.
- **Compute Imports:** Analizza le dipendenze tra i vari file, costruendo una mappa delle relazioni tra i moduli importati e le eventuali librerie

personalizzate. Questa rappresentazione è essenziale per comprendere come i file interagiscono tra loro e per supportare l'analisi semantica successiva.

Pipeline Analysis Area

- **Summary Import:** Avvia la creazione di summary sintetici per i file individuati come importati rispetto a dove è definita la pipeline, fornendo una prima panoramica delle funzionalità presenti nel codice. Ciò consente di orientare le fasi successive e di preparare il terreno per una descrizione più approfondita.

- **Pipeline Description:** Esamina la struttura della pipeline, focalizzandosi sul DAG e sui task definiti all'interno del codice. Vengono identificate le relazioni e le dipendenze, nonché la logica di esecuzione della pipeline ETL.

- **Extract Files:** Amplia la portata dell'analisi, includendo i file correlati alla pipeline principale, così da ottenere una visione completa delle trasformazioni ETL.

- **Summary Import e Summary:** Creano descrizioni dettagliate dei moduli e degli import che arricchiscono la pipeline, generando sommari contestualizzati che migliorano la qualità della documentazione.

- **Description Operator:** Analizza gli operator specifici (ad esempio, PythonOperator, BashOperator, ecc.) e ne descrive il funzionamento, le configurazioni e il ruolo all'interno della pipeline. Questo passaggio è cruciale per documentare gli aspetti più operativi del flusso di lavoro.

- **ETL Description:** Fornisce una sezione dedicata alle trasformazioni dei dati, descrivendo le logiche di ETL e le sequenze di elaborazione. Questa parte è essenziale per evidenziare come i dati vengano manipolati lungo il flusso ETL.

Documentation Generation Area

- **Validation:** Garantisce la correttezza e la coerenza delle descrizioni generate, confrontando i risultati di diverse fasi e rilevando eventuali incongruenze o omissioni.
- **Create Page:** Converte le informazioni raccolte in un documento strutturato, applicando template predefiniti che assicurano consistenza stilistica e chiarezza espositiva.
- **Publish Page:** Aggiorna o crea nuove pagine su Confluence, completando l'iter di documentazione. Durante questa fase, il sistema gestisce i metadati e i riferimenti incrociati, garantendo la continuità con la documentazione preesistente.

Flusso di dati e interazioni

Il file `genai_autodoc.py` funge da orchestratore, coordinando i vari step e monitorandone l'avanzamento. Il flusso di esecuzione, definito dalla sequenza di chiamate alle funzioni (es. `setup`, `summary_imported_pipeline_file`, `description_pipeline1`, ecc.), procede come segue:

1. **Cloning e setup:** Clona il repository tramite `GitLoader`, crea una directory temporanea e individua le pipeline da analizzare.
2. **Analisi preliminare:** Effettua il calcolo degli import, creando una mappa delle dipendenze.
3. **Generazione e raffinamento delle descrizioni:** Genera sommari e descrizioni dettagliate della pipeline, estendendo l'analisi ai file correlati e arricchendo progressivamente la rappresentazione semantica.
4. **Validazione e creazione della documentazione:** Confronta i risultati e, se coerenti, produce la documentazione finale, organizzata in base a template e pubblicata su Confluence.
5. **Gestione degli errori e logging:** Ogni fase implementa la gestione delle eccezioni, mentre la libreria `tqdm` fornisce feedback continuo sull'avanzamento.

Meccanismi di controllo

Per garantire la robustezza e la qualità del processo, il sistema implementa diversi livelli di controllo:

- **Feedback in tempo reale:** La barra di avanzamento e i messaggi di log offrono una panoramica immediata sullo stato dell'elaborazione.
- **Validazione incrementale:** La fase di validazione confronta le descrizioni generate nei diversi step, evidenziando eventuali discrepanze e prevenendo l'inclusione di informazioni errate.
- **Gestione delle eccezioni:** I blocchi try/except evitano arresti imprevisti, permettendo al sistema di segnalare gli errori e proseguire con le pipeline rimanenti.
- **Logging e tracciabilità:** I risultati di ogni step vengono salvati in strutture dati (come `partial_result`), consentendo un'analisi post-processo e la generazione di report diagnostici (in caso di debug attivo).

Considerazioni tecniche

La progettazione dell'architettura tiene conto di:

- **Integrazione con modelli GenAI:** Le funzioni di analisi e generazione di descrizioni si avvalgono di modelli di Generative AI, resi più efficaci da tecniche di prompt engineering.
- **Scalabilità:** L'approccio modulare e la possibilità di gestire repository di diverse dimensioni assicurano un buon comportamento del sistema anche in contesti di produzione.
- **Estendibilità:** La suddivisione in step indipendenti favorisce l'aggiunta di nuove funzionalità (es. ulteriori check, diversi formati di output, ecc.) senza modificare radicalmente l'architettura di base.
- **Qualità del risultato:** La fase di validazione, unita al controllo manuale opzionale, garantisce che la documentazione generata sia coerente, utile e di elevata qualità.

L'architettura dell'Automatic Documentation Drafting tool combina un'organizzazione modulare, un flusso di elaborazione chiaro e un insieme di meccanismi di controllo che assicurano la produzione di documentazione accurata e aggiornata. La sinergia tra la Pipeline Analysis e la Documentation Generation, orchestrata dal main, permette di mantenere un equilibrio tra complessità e manutenibilità, soddisfacendo le esigenze di team che necessitano di un processo di documentazione ETL efficiente e scalabile.

Parte IV

Implementazione

Capitolo 4

Metodologia e Implementazione

4.1 Approccio di prompt engineering: definizione e ottimizzazione dei prompt

L'approccio di prompt engineering adottato nel tool rappresenta il punto d'incontro tra le intenzioni umane e le capacità generative del modello Claude. In sostanza, il prompt engineering consiste nel progettare, definire e affinare in maniera iterativa le istruzioni testuali o prompt che guidano il modello AI nella generazione della documentazione a partire dal codice delle pipeline ETL. Nel sistema, l'intero processo è strutturato in moduli che sfruttano funzioni dedicate a ciascuna fase della generazione dei prompt. All'inizio, il modulo di setup (funzione setup in steps.py) inizializza una struttura dati predefinita, in cui vengono memorizzati sia gli output "non elaborati" (ovvero, i risultati grezzi restituiti dal modello AI) sia quelli "parsing" (ovvero, i dati successivamente trasformati in una forma strutturata e utilizzabile). Tale struttura funge da base per l'intero processo di prompt engineering, consentendo di mantenere una traccia delle varie iterazioni e facilitando il confronto tra output generati in momenti differenti.

```
{
   "prompt_pipeline_description": {
         "description_custom_imports_pipeline": {
             "unparsed_result": [],
             "parsed_result": [],
          },

          "description_pipeline": {
             "unparsed_result": [],
             "parsed_result": [],
          },

    },
   "prompt_page_structure": {

    },

   "prompt_etl_description": {

    },

}
```

Figura 4.1. Definizione della struttura dati

Per ogni fase del processo, ad esempio durante la creazione delle summary dei file importati o nella descrizione iniziale della pipeline, vengono generati prompt specifici tramite funzioni dedicate (come get_prompt_summary, get_prompt_task_id e get_prompt_pipeline). Questi prompt sono formulati in maniera tale da includere informazioni contestuali rilevanti, consentendo al modello di produrre output accurati e coerenti con le specifiche esigenze del dominio ETL. I risultati, ottenuti tramite la funzione send_prompt, vengono poi sottoposti a parsing (ad es. con parse_xml o parse_xml_pipeline) e integrati nella struttura dati di supporto.

```
return f"""
    ...
<instructions>
    Analyze the input and generate the ETL operations description step by
        step, focusing on this specific items:

     - Detect and list each database table referenced in the operations:
         a. Input tables.
         b. Output tables.
     - For each output table, provide a detailed description including:

        a. Any join applied, search in the code including:
          1. Detail the types of joins used on the table:
            1.1. Specify the type of join (e.g., INNER, OUTER, LEFT, RIGHT).
               1.2. Specify the input tables.
               1.3. Specify the keys and the conditions.
               1.4. Specify the output table.
       b. Any transformations applied to the table, detailed in the code
          including:

           2. Explain aggregations performed:
               2.1. Specify the type of aggregation functions used (e.g., SUM
                   , AVG, COUNT).
               2.2. Specify the input table.
               2.3. Specify the fields involved.
               2.4. Specify any alias.
               2.5. Specify any group.

           3. Describe filters and conditions used on the table:
               3.1. Specify the input table.
               3.2. Specify the condition.
               3.3. Specify the fields involved.

       c. Key fields with a description of their significance, as determined
           from the code context.

       Ensure all explanations are strictly derived from the provided input,
           avoiding assumptions about external context or functionalities
       not explicitly described.

</instructions>
      ...
"""
```

Figura 4.2. Istruzioni di un prompt per le descrizioni ETL

L'ottimizzazione dei prompt si realizza attraverso un approccio iterativo se il file da descrivere non è stato già descritto in precedenza il sistema genera nuovi prompt e ne conserva il risultato; in alternativa, viene utilizzata la documentazione precedentemente salvata. Questo meccanismo permette di affinare progressivamente i prompt, basandosi sui feedback derivanti dalla validazione dei risultati (tramite la funzione validation) e sul confronto tra descrizioni iniziali e finali. L'architettura del tool prevede inoltre il salvataggio dei prompt in file JSON, in modo da poter effettuare revisioni e

miglioramenti nel tempo senza perdere il contesto delle iterazioni passate. In sintesi, l'approccio di prompt engineering si fonda su:

- **Definizione modulare dei prompt:** Ogni step del processo (dall'estrazione delle importazioni alla generazione della descrizione finale) prevede la creazione di prompt specifici, progettati per guidare il modello AI in maniera precisa.

- **Ottimizzazione iterativa:** I prompt vengono continuamente affinati, utilizzando feedback e validazioni per migliorare la qualità e l'allineamento degli output con gli obiettivi tecnici e semantici richiesti.

- **Gestione e tracciabilità:** La conservazione dei risultati, sia grezzi che processati, consente una gestione trasparente del processo, fondamentale per garantire la ripetibilità e l'affidabilità della documentazione generata.

Questo approccio integrato non solo migliora la qualità della documentazione automatica, ma contribuisce anche a ridurre il debito di conoscenza, assicurando che le informazioni estratte dal codice siano sempre aggiornate, coerenti e facilmente comprensibili dagli sviluppatori e dagli stakeholder.

4.2 Pipeline di estrazione delle informazioni dal codice

La pipeline di estrazione delle informazioni dal codice rappresenta il primo, fondamentale stadio del tool, in cui viene acquisito e analizzato il codice sorgente delle pipeline ETL. Questo processo, alla base della generazione automatica della documentazione, si sviluppa in una serie di passaggi coordinati che consentono di trasformare il codice in dati strutturati e semanticamente arricchiti, pronti per essere ulteriormente elaborati.

Il percorso inizia con il **cloning del repository**. Utilizzando una funzione dedicata che si avvale di GitLoader, il tool scarica l'intero set di file in una directory temporanea. Questa operazione è cruciale per ottenere una copia locale del repository, garantendo così un ambiente isolato e controllato in cui eseguire le successive analisi. Successivamente, grazie alla funzione find_pipelines, il sistema individua in modo automatico le pipeline presenti all'interno del codice. Una volta individuati i file di interesse, la funzione

extract_content_from_file_name si occupa di estrarre il contenuto specifico di ciascun file, fornendo il testo grezzo che sarà poi sottoposto a ulteriori operazioni di elaborazione.

Un aspetto critico del processo riguarda **l'analisi delle dipendenze**: la funzione take_custom_imports esamina il codice per identificare e mappare le importazioni personalizzate, restituendo una rappresentazione strutturata delle relazioni tra i vari moduli.

Questo passaggio è essenziale perché consente di comprendere come i diversi componenti interagiscono tra loro e come le dipendenze influiscono sul funzionamento complessivo della pipeline. Successivamente, il tool procede alla **generazione dei sommari sintetici** per ciascun file importato, utilizzando la funzione summary_imported_pipeline_file. Questo step produce una prima rappresentazione semantica, che agisce come base per le fasi successive di descrizione e analisi del codice. I sommari offrono una visione d'insieme degli elementi chiave presenti nei file e facilitano la successiva elaborazione delle informazioni. Questi saranno poi usati insieme all'input per le descrizioni dei file che li importano. Una volta generata la knowledge base della pipeline Airflow con i sommari dei file che questa importa, viene generata una prima descrizione: description_pipeline1. La prima descrizione degli operatori è essenziale per estrarre informazioni di base riferite all'intera pipeline come ad esempio: i file e funizoni che un operatore usa, le strutture dati che vengono usare ed altro ancora.

```
<output>

   <operator>
       <operator_task_id>begin_execution</operator_task_id>

        <operator_type>EmptyOperator</operator_type>
   </operator>

   <operator>
       <operator_task_id>  upload_local_raw_data_to_s3 </operator_task_id>

       <operator_type> PythonOperator </operator_type>

       <data_structure>["countryinfo.txt","shapes_simplified_low.json"]
       </data_structure>
       <function>upload_file_to_s3</function>
       <file_name> build_emr_task.py</file_name>
   </operator>
  ...
</output>
```

Figura 4.3. Risultati unparsed

```
{
                {
                    "operator_task_id": "begin_execution",
                    "operator_type": "EmptyOperator"
                },
                {
                    "operator_task_id": "upload_local_raw_data_to_s3",
                    "operator_type": "PythonOperator",
                    "file_names": [
                        "countryinfo.txt",
                        "shapes_simplified_low.json"
                    ],
                    "data_structure": [
                        "countryinfo.txt",
                        "shapes_simplified_low.json"
                    ],
                    "functions": [
                        "upload_file_to_s3"
                    ]
                },

}
```

Figura 4.4. Risultati parsed

Il prossimo passo consiste nell'estrarre i file relativi ad ogni operatore e ripetere il procedimento, quindi vengono presi gli import, descritti e usati come input per la descrizione di questi file.

L'intero flusso, che si articola in 13 step ben definiti, viene eseguito in sequenza e monitorato attraverso una barra di avanzamento (implementata con la libreria tqdm). Questo meccanismo di monitoraggio garantisce non solo la trasparenza del processo, ma anche la tracciabilità delle operazioni, facilitando il debug e la validazione dei risultati. Ogni step contribuisce a raffinare progressivamente il livello di dettaglio e la coerenza dell'output, rendendo il processo complessivo robusto e affidabile.

La pipeline di estrazione delle informazioni dal codice non solo consente di acquisire e analizzare in maniera sistematica le pipeline ETL, ma rappresenta anche il fondamento su cui poggia l'intero sistema di documentazione automatica. Attraverso una serie di operazioni coordinate, il tool trasforma il codice sorgente in un insieme di dati strutturati e semanticamente arricchiti, che saranno poi utilizzati per generare una documentazione tecnica accurata e aggiornata.

4.3 Generazione automatica della documentazione

La generazione automatica della documentazione rappresenta il nucleo del sistema, dove le informazioni estratte dal codice vengono trasformate in descrizioni tecniche dettagliate. Questo processo, basato sull'applicazione del prompt engineering, guida il modello AI nella produzione di output coerenti e allineati agli standard richiesti, permettendo così di ridurre al minimo l'intervento manuale e di garantire una documentazione sempre aggiornata in relazione alle modifiche apportate al codice.

Il processo inizia con la generazione di una prima descrizione della pipeline, realizzata tramite la funzione description_pipeline1. In questa fase, il tool analizza il codice sorgente insieme ai sommari precedentemente prodotti, estraendo le informazioni rilevanti e sintetizzandole in una descrizione iniziale che fornisce un quadro generale del funzionamento della pipeline. Questa descrizione, sebbene preliminare, è fondamentale per impostare la base semantica su cui si svilupperanno le fasi successive.

Successivamente, il sistema impiega ulteriori funzioni in particolare description_pipeline2 e parsing_description ed etl_description per rafforzare

e perfezionare il contenuto generato. Queste funzioni si concentrano sull'approfondimento dell'analisi, integrando dettagli provenienti dalla valutazione degli operatori utilizzati nella pipeline e dall'analisi dei file correlati.

Il risultato di questo iter di elaborazione è una documentazione che non solo riporta le specifiche tecniche, ma evidenzia anche il ragionamento sottostante alle scelte implementative, contribuendo a ridurre il debito di conoscenza all'interno del team.

La documentazione, è data dalle due sezioni: una relativa alla pipeline Airflow con la descrizione approfondita degli operatori e una seconda sezione con la descrizione delle operazioni ETL.

Il contenuto finale, frutto di un processo iterativo di generazione e raffinamento, viene poi sottoposto a una rigorosa fase di validazione. Questa fase di validazione verifica la coerenza e la qualità delle descrizioni generate, assicurando che l'output sia semanticamente ricco e strutturato secondo gli standard tecnici richiesti. Grazie a questa metodologia automatizzata, il sistema riesce a mantenere la documentazione in linea con l'evoluzione del codice, consentendo aggiornamenti dinamici e una maggiore efficienza nella gestione delle informazioni tecniche.

file_name	type	tables	keys	table_out
cleaning_inventor_job.py	INNER	{inventor, inventor_cleaned_with_id}	{inventor.name, inventor_cleaned_with_id.name}	{inventor_cleaned_with_id}
wipo_classifications_etl_job.py	INNER	{wipo_data, wipo_field_data}	{wipo_data.field_id, wipo_field_data.id}	{transformed_data}

Tabella 4.1. Descrizione delle operazioni di join.

```
wipo_classifications_etl_job.py

    1. Database Tables Referenced:
        - Input tables:
        - wipo.parquet (Source: S3)
        - wipo_field.parquet (Source: S3)
        - Output table:
        - wipo_classifications (Source: Redshift)

    2. ETL:
        a. Join Transformation:
            i. Join:
            - Tables: wipo_data, wipo_field _data
            - Type: INNER
            - Keys: wipo_data.field_id, wipo_field _data.id
            - Output: transformed_data

        b. Transformation:
            i. Column Renaming:
                - Table: transformed_data
                - Columns renamed:
                - 'patent_id' to 'wipo_classification_id'
                - 'sector_title' to 'sector'
                - 'field_title' to 'field'

        c. Key Fields and Their Significance:
            - 'wipo_classification_id': Unique identifier for WIPO
                classifications, derived from the patent_id field.
            - 'sector': Represents the sector title of the WIPO
                classification.
            - 'field': Represents the field title of the WIPO classification.
            - 'field _id': Used to join wipo_data with wipo_field _data,
                representing the relationship between patents and their
                fields.
```

Figura 4.5. Risultati descrizione ETL

4.4 Integrazione con AWS SageMaker Studio e Confluence

L'integrazione con AWS SageMaker Studio e Confluence è un elemento chiave per il successo del tool, poiché unisce la potenza computazionale e la gestione collaborativa della documentazione. AWS SageMaker Studio offre un ambiente interattivo basato sul cloud, che facilita lo sviluppo, il testing e il debug delle pipeline di generazione della documentazione, permettendo di sfruttare risorse scalabili e un'infrastruttura affidabile.

Parallelamente, Confluence funge da repository centralizzato per la documentazione tecnica. Il tool interagisce con Confluence tramite API REST, automatizzando la creazione e l'aggiornamento delle pagine documentali. In questo modo, ogni modifica al codice si riflette immediatamente nella documentazione, migliorando la trasparenza e la collaborazione tra i membri del team. L'integrazione sinergica tra SageMaker Studio e Confluence garantisce che l'intero flusso di lavoro, dalla generazione del documento all'aggiornamento in tempo reale, avvenga in maniera fluida e automatizzata.

4.5 Descrizione CI/CD per l'implementazione

L'implementazione di una pipeline CI/CD (Continuous Integration/Continuous Delivery) è essenziale per garantire la rapidità, l'affidabilità e la coerenza del processo di generazione della documentazione automatica. Il tool è progettato per integrarsi perfettamente in un contesto CI/CD, automatizzando l'intero ciclo di aggiornamento e pubblicazione della documentazione tecnica.

In questo sistema, ogni esecuzione del tool avviene all'interno di un container Docker configurato appositamente. Ad ogni avvio, il container garantisce un ambiente isolato e ripetibile, predisponendo tutte le dipendenze necessarie per il corretto funzionamento del tool. Le configurazioni, inclusi i parametri di esecuzione e le credenziali di sessione per Confluence, vengono passate al container come variabili d'ambiente o parametri, assicurando una connessione sicura e automatizzata al sistema di pubblicazione. Il flusso CI/CD si articola in diverse fasi automatizzate:

- **Integrazione continua:** Ogni commit o modifica nel repository attiva automaticamente il processo di analisi del codice. Il sistema, infatti, è configurato in modo che ogni aggiornamento venga elaborato, trasformato e tradotto in documentazione senza necessità di intervento manuale.

- **Test e validazione:** Prima della pubblicazione, il tool esegue controlli rigorosi per verificare la coerenza e la qualità dell'output. Funzioni dedicate, come validation, confrontano la documentazione generata con le specifiche tecniche e semantiche richieste, garantendo che l'output sia accurato.

- **Deployment automatizzato:** Il modulo publish_page gestisce la pubblicazione della documentazione su Confluence, utilizzando le credenziali di sessione fornite al container. In questo modo, le nuove versioni sono immediatamente accessibili agli utenti e mantengono una coerenza con le versioni precedenti.
- **Versionamento e salvataggio:** Attraverso la funzione save_prompt, i prompt e i risultati intermedi vengono salvati in file JSON, facilitando il rollback e il tracking delle modifiche nel tempo. Questo approccio consente di monitorare la cronologia degli aggiornamenti e di intervenire rapidamente in caso di anomalie.

L'esecuzione del processo all'interno di un container Docker assicura ripetibilità e isolamento, elementi fondamentali per un ambiente CI/CD robusto. Questo metodo riduce drasticamente i tempi di rilascio, minimizza il rischio di errori e garantisce che la documentazione sia sempre aggiornata e in linea con lo stato corrente del codice. L'approccio adottato favorisce un ciclo di sviluppo agile e iterativo, in cui ogni fase del processo viene monitorata e integrata continuamente, contribuendo a creare un sistema di documentazione automatica efficiente e affidabile. I principali vantaggi di questo sistema integrato includono:

- **Ripetibilità:** L'uso di un container Docker garantisce che ogni esecuzione del processo sia identica, eliminando variabili ambientali indesiderate.
- **Sicurezza:** La gestione sicura delle credenziali e delle configurazioni riduce il rischio di errori e garantisce un'implementazione conforme alle policy aziendali.
- **Scalabilità:** Il sistema è progettato per essere scalabile, consentendo di gestire aumenti nei volumi di codice e nelle richieste di aggiornamento della documentazione senza compromettere le performance.

Questo approccio CI/CD consente, dunque, di mantenere una documentazione tecnica sempre aggiornata e accurata, supportando in modo efficace le esigenze dinamiche degli ambienti di sviluppo moderni.

Parte V

Valutazione

Capitolo 5

Casi di Studio e Valutazione

5.1 Descrizione dei casi d'uso implementati

Il primo caso d'uso implementato riguarda l'applicazione del tool al repository "Patent Analytics Data Pipeline" [41]. Questo repository, scelto appositamente per la sua complessità e ricchezza di informazioni, rappresenta un ambiente ideale per testare e validare l'efficacia del tool di Automatic Documentation Drafting.

La scelta di un repository così articolato offre numerosi benefici, poiché mette alla prova il sistema su scenari reali e sfidanti, in cui la documentazione deve coprire molteplici fasi e componenti di una pipeline ETL complessa. Il repository in questione si occupa del processamento dei dati relativi ai brevetti degli Stati Uniti, coprendo un arco temporale dal 2000 al 2021, e include oltre 5 milioni di record. All'interno del repository, il progetto si articola in diverse fasi, ciascuna delle quali contribuisce a fornire una panoramica completa del flusso di dati e delle operazioni di trasformazione:

Preparazione dei Dati:

Il repository integra dati provenienti da diverse fonti, come USPTO e Geonames, che vengono centralizzati in AWS S3. In questa fase, i dati vengono puliti, normalizzati e organizzati per essere successivamente elaborati, garantendo la qualità e la coerenza dei dati grezzi.

Elaborazione e Keyword Extraction:

Tramite l'utilizzo di Apache Spark e, in particolare, di tecniche avanzate di keyword extraction, il progetto analizza titoli e abstract dei brevetti per estrarre parole chiave significative. Questa operazione non solo facilita l'analisi delle tendenze tecnologiche, ma evidenzia anche il valore aggiunto derivante dalla capacità di interpretare in maniera automatica grandi volumi di dati.

Caricamento e Analisi dei Dati:

I dati trasformati vengono infine caricati in un data warehouse (AWS Redshift), strutturato in tabelle fact e dimension, che consentono di eseguire query analitiche complesse e di alimentare dashboard interattive. Questa fase permette di ottenere insight strategici sui trend tecnologici e sulle performance di mercato.

Il tool di documentazione automatica è stato applicato su questo repository per estrarre e generare, in maniera automatica, una documentazione tecnica esaustiva. Il processo di documentazione segue una serie di passaggi ben definiti, come descritto nel file genai_autodoc.py e nelle funzioni del modulo steps.py. In particolare, il tool:

1. **Clona il repository** utilizzando GitLoader, creando un ambiente isolato per l'analisi e garantendo l'accesso a una copia locale e aggiornata del codice sorgente.

2. **Individua le pipeline ETL** grazie alla funzione find_pipelines, concentrandosi sui file che implementano la logica di trasformazione dei dati. In questo caso due, e per ognuna di queste itera il seguente procedimento.

3. **Estrae il contenuto dei file** mediante extract_content_from_file_name, fornendo il testo grezzo su cui basare ulteriori operazioni di analisi.

4. **Analizza le dipendenze** con la funzione take_custom_imports, che mappa le importazioni personalizzate, evidenziando le relazioni tra i moduli e fornendo una base solida per la generazione dei sommari.

5. **Genera sommari sintetici** attraverso summary_imported_pipeline_file, creando una rappresentazione semantica iniziale che fungerà da base per la descrizione dettagliata della pipeline.

6. **Costruisce descrizioni dettagliate** riguardo gli operatori della pipeline tramite funzioni quali description_pipeline1 e description_pipeline2,

integrando informazioni relative agli operator e ai file associati, fino ad ottenere una documentazione tecnica completa.

7. **Valida i risultati** mediante funzioni di controllo, come validation, per garantire la coerenza e l'accuratezza delle descrizioni generate.
8. **Genera descrizioni dettagliate** riguardo i processi ETL, andando a recuperare i file e le loro summary.
9. **Pubblica la documentazione** su Confluence utilizzando le API REST, assicurando che il documento sia accessibile in modo centralizzato e aggiornato in tempo reale.

L'applicazione del tool al repository "Patent Analytics Data Pipeline" ha offerto numerosi vantaggi, mettendo in luce l'efficacia del sistema in contesti reali di elevata complessità. In primo luogo, l'uso di un repository così articolato ha permesso di verificare la robustezza del tool. Un ambiente complesso, infatti, mette alla prova la capacità del sistema di gestire e documentare processi ETL articolati, dimostrando come il tool sia in grado di operare in modo affidabile anche quando si confronta con flussi di lavoro intensi e variabili.

Inoltre, la presenza di molteplici fasi operative, dalla pulizia dei dati fino all'estrazione delle parole chiave, consente una validazione trasversale delle funzionalità implementate. Ogni modulo dalla generazione dei prompt alla pubblicazione su Confluence è sottoposto a test specifici, confermando che il sistema opera in maniera coordinata e integrata. Questa capacità di verificare la funzionalità a vari livelli rafforza la fiducia nel tool e ne evidenzia l'efficacia in scenari complessi.

Un ulteriore beneficio significativo è rappresentato dalla riduzione del debito di conoscenza. In un contesto in cui il codice viene continuamente aggiornato e modificato, la capacità del tool di generare documentazione automatica consente di mantenere una conoscenza sempre aggiornata e facilmente accessibile. Ciò facilita notevolmente la comprensione e la manutenzione del sistema, soprattutto per nuovi sviluppatori o team distribuiti che devono rapidamente acquisire familiarità con il funzionamento delle pipeline ETL.

La complessità del repository offre inoltre un contesto ricco di informazioni, che si traduce in un feedback dettagliato utile per l'ottimizzazione continua del prompt engineering. Questo processo iterativo permette di affinare la qualità degli output generati, migliorando progressivamente la precisione e la completezza della documentazione tecnica. Infine, la gestione di un repository con diverse pipeline dimostra la scalabilità e l'adattabilità del tool.

Tale complessità evidenzia come il sistema possa essere applicato con successo anche in contesti di grande dimensione, garantendo tempi di elaborazione efficienti e una documentazione accurata. In questo modo, il tool non solo risponde alle esigenze di ambienti complessi, ma contribuisce anche a stabilire un modello replicabile e robusto per la documentazione automatica in scenari di larga scala.

Nel complesso, la scelta di utilizzare un repository così articolato ha permesso di evidenziare, in modo chiaro e concreto, i molteplici benefici derivanti dall'adozione del sistema, rafforzando la validità dell' approccio e offrendo un vantaggio strategico nella gestione e condivisione della conoscenza tecnica.

5.2 Analisi qualitativa e quantitativa dei risultati

Panoramica e Impostazione Metodologica

I risultati ottenuti applicando il tool di Automatic Documentation Drafting al repository "Patent Analytics Data Pipeline" sono estremamente positivi e confermano l'efficacia del sistema nella generazione automatica di documentazione tecnica di alta qualità. La valutazione è stata condotta mediante un approccio duale, combinando analisi qualitative approfondite con metriche quantitative oggettive, al fine di ottenere una comprensione completa delle prestazioni del sistema in un caso d'uso reale.

Il repository scelto per questa valutazione rappresenta un caso emblematico di pipeline ETL ben strutturata, caratterizzata da docstring dettagliati, nomi di variabili descrittivi e una chiara organizzazione del codice. Queste caratteristiche hanno fornito un contesto ideale per testare le capacità del tool in condizioni operative ottimali, permettendo di valutare il suo potenziale massimo in termini di estrazione e organizzazione delle informazioni.

Analisi Qualitativa Approfondita

Sul fronte qualitativo, l'analisi è stata condotta attraverso un processo strutturato di revisione esperta, che ha coinvolto specialisti di dominio con significativa esperienza nello sviluppo e nella documentazione di pipeline ETL. I risultati di questa valutazione hanno evidenziato molteplici elementi:

Accuratezza Semantica

Il tool ha generato descrizioni complete e coerenti che rispecchiano fedelmente la logica e la struttura delle pipeline ETL. L'analisi manuale eseguita

dagli esperti ha confermato che la documentazione automatica cattura efficacemente anche le sfumature più sottili della semantica del codice sorgente. In particolare, è stata rilevata una notevole precisione nella descrizione delle relazioni causali tra i diversi componenti della pipeline, un aspetto che richiede solitamente una comprensione profonda del contesto operativo.

Completezza Informativa

Le informazioni relative agli operatori, alle dipendenze e ai file correlati sono state elaborate in modo da fornire una visione chiara e approfondita del flusso di lavoro. La documentazione generata include tutti gli elementi essenziali per comprendere sia la struttura macroscopica della pipeline che i dettagli implementativi di ciascun componente.

Coerenza Strutturale

Un aspetto particolarmente apprezzato è stata la coerenza strutturale della documentazione generata. Il tool ha mantenuto un formato uniforme attraverso tutti i componenti documentati, facilitando la navigazione e la comparazione tra diverse sezioni. Questa standardizzazione contribuisce significativamente alla leggibilità e all'usabilità della documentazione, aspetti spesso trascurati nella documentazione manuale.

Analisi Quantitativa e Metriche Operative

Dal punto di vista quantitativo, il tool ha prodotto metriche e tabelle che illustrano la qualità dei dati estratti, fornendo una base oggettiva per la valutazione delle prestazioni. L'analisi quantitativa si è concentrata su diversi aspetti misurabili, tra cui la completezza dell'estrazione, l'accuratezza delle relazioni identificate e l'efficienza del processo.

Descrizione degli Operatori

In questa tabella vengono dettagliati gli operatori identificati nella pipeline, inclusi i task id, i tipi di operatori utilizzati e le principali configurazioni associate. L'analisi quantitativa condotta sul repository di test ha rivelato che il tool identifica con consistenza assoluta gli operatori base in tutti i tentativi di documentazione. Tuttavia, durante 15-20 cicli di test effettuati sullo stesso repository, si verificano casi in cui alcuni metadati accessori degli operatori (come i file utilizzati o specifici elementi delle strutture dati usate dall'operatore) non sono stati catturati completamente. Questo rappresenta un tasso di successo elevato nella documentazione completa degli operatori. Nonostante questa rara imprecisione nei metadati periferici, i dati estratti

forniscono un quadro affidabile e dettagliato delle componenti operative della pipeline, dimostrando l'efficacia del tool nel documentare le relazioni e le interazioni tra i diversi moduli.

operator_task_id	operator_task_type	operator_type	file_names	data_structure	functions
begin_execution	single task	EmptyOperator	NaN	NaN	NaN
upload_local_raw_data_to_s3	task group	PythonOperator	{countryinfo.txt, shapes_simplified_low.json}	{countryinfo.txt, shapes_simplified_low.json}	upload_file_to_s3
check_raw_data_exists	task group	CheckS3KeyExistOperator	{assignee.tsv, countryinfo.txt, inventor.tsv, location.tsv, patent_assignee.tsv, patent_inventor.tsv, patent.tsv, shapes_simplified_low.json, wipo_field.tsv, wipo.tsv}	{assignee.tsv, countryinfo.txt, inventor.tsv, location.tsv, patent_assignee.tsv, patent_inventor.tsv, patent.tsv, shapes_simplified_low.json, wipo_field.tsv, wipo.tsv}	NaN
check_cleaned_data_exists	task group	CheckS3KeyExistOperator	{assignee.parquet, inventor.parquet, location.parquet, patent_assignee.parquet, patent_inventor.parquet, patent.parquet, wipo_field.parquet, wipo.parquet}	{assignee.parquet, inventor.parquet, location.parquet, patent_assignee.parquet, patent_inventor.parquet, patent.parquet, wipo_field.parquet, wipo.parquet}	NaN
download_raw_patent_data_to_s3	task group	DownloadZipDataAndPutInS3Operator	{patent.tsv.zip, patent_assignee.tsv.zip, assignee.tsv.zip, location.tsv.zip, wipo.tsv.zip, wipo_field.tsv.zip, inventor.tsv.zip, patent_inventor.tsv.zip}	{patent.tsv.zip, patent_assignee.tsv.zip, assignee.tsv.zip, location.tsv.zip, wipo.tsv.zip, wipo_field.tsv.zip, inventor.tsv.zip, patent_inventor.tsv.zip}	NaN
keyword_extraction_emr_task	task group	TaskGroup	{yake_keyword_extraction_job.py, rank_keyword_job.py}	{yake_keyword_extraction_job.py, rank_keyword_job.py}	build_emr_task
cleaning_emr_task	task group	TaskGroup	{cleaning_country_shapes_job.py, cleaning_location_job.py, cleaning_patent_job.py, cleaning_wipo_job.py, cleaning_wipo_field_job.py, cleaning_assignee_job.py, cleaning_inventor_job.py, cleaning_patent_assignee_job.py, cleaning_patent_inventor_job.py}	{cleaning_country_shapes_job.py, cleaning_location_job.py, cleaning_patent_job.py, cleaning_wipo_job.py, cleaning_wipo_field_job.py, cleaning_assignee_job.py, cleaning_inventor_job.py, cleaning_patent_assignee_job.py, cleaning_patent_inventor_job.py}	build_emr_task
check_keyword_data_exists	task group	CheckS3KeyExistOperator	{patent_keyword_raw.parquet, patent_keyword.parquet}	{patent_keyword_raw.parquet, patent_keyword.parquet}	NaN
check_tables_count	task group	SQLTableCheckOperator	NaN	{patents, dates, details, wipo_classifications, owners, locations, patent_keywords}	NaN
check_patents_table_quality	single task	SQLColumnCheckOperator	NaN	{id, granted_date, detail_id, wipo_classification_id, num_claims}	NaN
etl_emr_task	task group	TaskGroup	{dates_etl_job.py, details_etl_job.py, wipo_classifications_etl_job.py, intermediary_patent_etl_job.py, locations_etl_job.py, owners_etl_job.py, patents_etl_job.py, patent_keywords_etl_job.py}	{dates_etl_job.py, details_etl_job.py, wipo_classifications_etl_job.py, intermediary_patent_etl_job.py, locations_etl_job.py, owners_etl_job.py, patents_etl_job.py, patent_keywords_etl_job.py}	build_emr_task
check_dates_table_quality	single task	SQLColumnCheckOperator	NaN	{date, year}	NaN
check_details_table_quality	single task	SQLColumnCheckOperator	NaN	{detail_id, title}	NaN
check_wipo_classifications_table_quality	single task	SQLColumnCheckOperator	NaN	{wipo_classification_id, sector, field}	NaN
check_owners_table_quality	single task	SQLColumnCheckOperator	NaN	{owner_id, type, name}	NaN
check_locations_table_quality	single task	SQLColumnCheckOperator	NaN	{location_id, country, continent}	NaN
check_patent_keywords_table_quality	single task	SQLColumnCheckOperator	NaN	{patent_keyword_id, patent_id, keyword}	NaN
end_execution	single task	EmptyOperator	NaN	NaN	NaN

Tabella 5.1. Risultati Operatori.

Descrizione ETL

Queste tabelle riportano il nome di ciascun file analizzato, le tabelle di input e output derivanti dalle importazioni ed eventuali join, in modo da poter tracciare in modo semplice la struttura delle tabelle. Nei test condotti sul repository, il tool ha dimostrato un'elevata accuratezza nell'identificazione delle tabelle in input e output, insieme alle relative sorgenti dati. La sfida principale è emersa nella rilevazione di alcune operazioni di join, particolarmente quando queste coinvolgono tabelle rinominate attraverso variabili intermedie nel codice.

In queste specifiche istanze, che rappresentano una parte minoritaria ma significativa delle operazioni di join presenti nel codice analizzato, il tool non è riuscito a mappare correttamente le relazioni, sempre considerando singoli casi rispetto a decine di test. Questa limitazione, pur non compromettendo la validità complessiva dell'analisi del flusso dati, indica un'area specifica su cui concentrare gli sforzi di perfezionamento nelle future iterazioni del sistema.

file_name	**input_tables**	**output_tables**
cleaning_patent_assignee_job.py	patent_assignee.tsv	patent_assignee.parquet
cleaning_wipo_field_job.py	wipo_field.tsv	wipo_field.parquet
dates_etl_job.py	patent.parquet	dates
cleaning_wipo_job.py	wipo.tsv	wipo.parquet
patent_keywords_etl_job.py	patent_keyword.parquet	patent_keywords
cleaning_patent_job.py	patent.tsv	patent.parquet
details_etl_job.py	patent.parquet	details
cleaning_inventor_job.py	inventor.tsv	inventor.parquet
locations_etl_job.py	intermediary_patent.parquet	locations
cleaning_patent_inventor_job.py	patent_inventor.tsv	patent_inventor.parquet
patents_etl_job.py	patent.parquet, intermediary_patent.parquet	patents
owners_etl_job.py	intermediary_patent.parquet	owners
cleaning_assignee_job.py	assignee.tsv	assignee.parquet
cleaning_country_shapes_job.py	shapes_simplified_low.json, countryinfo.txt	country_shapes.csv
wipo_classifications_etl_job.py	wipo.parquet, wipo_field.parquet	wipo_classifications
yake_keyword_extraction_job.py	patent.parquet	patent_keyword_raw.parquet
rank_keyword_job.py	patent_keyword_raw.parquet	patent_keyword.parquet
cleaning_location_job.py	location.tsv, country_shapes.csv, countryinfo.txt	location.parquet
intermediary_patent_etl_job.py	patent.parquet, patent_assignee.parquet, assignee.parquet, patent_inventor.parquet, inventor.parquet, location.parquet	intermediary_patent.parquet

Tabella 5.2. Risultati ETL: Input/Output Tables

file_name	**type**	**tables**	**keys**	**table_out**
cleaning_inventor_job.py	INNER	, inventor_cleaned_with_id	inventor.name, inventor_cleaned_with_id.name	inventor_cleaned_with_id
patents_etl_job.py	INNER	_with_owner_and_location, patent	patent_with_owner_and_location.id, patent.id	patent_with_all_fields
cleaning_assignee_job.py	INNER	transformed_data, transformed_data_with_id	transformed_data.name, transformed_data_with_id.name, transformed_data.type_string_cleaned, transformed_data_with_id.type_string_cleaned	transformed_data_with_id
cleaning_country_shapes_job.py	INNER	country_shapes_cleaned, country_info}	country_shapes_cleaned.geonameid, country_info.geonameid	transformed_data
wipo_classifications_etl_job.py	INNER	wipo_data, wipo_field_data}	wipo_data.field_id, wipo_field_data.id	transformed_data
cleaning_location_job.py	LEFT	location_combined, country_info	location_combined.final_country, country_info.iso	location_combined
cleaning_location_job.py	LEFT	location_combined, additional_code	location_combined.final_country, additional_code.code	location_combined
cleaning_location_job.py	LEFT	data, location_with_country_id	data.new_country, location_with_country_id.new_country	location_combined
intermediary_patent_etl_job.py	LEFT	patent, patent_assignee	patent.id, patent_assignee.patent_id	patent_join_with_patent_assignee
intermediary_patent_etl_job.py	LEFT	patent_with_assignee, assignee	patent_with_assignee.assignee_id, assignee.id	patent_with_assignee
intermediary_patent_etl_job.py	LEFT	patent_with_assignee, location	patent_with_assignee.location_id, location.id	patent_with_assignee_location
intermediary_patent_etl_job.py	LEFT	patent_without_assignee, patent_inventor	patent_without_assignee.id, patent_inventor.patent_id	patent_without_assignee_join_inventor
intermediary_patent_etl_job.py	LEFT	patent_without_assignee_join_inventor, inventor	patent_without_assignee_join_inventor.inventor_id, inventor.id	patent_without_assignee_join_inventor
intermediary_patent_etl_job.py	LEFT	patent_without_assignee_join_inventor, location	patent_without_assignee_join_inventor.location_id, location.id	patent_without_assignee_join_inventor_loc

Tabella 5.3. Risultati ETL: Join.

Efficienza e Monitoraggio del Processo

L'utilizzo di strutture dati ben definite e il monitoraggio in tempo reale, implementato tramite la libreria tqdm, garantiscono la trasparenza e la tracciabilità di ogni fase del processo.

Ogni step del workflow viene monitorato e registrato, permettendo di analizzare in dettaglio l'avanzamento del processo e facilitando il debug in caso di anomalie. Le metriche di efficienza hanno evidenziato che il tool completa l'analisi di un repository di medie dimensioni (circa 30 file Python) in un tempo medio di 20 minuti. La raccolta sistematica di queste metriche ha permesso di aggregare dati di output estremamente accurati, che sono stati presentati in formati tabellari per una più facile interpretazione. Questo approccio quantitativo fornisce una solida base per valutare oggettivamente le prestazioni del tool e per identificare potenziali aree di miglioramento.

Correlazione tra Qualità del Codice e Risultati

Un'osservazione particolarmente rilevante emersa dall'analisi dei risultati riguarda la correlazione diretta tra la qualità della strutturazione del codice sorgente e l'efficacia della documentazione automatica generata. Il repository "Patent Analytics Data Pipeline", caratterizzato da un'eccellente organizzazione e da una documentazione interna di alta qualità, ha rappresentato un terreno fertile per l'applicazione del tool.

Questa correlazione sottolinea l'importanza di promuovere buone pratiche di sviluppo software all'interno delle organizzazioni, non solo come fine a sé stante, ma anche come prerequisito per l'efficace implementazione di strumenti di documentazione automatica. In particolare, l'adozione di convenzioni di denominazione chiare, l'uso sistematico di docstring e la strutturazione logica del codice sembrano essere fattori determinanti per il successo del processo di documentazione automatica.

Implicazioni Strategiche e Organizzative

L'analisi qualitativa e quantitativa dei risultati mostra chiaramente che il tool è in grado di produrre documentazione automatica di alta qualità, specialmente quando applicato a repository ben strutturati. I risultati dimostrano la robustezza e la precisione del sistema nell'estrazione e nell'organizzazione delle informazioni tecniche.

Questo caso d'uso evidenzia come l'adozione del tool possa rappresentare un significativo vantaggio strategico per le organizzazioni, con molteplici benefici:

1. **Miglioramento della gestione del know-how:** La documentazione automatica preserva e organizza la conoscenza tecnica, riducendo la dipendenza da singoli individui.

2. **Facilitazione della manutenzione dei sistemi:** Una documentazione completa e accurata semplifica l'identificazione e la risoluzione di problemi in ambienti complessi.

3. **Accelerazione dell'onboarding:** Nuovi membri del team possono acquisire rapidamente una comprensione approfondita dell'architettura e del funzionamento delle pipeline ETL.

4. **Promozione di standard di qualità:** L'implementazione del tool incentiva l'adozione di buone pratiche di sviluppo, creando un circolo virtuoso di miglioramento continuo.

5. **Ottimizzazione delle risorse:** La riduzione del tempo dedicato alla documentazione manuale permette di riallocare risorse preziose verso attività a maggiore valore aggiunto.

L'analisi dei risultati ottenuti nel caso d'uso "Patent Analytics Data Pipeline" conferma il potenziale trasformativo del tool di Automatic Documentation Drafting nel contesto della gestione delle pipeline ETL, offrendo una soluzione efficace a una delle sfide più persistenti nello sviluppo software: la creazione e il mantenimento di documentazione tecnica di alta qualità.

5.3 Benchmark

Approccio Metodologico alla Validazione

La definizione di un sistema di benchmark robusto ha rappresentato una sfida significativa nell'ambito del progetto di documentazione automatica. Durante la fase di progettazione, abbiamo considerato due approcci metodologici distinti per valutare l'accuratezza e l'affidabilità del tool:

- **Validazione tramite modello secondario**: Inizialmente, è stata valutata la possibilità di impiegare un secondo modello di AI generativa per validare gli output prodotti dal modello primario (Claude Sonnet). Questa metodologia avrebbe previsto l'utilizzo di un modello diverso che, applicando prompt progettati specificamente per la validazione, avrebbe verificato la correttezza e la completezza delle informazioni estratte.

- **Validazione basata su metriche di dissimilarità**: L'approccio alternativo prevedeva lo sviluppo di una metrica quantitativa di dissimilarità che confrontasse direttamente gli output generati dal tool con informazioni di riferimento fornite manualmente da esperti del dominio.

Dopo un'attenta analisi delle implicazioni metodologiche, abbiamo scartato il primo approccio per diverse ragioni critiche. In primo luogo, validare mediante un altro modello generativo avrebbe introdotto un ulteriore livello di incertezza, dato che anche il secondo modello sarebbe stato soggetto a potenziali errori o bias.

In secondo luogo, e più significativamente, esisteva il rischio concreto che entrambi i modelli commettessero errori sistematici simili, rendendo la validazione inaffidabile. Validare con un modello di AI ciò che è stato generato da un altro modello di AI risultava metodologicamente inopportuno, in quanto avrebbe potuto creare un circolo di conferma reciproca piuttosto che una verifica obiettiva.

Abbiamo quindi optato per il secondo approccio, sviluppando una metodologia di benchmark basata su metriche di dissimilarità con validazione umana, che ha garantito una valutazione più oggettiva e affidabile della qualità degli output.

Implementazione della Metrica di Dissimilarità

Il benchmark del sistema è stato realizzato adottando una metrica di dissimilarità, concepita per confrontare in maniera oggettiva le informazioni estratte dal tool con quelle fornite manualmente dall'utente come riferimento. Tale metrica, implementata nella funzione `calculate_dissimilarities`, consente di analizzare in dettaglio le differenze tra il DataFrame dei dati predetti e quello dei dati "veri", valutando la similarità a livello di ogni colonna per ciascun identificatore unico.

Il processo di benchmark si articola in più fasi:

1. **Normalizzazione dei Dati**: Le funzioni `normalize_df` e `normalize_df_join` sono impiegate per convertire le strutture dati dei risultati ETL in DataFrame standardizzati. Questa normalizzazione è fondamentale per garantire che formati di output potenzialmente eterogenei vengano resi omogenei e quindi comparabili. Il processo include:
   - Standardizzazione dei tipi di dati
   - Uniformazione dei nomi delle colonne

- Conversione di strutture nidificate in formato tabulare
- Gestione di valori mancanti o null

I DataFrame così ottenuti rappresentano il punto di partenza per il calcolo delle dissimilarità e forniscono una base coerente per confronti successivi.

2. **Calcolo delle Dissimilarità**: La funzione `calculate_dissimilarities` scorre ogni identificatore unico presente in entrambi i DataFrame, confrontando il contenuto di ogni colonna e calcolando una metrica di dissimilarità basata sulle differenze tra insiemi di valori. L'algoritmo implementato:

   - Identifica gli elementi comuni tra i due DataFrame
   - Per ogni identificatore comune, estrae i valori delle colonne corrispondenti
   - Calcola la differenza simmetrica tra gli insiemi di valori (elementi presenti in uno ma non nell'altro)
   - Normalizza questa differenza rispetto alla cardinalità totale degli insiemi
   - Aggrega i risultati per colonna e calcola uno score complessivo

Il risultato di questa operazione è un dizionario dettagliato che, oltre a indicare le differenze per ogni colonna, fornisce un punteggio globale (`"overall_score"`) che rappresenta la percentuale complessiva di dissimilarità. Un punteggio basso indica un alto grado di similarità, segnalando che il tool ha riprodotto con successo la documentazione attesa.

Parte VI

Conclusione

Capitolo 6

Discussione, Conclusioni e Sviluppi Futuri

6.1 Sintesi dei risultati e contributi della tesi

Questa tesi ha presentato un sistema innovativo per la generazione automatica di documentazione tecnica per pipeline ETL, affrontando una delle sfide più persistenti nello sviluppo software: la creazione e il mantenimento di documentazione accurata e aggiornata. I risultati ottenuti dimostrano in modo convincente che l'approccio basato su tecniche avanzate di prompt engineering con modelli generativi come Claude Sonnet rappresenta una soluzione efficace, scalabile e sostenibile a questo problema nel settore dello sviluppo software. I principali contributi di questa tesi possono essere sintetizzati come segue:

Metodologia di prompt engineering ottimizzata: Abbiamo sviluppato e perfezionato una serie di prompt strutturati che consentono l'estrazione efficace di informazioni da codice sorgente di pipeline ETL. Questa metodologia rappresenta un avanzamento significativo nell'applicazione di tecniche di GenAI a problemi di documentazione tecnica. L'approccio adottato ha permesso di identificare le strutture di prompt più efficaci, definendo un framework metodologico replicabile anche in altri contesti di documentazione automatica.

Sistema integrato end-to-end: Il tool realizzato offre un flusso di lavoro completo che va dall'analisi del codice sorgente alla pubblicazione della documentazione su Confluence, automatizzando l'intero processo e riducendo

significativamente il carico di lavoro manuale. L'integrazione con l'ecosistema AWS e con Confluence garantisce una soluzione enterprise-ready che si inserisce perfettamente nei workflow esistenti delle organizzazioni moderne.

Framework di validazione rigoroso: Abbiamo implementato un sistema di benchmark basato su metriche di dissimilarità che, in combinazione con la validazione umana, fornisce una valutazione oggettiva della qualità della documentazione prodotta. Questo approccio dual-track alla validazione consente di misurare l'efficacia del sistema sia dal punto di vista algoritmico che da quello dell'esperienza utente, offrendo una visione completa delle performance del tool.

Casi d'uso concreti: L'applicazione del sistema al repository "Patent Analytics Data Pipeline" ha dimostrato l'efficacia del tool in un contesto reale, con risultati quantitativi che evidenziano un'accuratezza elevata nell'estrazione delle informazioni chiave. L'estensione dell'analisi ad altri repository ha inoltre confermato la robustezza e l'adattabilità della soluzione a diversi stili di codice e architetture di pipeline ETL.

Soluzione per il debito tecnico documentale: Il sistema proposto offre una risposta concreta al problema del debito di conoscenza nelle organizzazioni, facilitando la condivisione delle informazioni e riducendo la dipendenza da conoscenze non documentate. L'automazione della documentazione non solo migliora la qualità e la consistenza della documentazione stessa, ma libera risorse preziose che possono essere impiegate in attività a maggior valore aggiunto.

Standardizzazione della documentazione: Il tool implementato garantisce una struttura coerente della documentazione, indipendentemente dall'autore originale del codice. Questa standardizzazione facilita la comprensione del codice da parte di nuovi membri del team e migliora la manutenibilità complessiva del software nel lungo periodo.

Riduzione dei costi di onboarding: La documentazione automatica e strutturata riduce significativamente il tempo necessario per l'onboarding di nuovi sviluppatori, accelerando la loro capacità di contribuire efficacemente al progetto e riducendo i costi associati all'inserimento di nuove risorse.

I test condotti hanno evidenziato la capacità del sistema di generare documentazione di alta qualità, con una precisione particolarmente elevata nell'identificazione degli operatori e delle tabelle in input e output, e con una buona accuratezza anche nella mappatura delle relazioni più complesse come

le operazioni di join e le trasformazioni dei dati. Le metriche di performance hanno inoltre dimostrato che il tempo necessario per generare documentazione completa è drasticamente ridotto rispetto all'approccio manuale tradizionale dedicato a questa attività. La validazione effettuata con sviluppatori esperti ha confermato che la qualità della documentazione prodotta soddisfa gli standard richiesti in ambito professionale, con un'elevata leggibilità e completezza delle informazioni. Particolarmente apprezzata è stata la capacità del sistema di mantenere un livello di astrazione appropriato, fornendo dettagli tecnici sufficienti senza perdersi in specificità irrilevanti, un equilibrio spesso difficile da raggiungere nella documentazione manuale.

6.2 Limiti e criticità riscontrate

Nonostante i risultati promettenti, l'implementazione e la valutazione del sistema hanno evidenziato alcune limitazioni e criticità che meritano un'analisi approfondita:

Dipendenza dalla qualità del codice sorgente: I risultati della documentazione automatica sono fortemente influenzati dalla qualità e dalla strutturazione del repository analizzato. Codice ben organizzato, con docstring dettagliati e nomi di variabili descrittivi, produce risultati significativamente migliori rispetto a codice poco documentato o strutturato in modo caotico. Questa dipendenza rappresenta una limitazione importante per l'applicazione del tool in contesti dove la qualità del codice è variabile o subottimale.

Tempi di elaborazione: L'analisi dettagliata del codice sorgente e la generazione della documentazione attraverso modelli di GenAI richiedono tempi di elaborazione significativi, specialmente per repository di grandi dimensioni. Questa limitazione può rappresentare un ostacolo all'adozione del tool in contesti dove la rapidità di aggiornamento della documentazione è un requisito fondamentale.

Tentativi di parallelizzazione non riusciti: Durante lo sviluppo del progetto, abbiamo esplorato la possibilità di parallelizzare il processo di analisi per migliorare le prestazioni. Tuttavia, questa strada è stata abbandonata a causa di problemi di integrazione con Confluence e limitazioni dell'ambiente di sviluppo. Le problematiche riscontrate includono conflitti nelle richieste API simultanee a Confluence e difficoltà nella gestione dei lock delle risorse condivise.

Validazione dipendente dall'utente: Il sistema di benchmark attuale richiede l'intervento umano per la definizione del gold standard e per l'analisi qualitativa delle discrepanze. Questa dipendenza introduce un elemento di soggettività nella valutazione e richiede risorse umane specializzate, limitando la scalabilità del processo di validazione. La creazione di un gold standard completamente automatizzato rimane una sfida aperta, in quanto richiederebbe capacità di comprensione del codice paragonabili a quelle di un esperto umano.

Dipendenza da Airflow e Pyspark: Il tool è stato progettato specificamente per analizzare processi ETL basati su Apache Airflow e PySpark, limitando la sua applicabilità ad altri framework o tecnologie ETL. L'estensione del supporto ad altre tecnologie richiederebbe una significativa rielaborazione dei prompt e dei meccanismi di analisi del codice, aumentando la complessità del sistema.

6.3 Proposte di miglioramento e possibili sviluppi futuri

Alla luce dei risultati ottenuti e delle limitazioni identificate, emergono diverse direzioni promettenti per lo sviluppo futuro del sistema:

Implementazione tramite RAG con agent AWS: Una delle direzioni più promettenti consiste nell'evoluzione dell'architettura verso un sistema basato su Retrieval-Augmented Generation (RAG) integrato con agent AWS. Questo approccio permetterebbe di orchestrare e customizzare i prompt in modo più dinamico, adattandoli al contesto specifico e migliorando la precisione dell'estrazione delle informazioni. L'utilizzo di agent AWS faciliterebbe inoltre l'integrazione con l'infrastruttura esistente e permetterebbe di sfruttare servizi cloud per migliorare scalabilità e prestazioni.

Approccio incrementale alla documentazione: Anziché rigenerare l'intera documentazione ad ogni modifica del codice, un sistema incrementale potrebbe analizzare solo le parti modificate e aggiornare selettivamente la documentazione esistente. Questo approccio ridurrebbe significativamente i tempi di elaborazione e faciliterebbe l'integrazione continua della documentazione nel processo di sviluppo.

Miglioramento del sistema di validazione: Il processo di benchmark

potrebbe essere perfezionato attraverso l'implementazione di tecniche di active learning che riducano progressivamente la necessità di intervento umano. Inoltre, l'introduzione di metriche di valutazione più sofisticate, specificamente adattate al dominio ETL, potrebbe fornire una valutazione più precisa e significativa della qualità della documentazione.

Estensione a diversi domini applicativi: Il framework metodologico sviluppato potrebbe essere adattato per la documentazione automatica di altri tipi di codice oltre alle pipeline ETL con Airflow e Pyspark, descrivendo processi ETL gestiti con altre tecnologie. Questo potrebbe essere integrato dopo uno studio approfondito e a parte sull'elasticità dei prompt.

6.4 Conclusioni finali

Questa ricerca ha affrontato il tema della documentazione automatica nei processi ETL, proponendo un approccio innovativo basato su tecniche di prompt engineering e sull'utilizzo di modelli di Generative AI per estrarre informazioni chiave dal codice sorgente e generare documentazione tecnica strutturata. L'obiettivo principale era sviluppare un tool capace di automatizzare un processo tradizionalmente manuale e spesso trascurato, migliorando la qualità, la coerenza e la fruibilità della documentazione all'interno dei team di sviluppo software.

I risultati ottenuti dimostrano che l'approccio proposto è efficace, scalabile e applicabile a contesti reali. L'implementazione del tool su un repository complesso come "Patent Analytics Data Pipeline" ha evidenziato la robustezza del sistema, confermando la capacità della soluzione di generare documentazione accurata e dettagliata per pipeline ETL basate su Apache Airflow e PySpark.

Uno dei principali contributi della tesi è stato lo sviluppo di una metodologia strutturata di prompt engineering, che ha permesso di ottimizzare l'interazione tra il codice sorgente e il modello AI. Questo approccio iterativo ha consentito di migliorare progressivamente l'accuratezza delle descrizioni generate, adattandole alle specifiche esigenze del dominio ETL. La generazione della documentazione ha beneficiato inoltre di un'integrazione diretta con Confluence, garantendo un aggiornamento automatico e continuo della knowledge base aziendale.

Contributi chiave della ricerca

I principali contributi della ricerca possono essere sintetizzati nei seguenti punti:

Automazione della documentazione: Il sistema sviluppato consente di generare documentazione dettagliata senza intervento manuale, riducendo significativamente il tempo necessario per mantenere aggiornati i documenti tecnici.

Applicazione pratica e validazione: Il tool è stato testato su un caso d'uso reale, dimostrando di poter estrarre informazioni strutturate e coerenti da pipeline ETL complesse.

Sistema di validazione quantitativa: L'adozione di una metrica di dissimilarità ha permesso di confrontare in modo oggettivo la documentazione generata con una documentazione di riferimento, fornendo un metodo sistematico per valutare la qualità dell'output.

Miglioramento della gestione della conoscenza: L'adozione di strumenti automatici per la documentazione riduce il debito di conoscenza all'interno dei team di sviluppo, facilitando la comprensione del codice anche per nuovi sviluppatori o team distribuiti.

Questa ricerca dimostra che l'automazione della documentazione tecnica nei processi ETL non solo è possibile, ma rappresenta un'opportunità concreta per migliorare la produttività e la qualità del software. L'applicazione del tool su un repository complesso ha confermato che l'adozione di modelli di AI per la documentazione automatica può ridurre significativamente il carico di lavoro degli sviluppatori, garantendo al contempo documenti tecnici più coerenti, aggiornati e facilmente consultabili.

L'automazione della documentazione non è solo un vantaggio per i team di sviluppo, ma rappresenta anche un passo avanti nella gestione della conoscenza aziendale. L'integrazione con strumenti collaborativi come Confluence assicura che le informazioni siano sempre accessibili e aggiornate, contribuendo a creare un ecosistema di sviluppo più efficiente e strutturato.

Il lavoro svolto in questa tesi pone le basi per future ricerche e sviluppi nel campo della documentazione automatica, con l'obiettivo di rendere questo processo sempre più preciso, veloce e adattabile a un numero crescente di contesti applicativi.

Bibliografia

[1] Jie-Cherng Chen and Sun-Jen Huang. 2009. An empirical analysis of the impact of software development problem factors on software maintainability. Journal of Systems and Software

[2] Golara Garousi, Vahid Garousi-Yusifoglu, Guenther ˘ Ruhe, Junji Zhi, Mahmoud Moussavi, and Brian Smith. 2015. Usage and usefulness of technical software documentation: An industrial case study. Information and Software Technology.

[3] Xing Hu, Xin Xia, David Lo, Zhiyuan Wan, Qiuyuan Chen, and Thomas Zimmermann. 2022. Practitioners' expectations on automated code comment generation. In 2022 IEEE/ACM 44rd International Conference on Software Engineering (ICSE). IEEE.

[4] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, et al. 2020. Codebert: A pre-trained model for programming and natural languages. arXiv preprint arXiv:2002.08155.

[5] Haotian Cui1, Chenglong Wang, Junjie Huang, Jeevana Priya Inala,Todd Mytkowicz, Bo Wang, Jianfeng Gao, Nan Duan CodeExp: Explanatory Code Document Generation

[6] Kimball, R. Caserta, J. "The Data Warehouse ETL Toolkit"

[7] Inmon, W.H. "Building the Data Warehouse"

[8] Apache Airflow Documentation (2023)

[9] Schervish, T. (2021). "The Evolution of Apache Airflow: History and Future Directions". Medium

[10] Sahoo, Sumit Kumar (2023)Open-source ETL Framework using Big Data tools Orchestration on AWS Cloud Platform.Masters thesis, Dublin, National College of Ireland.

[11] Bansal, S. K. (2014). Towards a semantic extract-transform-load (etl) framework for big data integration, 2014 IEEE International Congress on Big Data, IEEE, pp. 522–529

[12] Manconi A, Gnocchi M, Milanesi L, Marullo O, Armano G (2023) Framing Apache Spark in life sciences. Heliyon 9(2):e13368.

[13] Zaharia, M., Chowdhury, M., et al. (2012). "Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing". Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation.

[14] Armbrust, M., Das, T., et al. (2015). "Spark SQL: Relational Data Processing in Spark". Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, 1383-1394.

[15] Armbrust, M., Yavuz, B., et al. (2020). "Delta Lake: High-Performance ACID Table Storage over Cloud Object Stores". Proceedings of VLDB Endowment, 13(12), 3411-3424.

[16] Databricks (2021). "Koalas: pandas API on Apache Spark".

[17] Zaharia, M., Das, T., et al. (2018). "Structured Streaming: A Declarative API for Real-Time Applications in Apache Spark". Proceedings of the 2018 International Conference on Management of Data.

[18] Meng, X., Bradley, J., et al. (2016). "MLlib: Machine Learning in Apache Spark". Journal of Machine Learning Research, 17(34).

[19] Alammar, J. (2020). "The Illustrated Transformer: Visualization and Explanation". Distill.

[20] Bender, E. M., Gebru, T., McMillan-Major, A., Shmitchell, S. (2021). "On the Dangers of Stochastic Parrots: Can Language Models Be Too Big?". Proceedings of FAccT '21.

[21] Bommasani, R., et al. (2022). "On the Opportunities and Risks of Foundation Models". Stanford University HAI.

[22] Goodfellow, I., et al. (2014). "Generative Adversarial Networks". Advances in Neural Information Processing Systems.

[23] Vaswani, A., et al. (2017). "Attention Is All You Need". Advances in Neural Information Processing Systems.

[24] LeCun, Y. (2022). "A Path Towards Autonomous Machine Intelligence". OpenReview.

[25] Kaplan, J., et al. (2020). "Scaling Laws for Neural Language Models". arXiv preprint arXiv:2001.08361.

[26] Maynez, J., Narayan, S., Bohnet, B., McDonald, R. (2022). "On Faithfulness and Factuality in Abstractive Summarization". Association for Computational Linguistics (ACL).

[27] Touvron, H., et al. (2023). "LLaMA: Open and Efficient Foundation Language Models". arXiv preprint arXiv:2302.13971.

[28] Nakano, R., et al. (2023). "WebGPT: Browser-assisted Question-answering with Human Feedback". arXiv preprint arXiv:2112.09332.
[29] White, J., Fu, Q., Hays, S., Sandborn, M., Olea, C., Gilbert, H., Elnashar, A., Spencer-Smith, J., Schmidt, D. C. (2023). "A Prompt Pattern Catalog to Enhance Prompt Engineering with ChatGPT". arXiv preprint arXiv:2302.11382.
[30] Wei, J., et al. (2022). "Chain-of-Thought Prompting Elicits Reasoning in Large Language Models". Nature.
[31] Brown, T. B., et al. (2020). "Language Models are Few-Shot Learners". Advances in Neural Information Processing Systems.
[32] Mishra, S., Khashabi, D., Baral, C., Hajishirzi, H. (2022). "Cross-Task Generalization via Natural Language Crowdsourcing Instructions". ACL 2022.
[33] Antropic User-Guide 2024
[34] https://www.atlassian.com/software/confluence
[35] github.com
[36] https://aws.amazon.com/it/sagemaker-ai/studio/
[37] https://jupyter.org
[38] https://www.datachannel.co/blogs/what-is-etl-and-how-the-etl-process-works
[39] https://en.wikipedia.org/wiki/Apache_Airflow/media/File:AirflowLogo.png
[40] https://en.wikipedia.org/wiki/Apache_Spark/media/File:Apache_Spark_logo.svg
[41] https://github.com/tjoakarlina/Udacity-Patents-Analytics-Data-Pipeline