CS487 Final Project Proposal: Employing LLM-Based Evaluation for Memory Safety Bug Detection and Resolution C

Objective:

The goal of this project is to design and implement a system to evaluate multiple large language models (LLMs) in their ability to automatically identify and suggest fixes for memory safety bugs in C programs. The system will utilize several types of common memory safety vulnerabilities and evaluate LLM performance based on specific metrics.

This proposal aims to set the foundation for a software system that can both build and evaluate a LLM-powered bug detection and fixing for C programming memory safety vulnerabilities.

Scope:

This project will focus on testing two to three types of memory safety bugs and measure the ability of different LLMs to detect these issues and provide adequate solutions. The types of memory safety bugs that will be addressed include:

1. **Buffer Overflow**
   A buffer overflow occurs when more data is written to a buffer than it can hold, leading to memory corruption and potential security vulnerabilities.
2. **Use-After-Free**
   This bug occurs when a program continues to use a pointer to memory that has already been freed, which can lead to unpredictable behavior and security risks.
3. **Double Free**
   A double free occurs when a program attempts to free a block of memory more than once, leading to memory corruption or crashes.

Overview:

The system will be composed of the following key components:

1. **C Codebase with Memory Safety Bugs**
   A set of C programs will be intentionally written or sourced with the specific memory safety bugs mentioned above (buffer overflow, use-after-free, and double free).
2. **LLM Integration**
   The system will integrate two or more LLMs, such as OpenAI's GPT and Meta's LLaMA, to evaluate their performance on the tasks of bug detection and fix suggestions.
3. **Bug Detection and Fix Suggestion Pipeline**
   - Input: The C code containing memory safety bugs.
   - Process: Each LLM will analyze the code and suggest bug fixes.
   - Output: A list of identified bugs and recommended fixes from each LLM.

Evaluation Metrics:

To assess the performance of the LLMs in identifying and fixing memory safety bugs, the following metrics will be used:

1. **Accuracy**
   - **Bug Detection Rate**: The percentage of identified memory safety bugs versus the actual number of bugs present in the code.
   - **False Positives**: Instances where the LLM incorrectly identifies safe code as containing bugs.
2. **Solution Quality**
   - **Correctness of Fixes**: The percentage of correct fixes suggested for identified bugs.
   - **Improvement in Code Safety**: The degree to which the LLM's suggestion improves the memory safety of the program.
3. **Execution Performance**
   - **Latency**: Time taken by the LLM to analyze the code and provide suggestions.
   - **Resource Utilization**: Memory and computational resources consumed by the LLM during the evaluation.
4. **Comprehensibility**
   - **Clarity of Fixes**: How understandable and implementable the suggested fixes are for developers.
   - **Explanatory Quality**: How well the LLM explains the bug and its solution.

Conclusion:

This project aims to develop a system that can help evaluate and compare the performance of different LLMs in identifying and fixing memory safety issues in C code. By examining multiple types of memory bugs and utilizing detailed performance metrics, this system will provide valuable insights into the current capabilities of LLMs for automated software debugging.