DeCon: Обнаружение некорректных утверждений через постусловия, сгенерированные большой языковой моделью

Дата: 2025-01-06 00:00:00

Ссылка на исследование: https://arxiv.org/pdf/2501.02901

Рейтинг: 63
Адаптивность: 75

Ключевые выводы:

Исследование направлено на выявление и устранение проблемы некорректных утверждений (assertions), генерируемых большими языковыми моделями (LLM) при создании тестовых случаев. Основной результат: предложен новый подход DeCon, который использует постусловия, генерируемые LLM, для обнаружения некорректных утверждений. DeCon может обнаружить более 64% некорректных утверждений, генерируемых четырьмя современными LLM, и улучшить эффективность этих моделей в генерации кода на 4%.

Объяснение метода:

Исследование предлагает практичный метод обнаружения некорректных утверждений в автотестах, решая реальную проблему (62% утверждений LLM некорректны). Основные концепции (использование постусловий, фильтрация примерами ввода-вывода) могут быть адаптированы для диалога с LLM. Требуется базовое понимание программирования, но подход может значительно улучшить качество тестов и понимание требований к функциям.

Ключевые аспекты исследования: 1. **DeCon** - подход для обнаружения некорректных утверждений (assertions) в автоматически сгенерированных тестах с помощью постусловий, сгенерированных LLM. Исследование показало, что более 62% утверждений, генерируемых современными LLM, некорректны.

Использование постусловий - DeCon использует LLM для генерации постусловий (условий, которые должны выполняться после корректной работы функции), а затем фильтрует их с помощью имеющихся примеров ввода-вывода.

Фильтрация неверных утверждений - после фильтрации постусловий, оставшиеся постусловия используются для обнаружения некорректных утверждений в тестах.

Улучшение генерации кода - DeCon может улучшить эффективность LLM в генерации кода на 4% по метрике Pass@1.

Эффективность обнаружения - DeCon обнаруживает в среднем более 64% некорректных утверждений, сгенерированных четырьмя современными LLM.

Дополнение: Да, методы этого исследования можно адаптировать для применения в стандартном чате, без необходимости дообучения или API. Хотя авторы использовали API для автоматизации процесса, основная концепция может быть реализована в обычном диалоге с LLM.

Вот концепции и подходы, которые можно применить в стандартном чате:

Генерация постусловий для функций: Можно попросить LLM сгенерировать постусловия для функции на основе её описания и сигнатуры. Пример запроса: "Напиши постусловие в виде assert-выражения для функции, которая удаляет дубликаты из списка".

Проверка постусловий на примерах: Используя примеры ввода-вывода, можно попросить LLM проверить, выполняются ли постусловия для этих примеров. Это поможет отфильтровать некорректные постусловия.

Использование постусловий для проверки тестов: Можно представить LLM сгенерированные тесты и отфильтрованные постусловия и попросить проверить, соответствуют ли тесты постусловиям.

Итеративное улучшение: Можно использовать диалог для постепенного улучшения как постусловий, так и тестов.

Примеры результатов, которые можно получить: - Более качественные тесты с меньшим количеством некорректных утверждений - Лучшее понимание требований к функции через формализацию постусловий - Выявление противоречий в понимании функциональности

Ограничения при использовании в стандартном чате: - Процесс будет менее автоматизированным и более диалоговым - Для больших наборов тестов процесс может быть трудоемким - Эффективность зависит от качества формулировки запросов

Ключевое преимущество: подход фокусируется на логических условиях и не требует выполнения кода, что делает его подходящим для диалогового режима работы с LLM.

Анализ практической применимости: **1. Использование постусловий для валидации** - **Прямая применимость**: Пользователи могут адаптировать подход для проверки своих собственных тестов, запрашивая у LLM постусловия для функций и используя их для валидации. - **Концептуальная ценность**: Помогает

понять, что LLM могут генерировать некорректные утверждения, и предлагает метод для их фильтрации. - **Потенциал для адаптации**: Пользователи могут применять этот подход для улучшения качества тестов и понимания требований к функциям.

2. Фильтрация с помощью примеров ввода-вывода - **Прямая применимость**: Пользователи могут использовать имеющиеся примеры ввода-вывода для фильтрации сгенерированных постусловий. - **Концептуальная ценность**: Демонстрирует важность использования конкретных примеров для проверки абстрактных утверждений. - **Потенциал для адаптации**: Метод применим для проверки любых логических условий, не только для тестов.

3. Обнаружение некорректных утверждений - **Прямая применимость**: Пользователи могут применить эту технику для проверки сгенерированных LLM тестов. - **Концептуальная ценность**: Повышает осведомленность о проблеме некорректных утверждений в автоматически сгенерированных тестах. - **Потенциал для адаптации**: Метод может быть адаптирован для других задач, требующих валидации логических утверждений.

4. Улучшение генерации кода - **Прямая применимость**: Ограниченная для обычных пользователей, так как требует специальных инструментов. - **Концептуальная ценность**: Демонстрирует, что качество тестов влияет на качество генерируемого кода. - **Потенциал для адаптации**: Пользователи могут использовать похожий подход для выбора лучшего из нескольких сгенерированных LLM решений.

5. Статистика качества сгенерированных утверждений - **Прямая применимость**: Помогает пользователям оценить, насколько можно доверять сгенерированным LLM тестам. - **Концептуальная ценность**: Предоставляет количественную оценку проблемы некорректных утверждений. - **Потенциал для адаптации**: Пользователи могут использовать эту информацию для принятия решений о необходимости дополнительной проверки.

Prompt:

Применение знаний из исследования DeCon в промптах для GPT ## Ключевые идеи исследования для использования в промптах

Исследование DeCon показывает, что использование постусловий может значительно улучшить обнаружение некорректных утверждений в генерируемом коде. Основная идея заключается в проверке соответствия генерируемых утверждений ожидаемым постусловиям функции.

Пример промпта с применением знаний из DeCon

[======] # Задача: Написать функцию и тесты для неё

Контекст Я разрабатываю функцию find_second_smallest(numbers), которая

находит второе наименьшее число в списке.

Инструкции 1. Сначала опиши постусловия для функции find_second_smallest: - Какой результат должна возвращать функция для различных входных данных? - Какие граничные случаи следует учесть? - Какие инварианты должны сохраняться?

На основе этих постусловий напиши реализацию функции find_second_smallest.

Создай набор тестовых утверждений (assertions) для проверки функции, убедившись, что:

Каждое утверждение соответствует как минимум одному из описанных постусловий Проверяются все граничные случаи Тесты не содержат противоречивых ожиданий

Проанализируй каждое утверждение и объясни, какому постусловию оно соответствует.

Важно Обрати особое внимание на граничные случаи и неочевидные сценарии, такие как: - Пустой список - Список с одним элементом - Список с повторяющимися элементами - Список, где все элементы одинаковые [======]

Как работают знания из исследования в этом промпте

Генерация постусловий перед кодом: Промпт просит сначала определить постусловия функции, что соответствует методологии DeCon, где постусловия используются для проверки корректности утверждений.

Соответствие утверждений постусловиям: Явное требование проверять соответствие каждого тестового утверждения постусловиям, что помогает избежать некорректных утверждений.

Фокус на граничных случаях: Исследование показало, что многие некорректные утверждения связаны с непроверкой граничных случаев, поэтому промпт явно требует их рассмотрения.

Анализ утверждений: Требование объяснить, какому постусловию соответствует каждое утверждение, помогает выявить потенциально некорректные утверждения, не соответствующие ни одному постусловию.

Такой подход позволяет снизить количество некорректных утверждений примерно на 64% (согласно исследованию) и повысить качество генерируемого кода на ~4%.