Descrizione Preliminare del Sistema

Nicola Tortora, Gaspare Galasso

June 10, 2025

Contents

1 Introduzione

La crescente diffusione delle blockchain e degli smart contract ha portato alla necessità di strumenti avanzati per l'analisi automatica del codice, in particolare per l'individuazione di vulnerabilità che potrebbero compromettere la sicurezza e l'integrità dei sistemi decentralizzati. In questo contesto, il presente progetto propone un sistema di vulnerability assessment automatico che combina la capacità dei Large Language Models (LLM) con tecniche di Retrieval-Augmented Generation (RAG).

Questa prima versione del sistema è focalizzata esclusivamente su contratti scritti per la blockchain Algorand (Pyteal), ma la metodologia è stata progettata con un'architettura modulare e scalabile, che consentirà in futuro l'estensione ad altre piattaforme come Solana.

2 Panoramica

Il sistema ha come obiettivo l'analisi automatica di smart contract per l'identificazione delle vulnerabilità. Per ottenere una valutazione più accurata e ridurre il numero di falsi positivi, il sistema combina due approcci complementari: da un lato, l'analisi diretta del codice mediante modelli linguistici avanzati (LLM), dall'altro l'uso di un motore di retrieval semantico per il recupero di contratti simili, con vulnerabilità note, da un database vettoriale. Le informazioni dei contratti simili vengono poi fornite in contesto agli LLM per un analisi più accurata e risposte più corrette.

Il flusso generale prevede tre fasi principali:

- **Analisi Generale del Codice**: identificazione preliminare di vulnerabilità sospette tramite LLM.
- **Retrieval Semantico**: il codice sotto test viene convertito in un *embedding vector* e confrontato con una base di conoscenza di codice vulnerabile.
- **Analisi Dettagliata**: per ciascuna vulnerabilità sospetta, viene effettuata un'analisi mirata utilizzando anche il contesto derivato dal retrieval.

Ogni componente del sistema è stato progettato per operare in maniera modulare, facilitando la manutenzione, l'estensione e l'integrazione con strumenti futuri.

2.1 Architettura generale

Lo schema in Figura 1 spiega il funzionamento dei componenti principali (retrieval, analisi iniziale, analisi specifica).

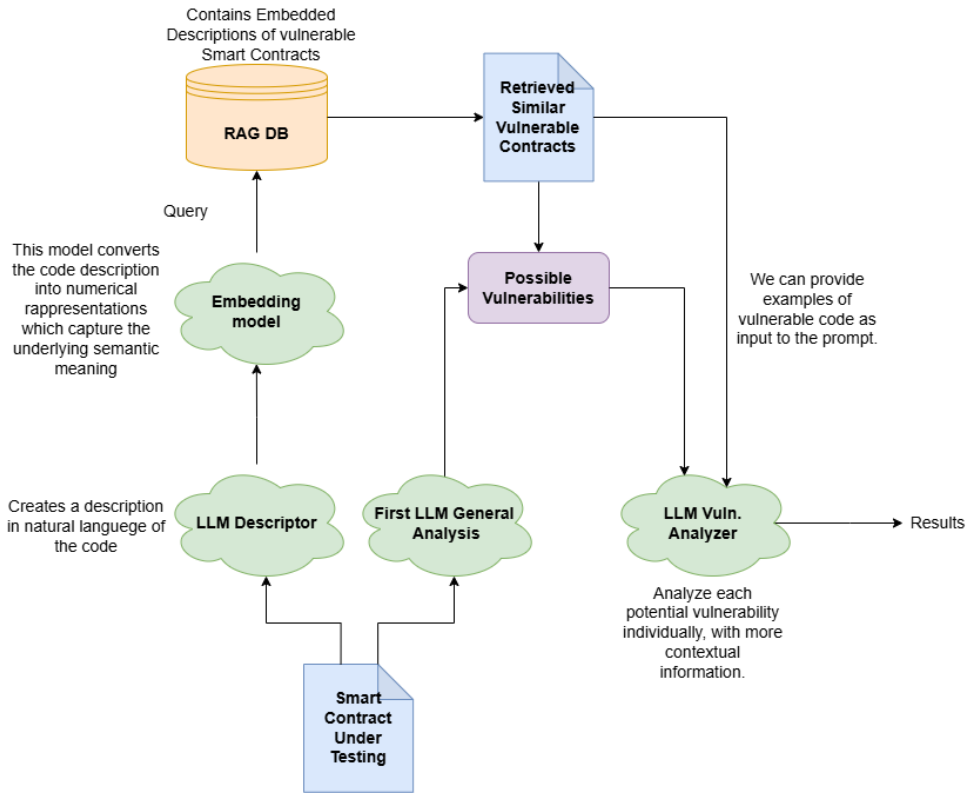

Figure 1: Architettura generale del sistema

2.2 Tecnologie Utilizzate

Linguaggi e Framework

- **Python**: linguaggio di programmazione principale per lo sviluppo di tutti i moduli del sistema.
- **FastAPI**: framework leggero per la realizzazione di API RESTful, utilizzate per esporre il servizio del database.

Modelli e LLM

- **Large Language Models (LLM)**: modelli linguistici avanzati (come Deepseek-V3 o equivalenti open-source).
- **Modelli di embedding**: modelli come `BAAI/bge-codehkunlp/instructor-xl` impiegati per generare rappresentazioni vettoriali del codice in uno spazio semantico.

Database

- **Qdrant**: motore di ricerca vettoriale utilizzato per effettuare operazioni di retrieval semantico basate su similarità tra embedding.

Strumenti di Supporto

- **Docker**: utilizzato per il deploy del database vettoriale *Qdrant*. L'uso di container Docker consente l'avvio rapido del servizio,

3 Requisiti del Sistema

Il sistema proposto per l'analisi automatica di smart contract PyTeal è stato progettato secondo un insieme di requisiti funzionali e non funzionali che ne guidano l'implementazione, la valutazione e l'evoluzione futura. In questa sezione vengono elencati e descritti in dettaglio.

3.1 Requisiti Funzionali

I requisiti funzionali definiscono il comportamento atteso del sistema e le operazioni che ciascun modulo deve essere in grado di svolgere.

1. Analisi Generale del Codice

- **RF1.1** – Il sistema deve accettare in input uno smart contract..
- **RF1.2** – Il sistema deve invocare un LLM per generare un'analisi generale del codice.
- **RF1.3** – Il sistema deve identificare un insieme di potenziali vulnerabilità nel codice.

2. Sistema di Retrieval

- **RF2.1** – Il sistema deve generare una descrizione in linguaggio naturale del codice tramite LLM.
- **RF2.2** – Il sistema deve convertire la descrizione testuale in un embedding vettoriale.
- **RF2.3** – Il sistema deve confrontare il vettore con un database vettoriale Qdrant contenente contratti vulnerabili.
- **RF2.4** – Il sistema deve recuperare i K contratti semanticamente più simili.
- **RF2.5** – Il sistema deve includere, alle potenziali vulnerabilità, anche quelle presenti nei contratti simili recuperati.

3. Analisi Dettagliata per Vulnerabilità

- **RF3.1** – Il sistema deve analizzare ogni vulnerabilità sospetta rilevata.
- **RF3.2** – Il sistema deve costruire un prompt specializzato con il codice e il contesto simile.
- **RF3.3** – Il sistema deve inviare il prompt a un LLM per un'analisi approfondita della vulnerabilità.
- **RF3.4** – Il sistema deve classificare la vulnerabilità come vera o falsa positiva sulla base della risposta.

4 Design del Sistema

Il sistema è progettato secondo un'architettura modulare, che consente elevata manutenibilità, testabilità e possibilità di estensione.

4.1 Architettura

L'architettura del sistema si articola in cinque macro-componenti:

- **User Interface (UI)**: permette all'utente di avviare l'analisi di un contratto smart.
- **Analysis Modules**: include i moduli `ModelAnalysis` e `VulnAnalysis` per l'analisi iniziale e approfondita.
- **Retrieval Module**: si occupa della descrizione testuale, embedding e retrieval semantico.
- **LLM API**: interfaccia di comunicazione con i modelli linguistici.
- **Vector DB (Qdrant)**: database vettoriale contenente contratti vulnerabili noti.

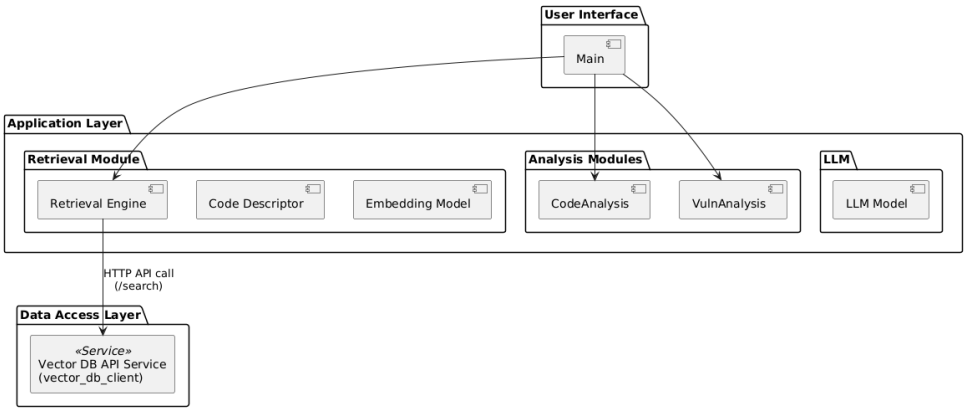

Figure 2: Architettura del sistema

4.2 Sequence Diagram

La Figura mostra il diagramma di sequenza del sistema, evidenziando le principali interazioni tra i componenti durante l'esecuzione dell'analisi.

1. L'utente carica uno smartcontract tramite la UI.
2. Il modulo `RetrievalModule` genera una descrizione naturale tramite LLM, calcola l'embedding e recupera contratti simili da `VectorDB`.
3. Il modulo `CodeAnalysis` analizza il codice e restituisce vulnerabilità sospette.
4. Per ogni vulnerabilità, `VulnAnalysis` costruisce un prompt RAG e chiede al LLM di classificare il rischio.

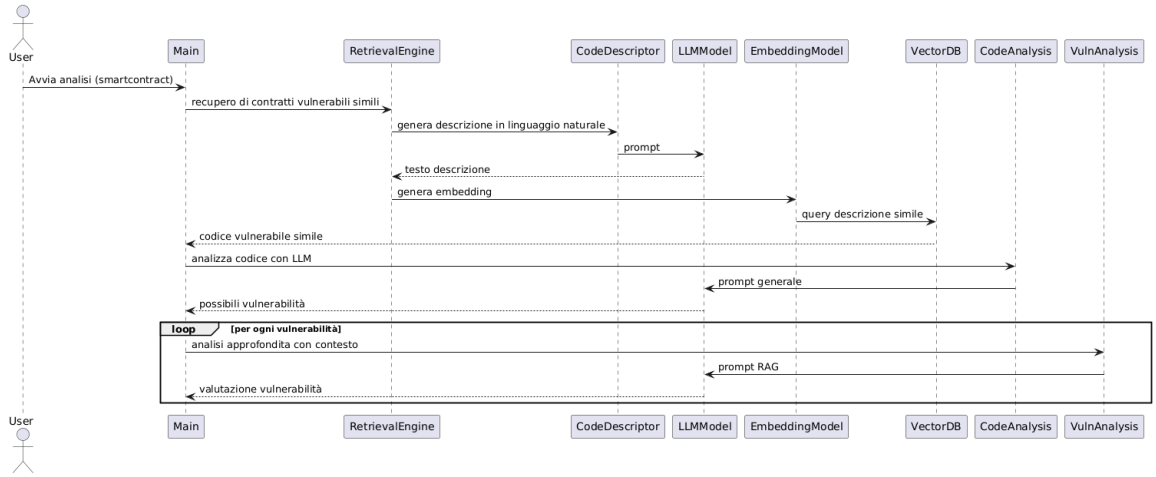

Figure 3: Diagramma di sequenza delle interazioni tra i moduli

4.3 Class Diagram

La Figura descrive la struttura ad oggetti del sistema, evidenziando le relazioni tra le classi.

- **RetrivalEngine**: È il motore principale del sistema di retrieval. Coordina la generazione della descrizione testuale del codice, la sua conversione in embedding e l'invocazione del servizio remoto per trovare i contratti simili.
- **CodeDescriptor**: Trasforma il codice in una descrizione in linguaggio naturale tramite un modello LLM.
- **EmbeddingModel**: Converte un testo in un embedding numerico usando SentenceTransformer.
- **LLMModel** (Classe astratta): Definisce l'interfaccia comune per i modelli LLM. Sottoclassi:
  - **OpenAILLM**: usa API OpenAI (es. DeepSeek).
  - **HFLLM**: usa modelli HuggingFace (es. LLaMA).
- **CodeAnalysis**: Analizza lo smartcontract tramite un LLM per trovare le possibili vulnerabilità
- **VulnAnalysis**: Analizza lo smartcontract su vulnerabilità singole con prompt RAG (arricchiti con informazioni di contesto)

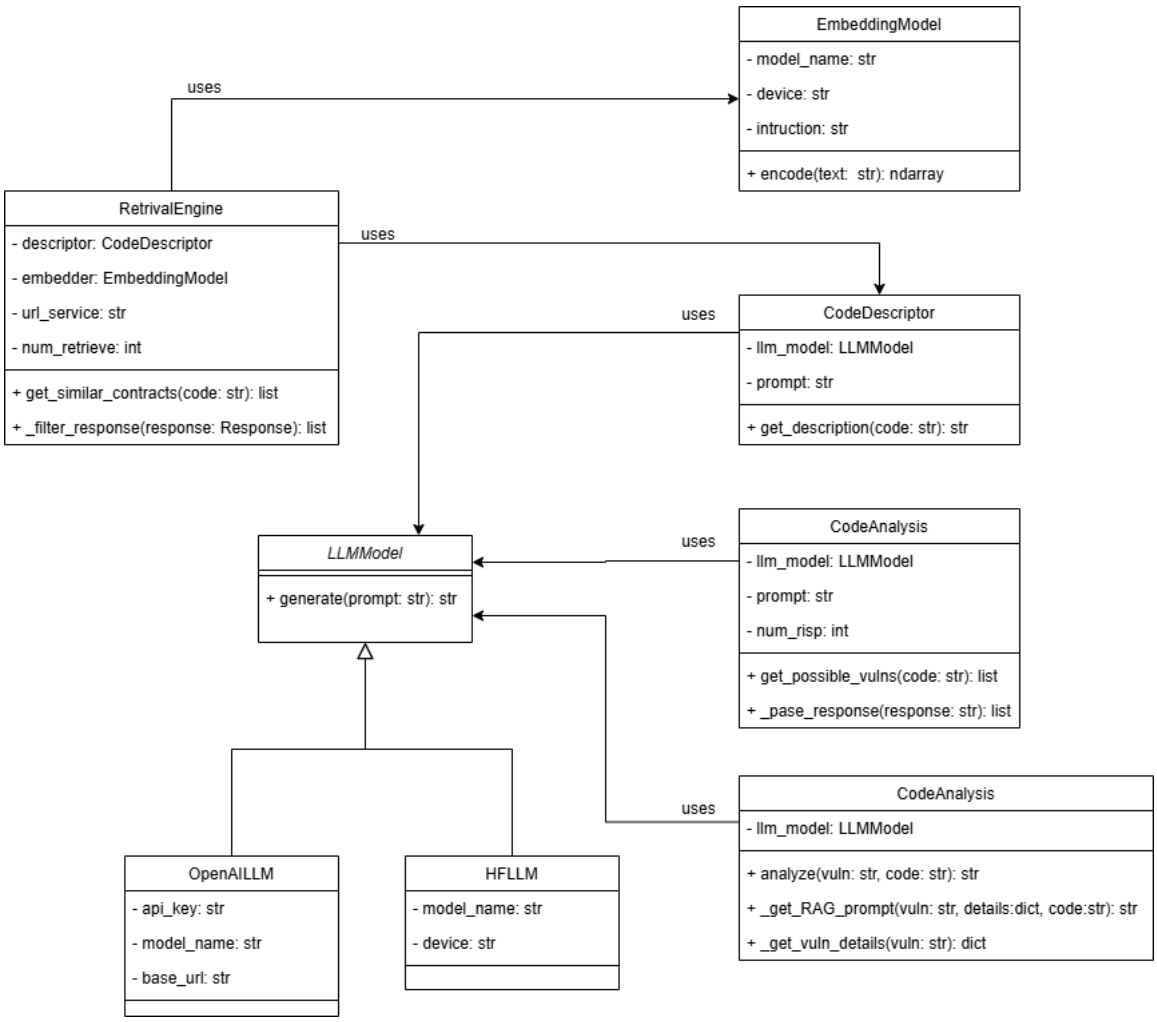

Figure 4: Diagramma delle classi