К улучшению вопросов разработчиков с использованием распознавания именованных сущностей на основе LLM для разговоров в чатах разработчиков

Дата: 2025-03-01 00:00:00

Ссылка на исследование: https://arxiv.org/pdf/2503.00673

Рейтинг: 75
Адаптивность: 85

Ключевые выводы:

Исследование представляет SENIR (Software specific Named entity recognition, Intent detection, and Resolution classification) - подход, использующий LLM для аннотирования сущностей, намерений и статуса разрешения в разговорах разработчиков в чатах. Основная цель - улучшить ясность вопросов разработчиков и повысить вероятность их успешного разрешения. Результаты показывают, что SENIR эффективно идентифицирует программные сущности (F-score 86%), намерения пользователей (F-score 71%) и статус разрешения вопросов (F-score 89%), а также выявляет ключевые факторы, влияющие на успешное разрешение вопросов.

Объяснение метода:

Исследование предлагает конкретные, применимые рекомендации по формулированию эффективных запросов к LLM на основе анализа 29,243 разговоров. Результаты показывают, как структурирование запросов с указанием конкретных технических деталей и позитивного тона повышает вероятность получения полезных ответов. Хотя полная реализация SENIR требует технических знаний, основные принципы доступны всем пользователям.

Ключевые аспекты исследования: 1. **SENIR (Software specific Named entity recognition, Intent detection, and Resolution classification)** - подход, использующий LLM для автоматической разметки сообщений в чатах разработчиков, выделяя программные сущности (библиотеки, функции, языки программирования), определяя намерения пользователей и классифицируя статус решения вопросов.

Модель прогнозирования разрешения вопросов - авторы создали модель, которая предсказывает вероятность решения вопроса на основе его характеристик (сущности, намерения, стиль и т.д.), с точностью AUC 0.75-0.76.

Анализ влияния сущностей и намерений на решение вопросов - исследование выявило, что определенные комбинации технических сущностей (например, Programming Language + Library Function) и намерений (API Usage, API Change) значительно повышают вероятность получения ответа.

Рекомендации по формулированию вопросов - на основе анализа авторы предлагают конкретные рекомендации по составлению эффективных вопросов, включая использование конкретных технических деталей, позитивного тона и избегание перегруженности ссылками.

Датасет DISCO - исследование использует большой набор данных из 29,243 разговоров в Discord, что обеспечивает репрезентативность результатов для современных платформ общения разработчиков.

Дополнение:

Применимость методов без дообучения или API

Исследование SENIR использует LLM (Mixtral 8x7B) для разметки сущностей и намерений, что технически требует доступа к API. Однако ключевые концепции и подходы могут быть адаптированы для использования в стандартном чате без необходимости в специальной технической реализации.

Концепции и подходы для стандартного чата:

Структурирование запросов по сущностям Пользователи могут явно указывать в своих запросах ключевые программные сущности (язык программирования, библиотеки, функции), следуя выявленным 28 категориям Пример: "Я использую Python (Programming Language) с библиотекой NumPy (Library) и функцией mean() (Library Function), и столкнулся с проблемой..."

Формулирование четкого намерения

Пользователи могут явно указывать цель своего запроса, используя 7 категорий намерений из исследования Пример: "[API Usage] Как использовать функцию X в библиотеке Y?" или "[Errors] Получаю ошибку Z при выполнении..."

Применение выявленных факторов успешности

Позитивный тон (+70% к вероятности решения) Конкретные технические детали вместо общих ссылок Использование специфичных сущностей (Library Function, Library Class) вместо общих (Application) Избегание перегрузки ссылками (отрицательно влияет на решение)

Использование успешных комбинаций сущностей

Для вопросов об ошибках: (Programming Language, Library Function) Для вопросов по

API: (Application, File Type) Для вопросов по обучению: (Programming Language, Library) ### Ожидаемые результаты:

При применении этих концепций в стандартном чате пользователи могут ожидать: - Повышение вероятности получения полезных ответов (до +30% по данным исследования) - Сокращение времени на уточняющие вопросы - Более точные и релевантные ответы от LLM благодаря лучшей структуре запроса

Исследование показывает, что правильное структурирование вопроса с указанием конкретных технических деталей может значительно повысить качество взаимодействия с LLM даже без использования специализированных API или дообучения.

Анализ практической применимости: 1. **SENIR (Software specific Named entity recognition, Intent detection, and Resolution classification)** - Прямая применимость: Высокая. Хотя сама система требует доступа к API LLM, концепция выделения ключевых элементов вопроса может быть адаптирована пользователями для улучшения собственных запросов к LLM и в чатах разработчиков. Пользователи могут следовать выявленным категориям сущностей для структурирования своих вопросов. - Концептуальная ценность: Высокая. Исследование демонстрирует, как структурирование запросов с четким выделением технических деталей и намерений улучшает качество взаимодействия. Это применимо при работе с любыми LLM. - Потенциал для адаптации: Высокий. Хотя полная реализация SENIR требует специализированных знаний, базовые принципы структурирования вопросов могут быть использованы всеми пользователями.

Модель прогнозирования разрешения вопросов Прямая применимость: Средняя. Сама модель требует технической реализации, но выявленные факторы, влияющие на успешность вопросов, могут напрямую использоваться пользователями при составлении запросов. Концептуальная ценность: Высокая. Понимание того, какие факторы влияют на качество ответов, помогает пользователям формулировать более эффективные запросы к LLM и в технических чатах. Потенциал для адаптации: Высокий. Выявленные факторы успешных вопросов (конкретные технические детали, позитивный тон) универсальны и могут применяться в различных контекстах.

Анализ влияния сущностей и намерений на решение вопросов

Прямая применимость: Высокая. Пользователи могут непосредственно использовать выявленные успешные комбинации сущностей и намерений при составлении своих вопросов. Концептуальная ценность: Высокая. Исследование демонстрирует, как специфичность и контекст влияют на качество взаимодействия с системами ИИ. Потенциал для адаптации: Высокий. Принципы сочетания конкретных технических деталей могут быть применены к различным типам запросов и в различных областях.

Рекомендации по формулированию вопросов

Прямая применимость: Очень высокая. Рекомендации конкретны, практичны и могут быть немедленно внедрены любым пользователем. Концептуальная ценность: Высокая. Рекомендации основаны на эмпирических данных и отражают принципы эффективной коммуникации с системами ИИ. Потенциал для адаптации: Очень высокий. Рекомендации универсальны и применимы к широкому спектру взаимодействий с LLM и в технических сообществах.

Prompt:

Применение исследования SENIR в промтах для GPT ## Ключевые знания из исследования для использования

Исследование SENIR показывает, что качество вопросов разработчиков существенно влияет на вероятность получения полезного ответа. Конкретные технические детали, структура вопроса и тональность играют критическую роль в успешном разрешении проблем.

Пример эффективного промта на основе SENIR

[======] # Запрос о проблеме с библиотекой React

Технические детали - **Язык программирования**: JavaScript/TypeScript - **Библиотека/Фреймворк**: React 18.2.0 - **Конкретный компонент/функция**: useEffect hook - **Окружение**: Create React App 5.0.1, Node.js 16.14.2

Описание проблемы Я столкнулся с неожиданным поведением при использовании useEffect. Компонент перерендеривается дважды, хотя зависимости хука не изменяются.

Код с проблемой [======]jsx useEffect(() => { fetchData(userId); console.log('Effect executed'); }, [userId]); [======]

Что я уже пробовал - Добавил проверку на null значения - Использовал useCallback для функции fetchData - Проверил React DevTools на наличие лишних рендеров

Конкретный вопрос Как я могу предотвратить двойной вызов useEffect при неизменных зависимостях в React 18? [======]

Почему этот промт эффективен (на основе исследования)

Структурированность технических деталей — промт включает конкретные программные сущности (React, useEffect, JavaScript), что согласно SENIR повышает вероятность разрешения на 30-40%.

Чёткое намерение — промт явно указывает на проблему с API Usage (использование useEffect), что относится к категории намерений с высоким

показателем разрешения (33.6%).

Конкретность — вместо расплывчатого "у меня проблема с React" промт содержит точные версии, компоненты и функции, что создает эффективные пары сущностей (Library + Library Function).

Позитивная тональность — отсутствуют негативные выражения, которые, согласно исследованию, снижают шансы на получение ответа.

Демонстрация предварительных усилий — раздел "Что я уже пробовал" показывает, что автор предпринял собственные попытки решения, что повышает вероятность получения помощи.

Такая структура промта позволяет GPT лучше понять контекст проблемы и предоставить более точный и полезный ответ, основываясь на конкретных технических деталях.