

Маленький Лог

Публикации

Архив

Поиск

Теги

Вопросы и ответы

поиск смайликов.приложение

Автономные агенты на базе LLM

Дата: 23 июня 2023 г. | Предполагаемое время чтения: 31 мин. | Автор: Лилиан Вэн

Содержание

Создание агентов с использованием LLM (large language model) в качестве основного контроллера - классная концепция. Вдохновляющими примерами служат несколько демонстрационных версий, подтверждающих правильность концепции, таких как AutoGPT, GPT-Engineer и BabyAGI. Потенциал LLM выходит за рамки создания хорошо написанных копий, рассказов, эссе и программ; его можно представить как мощный инструмент для решения общих проблем.

Обзор агентной системы

В автономной агентской системе на базе LLM LLM функционирует как мозг агента, дополненный несколькими ключевыми компонентами:

Планирование

Подцель и декомпозиция: Агент разбивает большие задачи на более мелкие, управляемые подцели, обеспечивая эффективное выполнение сложных задач.

Размышление и утончение: Агент может заниматься самокритикой и саморефлексией по поводу прошлых действий, учиться на ошибках и совершенствовать их для будущих шагов, тем самым улучшая качество конечных результатов.

Память

Кратковременная память: я бы рассматривал все обучение в контексте (см. Быстрое проектирование) как использование кратковременной памяти модели для обучения.

Долговременная память: это обеспечивает агенту возможность сохранять и вызывать (бесконечную) информацию в течение длительных периодов времени, часто за счет использования внешнего векторного хранилища и быстрого поиска.

Использование инструмента

Агент учится вызывать внешние API для получения дополнительной информации, которая отсутствует в весовых коэффициентах модели (часто ее трудно изменить после предварительной подготовки), включая текущую информацию, возможности выполнения кода, доступ к закрытым источникам информации и многое другое.

Рис. 1. Обзор автономной агентной системы на базе LLM.

Компонент первый: Планирование

Сложная задача обычно включает в себя множество шагов. Агент должен знать, в чем они заключаются, и планировать заранее.

Декомпозиция задачи

Цепочка размышлений (CoT; Вэй и др., 2022) стала стандартным методом подсказок для повышения производительности модели при выполнении сложных задач. Модели предписано “думать шаг за шагом”, чтобы использовать больше вычислений во время тестирования для разбиения сложных задач на более мелкие и простые шаги. CoT превращает большие задачи во множество управляемых и проливает свет на интерпретацию процесса мышления модели.

"Древо мыслей" (Яо и др., 2023) расширяет CoT, исследуя множество возможностей рассуждения на каждом этапе. Сначала он разбивает проблему на несколько этапов обдумывания и генерирует несколько мыслей на каждом шаге, создавая древовидную структуру. Процесс поиска может быть BFS (поиск в ширину) или DFS (поиск в глубину), при этом каждое состояние оценивается классификатором (с помощью запроса) или большинством голосов.

Декомпозиция задач может быть выполнена (1) LLM с помощью простых подсказок, таких как "Шаги для XYZ. \ n1.", "Каковы подцели для достижения XYZ?", (2) с помощью инструкций для конкретной задачи; например, "Напишите набросок истории". для написания романа, или (3) с помощью участия человека.

Другой совершенно иной подход, LLM + P (Liu et al., 2023), предполагает использование внешнего классического планировщика для выполнения долгосрочного планирования. Этот подход использует Язык определения предметной области планирования (PDDL) в качестве промежуточного интерфейса для описания проблемы планирования. В этом процессе LLM (1) переводит проблему в “Problem PDDL”, затем (2) запрашивает классический планировщик сгенерировать план PDDL на основе существующего “Domain PDDL” и, наконец, (3) переводит план PDDL обратно на естественный язык. По сути, этап планирования передается на аутсорсинг внешнему инструменту, при условии наличия специфичного для домена PDDL и подходящего планировщика, который распространен в определенных роботизированных установках, но не во многих других доменах.

Саморефлексия

Саморефлексия - жизненно важный аспект, который позволяет автономным агентам итеративно совершенствоваться, уточняя прошлые решения о действиях и исправляя предыдущие ошибки. Это играет решающую роль в реальных задачах, где неизбежны метод проб и ошибок.

ReAct (Яо и др., 2023) объединяет рассуждения и действия в рамках LLM, расширяя пространство действий до сочетания отдельных действий, специфичных для конкретной задачи, и языкового пространства. Первый позволяет LLM взаимодействовать с окружающей средой (например, использовать API поиска в Википедии), в то время как второй побуждает LLM генерировать трассировки рассуждений на естественном языке.

Шаблон запроса ReAct включает в себя четкие инструкции для LLM, которые должны быть продуманы, примерно в следующем формате:

Мысль: ...

Действие: ...

Наблюдение: ...

... (Повторяется много раз)

Рис. 2. Примеры траекторий рассуждений для наукоемких задач (например, HotpotQA, FEVER) и задач принятия решений (например, AlfWorld Env, WebShop). (Источник изображения: Яо и др., 2023).

В обоих экспериментах, посвященных наукоемким задачам и задачам принятия решений, ReAct работает лучше, чем базовый уровень, основанный только на действии, где удален шаг "Мысль: ...".

Reflexion (Shinn & Labash 2023) - это платформа, которая оснащает агентов динамической памятью и возможностями саморефлексии для улучшения навыков рассуждения. Reflexion имеет стандартную настройку RL, в которой модель вознаграждения обеспечивает простое бинарное вознаграждение, а пространство действий соответствует настройке в ReAct где пространство действий для конкретной задачи дополнено языком для выполнения сложных этапов рассуждения. После каждого действия

, агент вычисляет эвристический

и при необходимости могут принять решение сбросить среду для запуска нового испытания в зависимости от результатов самоанализа.

Рис. 3. Иллюстрация структуры рефлексии. ----- (Источник изображения: Shinn & Лабаш, 2023)

Эвристическая функция определяет, когда траектория неэффективна или содержит галлюцинации и ее следует остановить. Неэффективное планирование относится к траекториям, которые занимают слишком много времени и безуспешны. Галлюцинация определяется как столкновение с последовательностью идентичных действий, которые приводят к одному и тому же наблюдению в окружающей среде.

Саморефлексия создается путем показа LLM примеров с двумя кадрами, и каждый пример представляет собой пару (неудачная траектория, идеальное отражение для направления будущих изменений в плане). Затем отражения добавляются в рабочую память агента, до трех, которые будут использоваться в качестве контекста для запроса LLM.

Рис. 4. Эксперименты на AlfWorld Env и HotpotQA. Галлюцинации - более распространенная ошибка, чем неэффективное планирование в AlfWorld. (Источник изображения: Shinn & Лабаш, 2023)

Цепочка ретроспективного анализа (CoH; Liu et al., 2023) побуждает модель улучшать свои собственные результаты, явно представляя ее последовательностью прошлых результатов, каждый из которых снабжен комментариями обратной связи. Данные обратной связи с людьми представляют собой совокупность

, где

является подсказкой, каждая

является завершением модели,

- это человеческая оценка

, и

- это соответствующая обратная связь, предоставленная человеком задним числом.

Предположим, что кортежи отзывов ранжированы по вознаграждению,

Процесс находится под контролем точной настройки, когда данные представляют собой последовательность в форме

, где

. Модель отлажена только для прогнозирования

где зависит от префикса последовательности, так что модель может саморефлексироваться для получения более качественных выходных данных на основе последовательности обратной связи. Во время тестирования модель может дополнительно получать несколько раундов инструкций с помощью человеческих аннотаторов.

Чтобы избежать переобучения, CoH добавляет член регуляризации, чтобы максимизировать логарифмическую вероятность набора данных перед обучением. Чтобы избежать сокращения и копирования (поскольку в последовательностях обратной связи много общих слов), они случайным образом маскируют 0% - 5% прошлых токенов во время обучения.

Обучающий набор данных в их экспериментах представляет собой комбинацию сравнений с WebGPT, обобщения отзывов людей и набора данных о человеческих предпочтениях.

Рис. 5. После точной настройки с помощью CoH модель может следовать инструкциям для получения выходных данных с постепенным улучшением в определенной последовательности. (Источник изображения: Liu et al., 2023)

Идея CoH состоит в том, чтобы представить историю последовательного улучшения результатов в контексте и обучить модель учитывать тенденцию к получению более качественных результатов. Дистилляция алгоритмов (AD; Ласкин и др., 2023) применяет ту же идею к траекториям между эпизодами в задачах обучения с подкреплением, где алгоритм инкапсулирован в политику, обусловленную длительной историей. Учитывая, что агент взаимодействует с окружающей средой много раз, и в каждом эпизоде агент становится немного лучше, AD объединяет эту историю обучения и вводит ее в модель. Следовательно, нам следует ожидать, что следующее прогнозируемое действие приведет к повышению производительности по сравнению с предыдущими испытаниями. Цель состоит в том, чтобы изучить процесс RL вместо того, чтобы обучать саму политику для конкретной задачи.

Рис. 6. Иллюстрация того, как работает алгоритм дистилляции (AD).

(Источник изображения: Ласкин и др., 2023).

В статье выдвигается гипотеза, что любой алгоритм, генерирующий набор историй обучения, может быть преобразован в нейронную сеть путем выполнения поведенческого клонирования поверх действий. Данные истории генерируются набором исходных политик, каждая из которых обучена для выполнения конкретной задачи. На этапе обучения, во время каждого запуска RL, выборка случайной задачи выполняется, и для обучения используется последующая история из нескольких эпизодов, так что изученная политика не зависит от задачи.

На самом деле модель имеет ограниченную длину контекстного окна, поэтому эпизоды должны быть достаточно короткими, чтобы построить историю из нескольких эпизодов. Многосерийные контексты из 2-4 серий необходимы для изучения почти оптимального внутриконтекстного алгоритма RL. Появление внутриконтекстного RL требует достаточно длительного контекста.

По сравнению с тремя базовыми показателями, включая ED (экспертная дистилляция, клонирование поведения с использованием экспертных траекторий вместо истории обучения), политику источника (используется для генерации траекторий для дистилляции UCB), RL^2 (Дуан и др., 2017; используется в качестве верхней границы, поскольку для этого требуется онлайн-RL), AD демонстрирует контекстный RL с производительностью, приближающейся к RL^2 , несмотря на использование только автономного RL, и обучается намного быстрее, чем другие базовые показатели. При использовании частичной истории обучения исходной политике AD также улучшается намного быстрее, чем базовый уровень ED.

Рис. 7. Сравнение AD, ED, политики исходных текстов и RL^2 в средах, требующих памяти и исследования. Назначается только бинарное вознаграждение. Исходные политики обучены с помощью A3C для "темных" сред и DQN для watermaze.

(Источник изображения: Ласкин и др., 2023)

Компонент второй: Память

(Большое спасибо ChatGPT за помощь в подготовке этого раздела. Я многое узнал о человеческом мозге и структуре данных для быстрых MIPS в своих беседах с ChatGPT.)

Типы памяти

Память можно определить как процессы, используемые для получения, хранения, удержания и последующего извлечения информации. В человеческом мозге существует несколько типов памяти.

Сенсорная память: Это самая ранняя стадия памяти, обеспечивающая способность сохранять впечатления от сенсорной информации (визуальной, слуховой, и т.д.) после окончания действия первоначальных стимулов. Сенсорная память обычно длится всего несколько секунд. К подкатегориям относятся иконическая память (визуальная), эхо-память (слуховая) и тактильная память (осязательная).

Кратковременная память (STM) или Рабочая память: она хранит информацию, которая нам в данный момент известна и необходима для выполнения сложных когнитивных задач, таких как обучение и рассуждения. Считается, что кратковременная память содержит около 7 элементов (Miller, 1956) и длится 20-30 секунд.

Долговременная память (LTM): Долговременная память может хранить информацию в течение удивительно длительного времени, от нескольких дней до десятилетий, с практически неограниченным объемом памяти. Существует два подтипа LTM:

Явная / декларативная память: Это память о фактах и событиях, и относится к тем воспоминаниям, которые могут быть вызваны сознательно, включая эпизодическую память (события и переживания) и семантическую память (факты и концепции).

Неявная / процедурная память: Этот тип памяти является бессознательным и включает навыки и процедуры, которые выполняются автоматически, например, при езде на велосипеде или наборе текста на клавиатуре.

Рис. 8. Категоризация человеческой памяти.

Мы можем грубо рассмотреть следующие отображения:

Сенсорная память как обучающие встроенные представления для необработанных входных данных, включая текст, изображение или другие модальности;

Кратковременная память как обучение в контексте. Он короткий и конечный, поскольку он ограничен конечной длиной контекстного окна Transformer.

Долговременная память как внешнее хранилище векторов, к которому агент может обращаться во время запроса, доступное посредством быстрого извлечения.

Максимальный внутренний поиск продукта (MIPS)

Внешняя память может облегчить ограничение ограниченного объема внимания. Стандартной практикой является сохранение встроенного представления информации в базе данных векторного магазина, которая может поддерживать быстрый максимальный поиск внутри продукта (MIPS). Для оптимизации скорости поиска обычно используется алгоритм приближенных ближайших соседей (ANN), который возвращает приблизительно топ-k ближайших соседей, чтобы компенсировать небольшую потерю точности за огромное ускорение.....

Пара распространенных вариантов алгоритмов ANN для быстрых MIPS:

LSH (локально-зависимое хэширование): вводит функцию хэширования таким образом, что похожие входные элементы с высокой вероятностью сопоставляются с одними и теми же сегментами, где количество сегментов намного меньше количества входных данных.

РАЗДРАЖЕНИЕ (приблизительные ближайшие соседи, o Да): Основная структура данных - это деревья случайных проекций, набор двоичных деревьев, где каждый нелистовой узел представляет собой гиперплоскость, разделяющую входное пространство пополам, и в каждом листе хранится одна точка данных. Деревья строятся независимо и случайным образом, поэтому в какой-то степени это имитирует функцию хэширования. РАЗДРАЖАЮЩИЙ поиск выполняется во всех деревьях для итеративного поиска по той половине, которая ближе всего к запросу, а затем агрегирует результаты. Идея во многом связана с KD tree, но намного более масштабируема.

HNSW (иерархический судорожный малый мир): он вдохновлен идеей сетей малого мира, в которых до большинства узлов могут добраться любые другие узлы за небольшое количество шагов; например, функция социальных сетей “шесть степеней разделения”. HNSW создает иерархические слои этих графиков малого мира, где нижние слои содержат фактические точки данных. Слои в середине создают ярлыки для ускорения поиска. При выполнении поиска HNSW начинает со случайного узла в верхнем слое и перемещается к цели. Когда он не может подобраться ближе, он перемещается вниз к следующему слою, пока не достигнет нижнего слоя. Каждое перемещение в верхних слоях потенциально может охватить большое расстояние в пространстве данных, а каждое перемещение в нижних слоях повышает качество поиска.

FAISS (Facebook AI Similarity Search): Он основан на предположении, что в многомерном пространстве расстояния между узлами соответствуют гауссовскому распределению и, следовательно, должна существовать кластеризация точек данных. FAISS применяет векторное квантование, разбивая векторное пространство на кластеры, а затем уточняя квантование внутри кластеров. Поиск сначала ищет кандидатов в кластеры с грубым квантованием, а затем дополнительно изучает каждый кластер с более тонким квантованием.

ScaNN (масштабируемые ближайшие соседи): Основным нововведением в ScaNN является анизотропное векторное квантование. Он квантует точку данных

Для

such чтобы внутренний продукт

является таким же похожим на исходное расстояние

насколько возможно, вместо того, чтобы выбирать точки центроида квантования в шкафу.

Рис. 9. Сравнение алгоритмов MIPS, измеренных в $\text{recall}@10$. (Источник изображения: блог Google, 2020)

Ознакомьтесь с другими алгоритмами MIPS и сравнением производительности в разделе ann-benchmarks.com.

Компонент третий: Использование инструмента

Использование инструментов - замечательная и отличительная черта человека. Мы создаем, модифицируем и используем внешние объекты для выполнения действий, выходящих за пределы наших физических и когнитивных возможностей. Оснащение LLMs внешними инструментами может значительно расширить возможности модели.

Рис. 10. Изображение морской выдры, которая, плавая в воде, раскалывает камнем морскую раковину. В то время как некоторые другие животные могут использовать инструменты, сложность не сравнима с людьми. (Источник изображения: Животные, использующие инструменты)

MRKL (Карпас и др., 2022), сокращение от "Модульного мышления, знаний и языка", представляет собой нейросимвольную архитектуру для автономных агентов. Предлагается, чтобы система MRKL содержала набор "экспертных" модулей, а LLM общего назначения работает как маршрутизатор для направления запросов к наиболее подходящему экспертному модулю. Эти модули могут быть нейронными (например, модели глубокого обучения) или символьными (например, математический калькулятор, конвертер валют, погодный API).

Они провели эксперимент по точной настройке LLM для вызова калькулятора, используя арифметику в качестве тестового примера. Их эксперименты показали, что решать вербальные математические задачи было сложнее, чем явно сформулированные математические задачи, потому что LLMs (модель 7B Jurassic1-large) не смогла надежно извлечь правильные аргументы для базовой арифметики. Результаты показывают, когда

внешние символические инструменты могут работать надежно, знание того, когда и как использовать инструменты, имеет решающее значение, что определяется возможностями LLM.

Как TALM (Tool Augmented Language Models; Паризи и др., 2022), так и Toolformer (Шик и др., 2023) точно настраивают LM, чтобы научиться использовать API внешних инструментов. Набор данных расширяется в зависимости от того, может ли недавно добавленная аннотация вызова API улучшить качество выходных данных модели. Смотрите более подробную информацию в разделе “Внешние API” сайта Prompt Engineering.

Плагины ChatGPT и вызов функций API OpenAI являются хорошими примерами LLM, дополненных возможностями использования инструментов, работающих на практике. Набор инструментальных API может предоставляться другими разработчиками (как в плагинах) или определяться самостоятельно (как в вызовах функций).

HuggingGPT (Shen et al., 2023) - это платформа для использования ChatGPT в качестве планировщика задач для выбора моделей, доступных на платформе HuggingFace, в соответствии с описаниями моделей и обобщения ответов на основе результатов выполнения.

Рис. 11. Иллюстрация того, как работает HuggingGPT. (Источник изображения: Shen et al., 2023)

Система состоит из 4 этапов:

(1) Планирование задач: LLM работает как мозг и разбивает запросы пользователей на несколько задач. С каждой задачей связаны четыре атрибута: тип задачи, идентификатор, зависимости и аргументы. В них используются краткие примеры, помогающие LLM выполнять синтаксический анализ и планирование задач.

Инструкция:

Помощник искусственного интеллекта может анализировать вводимые пользователем данные для нескольких задач: [{"task": задача, "id", task_id, "dep": dependency_task_ids, "аргументы": {"text": текст, "image": URL, "audio": URL, "video": URL}}]. Поле "dep" обозначает идентификатор предыдущей задачи, которая генерирует новый ресурс, на который опирается текущая задача. Специальный тег "-task_id" ссылается

на сгенерированное текстовое изображение, аудио и видео в задаче зависимости с идентификатором `task_id`. Задачу НЕОБХОДИМО выбрать из следующих вариантов: `{{ Доступный список задач }}`. Между задачами существует логическая взаимосвязь, пожалуйста, обратите внимание на их порядок. Если пользовательский ввод невозможно разобрать, вам нужно ответить пустым JSON. Вот несколько примеров для вашей справки: `{{ Демонстрации }}`. История чата записывается как `{{ История чата }}`. Из этой истории чатов вы можете найти путь к ресурсам, упомянутым пользователем для планирования вашей задачи.

(2) Выбор модели: LLM распределяет задачи по экспертным моделям, где запрос оформляется как вопрос с несколькими вариантами ответов. LLM представлен список моделей на выбор. Из-за ограниченной длины контекста необходима фильтрация по типу задачи.

Инструкция:

Получив запрос пользователя и команду вызова, помощник искусственного интеллекта помогает пользователю выбрать подходящую модель из списка моделей для обработки запроса пользователя. Помощник по искусственному интеллекту просто выводит идентификатор наиболее подходящей модели. Выходные данные должны быть в строгом формате JSON: `"id": "id", "reason": "ваша подробная причина выбора"`. У нас есть список моделей, из которых вы можете выбрать `{{ Модели-кандидаты }}`. Пожалуйста, выберите одну модель из списка.

(3) Выполнение задачи: Экспертные модели выполняются для конкретных задач и регистрируют результаты.

Инструкция:

Используя входные данные и результаты логического вывода, ассистент с искусственным интеллектом должен описать процесс и результаты. Предыдущие этапы могут быть сформированы как - Пользовательский ввод: `{{ Пользовательский ввод }}`, Планирование задачи: `{{ Задачи }}`, Выбор модели: `{{ Назначение модели }}`, Выполнение задачи: `{{ Прогнозы }}`. Сначала вы должны прямо ответить на запрос пользователя. Затем опишите процесс выполнения задачи и покажите пользователю результаты вашего анализа и вывода модели от первого лица. Если результаты логического вывода содержат путь к файлу, необходимо указать пользователю полный путь к файлу.

(4) Генерация ответа: LLM получает результаты выполнения и предоставляет пользователям обобщенные результаты.

Чтобы использовать HuggingGPT в реальном мире, необходимо решить пару проблем: (1) Необходимо повысить эффективность, поскольку как этапы вывода LLM, так и взаимодействие с другими моделями замедляют процесс; (2) Он полагается на длинное контекстное окно для обмена информацией по сложному содержанию задачи; (3) Повышение стабильности результатов LLM и сервисов внешней модели.

API-Bank (Li et al., 2023) является эталоном для оценки производительности LLM, дополненных инструментами. Он содержит 53 часто используемых инструмента API, полный рабочий процесс LLM, дополненный инструментами, и 264 аннотированных диалоговых окна, которые включают 568 вызовов API. Выбор API довольно разнообразен, включая поисковые системы, калькулятор, запросы календаря, управление умным домом, управление расписанием, управление данными о состоянии здоровья, рабочий процесс аутентификации учетной записи и многое другое. Поскольку существует большое количество API, LLM сначала получает доступ к поисковой системе API, чтобы найти подходящий API для вызова, а затем использует соответствующую документацию для выполнения вызова.

Рис. 12. Псевдокод того, как LLM выполняет вызов API в API-Банке. (Источник изображения: Ли и др., 2023)

В рабочем процессе API-Банка LLM необходимо принять пару решений, и на каждом шаге мы можем оценить, насколько точным является это решение. Решения включают:

Нужен ли вызов API.

Определите правильный API для вызова: если этого недостаточно, LLM необходимо итеративно изменять входные данные API (например, определять ключевые слова для поиска в API поисковой системы).

Ответ на основе результатов API: модель может выбрать уточнение и вызвать его снова, если результаты не удовлетворены.

Этот тест оценивает возможности агента по использованию инструментов на трех уровнях:

Уровень-1 оценивает возможность вызова API. Учитывая описание API, модели необходимо определить, следует ли вызывать данный API, вызывать его правильно и должным образом реагировать на возвраты API.

Уровень 2 проверяет возможность извлечения API. Модели необходимо выполнить поиск возможных API-интерфейсов, которые могут удовлетворить требования пользователя, и научиться их использованию, прочитав документацию.

Уровень 3 оценивает способность планировать API помимо извлечения и вызова. Учитывая непонятно запросов пользователей (например, расписание матчей группы встреч, бронируйте рейс/отель/ресторан для поездки), модель, возможно, придется провести несколько вызовов API, чтобы решить его.

Примеры внедрения

Агент научных открытий

ChemCrow (Bran et al., 2023) - пример для конкретной предметной области, в котором LLM дополнен 13 разработанными экспертами инструментами для решения задач в области органического синтеза, разработки лекарств и проектирования материалов. Рабочий процесс, реализованный в LangChain, отражает то, что ранее было описано в ReAct и MRKLs, и сочетает в себе рассуждения CoT с инструментами, соответствующими задачам:

Магистру права предоставляется список названий инструментов, описания их полезности и подробная информация об ожидаемом вводе / выводе.

Затем ему дается указание ответить на запрос пользователя, используя при необходимости предоставленные инструменты. Инструкция предлагает модели следовать формату ReAct - Мысль, Действие, Ввод действия, наблюдение.

Одно интересное наблюдение заключается в том, что, хотя оценка на основе LLM показала, что GPT-4 и ChemCrow работают почти одинаково, человеческие оценки с привлечением экспертов, ориентированных на полноту и химическую корректность решений, показали, что ChemCrow значительно превосходит GPT-4. Это указывает на потенциальную проблему с использованием LLM для оценки собственной эффективности в областях, требующих глубоких знаний. Недостаток опыта может привести к тому, что LLM не будут знать о своих недостатках и, следовательно, не смогут правильно оценить правильность результатов задания.

Бойко и др. (2023) также изучили агентов, наделенных полномочиями LLM в области научных открытий, для автономного проектирования, планирования и проведения сложных научных экспериментов. Этот агент может использовать инструменты для просмотра Интернета, чтения документации, выполнения кода, вызова API для экспериментов с робототехникой и использования других LLM.

Например, при запросе "разработать новый противоопухолевый препарат" модель предложила следующие этапы обоснования:

поинтересовался текущими тенденциями в разработке противоопухолевых препаратов;

выбран объект;

запросил каркас, нацеленный на эти соединения;

Как только соединение было идентифицировано, модель предприняла попытку его синтеза.

Они также обсудили риски, особенно связанные с незаконными наркотиками и биологическим оружием. Они разработали тестовый набор, содержащий список известных агентов химического оружия, и попросили агента синтезировать их. 4 из 11 запросов (36%) были удовлетворены для получения обобщающего решения, и агент попытался ознакомиться с документацией для выполнения процедуры. 7 из 11 были отклонены, и среди этих 7 отклоненных случаев 5 произошли после веб-поиска, в то время как 2 были отклонены только на основании запроса.

Моделирование генеративных агентов

Generative Agents (Парк и др., 2023) - это супер веселый эксперимент, в котором 25 виртуальных персонажей, каждым из которых управляет агент с поддержкой LLM, живут и взаимодействуют в среде песочницы, вдохновленной The Sims. Генеративные агенты создают правдоподобные симулякры человеческого поведения для интерактивных приложений.

Дизайн генеративных агентов сочетает LLM с механизмами памяти, планирования и отражения, позволяя агентам вести себя в соответствии с прошлым опытом, а также взаимодействовать с другими агентами.

Поток памяти: представляет собой модуль долговременной памяти (внешняя база данных), который записывает полный список опыта агентов на естественном языке.

Каждый элемент - это наблюдение, событие, предоставленное непосредственно агентом. - Межагентная коммуникация может запускать новые утверждения на естественном языке.

Модель поиска: выявляет контекст для информирования о поведении агента в соответствии с релевантностью, новизной и важностью.

Новизна: недавние события имеют более высокие оценки

Важность: отличайте обычные воспоминания от основных. Спросите LM напрямую.

Актуальность: зависит от того, насколько это связано с текущей ситуацией / запросом.

Механизм отражения: со временем синтезирует воспоминания в умозаключения более высокого уровня и направляет будущее поведение агента. Это резюме прошлых событий более высокого уровня (<- обратите внимание, что это немного отличается от саморефлексии, приведенной выше)

Подсказать LM 100 самых последних наблюдений и сгенерировать 3 наиболее важных вопроса высокого уровня с учетом набора наблюдений / утверждений. Затем попросите LM ответить на эти вопросы.

Планирование и реагирование: преобразуйте размышления и информацию об окружающей среде в действия

Планирование, по сути, направлено на то, чтобы оптимизировать правдоподобие в данный момент по сравнению со временем.

Шаблон приглашения: {Вводная часть агента X}. Вот сегодняшний план Икс в общих чертах: 1)

Взаимоотношения между агентами и наблюдения за одним агентом другим принимаются во внимание при планировании и реагировании.

Информация об окружающей среде представлена в виде древовидной структуры.

Рис. 13. Архитектура генерирующего агента. (Источник изображения: Парк и др., 2023)

Эта забавная симуляция приводит к появлению нового социального поведения, такого как распространение информации, запоминание отношений (например, два агента продолжают тему разговора) и координация социальных мероприятий (например, организуйте вечеринку и пригласите многих других).

Примеры подтверждения концепции

AutoGPT привлек много внимания к возможности настройки автономных агентов с LLM в качестве основного контроллера. У него довольно много проблем с надежностью, учитывая интерфейс на естественном языке, но, тем не менее, это классная демонстрация концепции. Большая часть кода в AutoGPT посвящена разбору формата.

Вот системное сообщение, используемое AutoGPT, где `{{...}}` - входные данные пользователя:

Вы `{{ai-имя}}`, `{{описание AI-бота, предоставленное пользователем}}`.

Ваши решения всегда должны приниматься самостоятельно, без обращения за помощью к пользователю. Используйте свои сильные стороны как магистра права и придерживайтесь простых стратегий без каких-либо юридических сложностей.

ГОЛЫ:

1. `{{предоставленная пользователем цель 1}}`
2. `{{предоставленная пользователем цель 2}}`
3. ...
4. ...
5. ...

Ограничения:

1. ограничение в ~ 4000 слов для кратковременной памяти. Вашей кратковременной памяти мало, поэтому немедленно сохраняйте важную информацию в файлах.

2. Если вы не уверены, как вы ранее что-то делали, или хотите вспомнить прошлые события, размышления о похожих событиях помогут вам вспомнить.

3. Отсутствие помощи пользователю

4. Используйте исключительно команды, заключенные в двойные кавычки, например, "название команды"

5. Используйте подпроцессы для команд, которые не завершатся в течение нескольких минут

Команды:

1. Поиск в Google: "google", аргументы: "ввод": "<поиск>"

2. Просмотрите веб-сайт: "browse_website", аргументы: "url": "<url>", "вопрос": "<what_you_want_to_find_on_website>"

3. Запустите GPT-агент: "start_agent", аргументы: "name": "<имя>", "task": "<short_task_desc>", "prompt": "<подсказка>"

4. Сообщение GPT-агента: "message_agent", аргументы: "ключ": "<ключ>", "сообщение": "<сообщение>"

5. Перечислите GPT-агенты: "list_agents", аргументы:

6. Удалить GPT-агент: "delete_agent", аргументы: "ключ": "<ключ>"

7. Репозиторий для клонирования: "clone_repository", аргументы: "repository_url": "<url>", "clone_path": "<каталог>"

8. Записать в файл: "write_to_file", аргументы: "файл": "<файл>", "текст": "<текст>"

9. Прочитайте файл: "read_file", аргументы: "file": "<файл>"

10. Добавьте к файлу: "append_to_file", аргументы: "файл": "<файл>", "текст": "<текст>"

11. Удалить файл: "delete_file", аргументы: "file": "<файл>"

12. Поиск файлов: "search_files", аргументы: "каталог": "<каталог>"
13. Проанализируйте код: "analyze_code", аргументы: "code": "<full_code_string>"
14. Получаем улучшенный код: "improve_code", аргументы: "предложения": "<список_о_предложений>", "код": "<full_code_string>"
15. Напишите тесты: "write_tests", аргументы: "code": "<full_code_string>", "focus": "<list_of_focus_areas>"
16. Запустите файл Python: "execute_python_file", аргументы: "file": "<файл>"
17. Сгенерируйте изображение: "generate_image", аргументы: "подсказка": "<подсказка>"
18. Отправить твит: "send_tweet", аргументы: "текст": "<текст>"
19. Ничего не делать: "ничего не делать", аргументы:
20. Завершение задачи (завершение работы): "task_complete", аргументы: "причина": "<причина>"

Ресурсы:

1. Доступ в Интернет для поиска и сбора информации.
2. Управление долговременной памятью.
3. Агенты с поддержкой GPT-3.5 для делегирования простых задач.
4. Вывод файла.

Оценка эффективности:

1. Постоянно просматривайте и анализируйте свои действия, чтобы убедиться, что вы выполняете их в меру своих возможностей.
2. Постоянно конструктивно самокритикуйте свое поведение в целом.
3. Подумайте о прошлых решениях и стратегиях, чтобы усовершенствовать свой подход.

4. За каждую команду приходится платить, поэтому будьте умны и эффективны. Стремитесь выполнять задачи за наименьшее количество шагов.

Вы должны отвечать только в формате JSON, как описано ниже

Формат ответа:

```
{  
  
  "мысли": {  
  
    "текст": "мысль",  
  
    "рассуждение": "рассуждение",  
  
    "план": "- краткий маркированный список, содержащий \n- долгосрочный план",  
  
    "критика": "конструктивная самокритика",  
  
    "говорить": "краткое изложение мыслей, которое нужно высказать пользователю"  
  
  },  
  
  "команда": {  
  
    "name": "имя команды",  
  
    "аргументы": {  
  
      "arg name": "значение"  
  
    }  
  
  }  
}
```

Убедитесь, что ответ может быть проанализирован с помощью `python json.loads`

GPT-Engineer - это еще один проект по созданию целого репозитория кода с учетом задачи, заданной на естественном языке. GPT-инженеру дается указание продумать список

компонентов меньшего размера для сборки и запросить информацию пользователя для уточнения вопросов по мере необходимости.

Здесь приведен пример диалога для уточнения задачи, отправленного на конечную точку OpenAI ChatCompletion, используемую GPT-инженером. Входные данные пользователя заключены в {{текст пользовательского ввода}}.

[

{

"роль": "система",

"содержание": "Вы будете читать инструкции и не выполнять их, а только стремиться разъяснить их.\В частности, сначала вы составите краткий список областей, требующих уточнения.\Затем вы выберете один уточняющий вопрос и будете ждать ответа от пользователя.\n"

},

{

"роль": "пользователь",

"содержание": "Мы пишем {{игру Super Mario на python. Компоненты MVC разделены на отдельные файлы. Управление с клавиатуры.}}\n"

},

{

"роль": "ассистент",

"содержание": "Краткое описание областей, требующих уточнения:\n1. Особенности игры Super Mario (например, дизайн уровней, персонажи, механика игрового процесса)\n2. Подробные сведения о компонентах MVC (например, какие компоненты находятся в каждом файле)\n3. Реализация управления клавиатурой (например, какие клавиши использовать, как обрабатывать ввод) \n\nУточняющий вопрос: "Можете вы предоставить

более подробную информацию об игре Super Mario, такую как дизайн уровней, персонажи и механика игрового процесса?"

},

{

"роль": "пользователь",

"содержание": "{{Всего 10 уровней. Главный герой - водопроводчик по имени Марио, который умеет ходить и прыгать. Это классическая платформерная игра, похожая на Super Mario. Главный герой перемещается слева направо, пытаясь добраться до места назначения, где в процессе встречается множество препятствий и атак со стороны врагов.}}\n\nИс что-нибудь еще непонятно? Если да, отвечайте только в форме: \n\n{оставшиеся неясные области} оставшиеся вопросы.\n{Следующий вопрос}\nЕсли все достаточно ясно, ответьте только: "Больше нечего прояснять.\n"."

},

{

"роль": "ассистент",

"содержание": "Оставшиеся неясными области: 2 оставшихся вопроса.\nМожете вы предоставить дополнительную информацию о том, как компоненты MVC разделяются на отдельные файлы?"

},

{

"роль": "пользователь",

"содержание": "{{Сделайте свои собственные предположения и четко сформулируйте их перед началом}}"

}

]

Затем, после этих разъяснений, агент перешел в режим написания кода с другим системным сообщением. Системное сообщение:

Вы получите инструкции по написанию кода. Вы напишете очень длинный ответ. Убедитесь, что каждая деталь архитектуры, в конце концов, реализована в виде кода. Убедитесь, что каждая деталь архитектуры, в конце концов, реализована в виде кода.

Продумывайте шаг за шагом и приводите себя к правильным решениям, чтобы убедиться, что мы все делаем правильно. Сначала вы изложите названия основных классов, функций, методов, которые будут необходимы, а также кратко прокомментируете их назначение.

Затем вы будете выводить содержимое каждого файла, включая ВЕСЬ код. Каждый файл должен строго соответствовать формату блока кода markdown, где следующие маркеры должны быть заменены таким образом, что FILENAME - это имя файла в нижнем регистре, включая расширение файла, LANG - это язык блока кода разметки для языка кода, а CODE - это код:

ИМЯ ФАЙЛА

код

Вы начнете с файла “entrypoint”, затем перейдете к тем, которые импортированы этим файлом, и так далее. Пожалуйста, обратите внимание, что код должен быть полностью функциональным. Никаких заполнителей.

Следуйте рекомендациям по именованию файлов, соответствующим языку и структуре. Убедитесь, что файлы содержат все импортируемые файлы, типы и т.д. Убедитесь, что код в разных файлах совместим друг с другом. Убедитесь, что вы реализовали весь код, если вы не уверены, напишите правдоподобную реализацию. Включите файл определения зависимости модуля или менеджера пакетов. Прежде чем закончить, дважды проверьте, присутствуют ли в файлах все части архитектуры.

Полезно знать: вы почти всегда помещаете разные классы в разные файлы. Для Python вы всегда создаете соответствующий файл requirements.txt. Для NodeJS вы всегда создаете соответствующий файл package.json. Вы всегда добавляете комментарий, кратко описывающий назначение определения функции. Вы пытаетесь добавить комментарии, объясняющие очень сложные фрагменты логики. Вы всегда следуете лучшим практикам

для запрашиваемых языков с точки зрения описания кода, написанного как определенный пакет / проект.

Настройкиинструментов дляpython :,

пробный тест

КЛАССЫ ДАННЫХ

Разговорные образцы:

[

$$\{$$

"роль": "система",

"содержание": "Вы получите инструкции по написанию кода. Ты напишешь очень длинный ответ. Убедитесь, что каждая деталь архитектуры, в конце концов, реализована в виде кода. Убедитесь, что каждая деталь архитектуры, в конце концов, реализована в виде кода. \n N Продумывайте шаг за шагом и приводите себя к правильным решениям, чтобы убедиться, что мы все делаем правильно. Вы сначала изложите названия основных классов, функций, методов, которые будут необходимы, а также кратко прокомментируете их назначение. \n Затем вы будете выводить содержимое каждого файла, включая ВЕСЬ код. Каждый файл должен строго соответствовать формату блока кода markdown, где следующие маркеры должны быть заменены таким образом, что \n FILENAME - это имя файла в нижнем регистре, включая расширение файла, \n LANG - это язык блока кода разметки для языка кода, а CODE - это код: \n \n FILENAME \n `` LANG \n \n Code \n `` \n \n Вы будете начинаться с \"entrypoint\" файл, затем перейдите к тем, которые импортированы этим файлом, и так далее. Пожалуйста, обратите внимание, что код должен быть полностью функциональным. Никаких заполнителей. \n \n Follow соответствующее языку и структуре соглашение об именовании файлов, соответствующее передовой практике. \n Make убедитесь, что файлы содержат все импортируемые файлы, типы и т.д. Убедитесь, что код в разных файлах совместим друг с другом. Обязательно для реализации всего кода, если вы не уверены, напишите правдоподобную реализацию. Не включает файл определения зависимости модуля или менеджера пакетов. \n Прежде, чем вы закончите, дважды проверьте, присутствуют ли в файлах все части архитектуры. \n \n Полезно знать: Вы почти всегда помещаете разные классы в разные файлы. Для Python вы всегда создаете

соответствующий файл requirements.txt.\Для NodeJS вы всегда создаете соответствующий файл package.json.\Вы всегда добавляете комментарий, кратко описывающий назначение определения функции.\Вы пытаетесь добавлять комментарии, объясняющие очень сложные фрагменты логики.Вы всегда следуете лучшим практикам для запрашиваемых языков с точки зрения описания кода, написанного как определенный пакет/проект.\n\nPython toolbelt настройках:\n- pytest\n- классы данных\n"

},

... тот же разговор, что и ранее, закончился словами "Сделайте свои собственные предположения и четко изложите их перед началом".

{

"роль": "ассистент",

"содержание": "Предположения: \n1. Модель: Модель будет содержать данные игры, такие как информация об уровне, состояния персонажей и позиции врагов.\n2. Вид: вид будет обрабатывать визуальные элементы игры, включая визуализацию игровых объектов, фона и обновление дисплея. \n3. Контроллер: контроллер будет управлять пользовательским вводом, таким как элементы управления с клавиатуры, и соответствующим образом обновлять модель.\nИмея в виду эти предположения, есть ли какой-либо конкретный аспект реализации управления с клавиатуры, который вы хотели бы, чтобы я прояснил?"

},

{

"роль": "пользователь",

"содержание": "Теперь, пожалуйста, запомните шаги: \n\nПродумывайте шаг за шагом и приводите себя к правильным решениям, чтобы убедиться, что мы все делаем правильно.\nСначала изложите названия основных классов, функций, методов, которые будут необходимы, а также краткий комментарий об их назначении.\nЗатем вы будете выводить содержимое каждого файла, включая ВСЕ код.\nКаждый файл должен строго соответствовать формату блока кода markdown, где следующие маркеры должны быть заменены таким образом, что\nFILENAME - это имя файла в нижнем регистре, включая

расширение файла, \nLANG - это язык блока кода разметки для языка кода, а CODE - это код: \n\nFILENAME\n``LANG\nCode\n``\n\nPlease обратите внимание, что код должен быть полностью функциональным. Никаких заполнителей.\n\ Вы начнете с файла \"entrypoint\", затем перейдете к тем, которые импортированы этим файлом, и так далее.\nСледуйте соглашению об именовании файлов, соответствующему языку и фреймворку наилучшей практики.\nMake убедитесь, что файлы содержат все импортируемые файлы, типы и т.д. Код должен быть полностью функциональным. Убедитесь, что коды в разных файлах совместимы друг с другом.\nПрежде чем закончить, дважды проверьте, присутствуют ли в файлах все части архитектуры.\n"

}

]

Проблемы

После ознакомления с ключевыми идеями и демонстрациями создания агентов, ориентированных на LLM, я начинаю видеть пару общих ограничений:

Конечная длина контекста: ограниченная емкость контекста ограничивает включение исторической информации, подробных инструкций, контекста вызова API и ответов. Дизайн системы должен работать с такой ограниченной пропускной способностью связи, в то время как такие механизмы, как саморефлексия для извлечения уроков из прошлых ошибок, значительно выиграли бы от длинных или бесконечных контекстных окон. Хотя векторные хранилища и поисковые системы могут обеспечить доступ к большому объему знаний, их способность к представлению не так велика, как полное внимание.

Проблемы долгосрочного планирования и декомпозиции задач: Планирование на протяжении длительного периода и эффективное изучение пространства решений остаются сложными. LLM с трудом корректируют планы, сталкиваясь с неожиданными ошибками, что делает их менее надежными по сравнению с людьми, которые учатся методом проб и ошибок.

Надежность интерфейса на естественном языке: Текущая система агентов использует естественный язык в качестве интерфейса между LLMS и внешними компонентами, такими как память и инструменты. Однако надежность выходных данных

модели сомнительна, поскольку LLM могут допускать ошибки форматирования и иногда проявлять непослушное поведение (например, отказываться следовать инструкциям). Следовательно, большая часть демонстрационного кода агента сосредоточена на анализе выходных данных модели.

Цитата

Цитируется как:

Вэн, Лилиан. (июнь 2023 г.). “Автономные агенты на базе LLM”. Маленький Лог.
[https://lilianweng.github.io/posts/2023-06-23-agent /](https://lilianweng.github.io/posts/2023-06-23-agent/).

Или

```
@article{агент weng2023,
```

```
title = "Автономные агенты на базе LLM",
```

```
автор = "Вэн, Лилиан",
```

```
журнал = "lilianweng.github.io",
```

```
год = "2023",
```

```
месяц = "июнь",
```

```
url = "https://lilianweng.github.io/posts/2023-06-23-agent /"
```

```
}
```

Список литературы

[1] Вэй и др. “Подсказка цепочки мыслей выявляет рассуждения в больших языковых моделях”. НейрОНЫ 2022

[2] Яо и др. “Древо мыслей: свободное решение проблем с помощью больших языковых моделей”. arXiv препринт arXiv: 2305.10601 (2023).

[3] Лю и др. “Цепочка ретроспективного анализа приводит языковые модели в соответствие с обратной связью“ arXiv препринт arXiv: 2302.02676 (2023).

[4] Лю и др. “LLM + P: расширение возможностей больших языковых моделей с оптимальным уровнем планирования” arXiv препринт arXiv: 2304.11477 (2023).

[5] Яо и др. “Реагировать: синергия рассуждений и действий в языковых моделях”. ICLR 2023.

[6] Блог Google. “Анонсируем ScaNN: эффективный поиск векторного сходства” 28 июля 2020 года.

[7] <https://chat.openai.com/share/46ff149e-a4c7-4dd7-a800-fc4a642ea389>

[8] Шинн и Лабаш. “Рефлексия: автономный агент с динамической памятью и саморефлексией” arXiv препринт arXiv: 2303.11366 (2023).

[9] Ласкин и др. “Обучение с контекстным подкреплением с использованием алгоритма дистилляции” ICLR 2023.

[10] Карпас и др. “MRKL представляет собой модульную нейросимвольную архитектуру, которая сочетает в себе большие языковые модели, внешние источники знаний и дискретные рассуждения”. arXiv препринт arXiv: 2205.00445 (2022).

[11] Накано и др. “Webgpt: ответы на вопросы в браузере с обратной связью от человека”. arXiv препринт arXiv: 2112.09332 (2021).

[12] Паризи и др. “TALM: инструментальные расширенные языковые модели”

[13] Шик и др. “Формирователь инструментов: языковые модели могут сами научиться использовать инструменты”. arXiv препринт arXiv: 2302.04761 (2023).

[14] Weaviate. Почему векторный поиск такой быстрый? 13 сентября 2022 года.

[15] Li et al. “API-банк: эталон для LLM, дополненных инструментами” arXiv препринт arXiv: 2304.08244 (2023).

[16] Шен и др. “HuggingGPT: решение задач искусственного интеллекта с помощью ChatGPT и его друзей в HuggingFace” arXiv препринт arXiv: 2303.17580 (2023).

[17] Бран и др. “ChemCrow: расширение моделей на больших языках с помощью инструментов химии”. arXiv препринт arXiv: 2304.05376 (2023).

[18] Бойко и др. “Возникающие автономные возможности научных исследований больших языковых моделей”. arXiv препринт arXiv: 2304.05332 (2023).

[19] Чжун Пак Сон и др. “Генеративные агенты: интерактивные симулякры человеческого поведения”. arXiv препринт arXiv: 2304.03442 (2023).

[20] Автоматический захват. <https://github.com/Significant-Gravitas/Auto-GPT>

[21] GPT-инженер. <https://github.com/AntonOsika/gpt-engineer>

НЛП

языковая модель

агент

управляемость

подсказка

«

Состязательные атаки на LLM

»

Оперативное Проектирование

© 2024 Lill'Log Powered Хьюго и РаперМод