73.64

Temas Avanzados en Deep Learning

2º Cuatrimestre del 2024

Trabajo Práctico 2

Grupo 3

Integrantes:

- **Ezequiel Pérez** **61475**
- **Gonzalo Baliarda** **61490**

Contexto

El objetivo del trabajo práctico es implementar una herramienta que haga uso de RAG para generar tests automatizados para contratos inteligentes escritos en Solidity. Los tests generados deberán poder ser ejecutados con el framework de desarrollo Foundry.

Responsible AI y Safety

Si bien la salida generada por nuestra herramienta debería ser código Solidity y no debería afectar a los usuarios en un sentido moral y ético, es importante asegurarnos de que el modelo de lenguaje (LLM) no responda con otros formatos que no sean el deseado.

Por otro lado, debemos asegurarnos de que el LLM genere tests de buena calidad para detectar posibles errores en el código, que de otra forma podrían resultar en pérdidas monetarias para los usuarios. Sin embargo, debemos aclarar que detectar todos los errores es imposible, y los usuarios deberían tenerlo presente para complementar con otros métodos de detección de vulnerabilidades.

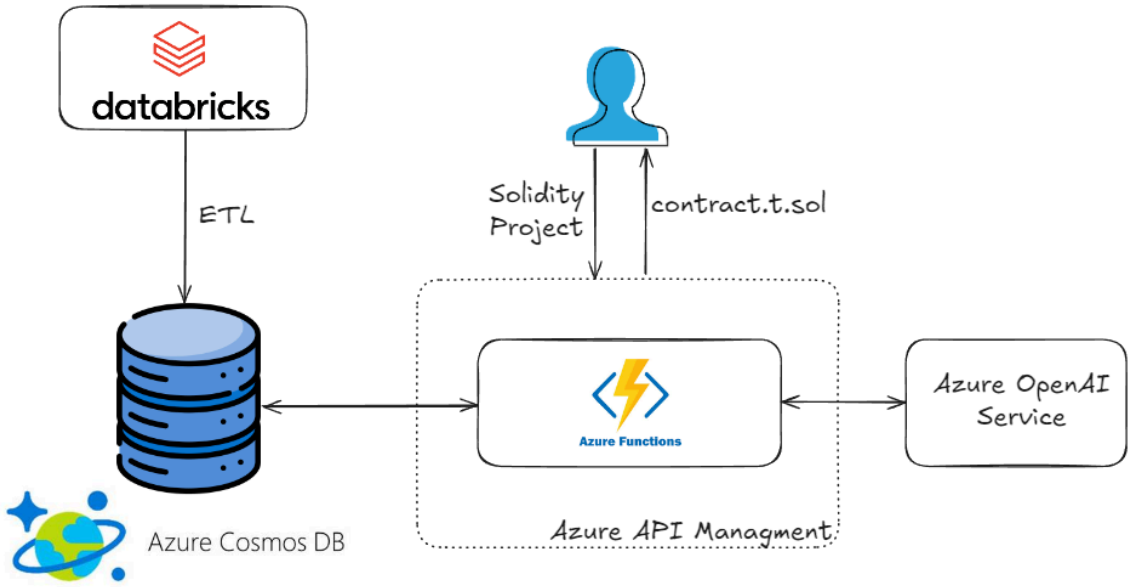
Finalmente, debemos asegurarnos de respetar las licencias de todos los códigos que ingresen a nuestro sistema, ya sea para la ingesta de datos en la base de datos vectorial, o a la hora de realizar la inferencia sobre código provisto por los usuarios.

Arquitectura y puesta en Producción

El funcionamiento de nuestra herramienta se puede separar en dos etapas. La primera es la creación de tests a partir del código de un contrato inteligente, y la segunda es un proceso de corrección iterativo para lograr que todos los tests devueltos por la herramienta sean compilables sin necesidad de intervención humana. Si bien la herramienta podría correr de forma local, a continuación proponemos una arquitectura montada en Microsoft Azure para poner el sistema en producción y ofrecerlo como un SaaS.

Dado que nuestra herramienta está basada en un sistema de RAG, necesitaremos una base de datos vectorial. Para esto elegimos **Azure Cosmos DB**, la cual soporta búsquedas vectoriales y permite ingesta y consulta de datos en tiempo real, además de escalabilidad, alta disponibilidad, fiabilidad y seguridad. El proceso de ETL para cargar esta base de datos estará hosteado en **Azure Databricks**, un servicio especializado para este caso de uso que además provee soporte para Jupyter Notebooks colaborativas.

Para ejecutar la lógica de negocio de la herramienta, implementaremos una API REST serverless basada en **Azure Functions** y **Azure API Management**, servicios que nos brindarán escalabilidad y alta disponibilidad a la vez que nos ahorran la necesidad de implementar un servidor propio. A su vez, haremos uso del **Azure OpenAI Service**, el cual facilita el acceso a modelos de lenguaje (e.g., gpt-4o) y embeddings (e.g., text-embedding-3-large) de OpenAI.


databricks
ETL
Solidity
Project
contract.t.sol
Azure Functions
Azure OpenAI
Service
Azure API Managment
Azure Cosmos DB