UNIVERSITÀ DEGLI STUDI DI SALERNO
DIPARTIMENTO DI INFORMATICA

Esame di Ingegneria, Gestione ed Evoluzione del Software

PROGETTO CODESMILE

Report

TEAM MEMBERS

Dario Mazza - 0522501553

Nicolò Delogu - 0522501556

REPOSITORY

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

VERSIONE

1.0

Anno Accademico 2024-2025

1 Descrizione del Tool

1.1 Contesto

Il tool CodeSmile si colloca all'interno del macro contesto del Technical Debt e, in particolare, fa fronte alla problematica dell'AI-Specific Technical Debt, ossia l'accumulo di compromessi tecnici all'interno di una tipologia precisa di sistemi: quella degli Artificial Intelligence-Enabled Systems. Più specificatamente, CodeSmile si propone di ridurre le seguenti tipologie di debito tecnico legate all'IA: Data Debt, Model Debt, Configuration Debt e Ethics Debt, attraverso il rilevamento (basato su regole di identificazione) di code smells specifici per il Machine Learning (ML-specific Code Smells) all'interno di progetti software basati su Python. L'obiettivo del tool è migliorare la qualità del codice individuando pattern sub-ottimali che potrebbero comprometterne la manutenibilità, la leggibilità o la robustezza.

1.2 Funzionamento Generale

CodeSmile effettua un'analisi statica del codice utilizzando Abstract Syntax Trees (AST) e regole specifiche per il rilevamento dei code smells. Il tool analizza i moduli Python di un progetto, costruisce l'AST per ciascuno di essi e applica regole per individuare i code smells. Attualmente, CodeSmile supporta il rilevamento di 16 code smells specifici per il machine learning, come ad esempio l'uso errato delle API in-place di Pandas, la mancata liberazione della memoria nei cicli, e l'uso improprio dei metodi forward in PyTorch. Questo contribuisce a ridurre il debito tecnico, rendendo il codice più semplice da mantenere e migliorare.

2 Struttura e Architettura del Tool

Il tool è suddiviso in più moduli Python, ognuno con una responsabilità specifica. La struttura riflette un'organizzazione funzionale, dove ogni file Python contribuisce a un particolare aspetto del rilevamento dei *code smells*.

2.1 Moduli Principali e Architettura

- Modulo **`components`**, contiene componenti principali del tool, tra cui:
  - **`detector.py`**: Questo modulo contiene la logica di base per il rilevamento degli smells. Utilizza librerie specifiche, dizionari e regole di rilevamento per identificare code smells nei progetti ML. Il file `detector.py` applica diverse regole, come il controllo dell'uso improprio delle operazioni su DataFrame di Pandas e la corretta gestione delle operazioni sui tensori. Inoltre, salva i risultati dell'analisi in formato CSV.
  - **`cloner.py`**: Questo modulo si occupa della clonazione dei repository di progetto. Utilizza un dataset di progetti ML (NICHE dataset) per recuperare e preparare i progetti da analizzare, filtrando i repository sulla base di metriche come il numero di stelle e di commit.

- Modulo **`controller`**, funge da intermediario, gestendo sia l'interfaccia utente che il flusso di controllo del tool:
  - **`analyzer.py`**: Gestisce l'analisi dei progetti, integrando il rilevamento degli smells con i moduli `components`. Questo modulo è in grado di analizzare i file Python di un progetto, identificare i code smells tramite il modulo `detector.py` e salvare i risultati in file CSV. Inoltre, supporta l'analisi in parallelo utilizzando `ThreadPoolExecutor` per migliorare l'efficienza del processo di rilevamento.
  - **`GUI.py`**: Include una piccola interfaccia grafica sviluppata con Tkinter, che consente agli utenti di selezionare le directory di input e output, configurare il numero di "walker" da usare e avviare l'analisi. La GUI include diverse opzioni per personalizzare l'analisi, come la scelta della parallelizzazione e la configurazione del numero di thread. Anche se sono presenti opzioni per il refactoring, queste non sono ancora collegate a una logica di codice. La GUI permette inoltre di visualizzare i risultati dell'analisi in una finestra di testo integrata, utilizzando la classe `TextboxRedirect` per reindirizzare l'output della console.
- Modulo **`cs_detector`**, è il cuore del sistema di rilevamento dei code smells, diviso in due sotto-moduli principali:
  - **`code_extractor`**, gestisce l'estrazione delle strutture di codice:
    - **`dataframe_detector.py`**: Utilizza il file `dataframes.csv` per identificare operazioni sui DataFrame di Pandas che potrebbero rappresentare potenziali code smells. Implementa un approccio ricorsivo per cercare variabili che derivano da oggetti DataFrame di Pandas e ne traccia l'utilizzo tramite l'analisi AST, verificando se le operazioni svolte su di esse potrebbero costituire code smells.
    - **`libraries.py`**: Questo modulo si occupa di estrarre le librerie utilizzate all'interno del codice tramite l'analisi dell'AST. Viene utilizzato per identificare le librerie importate e associare i metodi chiamati alle librerie corrispondenti.
    - **`models.py`**: Fornisce il supporto per caricare e verificare i dizionari dei modelli di machine learning e delle operazioni sui tensori. Permette di verificare l'uso corretto delle operazioni sui modelli.
    - **`variables.py`**: Contiene funzioni per identificare e tracciare le definizioni delle variabili all'interno del codice. Utilizza pattern di ricerca tramite espressioni regolari per individuare le assegnazioni di variabili e determinare se queste potrebbero rappresentare un problema.
  - **`detection_rules`**, contiene le regole di rilevamento definite per l'analisi:
    - **`APISpecific.py`**: Include regole specifiche per rilevare l'uso improprio di determinate API comuni nei progetti ML. Ad esempio, rileva problemi come l'uso improprio della funzione `dot` di NumPy per la moltiplicazione di matrici o il mancato utilizzo di `TensorArray`

- **`Generic.py`**: Include regole generali per identificare code smells comuni che possono verificarsi in vari contesti, come l'utilizzo improprio di algoritmi deterministici o la mancata impostazione esplicita di parametri nelle API.

- Modulo **`obj_ditionaries`**: contiene i file `dataframes.csv`, `models.csv`, e `tensors.csv`, che mappano le operazioni tipiche di Pandas e altre librerie per il machine learning, fornendo una base strutturata per l'analisi dei code smells specifici per ML.

- Modulo **`input`**: contiene il dataset di input e i progetti di esempio utilizzati per l'analisi. Il modulo `cloner.py` utilizza i dati contenuti nella directory `dataset` per clonare i repository GitHub da analizzare.

- Modulo **`test`**: Include test specifici per valutare le funzionalità del rilevatore di code smells (`cs_detector`). Il file `cli.py` viene utilizzato per testare l'interfaccia a linea di comando e verificare la corretta esecuzione delle operazioni di rilevamento. Inoltre, il file `test_models.py` verifica la corretta identificazione delle librerie e dei metodi all'interno del codice, assicurandosi che il modulo `models.py` sia in grado di caricare correttamente i dizionari e applicare le regole di verifica sui modelli.

2.2 Interazione tra Componenti

1. **Clonazione dei Progetti:** Il processo inizia con il modulo `cloner.py`, che clona i repository GitHub da analizzare, utilizzando criteri di filtraggio per selezionare progetti rilevanti dal dataset NICHE.

2. **Estrazione del Codice:** Una volta clonati i progetti, il modulo `cs_detector/code_extractor` estrae le strutture di codice rilevanti. Utilizza i file `dataframes.csv`, `models.csv`, e `tensors.csv` per costruire una rappresentazione strutturata delle operazioni sui dati e sui modelli. Inoltre, `dataframe_detector.` analizza l'uso dei DataFrame per individuare potenziali smells.

3. **Rilevamento dei Code Smells**: Il modulo `components/detector.py` applica le regole di rilevamento del modulo `detection_rules` per identificare code smells specifici. Il rilevamento viene effettuato tramite l'analisi AST e il confronto con i dizionari e le regole definite.

4. **Analisi e Output**: I risultati dell'analisi vengono elaborati dal modulo `controller/analyzer.py`, che li integra e li salva. I risultati possono essere visualizzati tramite la GUI sviluppata con Tkinter, oppure stampati sulla console se l'interfaccia CLI è utilizzata. Il modulo `analyzer.py` supporta sia l'analisi sequenziale che parallela, in base alle opzioni selezionate dall'utente. Infine, i risultati aggregati sono salvati in un file `overview_output.csv`, che offre una visione d'insieme dell'analisi.

3 Flusso di Utilizzo del Tool

3.1 Modalità di Utilizzo

Attualmente, CodeSmile è utilizzabile sia tramite un'interfaccia a linea di comando (CLI) che tramite un'interfaccia grafica (GUI). La CLI consente agli utenti di specificare i file da analizzare e configurare l'analisi, di

validare i file output ottenuti e di ottenere dei report riguardo la quantità e la tipologia di ogni smell rilevato. La GUI, d'altra parte, permette un'analisi più accessibile tramite opzioni visive.

3.1.1 Versione con GUI

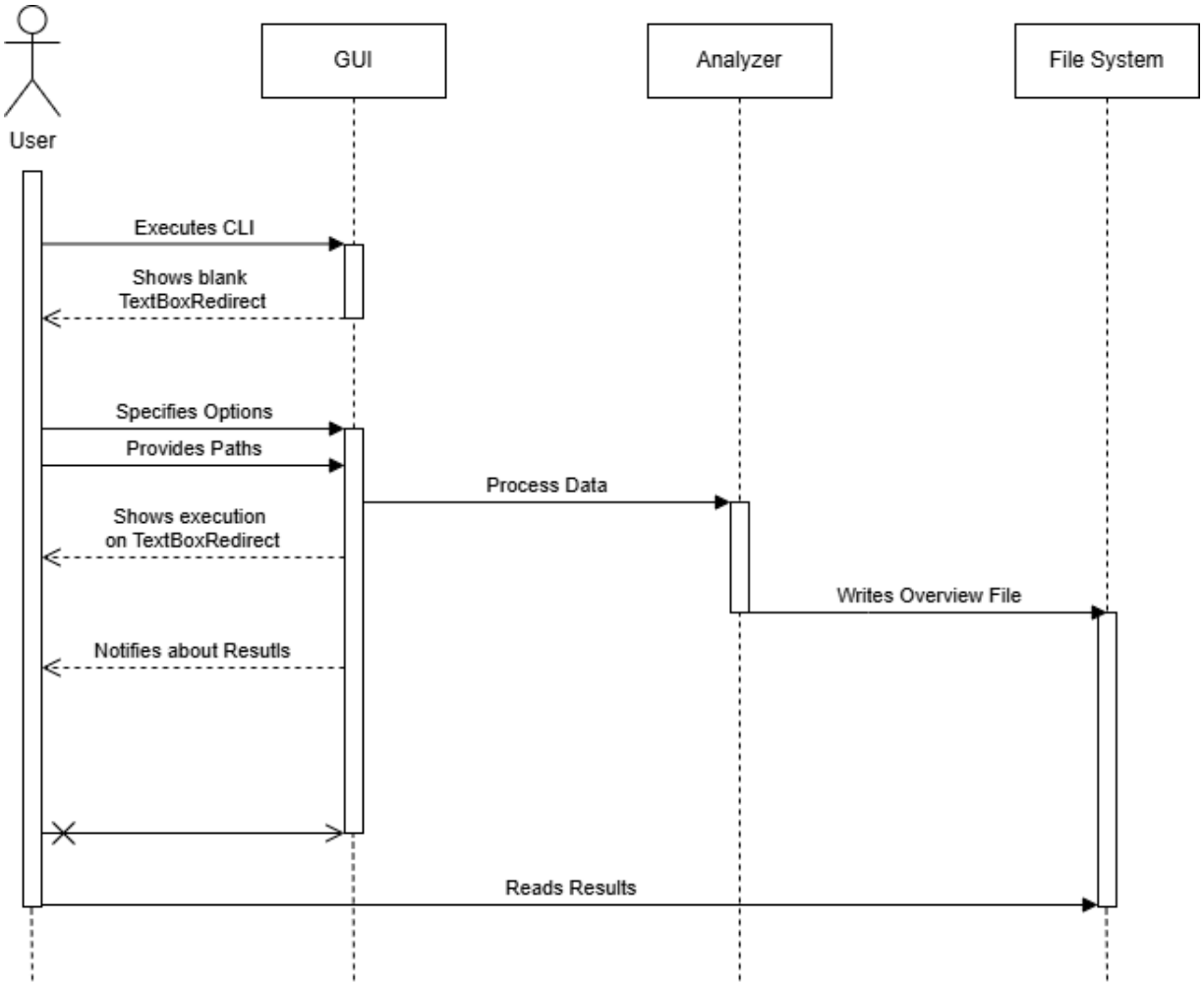

Figura 1: Sequence Diagram per Analisi

3.1.2 Versione con CLI

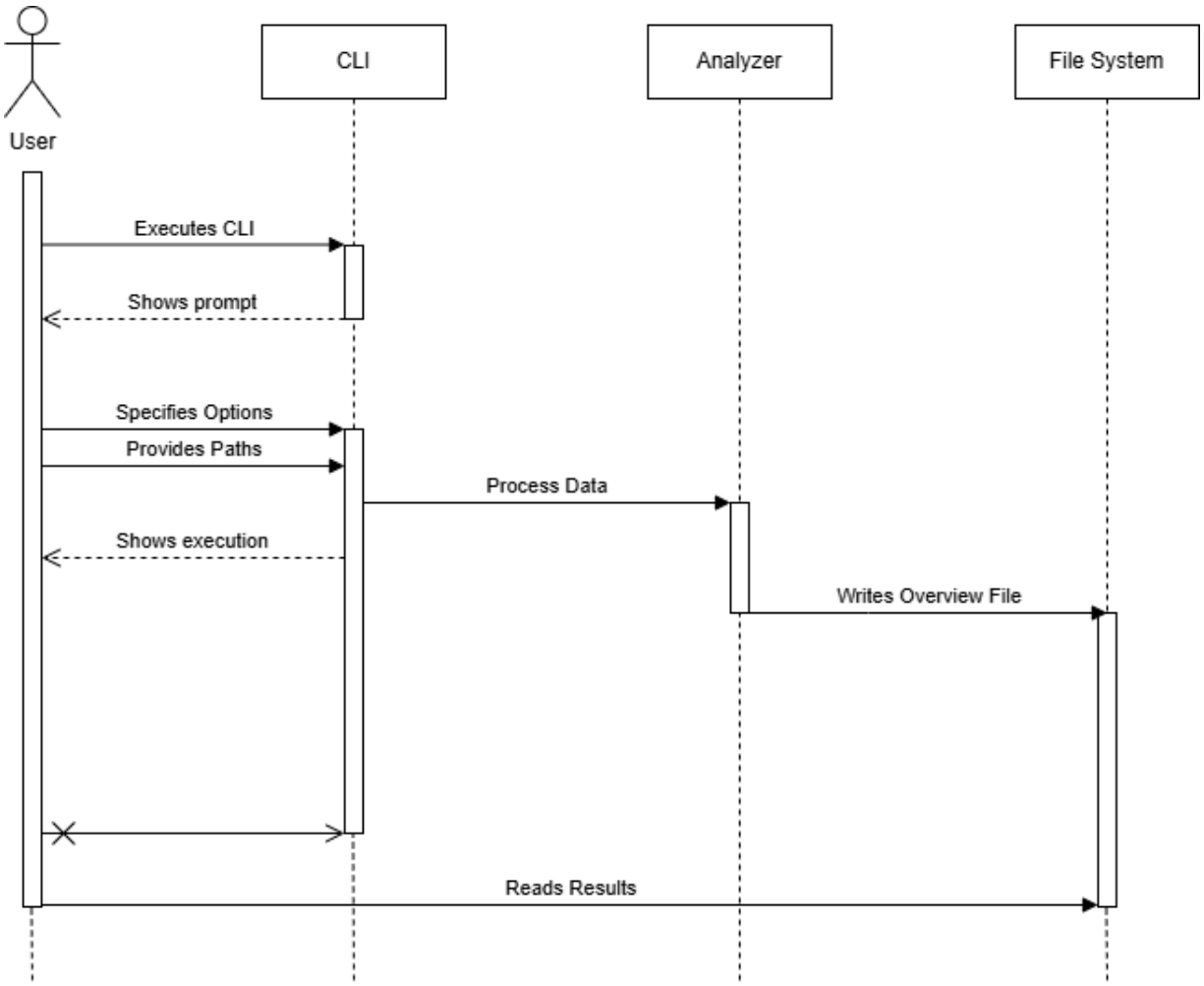

Figura 2: Sequence Diagram per Analisi

3.1.3 Funzionalita di Report

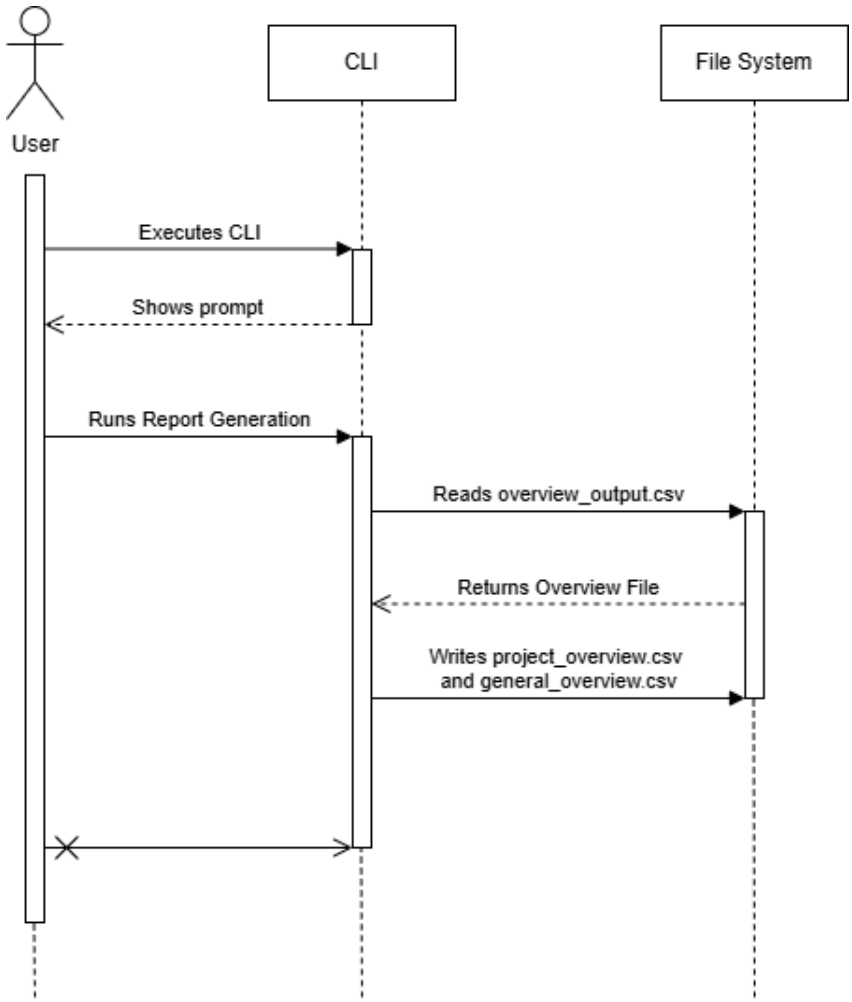

Figura 3: Sequence Diagram per Report

3.2 Output e Risultati

Il risultato dell'analisi viene fornito in formato CSV. Ogni progetto analizzato genera un file `to_save.csv`, che contiene dettagli sugli smells rilevati, tra cui il nome del file, il nome della funzione, il tipo di smell, il nome dello smell, e un messaggio descrittivo. Questo file viene salvato nella directory di output corrispondente al progetto analizzato.

Esempio di output `to_save.csv`:

- **filename**: Percorso del file in cui è stato rilevato il code smell.
- **function_name**: Nome della funzione in cui è stato rilevato il code smell.
- **smell**: Numero di occorrenze del code smell.
- **name_smell**: Nome del code smell rilevato.
- **message**: Messaggio descrittivo che spiega il problema individuato.

Inoltre, il tool mostra l'andamento direttamente sulla console o tramite la GUI, notificando l'utente con il tempo di esecuzione impiegato per l'analisi e con le opzioni che sono state fornite in input al tool. La sintesi dei risultati viene aggregata tramite il modulo `analyzer`, che combina tutti i risultati parziali in un unico file `overview_output.csv`, utile per una visione complessiva dell'analisi. Il file `overview_output.csv` viene salvato nella directory `general_output`, insieme al file `count_report.py`, che ha lo scopo di aiutare a sintetizzare i risultati dell'analisi fornendo report sia a livello di smell che di progetto, rendendo più semplice il confronto e la valutazione della qualità del codice analizzato. I file `random_stratifying.py` e altri strumenti di validazione si trovano nella sottocartella `validation`, con l'obiettivo di costruire un campione rappresentativo per ogni smell da utilizzare per ulteriori studi e report dettagliati.