Большие языковые модели для локализации уязвимостей в файле могут оказаться «потерянными в конце»

Дата: 2025-02-09 00:00:00

Ссылка на исследование: https://arxiv.org/pdf/2502.06898

Рейтинг: 78
Адаптивность: 85

Ключевые выводы:

Исследование оценивает эффективность современных LLM (GPT-3.5, GPT-4, Mixtral, Llama) в обнаружении уязвимостей в файлах кода. Основной вывод: LLM значительно хуже обнаруживают уязвимости, расположенные ближе к концу больших файлов (эффект «lost in the end»), что противоречит ранее известному эффекту «lost in the middle».

Объяснение метода:

Исследование выявляет "lost in the end" эффект в LLM и предлагает простую стратегию "chunking" для повышения эффективности обнаружения уязвимостей на 37%. Предоставляет конкретные рекомендации по оптимальным размерам фрагментов для анализа кода (500-6500 символов), которые любой пользователь может немедленно применить без специальных инструментов. Ограничения: исследованы только три типа уязвимостей и ограниченный набор моделей.

Ключевые аспекты исследования: 1. **"Lost in the End" эффект** - исследование выявило, что современные LLM (GPT-4, Llama 3, Mixtral) имеют тенденцию пропускать уязвимости, расположенные в конце длинных файлов, что авторы назвали эффектом "lost in the end".

Влияние размера файла и позиции уязвимости - обнаружена отрицательная корреляция между размером файла/позицией уязвимости и вероятностью её обнаружения LLM. Чем больше файл или чем дальше к концу файла расположена уязвимость, тем ниже вероятность её обнаружения.

Стратегия "chunking" - разделение больших файлов на меньшие фрагменты повышает эффективность обнаружения уязвимостей LLM-моделями (в среднем на 37% по всем моделям).

Оптимальные размеры входных данных - исследование определило

оптимальные размеры фрагментов для разных типов уязвимостей (CWE-22: 3000-6500 символов, CWE-89 и CWE-79: 500-1500 символов).

Сравнение коммерческих и открытых моделей - при уменьшении размера входных данных разница в производительности между коммерческими и открытыми моделями сокращается, что указывает на то, что преимущество коммерческих моделей может заключаться в лучшей обработке контекстных окон.

Дополнение: Действительно ли для работы методов этого исследование требуется дообучение или API? Или методы и подходы можно применить в стандартном чате, а ученые лишь для своего удобства использовали расширенные техники?

Методы и подходы из этого исследования **не требуют дообучения или специального API** и могут быть применены в стандартном чате с LLM. Исследователи использовали модели "off-the-shelf" (без дополнительной настройки), как указано в разделе 5: "мы использовали эти модели в их конфигурациях по умолчанию без какой-либо тонкой настройки".

Основные концепции, которые можно применить в стандартном чате:

Стратегия "chunking" - разделение больших файлов кода на меньшие фрагменты перед отправкой в чат. Это самый важный и простой в реализации метод, который значительно повышает эффективность обнаружения проблем в коде.

Оптимальные размеры фрагментов - использование рекомендованных размеров: 500-1500 символов для большинства проблем безопасности и до 6500 символов для более сложных случаев.

Приоритизация начала файла - понимание того, что LLM хуже обрабатывают конец длинных файлов, и соответственно структурирование запросов.

Структурированный промпт - использование формата промпта с четкой структурой ожидаемого ответа, как в исследовании: "Se: [объяснение], BI: [проблемная строка], BUG FOUND: YES/NO".

Результаты от применения этих концепций: - Значительное повышение эффективности обнаружения проблем в коде (до 95% для некоторых типов проблем) - Более точное указание проблемных мест в коде - Снижение влияния размера файла и позиции проблемы на эффективность анализа - Более структурированные и полезные ответы от LLM

Эти подходы применимы не только к поиску уязвимостей, но и к другим задачам анализа кода: поиску багов, code review, оптимизации и рефакторингу.

Анализ практической применимости: 1. **"Lost in the End" эффект** - Прямая применимость: Высокая. Пользователи должны осознавать, что LLM могут пропускать уязвимости в конце длинных файлов, и соответственно адаптировать

свои запросы. - Концептуальная ценность: Очень высокая. Понимание этого ограничения фундаментально для эффективного использования LLM в задачах безопасности кода. - Потенциал для адаптации: Высокий. Пользователи могут применять простые стратегии разделения кода для повышения эффективности обнаружения уязвимостей.

Влияние размера файла и позиции уязвимости Прямая применимость: Высокая. Результаты напрямую указывают, что пользователи должны учитывать размер анализируемого кода. Концептуальная ценность: Высокая. Результаты опровергают распространенное мнение о "lost in the middle" для задач анализа кода. Потенциал для адаптации: Средний. Знание этих закономерностей помогает формировать лучшие запросы, но требует дополнительных действий от пользователя.

Стратегия "chunking"

Прямая применимость: Очень высокая. Любой пользователь может немедленно применить эту стратегию без специальных инструментов. Концептуальная ценность: Средняя. Подход интуитивно понятен и не требует глубокого понимания работы LLM. Потенциал для адаптации: Высокий. Стратегия применима к различным задачам анализа кода, не только поиску уязвимостей.

Оптимальные размеры входных данных

Прямая применимость: Высокая. Конкретные числа могут использоваться как руководство при разделении кода. Концептуальная ценность: Средняя. Значения специфичны для конкретных типов уязвимостей, но концепция определения оптимального размера применима шире. Потенциал для адаптации: Высокий. Методология определения оптимального размера может быть применена к другим задачам.

Сравнение коммерческих и открытых моделей

Прямая применимость: Средняя. Помогает в выборе модели для конкретных задач, но ограничена исследованными моделями. Концептуальная ценность: Средняя. Предоставляет понимание сильных и слабых сторон различных моделей. Потенциал для адаптации: Низкий. Выводы специфичны для конкретных моделей и могут устареть с выходом новых версий.

Prompt:

Применение исследования для оптимизации промптов при поиске уязвимостей ## Ключевые инсайты из исследования

Исследование показывает, что большие языковые модели (LLM) значительно хуже обнаруживают уязвимости в конце больших файлов кода ("эффект lost in the end"). Разделение файлов на меньшие фрагменты существенно улучшает результаты.

Пример оптимизированного промпта

[======] # Задача: Проверка кода на уязвимости XSS (CWE-79)

Контекст Я разрабатываю стратегию анализа безопасности кода. Исследования показывают, что LLM часто пропускают уязвимости в конце файлов, поэтому я разделил код на фрагменты размером ~500 символов.

Инструкции 1. Проанализируй следующий фрагмент кода на наличие XSS-уязвимостей (CWE-79) 2. Обрати ОСОБОЕ внимание на код в конце фрагмента 3. Рассмотри, как пользовательский ввод обрабатывается и выводится 4. Проверь все переменные в контексте данного фрагмента 5. Если найдешь уязвимость, опиши её, почему она возникает и как её исправить

Фрагмент кода для анализа [======]javascript // Код фрагмента здесь [======]

Дополнительный контекст Этот фрагмент является частью [описание функциональности]. Переменные [X, Y, Z] получают данные от пользователя. [======]

Почему этот промпт работает эффективнее

Оптимальный размер фрагмента: Промпт учитывает рекомендации исследования по оптимальному размеру фрагментов (500 символов для XSS-уязвимостей)

Акцент на конец фрагмента: Явно обращает внимание модели на конец фрагмента, где уязвимости чаще пропускаются

Сохранение контекста: Включает информацию о переменных и их происхождении, что важно для определения уязвимостей, связанных с пользовательским вводом

Специфика типа уязвимости: Промпт сфокусирован на конкретном типе уязвимости (XSS/CWE-79), что улучшает точность анализа

Для других типов уязвимостей размер фрагментов нужно корректировать: 500-1500 символов для SQL-инъекций (CWE-89) и 3000-6500 символов для уязвимостей обхода пути (CWE-22).