

UnitGPT

Generación de tests mediante RAG

- Baliarda, Gonzalo
- Pérez, Ezequiel

Objetivos

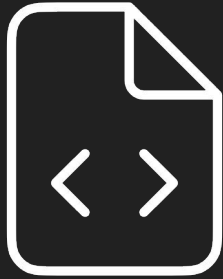
- Generación automática de unit tests para contratos inteligentes escritos en Solidity.
- Compilables en el framework de desarrollo Foundry.



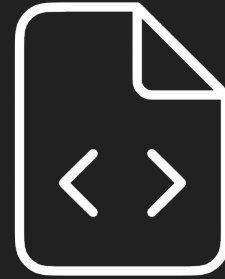
Dataset

Dataset

Contratos inteligentes con sus respectivos tests.



MyContract.sol



MyContract.t.sol

Dataset

Emparejar funciones con sus respectivos tests.



```
function transfer(address to, uint256 amount) public {  
    // ...  
}
```

MyContract.sol



```
function testTransfer() public {  
    // ...  
}
```

MyContract.t.sol

Dataset

Generar descripción de las funciones.



```
function transfer(address to, uint256 amount) public {  
    // ...  
}
```

MyContract.sol

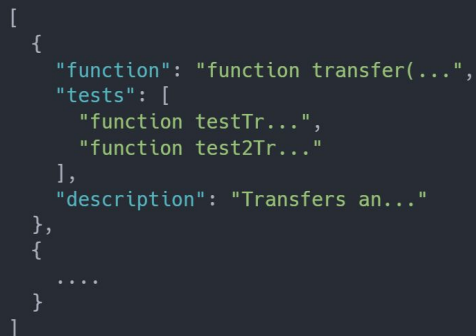


llama3.2:3b

Transfers a specified amount from one account to another. It reduces the sender's account balance [...]

Dataset

Combinamos funciones, tests y descripciones.



```
[
  {
    "function": "function transfer(...",
    "tests": [
      "function testTr...",
      "function test2Tr..."
    ],
    "description": "Transfers an..."
  },
  {
    ....
  }
]
```

dataset.json

Vector Database

Vector Database

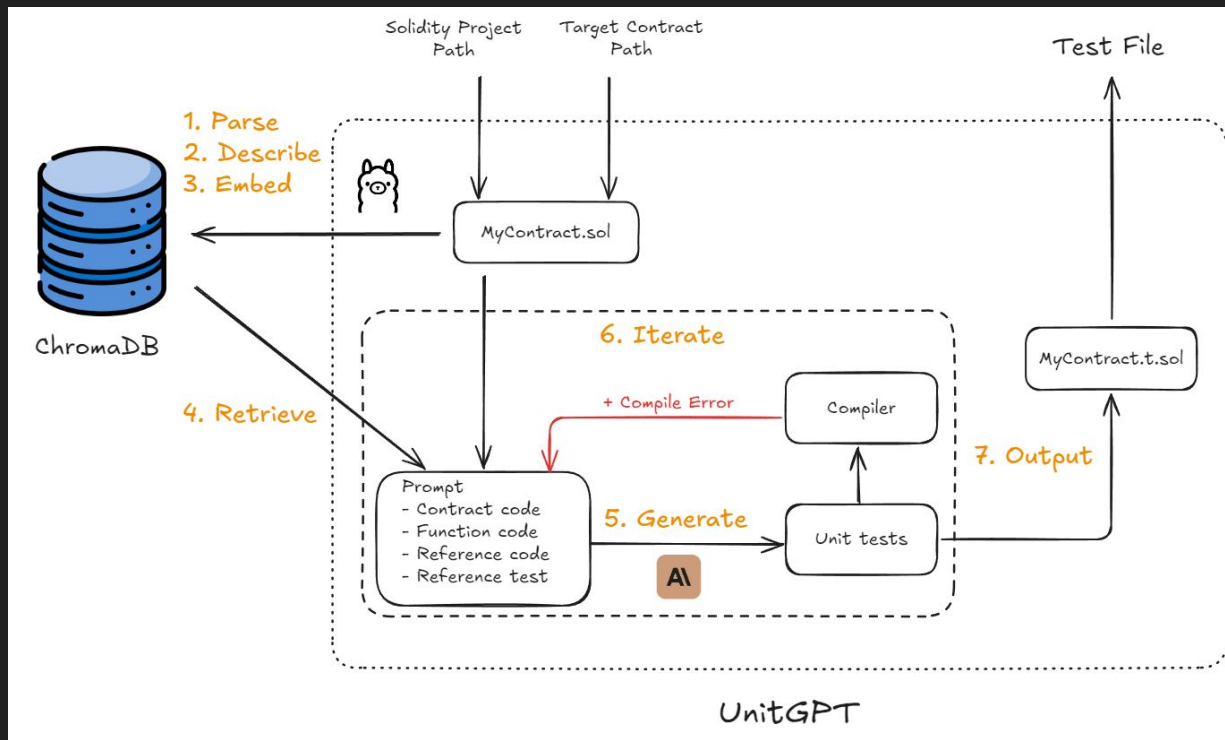
ChromaDB

- **Embeddings:** descripciones \longrightarrow *mxbai-embed-large:335m (Ollama)*
- **Metadata:** función y tests
- **Distancia:** L2 (Euclidean Distance Squared)

Distance	parameter	Equation
Squared L2	<code>l2</code>	$d = \sum (A_i - B_i)^2$
Inner product	<code>ip</code>	$d = 1.0 - \sum (A_i \times B_i)$
Cosine similarity	<code>cosine</code>	$d = 1.0 - \frac{\sum (A_i \times B_i)}{\sqrt{\sum (A_i^2)} \cdot \sqrt{\sum (B_i^2)}}$

Arquitectura

Arquitectura



Desafíos

Desafíos

- Modelos chicos locales no dan buenos resultados.
- Generación del dataset.
- Prompt engineering.

Resultados

Métricas

- Cantidad de tests compilables
- Cantidad de fuzzing tests
- Cantidad / Tipo de errores de compilación
- Calidad y relevancia de los tests

RAG vs NO RAG

- OpenZeppelin's ERC20, 12 funciones públicas
- Claude 3 Haiku, *temperature* = 0.8
- RAG (*K* = 1, *subtests* = 1), *recompile_tries* = 3

CON RAG

4 tests passed, 8 failed, 0 skipped (12 total tests)

- 7 fuzzing tests
- 5 errores de zero address
- 0 errores de compilación

SIN RAG

7 tests passed, 2 failed, 0 skipped (9 total tests)

- 0 fuzzing tests
- Uso de assert
- Errores críticos en tests
- 8 errores de compilación (Undeclared identifier)

K y Subtests

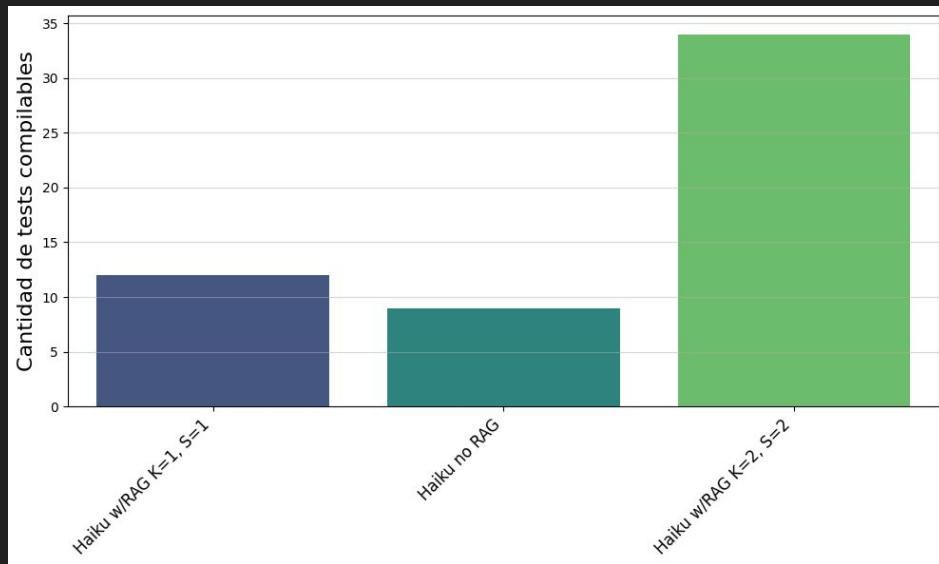
- OpenZeppelin's ERC20, 12 funciones públicas
- Claude 3 Haiku, *temperature* = 0.8
- RAG (*K* = 2, *subtests* = 2), *recompile_tries* = 3

```
22 tests passed, 12 failed, 0 skipped (34 total tests)
```

- Múltiples signatures de funciones
- Errores de compilación: variables y funciones no declaradas, conversión de tipo

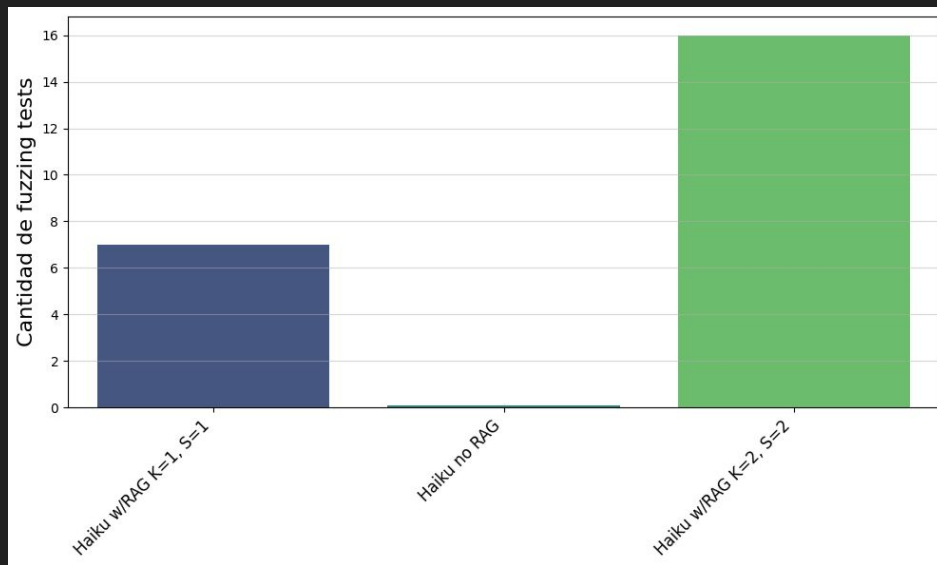
Resultados

- OpenZeppelin's ERC20, 12 funciones públicas
- Claude 3 Haiku, *temperature* = 0.8



Resultados

- OpenZeppelin's ERC20, 12 funciones públicas
- Claude 3 Haiku, *temperature* = 0.8



Generalización

- *Fallback*, Ethernaut CTF, 4 funciones públicas
- Claude 3 Haiku, *temperature* = 0.8
- RAG (*K* = 1, *subtests* = 1), *recompile_tries* = 3

CON RAG

```
0 tests passed, 4 failed, 0 skipped (4 total tests)
```

- 1 fuzzing test
- 0 errores compilación
- tests incorrectos

SIN RAG

```
0 tests passed, 3 failed, 0 skipped (3 total tests)
```

- 0 fuzzing test
- 1 error de compilación
(unexpected function call)
- tests incorrectos

Errores comunes

- Mint/Approve a la dirección 0
- Reducir allowance a menor que 0
- Allowance insuficiente
- Reverts no llamados
- Falta de *vm.prank*
- Emisión de eventos no declarados (compilación)

Conclusiones

Conclusiones

- El uso de RAG brinda varias ventajas
 - Reduce errores de compilación
 - Permite el uso de más features del framework de testing
 - Brinda inspiración para variedad de tests
- No se generalizan bien las ventajas
- El prompt es importante
- Modelos chicos todavía no sirven para tareas complejas

Demo

Muchas Gracias

¿Preguntas?