

Code Analysis with Chatgpt

胡晓文

June 1, 2023

A brief introduction of my first attempt on code analysis using ChatGPT.

1. What is code analysis.
2. Brief introduction on Vectorstore Database and Langchain Library.
3. Examples and Case Study.

Code Analysis

- Examine property of source code without executing them.

Code Analysis

- Examine property of source code without executing them.
- Checking for a wide range of issues such as
 - syntax errors
 - security vulnerabilities
 - performance inefficiencies, and adherence to coding conventions.

Code Analysis

- Examine property of source code without executing them.
- Checking for a wide range of issues such as
 - syntax errors
 - security vulnerabilities
 - performance inefficiencies, and adherence to coding conventions.
- Hard! Trade-off between accuracy and efficiency.
 - Points-to Analysis.
 - Difficult because Aliasing, indirect assignment, dynamic nature, external libraries.
 - Limit context search, abstract interpretation, etc.

LLM and Code Analysis

- Code generation with LLM usually focuses on simple, self-contained function with small amount of context.
- Code analysis usually require larger context and documentation. ¹
- Current tools use LLM simply as reasoning and assistant model. ^{2 3 4}

¹Improving Few-shot Prompts with Relevant Static Analysis Products

²<https://socket.dev/>

³<https://github.com/anc95/ChatGPT-CodeReview>

⁴<https://useadrenaline.com/app>

Potential Approach:

- Fine tuning with documentation and method of code analysis.
 - Expensive and require tons of training data.

Potential Approach:

- Fine tuning with documentation and method of code analysis.
 - Expensive and require tons of training data.
- Feed directly with all the documentation and code examples.
 - 8k token limits on gpt4. (In reality even less...)
 - Non-relevant information confuses GPT.
 - Token Compression

Potential Approach:

- Fine tuning with documentation and method of code analysis.
 - Expensive and require tons of training data.
- Feed directly with all the documentation and code examples.
 - 8k token limits on gpt4. (In reality even less...)
 - Non-relevant information confuses GPT.
 - Token Compression
- Hard training by manual pick relevant information
 - Tedious and can't be general

Potential Approach:

- Fine tuning with documentation and method of code analysis.
 - Expensive and require tons of training data.
- Feed directly with all the documentation and code examples.
 - 8k token limits on gpt4. (In reality even less...)
 - Non-relevant information confuses GPT.
 - Token Compression
- Hard training by manual pick relevant information
 - Tedious and can't be general
- Tradeoff between tokens, precision, and manual effort
 - Documentation and code examples are stored as VectorStore DB
 - Langchain and DeepLake for retrieving relevant context based on user query.
 - Feed context into ChatGPT.

VectorStore is a powerful database system for efficient storage and retrieval of information using vector representations.

- Preprocessing:
 - Remove stop words and punctuation.
 - Apply stemming or lemmatization. (running -> run)

VectorStore

VectorStore is a powerful database system for efficient storage and retrieval of information using vector representations.

- Preprocessing:
 - Remove stop words and punctuation.
 - Apply stemming or lemmatization. (running -> run)
- Word Embedding:
 - Encode words into high-dimensional vectors using techniques like Word2Vec or GloVe.

VectorStore is a powerful database system for efficient storage and retrieval of information using vector representations.

- Preprocessing:
 - Remove stop words and punctuation.
 - Apply stemming or lemmatization. (running -> run)
- Word Embedding:
 - Encode words into high-dimensional vectors using techniques like Word2Vec or GloVe.
- Document Encoding:
 - Combine word vectors to form document embeddings.
 - Use averaging or methods like Doc2Vec.
- Indexing:
 - Index document embeddings for fast querying and similarity calculations.

Extracting Relevant Information:

- Query Encoding:
 - Preprocess query string (remove stop words, punctuation, etc.).
 - Apply stemming or lemmatization.

Extracting Relevant Information:

- Query Encoding:
 - Preprocess query string (remove stop words, punctuation, etc.).
 - Apply stemming or lemmatization.
- Vector Representation:
 - Transform the processed query into a vector using word embedding.

Extracting Relevant Information:

- Query Encoding:
 - Preprocess query string (remove stop words, punctuation, etc.).
 - Apply stemming or lemmatization.
- Vector Representation:
 - Transform the processed query into a vector using word embedding.
- Similarity Calculation:
 - Compare the query vector with document vectors using cosine similarity.
 - Measure the similarity between vectors in the vector space.

Extracting Relevant Information:

- Query Encoding:
 - Preprocess query string (remove stop words, punctuation, etc.).
 - Apply stemming or lemmatization.
- Vector Representation:
 - Transform the processed query into a vector using word embedding.
- Similarity Calculation:
 - Compare the query vector with document vectors using cosine similarity.
 - Measure the similarity between vectors in the vector space.
- Retrieval:
 - Retrieve documents with highest cosine similarity scores.
 - Provide the most relevant results to the user.

Prompt:

I would like you to simulate a static code analysis system. Your task is to scrutinize various code snippets for potential security vulnerabilities. These might include, but are not limited to, memory leaks, buffer overflows, integer overflows, and unchecked return values. For each code snippet provided, please identify and thoroughly explain any detected security issues, elaborating why they pose a risk. Additionally, recommend appropriate modifications or best practices to mitigate these identified vulnerabilities.

Raw Performance

Input: `simple_udt.c` (no overflow checks)

Response:

... Integer Overflow: It's possible that the addition of `input_amount` and `output_amount` may result in integer overflow, as both are `uint128_t` types. Although it's highly unlikely to reach maximum value of `uint128_t`, it's good practice to check for overflow. ...

Solution: Check for integer overflow when adding amounts.

```
if (UINT128_MAX - input_amount < current_amount) {  
    return ERROR_OVERFLOW;  
}  
input_amount += current_amount;
```

Raw Performance

Input: `simple_udt.c` (buffer overflow)

Response:

... Buffer overflow: There is a potential buffer overflow vulnerability in the following line:

```
if (memcmp(buffer, args_bytes_seg.ptr, BLAKE2B_BLOCK_SIZE + 1) == 0)
```

The `memcmp` function is comparing `BLAKE2B_BLOCK_SIZE + 1` bytes, but both `buffer` and `args_bytes_seg.ptr` are of size `BLAKE2B_BLOCK_SIZE`. This may lead to a buffer overflow and cause undefined behavior or even security risks.

To fix this issue, change the comparison length to `BLAKE2B_BLOCK_SIZE`:

```
if (memcmp(buffer, args_bytes_seg.ptr, BLAKE2B_BLOCK_SIZE) == 0) {
```

Raw Performance

Potential Buffer Overflow:

```
#include <stdio.h>
#include <string.h>

#define BUFFER_SIZE 10

void copy_to_buffer(char *input) {
    char buffer[BUFFER_SIZE];
    strcpy(buffer, input);
    printf("Buffer contains: %s\n", buffer);
}

int main(int argc, char *argv[]) {
    if (argc != 2) {
        printf("Please provide one argument!\n");
        return 1;
    }

    copy_to_buffer(argv[1]);
    return 0;
}
```

Combine With Code Understanding

1. RFCs and ckb-c-stdlib as Database
2. Extract CKB relevant context from the target code.
3. Through langchain and Deeplake, extract document & examples of relevant context.
4. Feed context to chatgpt with proper prompt.

Prompt:

Please identify and extract all function calls within the following C code. Only include those that are not come from the standard C library. Please reply me with a list: [function1, function2, ...]. Please do not include any other responses except the list itself. ⁵

⁵A simple parser can does that as well.

- `k`: Number of Documents to return.
- `distance_metric`: 'L2' for Euclidean, 'L1' for Nuclear, 'max' L-infinity distance, 'cos' for cosine similarity, 'dot' for dot product.
- `filter`: Attribute filter by metadata.
- `maximal_marginal_relevance`: Whether to use maximal marginal relevance.

Retrieve Document from DB

Case study: `ckb_load_script`

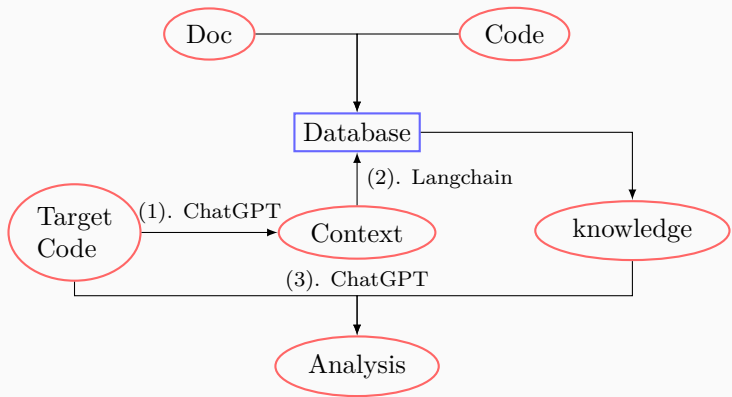
- 5 returned documents, 1200 tokens
- Relevant Info: 1
 1. Code snippet that contains usage of `ckb_load_script`
- Non-critical Information: 2
 1. Both are from RFCs documents, but aren't less important than code example.
- Wrong information: 2
 1. `ckb_load_script_hash`
 2. `load_script`
- Can be improved by forcing the returned context to contain the **exact keyword** `ckb_load_script`

Summarize knowledge through ChatGPT

Prompt:

I would like to familiarize myself with the usage standards of certain function calls. For each function, please provide guidance on how to ensure the success of the function call by looking at the RFC and syscalls code, including what return value should be checked. Please provide accurate information and if you are unable to find relevant details in the given context, kindly indicate so in your response.

Pipeline



Example

Input: `simple_udt.c` (unchecked return)

Response:

... as it doesn't check for the return value of `ckb_load_script`. If the function fails, the script may not be loaded properly, and the following code may access uninitialized memory. To fix this issue, you should check the return value of `ckb_load_script`:

```
ret = ckb_load_script(script, &len, 0);  
if (ret != CKB_SUCCESS) {  
    return ret;  
}
```

Example

Input: simple_udt.c (Script too long)

Response:

```
ret = ckb_load_script(script, &len, 0);  
if (len > SCRIPT_SIZE) {  
    return ERROR_SCRIPT_TOO_LONG;  
}
```

Unrelated / Non-accurate Responses:

1. Possibility of Overflow: Variables like `len`, `i`, and `witness_len` are used in loops and buffer size calculations. If their values grow large enough, they could overflow, which could lead to incorrect calculations and potentially buffer overflows.
2. Buffer overflow: Functions such as `memcpy()` and `ckb_load_script()` may cause buffer overflows if the length of the data exceeds the size of the buffer. Even though the code seems to be checking the length against the buffer size, it's better to double-check this part.
3. Secure Hashing: While Blake2b is a fast and secure hashing function, note that it's susceptible to length extension attacks unlike its successor Blake3.

Exceed token limits

- Knowledge generated from Database can easily exceeds token limits.
- Large context causes slowdown or even timeout.
- Solution: Split context into pieces and perform analysis in multiple chat instances.
 1. Generate context as usual as in (1).
 2. For each related function / header file, generate separate knowledge.
 3. For each related knowledge, ask ChatGPT to perform analysis.
 4. Expensive and slow.

Missed Document

- Information that are important but missed by knowledge retriever.
 - Partial Loading for syscall.
- Reason: Doc does not contain relevant keywords that can be picked up by the similarity function.
 - Partial Loading does not mention `ckb_load_script`.
- Potential Solution:
 - Manually adding information as knowledge.
 - Write AI-oriented documentation.

From the experience, information of partial loading is not **necessary** for GPT to figure out that it needs to check the size of the script being loaded. Simple code snippets can help its understanding as well.

Unable to understand implicit intention

- `secp256k1_blake160_sighash_all.c`
- Remove part of the hashing (no hash of len)
- ChatGPT thinks it is OK.
- But **understand** the issue when asked explicitly.

- Incorrect response happens occasionally... No real fix...
- Analysis are expensive, each request can easy go up to 8-10k tokens.