LLM Agents for Automatic Code Debugging

Nirjhar & Vardhan

June 15, 2024

Outline

Problem Statement

We desire to achieve the following using LLM agents:

- Identify and locate bugs in the code based on human input

Problem Statement

We desire to achieve the following using LLM agents:

- Identify and locate bugs in the code based on human input
- Suggest a corrected version of the code

Problem Statement

We desire to achieve the following using LLM agents:

- Identify and locate bugs in the code based on human input
- Suggest a corrected version of the code
- Validate that the proposed fix resolves the bug

Problem Relevance

1. Problems in debugging code manually:

Problem Relevance

1. Problems in debugging code manually:
    - Time-consuming process

Problem Relevance

1. Problems in debugging code manually:
    - Time-consuming process
    - Requires deep understanding of the codebase

Problem Relevance

1. Problems in debugging code manually:
    - Time-consuming process
    - Requires deep understanding of the codebase
    - Prone to human error

Problem Relevance

1. Problems in debugging code manually:
    - Time-consuming process
    - Requires deep understanding of the codebase
    - Prone to human error
2. Benefits of automating with LLM agents:

Problem Relevance

1. Problems in debugging code manually:
    - Time-consuming process
    - Requires deep understanding of the codebase
    - Prone to human error
2. Benefits of automating with LLM agents:
    - Improve efficiency and accuracy

Problem Relevance

1. Problems in debugging code manually:
   - Time-consuming process
   - Requires deep understanding of the codebase
   - Prone to human error
2. Benefits of automating with LLM agents:
   - Improve efficiency and accuracy
   - Suggest and implement code fixes

Problem Relevance

1. Problems in debugging code manually:
   - Time-consuming process
   - Requires deep understanding of the codebase
   - Prone to human error
2. Benefits of automating with LLM agents:
   - Improve efficiency and accuracy
   - Suggest and implement code fixes
   - Provide comprehensive code reviews

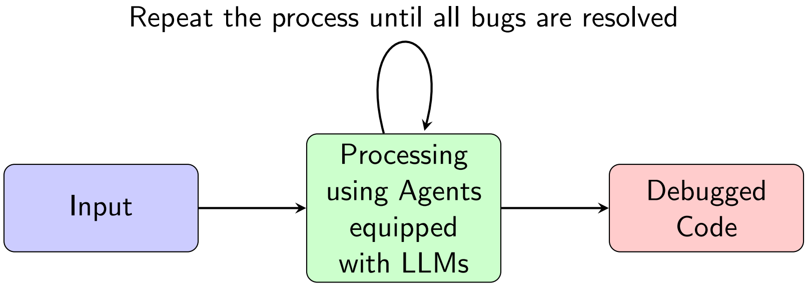
What are LLMs?

- Advanced AI models trained on vast text data
- Understand and generate human-like language
- Perform tasks such as:
  - Text generation
  - Translation
  - Summarization
  - Code generation
- Examples:
  - OpenAI's GPT series
  - Google's BERT
  - Meta's LLaMA

What are Agents?

- Software entities performing specific tasks
- Equipped with an LLM
- Have a defined role and goal
- Examples:
  - Article Generating System: Planner agent, Writer agent, Editor agent
  - Automatic Mailing System: Web Scraping agent, Email Drafting agent, Email Reviewing agent, Automatic Email Sending agent

System Architecture


Repeat the process until all bugs are resolved
Input
Processing using Agents equipped with LLMs
Debugged Code

Implementation

- LLM used: mixtral-8x7b-32768 from Groq

Implementation

- LLM used: mixtral-8x7b-32768 from Groq
- Agent system built using the ‘crewai’ library

Implementation

- LLM used: mixtral-8x7b-32768 from Groq
- Agent system built using the ‘crewai’ library
- Four main agents:

Implementation

- LLM used: mixtral-8x7b-32768 from Groq
- Agent system built using the ‘crewai’ library
- Four main agents:
  - Analyzer: analyzes the code

Implementation

- LLM used: mixtral-8x7b-32768 from Groq
- Agent system built using the ‘crewai’ library
- Four main agents:
    - Analyzer: analyzes the code
    - Fixer: fixes the bug

Implementation

- LLM used: mixtral-8x7b-32768 from Groq
- Agent system built using the ‘crewai’ library
- Four main agents:
    - Analyzer: analyzes the code
    - Fixer: fixes the bug
    - Reviewer: reviews the code

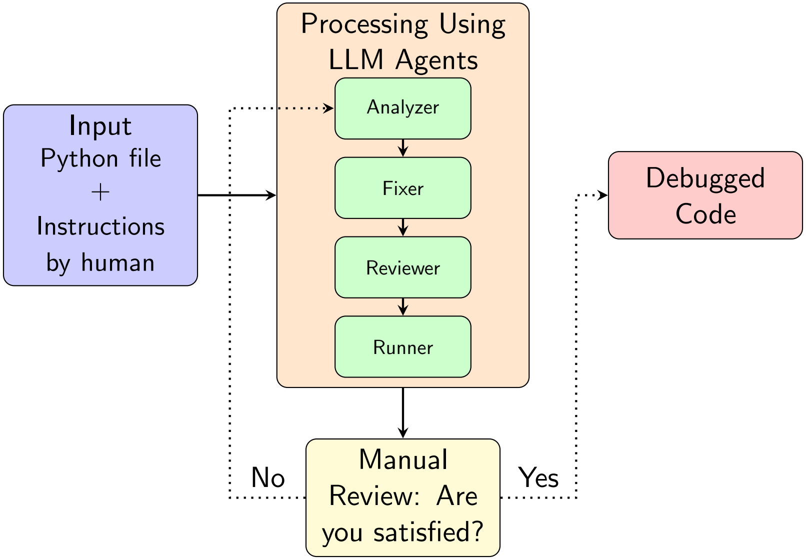
Implementation

- LLM used: mixtral-8x7b-32768 from Groq
- Agent system built using the 'crewai' library
- Four main agents:
  - Analyzer: analyzes the code
  - Fixer: fixes the bug
  - Reviewer: reviews the code
  - Runner: runs the code

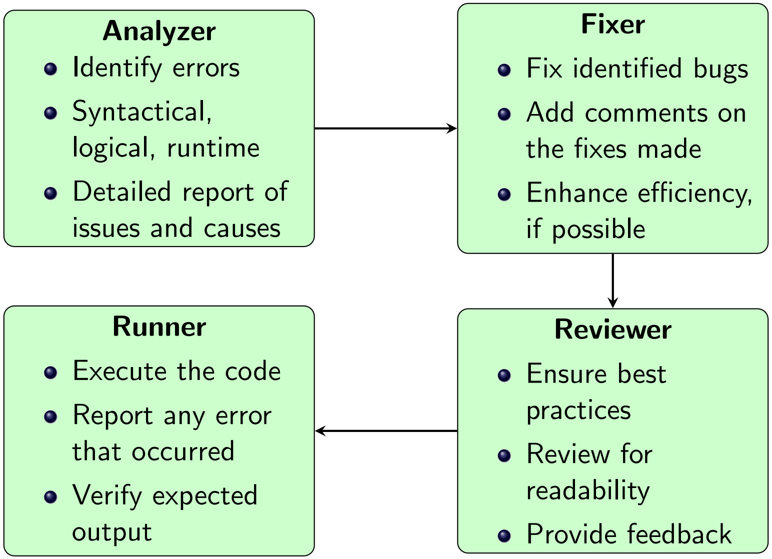
Implementation

- LLM used: mixtral-8x7b-32768 from Groq
- Agent system built using the ‘crewai’ library
- Four main agents:
  - Analyzer: analyzes the code
  - Fixer: fixes the bug
  - Reviewer: reviews the code
  - Runner: runs the code
- Iterative process for thorough debugging

Implementation flowchart



Workflow of Agents



Example - Code Comparison

Buggy Code

```python
class  Arithmetic:
    def __init__(self, number):
        self.number = number
        print("The square of"+self.number+" is "+self.square())
        print(f"The factorial of {self.number} is {self.factorial(self.number)}")

    def square(self):
        return self.multiply(self.number, self.number)

    def multiply(self,a,b):
        result = a*b

    def factorial(self,num):
        if num == 0:
          return 1
        else:
          return self.multiply(num, self.factorial(num))

print("Enter a positive integer:",end=" ")
n = input()
help = Arithmetic(n)
```

Debugged Code (by human)

```python
class  Arithmetic:
    def __init__(self, number):
        self.number = number
        print(f"The square of {self.number} is {self.square}")
        print(f"The factorial of {self.number} is {self.factorial(self.number)}")

    def square(self):
        return self.multiply(self.number, self.number)

    def multiply(self,a,b):
        result = a*b
        return result

    def factorial(self,num):
        if num == 0:
          return 1
        else:
          return self.multiply(num, self.factorial(num-1))

print("Enter a positive integer:",end=" ")
n = int(input())
help = Arithmetic(n)
```

Example - Input for our Model

Buggy Code

```python
class  Arithmetic:
    def __init__(self, number):
        self.number = number
        print("The square of"+self.number+" is "+self.square())
        print(f"The factorial of {self.number} is {self.factorial(self.number)}")

    def square(self):
        return self.multiply(self.number, self.number)

    def multiply(self,a,b):
        result = a*b

    def factorial(self,num):
        if num == 0:
          return 1
        else:
          return self.multiply(num, self.factorial(num))

print("Enter a positive integer:",end=" ")
n = input()
help = Arithmetic(n)
```

Instructions by Human

```
The input code is riddled with bugs that need to be addressed and needs to looks more polished.

Firstly, the user-entered number is not in the correct format, causing errors.

Additionally, when an object of the class is instantiated, it fails to print the square of the number as intended.

Moreover, the multiplication function does not produce the correct results, failing to operate as expected.

Lastly, the factorial function encounters an issue where it enters an infinite recursion loop, causing the program to crash.
```

Example - Code Comparison

Debugged Code (by model)

```python
import sys

class Arithmetic:
    def __init__(self, number: int): # Initialize the object with a positive integer.

        self.number = number
        self.square = self.square()
        self.factorial = self.factorial()

    def square(self): #Calculate the square of the number.
        result = self.multiply(self.number, self.number)
        print(f"The square of {self.number} is {result}")
        return result

    def multiply(self, a: int, b: int) -> int: # return: The result of the multiplication of a and b.
        result = a * b
        return result

    def factorial(self): # Calculate the factorial of the number using an iterative approach.
        result = 1
        for i in range(1, self.number + 1):
            result *= i
        return result

if __name__ == "__main__":
    print("Enter a positive integer: ", end="")
    try:
        n = int(input())
    except ValueError:
        print("Error: Input must be a positive integer.")
        sys.exit(1)

    if n < 1:
        print("Error: Input must be a positive integer.")
        sys.exit(1)

    help = Arithmetic(n)
```

Drawbacks - Working without and with context

Code with logical error

```python
def square(num):
    return num**3

```

Instructions by Human

```

```

Debugged Code 1

```python
def square(num): # The function prints square of num
    return num**2
```

Debugged Code 2

```python
def cube(num): # The function prints cube of num
    return num**3
```

Code with logical error

```python
def square(num):
    return num**3

```

Instructions by Human

```
I want to print the sqaure of number 5 using the function defined.

```

Debugged Code

```python
def square(num): # The function prints square of num
    return num**2
print(f"Square of 5 is {square(5)}")
```

Further Improvements

- Implement complex debugging strategies

Further Improvements

- Implement complex debugging strategies
- Improve the code to work for a repository

Further Improvements

- Implement complex debugging strategies
- Improve the code to work for a repository
- Enhance user interface

References

1. OpenAI's GPT: openai.com/research/gpt-3
2. Google's BERT: ai.googleblog.com/open-sourcing-bert
3. Meta's LLaMA: ai.facebook.com/llama-meta-ai
4. CrewAI: https://www.crewai.com/
5. Prompt Engineering: https://www.promptingguide.ai/