Интеграция различных программных артефактов для улучшенной локализации ошибок и ремонта программ на основе LLM

Дата: 2025-03-04 00:00:00

Ссылка на исследование: https://arxiv.org/pdf/2412.03905

Рейтинг: 65
Адаптивность: 75

Ключевые выводы:

Основная цель исследования - улучшить локализацию и исправление программных ошибок с помощью LLM путем интеграции различных программных артефактов. Главные результаты: разработан фреймворк DEVLoRe, который успешно локализует 49,3% ошибок в одиночных методах и 47,6% ошибок в нескольких методах, а также генерирует 56,0% правдоподобных исправлений для одиночных методов и 14,5% для нескольких методов, превосходя современные методы APR.

Объяснение метода:

Исследование предлагает ценные принципы для всех пользователей LLM: комбинирование разных типов контекста улучшает ответы; двухэтапный подход (анализ, затем решение) повышает точность; структурированные запросы с четким ожидаемым форматом снижают галлюцинации. Несмотря на технический фокус на Java, эти концепции применимы к широкому спектру задач и могут быть адаптированы пользователями любого уровня подготовки.

Ключевые аспекты исследования: 1. **Интеграция различных программных артефактов для локализации и исправления ошибок с помощью LLM**. Исследование предлагает фреймворк DEVLoRe, который использует комбинацию трех типов информации: содержимое отчетов об ошибках, трассировки стека ошибок и отладочную информацию для более эффективной работы LLM.

Двухэтапный подход к исправлению ошибок. Сначала LLM локализует проблемные методы и строки кода, а затем генерирует исправления, что имитирует процесс работы человека-разработчика.

Сравнительный анализ эффективности различных типов информации. Исследование показывает, что содержимое отчетов об ошибках наиболее эффективно для локализации ошибок, а трассировки стека — для их исправления, при этом комбинация всех типов информации дает наилучшие результаты.

Работа с ошибками, затрагивающими несколько методов. В отличие от большинства существующих подходов, DEVLoRe способен локализовать и исправлять ошибки, распространяющиеся на несколько методов, что делает его более универсальным.

Строгий формат запросов к LLM. Исследователи разработали четкую структуру запросов, которая помогает LLM более точно локализовать и исправлять ошибки, снижая вероятность галлюцинаций.

Дополнение:

Применимость методов исследования в стандартном чате

Хотя в исследовании использовались специальные инструменты для сбора отладочной информации (Method Recorder и Debug Recorder), ключевые концепции и подходы могут быть применены в стандартном чате с LLM без необходимости дообучения или API.

Применимые концепции:

Комбинирование разных типов информации Пользователи могут предоставлять LLM несколько видов контекста одновременно (описание проблемы, сообщения об ошибках, наблюдаемое поведение) Пример: "Вот код [код], вот сообщение об ошибке [ошибка], и вот что я наблюдаю при запуске [наблюдение]"

Двухэтапный подход к решению проблем

Сначала запрос на анализ и локализацию проблемы Затем запрос на исправление с учетом выявленной проблемы Пример: "Сначала проанализируй, где может быть проблема в этом коде. Теперь, исходя из этого анализа, предложи исправление."

Структурированные запросы с ожидаемым форматом вывода

Четко указывать, в каком формате должен быть представлен ответ Пример: "Предложи исправление в формате: 1) Проблемная строка, 2) Исправленная версия, 3) Объяснение исправления"

Предоставление контекста выполнения

Пользователи могут вручную собрать и предоставить некоторую отладочную информацию Пример: "При выполнении этого кода переменная X имеет значение Y в строке Z" #### Ожидаемые результаты:

При использовании этих концепций в стандартном чате можно ожидать: - Более точной локализации проблем в коде - Более качественных предложений по исправлению - Снижения вероятности галлюцинаций модели - Более

структурированных и понятных ответов

Исследование показывает, что даже без специальных инструментов, правильная структуризация запросов и комбинирование разных типов информации может значительно улучшить качество взаимодействия с LLM при решении технических задач.

Анализ практической применимости: 1. **Интеграция различных программных артефактов** - Прямая применимость: Пользователи могут улучшить свои запросы к LLM, предоставляя не только описание проблемы, но и трассировки стека ошибок и отладочную информацию о значениях переменных. Это может значительно повысить точность ответов LLM при решении проблем с кодом. - Концептуальная ценность: Понимание того, что разные типы информации дополняют друг друга при диагностике проблем, позволяет пользователям более эффективно формулировать запросы к LLM. - Потенциал для адаптации: Этот принцип может быть перенесен на взаимодействие с LLM в других контекстах, где комбинирование разных источников информации может улучшить результат.

Двухэтапный подход к исправлению ошибок Прямая применимость: Пользователи могут структурировать свои запросы к LLM в два этапа: сначала попросить локализовать проблему, а затем предоставить решение, что соответствует естественному подходу разработчиков. Концептуальная ценность: Понимание того, что LLM может работать последовательно, сначала анализируя проблему, а затем решая ее, помогает пользователям строить более эффективные цепочки запросов. Потенциал для адаптации: Этот подход может быть применен к различным задачам, требующим сначала анализа, а затем решения.

Сравнительный анализ эффективности различных типов информации

Прямая применимость: Пользователи узнают, какая информация наиболее полезна для каких задач (например, описания проблем для локализации и трассировки стека для исправления), что помогает им предоставлять LLM более релевантные данные. Концептуальная ценность: Понимание относительной ценности разных типов входных данных для LLM помогает построить более эффективные взаимодействия. Потенциал для адаптации: Этот принцип может быть применен к другим областям, где необходимо определить, какая информация наиболее ценна для LLM.

Работа с ошибками, затрагивающими несколько методов

Прямая применимость: Пользователи узнают, что LLM способны анализировать сложные проблемы, затрагивающие несколько частей кода, если предоставить им достаточно контекста. Концептуальная ценность: Это расширяет понимание возможностей LLM для решения сложных, взаимосвязанных проблем. Потенциал для адаптации: Можно применить подход к другим сложным задачам, требующим понимания взаимосвязей между разными компонентами.

Строгий формат запросов к LLM

Прямая применимость: Пользователи могут адаптировать подход к структурированию запросов (с четкими инструкциями и ожидаемыми форматами вывода) для своих задач. Концептуальная ценность: Понимание того, что структурированные запросы с четкими ожиданиями снижают вероятность галлюцинаций LLM. Потенциал для адаптации: Этот подход к формату запросов может быть применен практически в любом контексте взаимодействия с LLM.

Prompt:

Применение исследования DEVLoRe в промтах для GPT ## Ключевые знания из исследования

Исследование DEVLoRe демонстрирует эффективность интеграции различных программных артефактов для локализации и исправления ошибок с помощью LLM. Ключевые аспекты:

Комбинирование артефактов: Использование нескольких источников (описание проблемы, трассировка стека, отладочная информация) значительно эффективнее, чем один источник **Двухэтапный подход**: Сначала локализация ошибочных методов, затем конкретных строк кода **Структурированные промпты**: Чёткая структура запросов с разделителями повышает точность ответов ## Пример промта для отладки и исправления кода

[======] # Запрос на локализацию и исправление ошибки в коде

Описание проблемы При вызове метода calculateDiscount() приложение выдаёт ошибку NullPointerException при обработке заказов с нулевой стоимостью.

Трассировка стека [======] java.lang.NullPointerException: Cannot invoke "Double.doubleValue()" because "this.totalPrice" is null at com.example.shop.Order.calculateDiscount(Order.java:45) at com.example.shop.CheckoutService.processOrder(CheckoutServic e.java:28) at com.example.shop.Main.main(Main.java:12) [======]

Отладочная информация - Метод calculateDiscount() вызывается для объектов Order с параметрами: - При успешном выполнении: totalPrice=150.0, quantity=3 - При ошибке: totalPrice=null, quantity=0 - Переменная totalPrice инициализируется в методе setOrderItems() - Для заказов с нулевой стоимостью setOrderItems() не вызывается

Исходный код [======]java public class Order { private Double totalPrice; private int quantity;

public Double calculateDiscount() { if (quantity > 5) { return totalPrice * 0.1; // 10% discount for large orders } return totalPrice * 0.05; // 5% default discount }

public void setOrderItems(List items) { this.quantity = items.size(); this.totalPrice =

items.stream() .mapToDouble(Item::getPrice) .sum(); }

// Other methods... } [======]

Задача 1. Локализуйте точную строку кода, вызывающую ошибку 2. Предложите исправление, которое обрабатывает случай с нулевым totalPrice 3. Объясните, почему ваше решение работает [======]

Объяснение эффективности

Данный промт использует ключевые принципы из исследования DEVLoRe:

Интеграция артефактов - включены все три типа информации: Описание проблемы (что происходит) Трассировка стека (где происходит ошибка) Отладочная информация (контекст выполнения)

Структурированный формат - четкое разделение разделов помогает модели лучше понять информацию и уменьшает вероятность галлюцинаций.

Двухэтапный подход - промт сначала направляет модель на локализацию ошибки (строка 1 в задаче), а затем на её исправление (строка 2).

Такой подход, согласно исследованию, значительно повышает вероятность получения корректного решения по сравнению с предоставлением только одного типа информации.