Сравнение кода, написанного человеком, и кода, сгенерированного ИИ: Вердикт всё ещё не вынесен!

Дата: 2025-01-28 00:00:00

Ссылка на исследование: https://arxiv.org/pdf/2501.16857

Рейтинг: 75
Адаптивность: 85

Ключевые выводы:

Исследование сравнивает качество кода, написанного людьми и сгенерированного большими языковыми моделями (LLM, конкретно GPT-4). Основная цель - оценить, насколько эффективны LLM в создании программного кода по сравнению с человеческими программистами. Результаты показывают, что код, написанный людьми, лучше соответствует стандартам кодирования, но код GPT-4 чаще проходит функциональные тесты. При этом LLM часто создают более сложный код и испытывают трудности с задачами, требующими глубоких предметных знаний.

Объяснение метода:

Исследование предоставляет практически применимые выводы о сравнении кода, написанного людьми и сгенерированного LLM. Результаты показывают, что LLM лучше в стандартных задачах, но отстают в сложных. Выводы о функциональных различиях, безопасности и сложности кода напрямую полезны для широкого круга пользователей.

Ключевые аспекты исследования: 1. **Сравнительный анализ кода**: Исследование проводит систематическое сравнение кода, написанного людьми и сгенерированного LLM (GPT-4), используя 72 различных задачи программирования на Python.

Многомерная оценка качества: Работа оценивает код по четырем ключевым критериям: соответствие стандартам кодирования Python (с использованием Pylint), безопасность и уязвимости (с использованием Bandit), сложность кода (с использованием Radon) и функциональная корректность (с использованием тестов Pytest).

Функциональные различия: Выявлены области, где LLM превосходит людей (стандартные задачи, проходимость тестов) и где отстает (сложные задачи, требующие глубоких доменных знаний и творческого мышления).

Безопасность кода: Обнаружены уязвимости как в коде, написанном людьми, так и сгенерированном LLM, с более серьезными выбросами в коде LLM.

Сложность кода: LLM генерирует в среднем более сложный код (на 61% выше по цикломатической сложности), что может затруднять его поддержку и понимание.

Дополнение: Исследование не требует дообучения моделей или специального API для применения его методов и выводов. Все концепции и подходы могут быть адаптированы для работы в стандартном чате с LLM.

Основные концепции, которые можно применить в стандартном чате:

Выбор типа задач для LLM: Исследование показывает, что LLM лучше справляются со стандартными, хорошо определенными задачами, но отстают в сложных задачах, требующих глубоких доменных знаний. Пользователи могут использовать LLM для рутинных задач программирования, но полагаться на свои навыки для более сложных проблем.

Проверка безопасности: Зная о типичных уязвимостях в коде, сгенерированном LLM (небезопасное использование подпроцессов, жестко закодированные конфиденциальные данные, использование небезопасных библиотек), пользователи могут проверять сгенерированный код на эти проблемы.

Упрощение сложного кода: Понимая, что LLM генерирует более сложный код, пользователи могут запрашивать более простые решения или просить упростить полученный код.

Оценка функциональности: Исследование показывает, что код LLM часто проходит больше тестов, чем человеческий код. Пользователи могут ожидать высокой функциональности от сгенерированного кода, но также должны проверять его на соответствие требованиям.

Улучшение документации: Зная о проблемах с документацией в коде LLM, пользователи могут специально запрашивать хорошо документированный код или добавлять документацию самостоятельно.

Применение этих концепций позволит пользователям более эффективно использовать LLM для генерации кода, понимая их сильные и слабые стороны, и получать более качественные результаты.

Анализ практической применимости: 1. **Сравнительный анализ кода**: - Прямая применимость: Высокая. Пользователи могут использовать эти выводы для понимания, когда лучше применять LLM для генерации кода, а когда полагаться на ручное программирование. - Концептуальная ценность: Значительная. Исследование помогает понять сильные и слабые стороны LLM в контексте программирования. - Потенциал для адаптации: Высокий. Методология сравнения

может быть использована пользователями для оценки собственных решений.

Многомерная оценка качества: Прямая применимость: Средняя. Обычные пользователи могут не иметь доступа к инструментам статического анализа, но понимание критериев оценки полезно. Концептуальная ценность: Высокая. Представление о различных аспектах качества кода помогает формулировать лучшие запросы. Потенциал для адаптации: Высокий. Пользователи могут адаптировать критерии для собственной оценки сгенерированного кода.

Функциональные различия:

Прямая применимость: Очень высокая. Пользователи могут напрямую применять знания о том, для каких задач лучше использовать LLM. Концептуальная ценность: Высокая. Понимание ограничений LLM в сложных задачах помогает реалистично оценивать возможности. Потенциал для адаптации: Высокий. Можно адаптировать подход к использованию LLM в зависимости от типа задачи.

Безопасность кода:

Прямая применимость: Высокая. Выводы о типичных уязвимостях помогают пользователям проверять сгенерированный код. Концептуальная ценность: Значительная. Исследование показывает необходимость проверки безопасности, независимо от источника кода. Потенциал для адаптации: Средний. Требуются специализированные знания для адаптации методов проверки безопасности.

Сложность кода:

Прямая применимость: Высокая. Пользователи могут ожидать более сложный код от LLM и учитывать это при использовании. Концептуальная ценность: Средняя. Понимание тенденции к переусложнению помогает оценивать генерируемый код. Потенциал для адаптации: Высокий. Пользователи могут запрашивать более простые решения или упрощать полученный код.

Prompt:

Использование знаний из исследования в промтах для GPT ## Ключевые выводы для создания эффективных промтов

Исследование показывает, что код GPT-4: - Лучше проходит функциональные тесты (87.3% vs 54.9% у людей) - Имеет более высокую цикломатическую сложность (5.0 vs 3.1) - Хуже справляется с задачами, требующими глубоких предметных знаний - Может содержать проблемы безопасности

Пример промта с учетом этих знаний

[======] # Задача: Создать функцию для обработки пользовательских данных

Требования: 1. Напиши Python-функцию process_user_data(user_input), которая

валидирует и очищает пользовательский ввод. 2. Функция должна обрабатывать строки, содержащие имя, email и возраст.

Специальные инструкции (с учетом исследования): - Стремись к низкой цикломатической сложности (не более 3-4) для лучшей поддерживаемости - Уделяй особое внимание безопасности кода, особенно при валидации пользовательского ввода - Следуй стандартам PEP 8 для Python - Добавь комментарии, объясняющие логику работы - Включи простые примеры использования функции - Предоставь несколько тестовых случаев для проверки функциональности

Ожидаемый результат: Хорошо структурированная, безопасная и эффективная функция с низкой сложностью. [======]

Объяснение эффективности

Этот промт учитывает ключевые выводы исследования следующим образом:

Контроль сложности: Явно ограничивает цикломатическую сложность, так как исследование показало, что код GPT-4 обычно более сложный (5.0 vs 3.1 у людей)

Акцент на безопасности: Требует особого внимания к безопасности, что решает выявленную проблему с уязвимостями в коде LLM

Соблюдение стандартов: Запрашивает соответствие PEP 8, что улучшает структурированность кода (где люди показали преимущество)

Документация и тесты: Запрашивает комментарии и тестовые случаи, чтобы использовать сильную сторону GPT-4 в прохождении функциональных тестов

Такой структурированный промт помогает компенсировать выявленные в исследовании слабости GPT и использовать его сильные стороны, что приводит к более качественному и поддерживаемому коду.