Initial Report Document

Versione 1.0

CodeSmile

Team Members:
Matteo Ercolino — 0522501462
Simone Silvestri — 0522501419

Repository: GitHub link

Anno Accademico 2024/2025

Indice

1 Descrizione del Sistema **3**
1.1 Introduzione . . . 3

2 Struttura e Architettura del Sistema **4**
2.1 Package del Sistema . . . 4
2.1.1 Package `cli` . . . 4
2.1.2 Package `code_extractor` . . . 4
2.1.3 Package `components` . . . 4
2.1.4 Package `detection_rules` . . . 5
2.1.5 Package `gui` . . . 6
2.1.6 Package `utils` . . . 7
2.1.7 Package `report` . . . 7
2.2 Web App . . . 7
2.2.1 Package `gateway` . . . 7
2.2.2 Package `services` . . . 7
2.2.3 Funzionamento e Interazione del Sistema Web . . . 8

3 Flusso di Utilizzo del Tool **10**
3.1 Utilizzo tramite Interfaccia a Linea di Comando (CLI) . . . 10
3.2 Utilizzo tramite Web App . . . 10
3.3 Analisi AI-Based e Static Analysis . . . 10
3.4 Analisi di Progetti Open Source . . . 10
3.5 Generazione di Report e Visualizzazioni . . . 11

1 Descrizione del Sistema

1.1 Introduzione

CodeSmile è uno strumento sviluppato per affrontare il debito tecnico, con un focus particolare sul debito tecnico specifico per l'intelligenza artificiale, noto come *AI-Specific Technical Debt*. Questo tipo di debito tecnico emerge all'interno dei sistemi che utilizzano tecnologie di Intelligenza Artificiale (AI-Enabled Systems).

Il principale obiettivo di CodeSmile è ridurre le seguenti tipologie di debito tecnico legato all'IA:

- **Data Debt** (Debito dei dati),
- **Model Debt** (Debito del modello),
- **Configuration Debt** (Debito della configurazione),
- **Ethics Debt** (Debito etico).

CodeSmile raggiunge questo obiettivo identificando *code smells* specifici per il Machine Learning, conosciuti come *ML-specific Code Smells*, all'interno di progetti software scritti in Python. Il processo di rilevamento si basa su regole predefinite di identificazione e sull'analisi statica del codice, utilizzando gli Abstract Syntax Trees (AST).

Il sistema è progettato per migliorare la qualità del codice rilevando pattern subottimali che potrebbero compromettere la manutenibilità, la leggibilità o la robustezza. Al momento, CodeSmile supporta il rilevamento di 16 diversi *code smells* specifici per il Machine Learning, come l'uso errato delle API in-place di Pandas, la mancata liberazione della memoria durante i cicli, e l'uso improprio dei metodi `forward` in PyTorch.

Il design del sistema è basato su un'architettura orientata agli oggetti, che ottimizza la modularità, la manutenibilità e la scalabilità del codice. È stata sviluppata anche una Web-App che consente interazioni via web, migliorando l'esperienza utente e l'accessibilità del tool. Inoltre, il sistema sfrutta modelli di linguaggio (LLM) per l'iniezione di *smell* nei codebase e per addestrare un modello di intelligenza artificiale finalizzato al rilevamento degli *ML-specific smells*.

2 Struttura e Architettura del Sistema

Il sistema è organizzato secondo un'architettura a oggetti, con una suddivisione in package, ognuno con una responsabilità specifica.

2.1 Package del Sistema

2.1.1 Package `cli`

Gestisce l'interfaccia a linea di comando per l'esecuzione del tool.

- `cli_runner.py`: Gestisce l'interfaccia a linea di comando (CLI) per l'esecuzione del tool CodeSmile. Permette di configurare e avviare il processo di analisi dei progetti Python, supportando parametri per l'input/output, l'esecuzione parallela, la ripresa di analisi interrotte e l'elaborazione di più progetti. Utilizza `ProjectAnalyzer` per eseguire il rilevamento dei code smells e salvare i risultati.

2.1.2 Package `code_extractor`

Contiene i moduli per l'estrazione delle strutture di codice.

- `dataframe_extractor.py`: Fornisce strumenti per l'analisi statica del codice Python al fine di identificare l'uso dei DataFrame di Pandas. Utilizzando Abstract Syntax Trees (AST), il modulo rileva variabili inizializzate come DataFrame, traccia i metodi chiamati su di essi e individua gli accessi alle colonne. Inoltre, carica un dizionario di metodi di Pandas per affinare il processo di rilevamento.
- `library_extractor.py`: Analizza il codice Python tramite Abstract Syntax Trees (AST) per identificare le librerie importate e i relativi alias. Fornisce metodi per estrarre informazioni sulle importazioni, mappare gli alias delle librerie e determinare a quale libreria appartiene una determinata chiamata di funzione o metodo.
- `model_extractor.py`: Carica e gestisce informazioni sui modelli di Machine Learning e le operazioni sui tensori. Utilizzando dati estratti da file CSV, permette di identificare metodi associati a specifiche librerie, filtrare operazioni sui tensori con input multipli e verificare se un determinato modello appartiene a una specifica libreria.
- `variable_extractor.py`: Analizza il codice Python tramite Abstract Syntax Trees (AST) per identificare le definizioni e gli utilizzi delle variabili all'interno di una funzione. Fornisce metodi per estrarre le variabili dichiarate e tracciare il loro uso, associandole ai relativi nodi AST per facilitare l'analisi statica del codice.

2.1.3 Package `components`

Include i moduli responsabili delle operazioni principali di analisi e rilevamento.

- `inspector.py`: Analizza il codice Python per individuare code smells utilizzando l'analisi statica basata su Abstract Syntax Trees (AST). Coordina l'estrazione di informazioni relative a variabili, DataFrame, modelli di Machine Learning e operazioni sui tensori, applicando regole di rilevamento specifiche per identificare pattern problematici nel codice.

- `project_analyzer.py`: Gestisce l'analisi di interi progetti Python, coordinando l'elaborazione dei file sorgente, l'esecuzione parallela o sequenziale dell'analisi e la generazione di report sui code smells rilevati. Si interfaccia con il modulo `Inspector` per eseguire le verifiche sui singoli file.
- `project_repository_cloner.py`: Si occupa della clonazione e gestione dei repository di progetti da analizzare, utilizzando un dataset di repository Machine Learning per filtrare quelli più rilevanti in base a metriche come numero di stelle e commit. Fornisce funzionalità per la pulizia e la configurazione della directory di progetti.
- `rule_checker.py`: Applica regole di rilevamento dei code smells al codice sorgente utilizzando l'analisi AST. Coordina l'esecuzione delle regole specifiche per API di Machine Learning e quelle generali, aggregando i risultati e fornendo un report dettagliato sulle problematiche individuate.

2.1.4 Package `detection_rules`

Definisce le regole di rilevamento dei *code smells*, suddivise in due sotto-package.

- `smell.py`: Superclasse che definisce la struttura base di un *code smell*.

Sotto-package `api_specific_smells`

Contiene le regole specifiche per il rilevamento di errori nelle API di Machine Learning.

- `chain_indexing_smell.py`: Identifica l'uso della *chain indexing* in Pandas, che può causare problemi di prestazioni e ambiguità nell'accesso ai dati. Suggerisce l'uso di `.loc` o `.iloc` per una manipolazione più efficiente dei DataFrame.
- `dataframe_conversion_api_misused.py`: Rileva l'uso improprio dell'attributo `values` nei DataFrame di Pandas, deprecato in alcune versioni e poco chiaro nel tipo di ritorno. Consiglia l'uso di NumPy o metodi più espliciti per la conversione.
- `gradients_not_cleared_before_backward_propagation.py`: Verifica che i gradienti vengano azzerati con `zero_grad()` prima della chiamata a `backward()` nei loop di addestramento di PyTorch, evitando errori e comportamenti indesiderati.
- `matrix_multiplication_api_misused.py`: Individua l'uso improprio di `dot()` per la moltiplicazione di matrici in NumPy, suggerendo l'uso di `matmul()` o dell'operatore `@` per maggiore chiarezza e compatibilità con array multidimensionali.
- `pytorch_call_method_misused.py`: Identifica chiamate dirette a `forward()` nei modelli PyTorch, pratica sconsigliata. Suggerisce di invocare il modello direttamente tramite la sua istanza per garantire il corretto funzionamento delle forward hooks.
- `tensor_array_not_used.py`: Rileva l'uso improprio di `tf.constant()` per creare array modificabili all'interno di un ciclo, suggerendo invece l'uso di `tf.TensorArray` per una gestione più efficiente delle operazioni iterative in TensorFlow.

Sotto-package `generic_smells`

Contiene le regole generali per il rilevamento di *code smells* comuni.

- `broadcasting_feature_not_used.py`: Identifica l'uso non necessario della funzione `tf.tile()` in TensorFlow per replicare tensori, suggerendo invece l'uso del *broadcasting*, che migliora l'efficienza computazionale e l'uso della memoria.

- `columns_and_datatype_not_explicitly_set.py`: Rileva l'assenza della specifica esplicita del parametro `dtype` nelle operazioni di creazione di DataFrame o lettura da CSV in Pandas, il che potrebbe portare a comportamenti inattesi.
- `deterministic_algorithm_option_not_used.py`: Verifica se la funzione `torch.use_deterministic_algorithms(True)` viene utilizzata in PyTorch, avvertendo che il suo utilizzo può comportare una significativa perdita di prestazioni se non strettamente necessario.
- `empty_column_misinitialization.py`: Rileva inizializzazioni di colonne di un DataFrame Pandas con valori predefiniti come `0` o stringhe vuote, suggerendo invece l'uso di `NaN` per una gestione più appropriata dei dati mancanti.
- `hyperparameters_not_explicitly_set.py`: Identifica la creazione di modelli di Machine Learning senza l'impostazione esplicita di iperparametri, promuovendo una maggiore trasparenza e riproducibilità dei risultati.
- `in_place_apis_misused.py`: Analizza l'uso errato delle API di Pandas con il parametro `inplace`, segnalando i casi in cui viene erroneamente omesso o impostato a `False`, causando confusione nella gestione della memoria.
- `memory_not_freed.py`: Rileva casi in cui la memoria non viene esplicitamente liberata dopo la creazione di un modello TensorFlow all'interno di cicli, suggerendo l'uso di `tf.keras.backend.clear_session()` per evitare *memory leaks*.
- `merge_api_parameter_not_explicitly_set.py`: Verifica che le operazioni di `merge()` in Pandas includano parametri fondamentali come `how`, `on` e `validate`, per evitare comportamenti imprevisti nella fusione di DataFrame.
- `nan_equivalence_comparison_misused.py`: Individua confronti diretti con il valore `NaN` utilizzando operatori di uguaglianza o disuguaglianza, suggerendo invece l'uso della funzione `np.isnan()` per una corretta gestione dei valori NaN.
- `unnecessary_iteration.py`: Rileva iterazioni non necessarie sui DataFrame Pandas tramite metodi inefficienti come `iterrows()`, `itertuples()`, `apply()` e `applymap()`, suggerendo l'uso di operazioni vettorializzate per migliorare le prestazioni.

2.1.5 Package `gui`

Contiene i moduli per l'interfaccia grafica del tool.

- `code_smell_detector_gui.py`: Implementa l'interfaccia grafica per l'analisi dei code smells nei progetti Python. Fornisce un'interfaccia utente basata su Tkinter per la selezione dei parametri di analisi, la configurazione delle directory di input/output e l'esecuzione dell'analisi con supporto per elaborazione parallela e ripresa delle esecuzioni interrotte.
- `gui_runner.py`: Gestisce l'inizializzazione e l'esecuzione dell'interfaccia grafica del tool. Avvia l'applicazione GUI e mantiene il ciclo di esecuzione dell'interfaccia utente, permettendo agli utenti di interagire con il sistema in modo intuitivo.
- `textbox_redirect.py`: Fornisce una classe per il reindirizzamento dell'output della console alla finestra di testo della GUI. Consente la visualizzazione in tempo reale dei log e dei messaggi generati dal tool all'interno dell'interfaccia grafica.

2.1.6 Package `utils`

Include utility per la gestione di file e altre operazioni comuni.

- `file_utils.py`: Fornisce funzioni di utilità per la gestione dei file e delle directory, tra cui la pulizia delle cartelle di output, la ricerca di file Python all'interno di un progetto e l'unione dei risultati di analisi da più file CSV. Include anche funzionalità per la gestione di log di esecuzione e la scrittura sicura in ambienti multi-thread.

2.1.7 Package `report`

Gestisce la generazione dei report finali.

- `report_generator.py`: Genera report dettagliati sui code smells rilevati nei progetti analizzati. Supporta la creazione di riepiloghi generali, report specifici per progetto e visualizzazioni grafiche, oltre alla generazione di file CSV ed Excel contenenti i risultati dell'analisi. Include anche un'interfaccia a menu per permettere agli utenti di selezionare il tipo di report da produrre.

2.2 Web App

Il sistema include una Web App che consente l'interazione con il tool tramite un'interfaccia web.

La Web App è sviluppata in React e Node.js ed è integrata con il progetto attraverso un gateway che espone una serie di servizi per l'analisi del codice e la generazione di report.

2.2.1 Package `gateway`

Il gateway funge da punto di accesso centralizzato per la Web App, permettendo di instradare le richieste ai servizi sottostanti.

- `main.py`: Implementa un'API Gateway utilizzando FastAPI per instradare le richieste tra i vari servizi. Gestisce le richieste ai servizi di analisi AI, analisi statica e generazione di report, offrendo un'astrazione centralizzata per l'interazione con il backend.

2.2.2 Package `services`

I servizi della Web App sono suddivisi in tre moduli principali: AI Analysis, Static Analysis e Report Generation.

Modulo `AI Analysis Service`

Questo modulo si occupa del rilevamento di code smells utilizzando tecniche di intelligenza artificiale.

- `main.py`: Avvia il servizio di analisi AI tramite FastAPI, gestendo le richieste di rilevamento di code smells basate su modelli di machine learning.
- `routers/detect_smell.py`: Definisce gli endpoint per l'analisi del codice tramite AI. Contiene la logica per ricevere snippet di codice, validarne la sintassi ed eseguire il rilevamento dei code smells utilizzando il modello AI.
- `schemas/requests.py`: Definisce lo schema per le richieste di rilevamento dei code smells, specificando il formato dei dati in ingresso.
- `schemas/responses.py`: Definisce lo schema per le risposte fornite dal servizio di analisi AI, includendo i dettagli sui code smells rilevati.

- `utils/model.py`: Contiene la logica per interagire con il modello AI per il rilevamento dei code smells. Invia snippet di codice a un'API esterna per l'analisi e interpreta i risultati restituiti.

Modulo `Static Analysis Service`

Questo modulo si occupa del rilevamento dei code smells tramite tecniche di analisi statica del codice.

- `main.py`: Avvia il servizio di analisi statica tramite FastAPI, gestendo le richieste di rilevamento dei code smells basate sull'analisi statica del codice sorgente.
- `routers/detect_smell.py`: Definisce gli endpoint per l'analisi statica del codice. Riceve snippet di codice come input, esegue l'analisi statica e restituisce i code smells rilevati.
- `schemas/requests.py`: Definisce lo schema per le richieste di rilevamento dei code smells, specificando il formato del codice da analizzare.
- `schemas/responses.py`: Definisce lo schema per le risposte fornite dal servizio di analisi statica, includendo dettagli sui code smells rilevati, come funzione, linea e descrizione.
- `utils/static_analysis.py`: Implementa il processo di analisi statica utilizzando l'Inspector del sistema. Converte il codice ricevuto in un file temporaneo, esegue l'analisi e restituisce i risultati strutturati in base ai code smells individuati.

Modulo `Report Service`

Questo modulo gestisce la generazione e l'elaborazione dei report sui code smells rilevati nei progetti analizzati.

- `main.py`: Avvia il servizio di generazione report tramite FastAPI, gestendo le richieste per l'aggregazione e visualizzazione dei dati sui code smells.
- `routers/report.py`: Definisce gli endpoint per la generazione dei report. Riceve dati sui code smells rilevati nei progetti e restituisce report aggregati per l'analisi e la visualizzazione.
- `schemas/requests.py`: Definisce il formato delle richieste per la generazione dei report, includendo informazioni sui progetti analizzati e i relativi file e smells rilevati.
- `schemas/responses.py`: Definisce il formato delle risposte del servizio di reportistica, fornendo dati aggregati per la creazione di visualizzazioni e statistiche sui code smells.
- `utils/report_generator.py`: Implementa la logica per aggregare i risultati dell'analisi dei progetti e generare report strutturati. Utilizza Pandas per elaborare i dati e produrre un formato adatto alla visualizzazione e analisi.

2.2.3 Funzionamento e Interazione del Sistema Web

L'architettura della Web App prevede che le richieste provenienti dall'interfaccia sviluppata in React e Node.js siano inoltrate a un *gateway* basato su FastAPI, che agisce come punto di ingresso unico per i servizi sottostanti. Quando l'utente invia una richiesta (ad esempio per rilevare code smells o generare un report), il gateway verifica il tipo di operazione e smista la richiesta al servizio corrispondente:

1. **AI Analysis Service:** gestisce le analisi basate su modelli di machine learning. Una volta ricevuta la richiesta dal gateway, il servizio ne valida il contenuto, interagisce con il modello di AI per individuare code smells, quindi restituisce i risultati in formato JSON.

2. **Static Analysis Service:** realizza l'analisi statica del codice utilizzando l'`Inspector` per generare una lista di *smells* basati sulla struttura del codice. Il gateway inoltra richieste che contengono snippet di codice a questo servizio, il quale crea dinamicamente file temporanei, esegue le analisi e fornisce un elenco di code smells rilevati.

3. **Report Service:** si occupa di aggregare e combinare i risultati delle analisi ottenuti dai progetti o dai file analizzati. Tramite il gateway, il servizio riceve i dati di più progetti, procede all'elaborazione con Pandas, quindi restituisce report strutturati (ad esempio liste di code smells aggregati per progetto).

Dal punto di vista del flusso di lavoro, il gateway espone endpoint specifici (`/api/detect_smell_ai`, `/api/detect_smell_static`, `/api/generate_report`) che incapsulano la logica di instradamento verso i servizi appropriati. Dopo aver validato i parametri di input, il gateway crea una richiesta verso il microservizio corrispondente, attendendo in modo asincrono la risposta. Una volta ricevuta, questa viene aggregata o trasformata in base alle necessità del front-end, quindi inviata nuovamente al *client* React/Node.js per la visualizzazione o l'ulteriore elaborazione.

3 Flusso di Utilizzo del Tool

Il tool offre diverse modalità di utilizzo, adattandosi a scenari differenti a seconda delle esigenze dell'utente. In questa sezione vengono analizzati i principali flussi di utilizzo.

3.1 Utilizzo tramite Interfaccia a Linea di Comando (CLI)

L'utente può eseguire il tool attraverso un'interfaccia a linea di comando, specificando i parametri necessari per avviare l'analisi. Tramite CLI è possibile:

- Analizzare singoli file o interi progetti;
- Specificare directory di input e output;
- Configurare l'esecuzione in modalità sequenziale o parallela;

Questo approccio è utile per sviluppatori e ricercatori che desiderano eseguire analisi mirate in maniera rapida ed efficiente.

3.2 Utilizzo tramite Web App

La Web App fornisce un'interfaccia grafica intuitiva che consente di interagire con il tool senza dover scrivere comandi manualmente. Gli utenti possono:

- Caricare file o progetti direttamente dal browser;
- Visualizzare i risultati;
- Esportare report per analisi successive.

Questa modalità è ideale per utenti meno esperti o per contesti in cui è preferibile un'interazione più visuale.

3.3 Analisi AI-Based e Static Analysis

Il tool supporta due tipi di analisi:

- **Analisi AI-Based**: sfrutta modelli di Machine Learning per rilevare *code smells* attraverso un'analisi semantica del codice.
- **Analisi Static Analysis**: utilizza tecniche basate su AST (Abstract Syntax Tree) per individuare problemi specifici nelle API e nelle strutture di codice.

3.4 Analisi di Progetti Open Source

Il tool può essere utilizzato per analizzare repository open source, individuando problemi di qualità del codice e potenziali ottimizzazioni. È possibile:

- Clonare e analizzare automaticamente progetti da GitHub;
- Filtrare i progetti in base a criteri come numero di stelle o contributori attivi;
- Generare statistiche sulla diffusione di determinati *code smells*.

3.5 Generazione di Report e Visualizzazioni

Una volta completata l'analisi, il tool consente di generare report dettagliati contenenti:

- Riepiloghi dei *code smells* rilevati;
- Statistiche aggregate per progetto o file;

I report possono essere esportati in formati come CSV per ulteriori elaborazioni.