Damien Ouzillou

Blockchain Intelligence — Analyse de Smart Contracts par LLM

1. Choix de conception

L'objectif principal du projet Blockchain Intelligence est de permettre à un utilisateur d'interagir avec un contrat intelligent Ethereum à l'aide du langage naturel, tout en garantissant une exécution locale, sans dépendance à une API externe d'intelligence artificielle.

Le projet repose sur cette architecture :

- Ingest_contract.py : récupération automatique du code source et de l'ABI d'un contrat depuis Etherscan via son adresse Ethereum. Ces données sont ensuite stockées localement pour éviter les appels redondants.
- Parse_source.py : extraction syntaxique des éléments clés du code Solidity (fonctions, modificateurs, variables d'état, héritages, visibilités…). Cette étape repose sur des expressions régulières (regex) et permet d'obtenir une représentation JSON normalisée du contrat.
- Analyze_contract.py : analyse de l'ABI pour produire un résumé fonctionnel du contrat, notamment les fonctions publiques, les événements, et la logique d'accès.
- Query_assistant.py : cœur de l'intelligence du projet. Ce module combine :
    - Un moteur de règles simples pour répondre instantanément à des questions récurrentes (ex. « What functions can be called by anyone ? »)
    - Un moteur d'analyse sémantique basé sur un LLM local (Phi-3 via Ollama) pour les questions nécessitant du raisonnement contextuel. Le choix du LLM local était pour ne pas devoir payer comme cela aurait été le cas avec l'API OpenAI par exemple. En plus, Phi-3 est assez léger pour un LLM (environ 2Go) comparé à d'autres qui sont plus lourds (Llama 3 pesait quasiment 5Go, cependant il aurait peut-être été plus fort que Phi-3). Enfin, comme j'utilise un LLM local, aucune donnée n'est transmise à un service externe comme ça aurait pu être le cas avec un LLM via une API comme OpenAI par exemple.
- Main.py : interface utilisateur en ligne de commande. Elle orchestre le chargement du contrat, la génération du résumé, et la boucle interactive de questions/réponses.

2. Limites actuelles

Le projet présente quelques limitations techniques :

- Le parsing est basé sur des expressions régulières (regex) : c'est très rapide et efficace mais assez fragile face à des codes sources Solidity complexes (héritages multiples, bibliothèques externes…). C'est assez difficile de gérer tous les cas possibles avec des regex, mais c'est la solution qui me semblait la plus simple à mettre en place avec l'objectif de temps fixé pour avoir un PoC.
- Le modèle analyse un contrat isolé, que j'obtiens sur la base de son adresse. S'il y a des dépendances entre contrats, ou tout simplement des liens inter-contractuels (j'imagine que ça doit exister), mon algorithme ne sait pas le gérer.
- Mon code lit le code statiquement, il n'aborde pas le code comme le ferait un réel flux d'exécution avec les appels de fonctions etc.

- J'utilise le LLM Phi-3, qui n'est probablement pas le meilleur car léger et open-source (et c'est pour cela que je l'ai choisi en grande partie). Il me paraît probable que j'aurais obtenu des meilleures réponses avec Mistral, OpenAI etc.
- Le modèle me répond en langage naturel, mais il n'y a pas de formatting particulier. J'imagine que ça pourrait être utile pour intégrer le modèle à d'autres outils.

3. Pistes d'évolution et extentions potentielles

Avec plus de temps et de moyens, j'aurais selon moi pu faire un meilleur projet sur la base de plusieurs axes :

- Sur le plan technique :
    - D'abord, j'aurais probablement pu améliorer mon parsing (plutôt que d'utiliser des regex assez difficiles à écrire et lire), il doit y avoir de meilleurs outils pour cela.
    - Je pourrais aussi faire en sorte de pouvoir écrire du code pour que le modèle soit capable de répondre à des questions sur plusieurs contrats à la fois.
- Améliorer le LLM : potentiellement, il pourrait être pertinent de laisser l'utilisateur choisir le modèle de LLM qu'il veut parmi une liste, mais j'étais assez bridé à ce sujet. J'aurais aussi sûrement pu faire du meilleur prompt engineering, et gérer des edges cases que je n'avais pas anticipées. Je me suis essentiellement basé sur le dataset « ERC20 Token (baseline) **:** USDC » dans mon travail.
- Expérience utilisateur : pour faire un travail plus propre, avec plus de temps j'aurais pu développer un petit front-end pour visualiser les fonctions, events… et poser des questions de façon plus ludique et agréable que dans un terminal. J'aurais aussi pu imaginer aller sur d'autres blockchains avec des APIs compatibles Etherscan. Je me suis concentré exclusivement sur Ethereum Mainnet.