UNIVERSITÀ DEGLI STUDI DI SALERNO
DIPARTIMENTO DI INFORMATICA

Esame di Ingegneria, Gestione ed Evoluzione del Software

PROGETTO CODESMILE

Final Project Report

TEAM MEMBERS

Dario Mazza - 0522501553

Nicolò Delogu - 0522501556

REPOSITORY

`https://github.com/xDaryamo/smell_ai`

VERSIONE

1.0

Anno Accademico 2024-2025

1 Introduzione

Il presente documento descrive i risultati delle modifiche apportate al sistema. Le attività progettuali si sono concentrate sul miglioramento della struttura architetturale e sull'introduzione di nuove funzionalità. Gli obiettivi principali erano incrementare la modularità e la scalabilità del sistema e ampliare le capacità analitiche, assicurando nel contempo delle buone performance.

2 Esito delle modifiche

2.1 Esito CR1

La ristrutturazione del sistema ha comportato la transizione da un'architettura procedurale a una orientata agli oggetti. Questa modifica ha portato a una struttura più modulare e manutenibile, migliorando l'organizzazione del codice. Il risultato finale è un sistema che risulta più scalabile e comprensibile rispetto alla versione precedente. Nella repository del progetto è stato aggiunto il diagramma delle classi che rispecchia la nuova architettura del tool. Inoltre, è stata aggiunta una suite di testing automatizzata, integrata tramite pipeline di *CI/CD* (Continuous Integration/Continuous Deployment). Questa soluzione permette di effettuare controlli automatici e continui sul codice, riducendo i tempi necessari per identificare e correggere eventuali errori.

2.2 Esito CR2

È stata sviluppata una nuova web application indipendente per l'analisi dei code smells. L'applicazione offre un'interfaccia intuitiva, con funzionalità di analisi interattiva e un'esperienza utente migliorata. L'architettura dell'applicazione è stata progettata seguendo un approccio a servizi, utilizzando **FastAPI** per implementare il backend. Questo ha permesso di creare un sistema modulare e facilmente integrabile con le altre componenti. Per il frontend, è stato utilizzato il framework **Next.js**, che ha consentito di realizzare un'interfaccia utente fluida e intuitiva. La web application è stata integrata nel sistema senza alterare il funzionamento delle componenti esistenti, offrendo una soluzione efficace, scalabile e user-friendly.

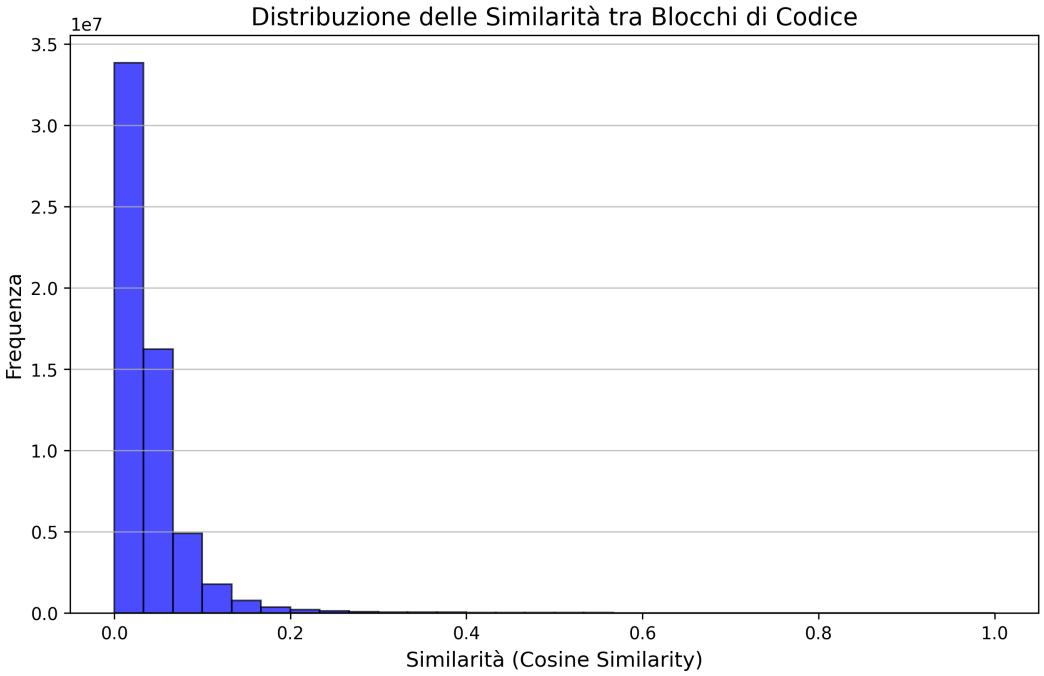
2.3 Esito CR3

È stato introdotto un sistema per la generazione automatica di dataset con smells specifici utilizzando Modelli di Linguaggio di Grandi Dimensioni (LLM). L'integrazione del modello Qwen2.5-Coder-14B ha permesso di creare varianti di codice con code smells in maniera relativamente efficiente rispetto ai vincoli hardware del progetto. Per garantire la validità del dataset, sono stati effettuati due controlli principali. Il primo controllo è stata la validazione sintattica, che ha confermato l'assenza di errori sintattici nel 97.38% di tutti gli snippet di codice generati. Il secondo controllo ha riguardato la similarità tra gli snippet, con l'obiettivo di verificare la dispersione e l'unicità dei dati. A tal fine, sono state calcolate le caratteristiche principali della distribuzione delle similarità tra snippet di codice. I risultati sono i seguenti:

- **Media:** 0.0410

- **Mediana:** 0.0284
- **Deviazione standard:** 0.0501
- **Valore massimo:** 1.0
- **Valore minimo:** 0.0

Queste statistiche dimostrano una bassa similarità media tra gli snippet, indicando una buona diversità nei dati generati. Il valore massimo di 1.0 evidenzia che alcuni snippet potrebbero essere identici, ma rappresentano casi isolati e non influenzano negativamente la qualità complessiva del dataset. Al fine di inquadrare meglio i risultati ottenuti, è stato prodotto un plot per rappresentare graficamente i valori delle statistiche di similarità:



2.4 Esito CR4

Durante l'implementazione della CR4, è stato sviluppato e addestrato un modello di intelligenza artificiale per il rilevamento automatico di code smells, sfruttando i dataset generati nella CR3. Il modello è stato progettato utilizzando Qwen2.5-Coder-Instruct tra cui le varianti da 0.5B, 1.5B e 3B parametri. In fase di valutazione, il modello 3B è risultato essere il più performante, raggiungendo le seguenti metriche di classificazione:

Classe	Precision	Recall	F1-score
Unnecessary Iteration	0.80	0.93	0.86
Hyperparameter Not Explicitly Set	0.84	0.88	0.86
NaN Equivalence Comparison Misused	0.98	0.97	0.98
Empty Column Misinitialization	0.98	0.96	0.97
Gradients Not Cleared Before Backward Propagation	0.82	0.87	0.85
TensorArray Not Used	0.94	0.93	0.94
Dataframe Conversion API Misused	0.90	0.88	0.89
Deterministic Algorithm Option Not Used	0.99	0.99	0.99
Chain Indexing	0.88	0.76	0.81
No Smell	0.98	0.98	0.98
Matrix Multiplication API Misused	0.97	0.91	0.94
Merge API Parameter Not Explicitly Set	0.93	0.93	0.93
Memory Not Freed	0.93	0.90	0.92
Broadcasting Feature Not Used	0.98	0.97	0.98
In-Place APIs Misused	0.83	0.74	0.78
Columns and DataType Not Explicitly Set	0.89	0.94	0.92
PyTorch Call Method Misused	0.94	0.92	0.93
Micro avg	0.95	0.95	0.95
Macro avg	0.92	0.91	0.91
Weighted avg	0.95	0.95	0.95
Samples avg	0.95	0.95	0.95

Tabella 1: Metriche di valutazione del modello più performante.

Il modello ha ottenuto un'accuratezza complessiva del 94.46% e ha dimostrato prestazioni elevate su diverse tipologie di code smells. L'F1-score medio è pari a 0.95, evidenziando un bilanciamento ragionevole tra precisione e richiamo.

3 Deviazioni da quanto pianificato

Durante l'esecuzione del progetto, si sono manifestate alcune limitazioni legate alla disponibilità di risorse hardware. In particolare, non è stato possibile fare uso di modelli altamente complessi per la fase di injection dei code smells e per l'addestramento del modello di rilevazione, a causa dei requisiti computazionali elevati

richiesti da tali approcci. Queste limitazioni hanno impedito l'iniezione di code smells multipli all'interno di un singolo snippet e la realizzazione della detection con più etichette (multi-label classification). Di conseguenza, consideriamo le change requests CR3 e CR4 realizzate con successo solo in parte. Un'altra differenza significativa riguarda le funzionalità offerte dal tool statico rispetto al tool AI. Il tool statico, a differenza del tool AI, fornisce la riga specifica in cui si manifesta il code smell e offre suggerimenti su come correggerlo. Il tool AI, invece, non dispone di queste funzionalità a causa delle limitazioni hardware e della dimensione del contesto gestibile dai modelli utilizzati. L'attenzione è quindi stata focalizzata sul miglioramento della qualità del *prompt engineering*, sfruttando al massimo le capacità dei modelli disponibili. Grazie a questa strategia, è stato possibile raggiungere comunque risultati promettenti, dimostrando l'efficacia dell'approccio adottato, pur operando in un contesto limitato.