

Создание ИИ-агента для взаимодействия с внешними инструментами

Юрчук Анна Михайловна

Приамурский государственный университет имени Шолом-Алейхема

Студент

Аннотация

Цель исследования — разработка интерактивного инструмента для создания ИИ-агента, способного взаимодействовать с внешними инструментами (например, API для получения погоды) и облачных решений Hugging Face. Методология включает пошаговое построение системы с демонстрацией кода и обсуждением ключевых проблем, таких как галлюцинации моделей. Результаты показывают, что предлагаемый подход позволяет обучать студентов базовым принципам агентов без углубления в сложные фреймворки.

Ключевые слова: ИИ-агенты, Hugging Face, галлюцинации, демонстрационные библиотеки, Google Colab, Python.

Creation of AI-agent for interaction with external tools

Yurchuk Anna Mikhailovna

Sholom Aleichem Priamurskiy State University

Student

Abstract

The aim of the research is to develop an interactive tool for building an AI agent capable of interacting with external tools (e.g. weather APIs) and cloud-based Hugging Face solutions. The methodology includes a step-by-step construction of the system with code demonstrations and discussion of key issues such as model hallucinations. The results show that the proposed approach allows teaching students the basic principles of agents without delving into complex frameworks.

Keywords: AI agents, Hugging Face, hallucination, demonstration libraries, Google Colab, Python.

1. Введение

1.1 Актуальность

Исследование и разработка ИИ-агентов становится ключевым направлением в современном мире искусственного интеллекта, так как они позволяют автоматизировать задачи, требующие взаимодействия с внешними системами (API). В условиях быстрого развития технологий и роста спроса на автоматизацию, агенты играют важную роль в различных областях, таких как финансы, здравоохранение, робототехника и автоматизация бизнес-процессов.

Однако, несмотря на широкое распространение и важность ИИ-агентов, существует значительная проблема в области образования: отсутствие простых и доступных учебных материалов для студентов и начинающих разработчиков. Большинство существующих учебников и курсов сосредоточены на конкретных фреймворках и инструментах, что усложняет процесс обучения и делает его менее гибким. В результате, студенты часто сталкиваются с трудностями при переходе от теории к практике, особенно если они хотят использовать свои знания в различных проектах.

Для решения этой проблемы важно создать образовательные материалы, которые будут независимы от конкретных фреймворков и сосредоточены на понимании концепций ИИ-агентов. В данной статье применяется подход, использующий демонстрационные библиотеки (`huggingface_hub` и `transformers`) и облачные решения (Hugging Face [1] и Google Colab) для обучения студентов созданию простых агентов. Этот подход позволяет учащимся освоить базовые принципы работы агентов без необходимости углубляться в сложные технические детали, что делает процесс обучения более доступным и интересным.

Кроме того, использование облачной платформы Google Colab, устраняет необходимость в установке дополнительного программного обеспечения и предоставляет возможность работать с мощными вычислительными ресурсами прямо из браузера.

1.2 Обзор исследований

В статье А. О. Исакова, Н. Ф. Гусаровой, Д. А. Добренко и А. А. Голубевой рассматривается, как сделать поведение искусственных агентов в системах поддержки принятия клинических решений более понятным для людей. Авторы говорят о том, что важно не только чтобы агенты принимали правильные решения, но и могли объяснить, как они к ним пришли. Для этого используются специальные методы обучения, которые помогают агентам объяснять свои действия [2].

Ольга Ведерникова в своей работе рассказывает о создании ИИ-агентов на основе языковых моделей (LLM), таких как ChatGPT, которые могут взаимодействовать с людьми и другими системами для автоматизации задач, таких как анализ данных и принятие решений. Авторы также демонстрируют, как мультиагентные системы и платформа Epsilon Workspace используют эти технологии для улучшения бизнес-аналитики и интеграции ИИ в процессы обработки данных [3].

Применение нейронных сетей для прогнозирования финансовых временных рядов, таких как курсы акций или другие рыночные показатели, описан в работе А. А. Архиповой. Автор показывает, что нейросетевые модели (например, LSTM) могут точнее предсказывать будущие значения, учитывая сложные зависимости в данных, по сравнению с традиционными методами анализа, и предлагает алгоритмы для улучшения качества прогнозов в условиях нестабильных рынков [4].

О. С. Видмант в своей работе использовал рекуррентные нейронные сети LSTM для прогнозирования финансовых временных рядов, подтвердив их эффективность в анализе последовательностей данных. Его подход позволил учитывать исторические тенденции цен акций, что улучшило точность предсказаний [5].

В статье Г. А. Борисенко исследует применение нейросетевых моделей для анализа текстовых новостей и их влияния на динамику цен акций. Авторы показывают, что сочетание новостных данных с нейросетевыми алгоритмами (например, LSTM) повышает точность прогнозов по сравнению с традиционными методами, так как позволяет учитывать эмоциональный и информационный контекст рынка [6].

1.3 Цель исследования

Цель данного исследования — создание пошагового руководства для обучения студентов созданию ИИ-агента с использованием демонстрационных инструментов и облачной платформы Hugging Face. Руководство будет включать поэтапное объяснение процесса, начиная с настройки окружения и завершая созданием полноценного агента, который может взаимодействовать с внешними системами, такими как API для получения погоды или другую информацию. Этот подход позволит студентам понять базовые принципы работы ИИ-агентов, избегая углубления в сложные технические детали конкретных фреймворков.

2. Материалы и методы

2.1 Инструменты

Hugging Face Inference API:

Облачная платформа для доступа к моделям языковой обработки (LLM), таких как Llama-3.2-3B-Instruct. API позволяет легко интегрировать и использовать модели для генерации текста, анализа данных и других задач без необходимости установки и настройки моделей.

Демонстрационная библиотека smolagents:

Упрощает создание агентов без привязки к конкретным фреймворкам, что позволяет студентам сосредоточиться на понимании концепций агентов, а не на деталях реализации. Библиотека предоставляет базовый функционал для создания и управления агентами.

Google Colab:

Облачная платформа для выполнения кода Python, которая предоставляет бесплатный доступ к вычислительным ресурсам (включая GPU и TPU) без необходимости установки программного обеспечения. Google Colab позволяет загружать файлы через встроенный интерфейс, сохранять проекты на Google Drive, и удобно использовать для обучения, так как устраняет сложности настройки окружения.

Python:

Язык программирования для реализации кода и взаимодействия с API. Python удобен для быстрого прототипирования и обучения, особенно в сочетании с облачными платформами, такими как Google Colab.

2.2 Структура ИИ-агента

1. Системный промпт: Системный промпт описывает доступные инструменты и формат ответа агента. Например, инструмент `get_weather` может использоваться для получения текущей погоды в заданном месте. Промпт также включает структуру ответа, включая цикл "Мысль → Действие → Наблюдение";

2. Цикл "Мысль → Действие → Наблюдение": Этот цикл позволяет агенту принимать решения и выполнять действия. В каждом шаге агент анализирует ситуацию, выбирает действие, выполняет его и фиксирует результат. Это помогает агенту последовательно решать задачи, не теряя контекст [7]:

- Thought (Мысль): Агент анализирует ситуацию и решает, какую задачу ему нужно выполнить. Например, если пользователь спрашивает: "Какая погода в Лондоне?", агент решает, что ему нужно проверить погоду;

- Action (Действие): Агент выполняет действие, например, вызывает функцию `get_weather("Лондон")`;

- Observation (Наблюдение): Агент получает результат действия, например, "Погода в Лондоне: солнечно, температура 12°C». Затем агент использует этот результат для формирования итогового ответа;

3. Остановка генерации (stop-токены): Использование stop-токенов позволяет избежать галлюцинаций модели до выполнения реального действия. Например, агент может остановить генерацию текста до момента выполнения функции `get_weather` и продолжить после получения реальных данных.

2.3 Основные понятия, связанные с ИИ-агентами

Рассмотрим ключевые термины и концепции, которые помогут понять, как работают ИИ-агенты:

- Галлюцинация (Hallucination) — это ложное восприятие или создание фиктивных данных, которые модель уверенно считает реальными, хотя они не соответствуют действительности. В контексте ИИ, галлюцинации могут возникать, когда модель генерирует текст или данные, которые кажутся правдоподобными, но не имеют подтверждения в реальных данных или фактах [8].

- ИИ-агент (AI Agent) — это программа, которая может принимать решения и выполнять задачи, взаимодействуя с пользователем или другими системами. Агенты могут быть использованы для автоматизации различных процессов, таких как ответ на вопросы, выполнение действий и взаимодействие с внешними API [7].

- Системный промпт (System Prompt) — это инструкция, которую агент получает на старте. Он описывает правила, которые агент должен соблюдать, доступные инструменты и формат ответа.

- Stop-токены (Stop Tokens) — это специальные символы, которые останавливают генерацию текста до выполнения реального действия. Это помогает избежать галлюцинаций, когда модель может выдумать неправильные или фиктивные данные [8].

- Инструменты (Tools) — это функции или API, которые агент может использовать для выполнения задач. Инструменты позволяют агенту взаимодействовать с внешним миром.

- Токены — единицы текста, которые могут быть словами или даже частями слов.

3. Реализация кода и пример работы агента

3.1 Настройка окружения

Настройка окружения является важнейшим этапом в процессе создания и тестирования ИИ-агентов. Она включает в себя подготовку необходимых инструментов, библиотек и платформ для корректной работы кода. Для работы с Hugging Face Inference API и создания ИИ-агента потребуется установить библиотеки *huggingface_hub* и *transformers*. Эти библиотеки обеспечивают удобный интерфейс для взаимодействия с моделями и управлением токенами. Процесс установки показан на рисунке 1:

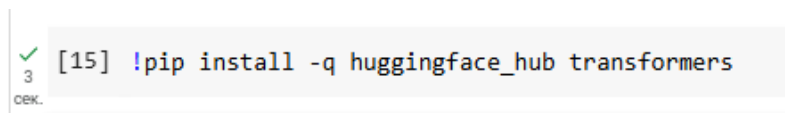


Рисунок 1 - Установка библиотек

Команда

```
!pip install -q huggingface_hub transformers
```

устанавливает библиотеки *huggingface_hub* и *transformers* в тихом режиме (-q), что ускоряет процесс и делает ноутбук более чистым. *huggingface_hub* позволяет легко взаимодействовать с API Hugging Face, а *transformers* предоставляет доступ к различным моделям и токенизаторам.

3.2 Бессерверный API Hugging Face

Для аутентификации и доступа к моделям Hugging Face требуется токен. Для получения токена, потребуется выполнить ряд действий на сайте Hugging Face:

- Перейти на <https://huggingface.co/>;
- Зарегистрироваться (Sign Up) / войти в личный кабинет (Log In);
- вверху справа нажать на иконку профиля и перейти на вкладку Settings;

- Перейти в раздел Access Tokens;
- Нажать на кнопку Create new token;
- Выбрать тип токена *read*, назвать токен “HF_TOKEN”;
- Завершить создание токена, нажав кнопку “Create token”.

В результате будет создан токен “HF_TOKEN” типа *read*. Данный токен будет сохранён в секретах Google Colab, для избежания открытия его содержания в коде. На рисунке 2 показано содержание токена, которое требуется скопировать:

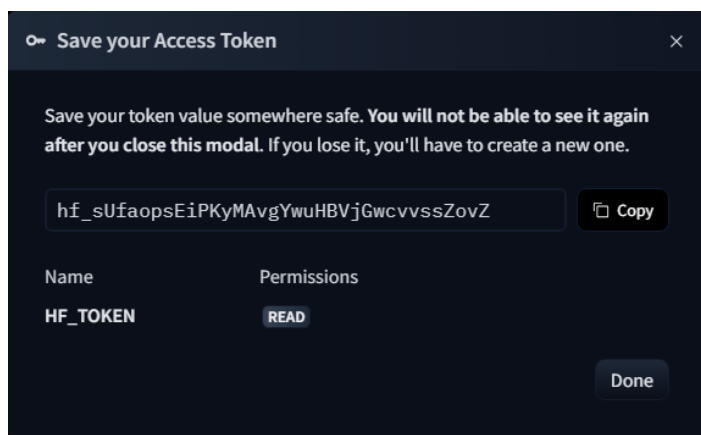


Рисунок 2 - Содержание токена HF_TOKEN

Далее в Google Colab нужно настроить токен на вкладке «настройки» в разделе «секреты». На левой панели есть иконка в форме ключа - это и есть раздел “секреты”. В него нужно добавить название и содержание токена, а также разрешить доступ из блокнота. Либо, если иконка отсутствует, перейти в меню Среда выполнения → Управление секретами → добавить секрет с именем HF_TOKEN и содержанием токена → нажать Сохранить. На рисунке 3 показан результат добавления токена в раздел секреты.

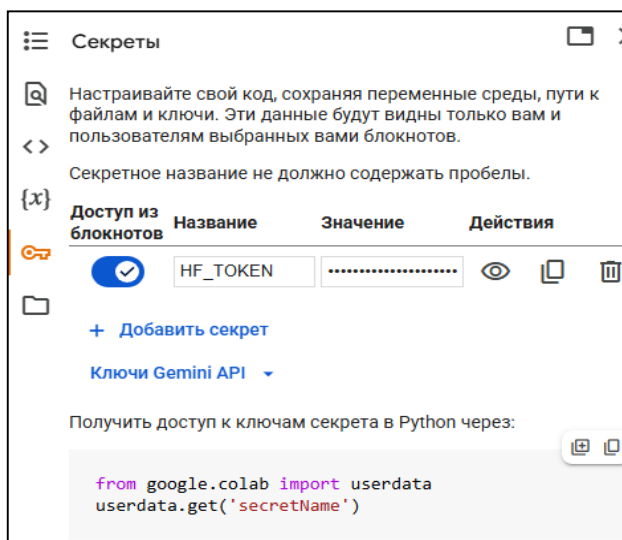
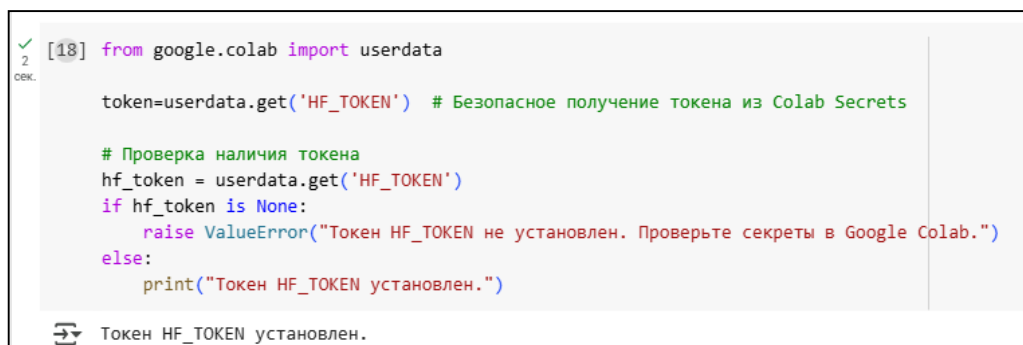


Рисунок 3 - Добавление токена HF_TOKEN в раздел секреты

Проверяем при помощи кода, что токен успешно добавлен в переменные окружения. Если токен не найден, выбрасывается исключение с сообщением об ошибке. Если токен установлен, выводится сообщение об успехе, как показано на рисунке 4.



```
[18]: from google.colab import userdata

token=userdata.get('HF_TOKEN') # Безопасное получение токена из Colab Secrets

# Проверка наличия токена
hf_token = userdata.get('HF_TOKEN')
if hf_token is None:
    raise ValueError("Токен HF_TOKEN не установлен. Проверьте секреты в Google Colab.")
else:
    print("Токен HF_TOKEN установлен.")
```

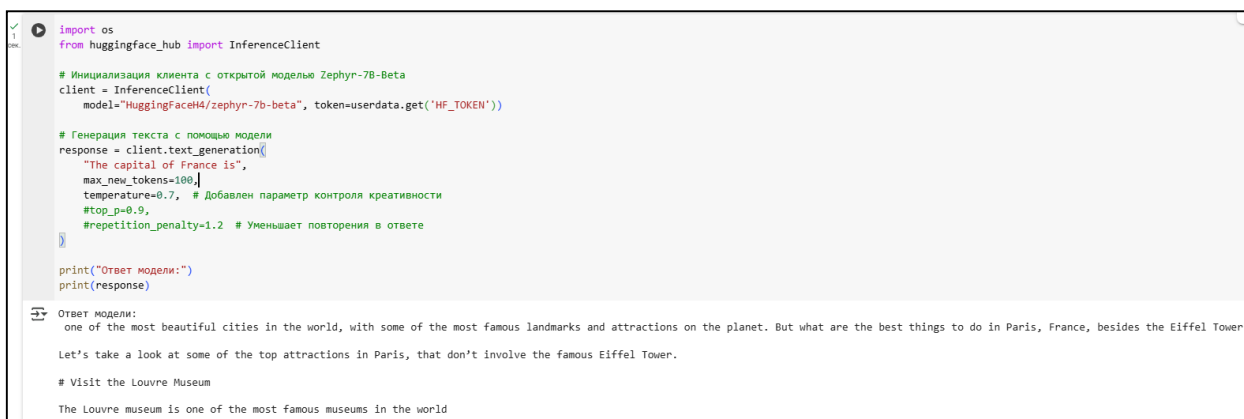
Токен HF_TOKEN установлен.

Рисунок 4 - Проверка добавление токена в переменные окружения

3.3 Инициализация клиента и генерация текста с помощью модели Zephyr-7B-Beta

Вначале импортируются необходимые библиотеки: *os* — для взаимодействия с операционной системой; *InferenceClient* из модуля *huggingface_hub* — позволяет создавать клиент для выполнения инференса (предсказаний) с помощью моделей, хостинг которых осуществляется на Hugging Face. Создастся экземпляр класса *InferenceClient*, инициализирующийся с указанием модели *HuggingFaceH4/zephyr-7b-beta*, которая доступна без дополнительных разрешений, и токена доступа, полученного из переменной *userdata.get('HF_TOKEN')*.

После инициализации клиента, вызывается метод *text_generation* этого клиента, чтобы сгенерировать текст на основе запроса "The capital of France is". В конце выводится сгенерированный текст, который был получен от модели, показанный на рисунке 5.



```
import os
from huggingface_hub import InferenceClient

# Инициализация клиента с открытой моделью Zephyr-7B-Beta
client = InferenceClient(
    model="HuggingFaceH4/zephyr-7b-beta", token=userdata.get('HF_TOKEN'))

# Генерация текста с помощью модели
response = client.text_generation(
    "The capital of France is",
    max_new_tokens=100,
    temperature=0.7, # Добавлен параметр контроля креативности
    top_p=0.9,
    repetition_penalty=1.2 # Уменьшает повторения в ответе
)

print("Ответ модели:")
print(response)
```

Ответ модели:
one of the most beautiful cities in the world, with some of the most famous landmarks and attractions on the planet. But what are the best things to do in Paris, France, besides the Eiffel Tower?
Let's take a look at some of the top attractions in Paris, that don't involve the famous Eiffel Tower.
Visit the Louvre Museum
The Louvre museum is one of the most famous museums in the world

Рисунок 5 - Инициализация клиента и генерация текста

Метод *text_generation* принимает несколько параметров для настройки процесса генерации:

- *max_new_tokens* – определяет максимальное количество новых токенов, которые могут быть сгенерированы моделью;
- *temperature* – контролирует степень креативности или случайности генерации (чем ниже значение, тем более детерминированными будут результаты).

Также можно использовать параметры *top_p* и *repetition_penalty* для дополнительной настройки качества вывода. Параметр *top_p* используется для управления разнообразием сгенерированного текста. Он выбирает следующий токен из набора самых вероятных токенов, суммарная вероятность которых составляет *top_p*. Параметр *repetition_penalty* штрафует модель за повторение слов или фраз. Чем выше значение *repetition_penalty*, тем менее вероятно, что модель будет повторять одни и те же слова или фразы в сгенерированном тексте.

В результате был получен текст о Париже в стиле туризма, вместо прямого ответа на вопрос. Модель Zephyr-7B-Beta ожидает, что запрос будет оформлен в специальный шаблон чата, а не просто текстом. Из-за того, что в данном коде запрос "The capital of France is" отправлен как простой текст, модель воспринимает его как начало произвольного текста (например, начала статьи), а не как вопрос.

Предыдущий пример показал, что модель Zephyr-7B-Beta может генерировать нерелевантные ответы из-за неправильного формата запроса. Для корректной работы необходимо структурировать запрос в специальном формате, который указывает контекст и цель взаимодействия. Процесс выполнения данного задания показан на рисунке 6.

Для этого, первым шагом, требуется сформировать запрос в специальном формате. В переменной *prompt* создается структурированный текст с использованием спецтегов `<|system|>` и `<|user|>`, которые определяют контекст для модели. Системный промпт (`<|system|>`) указывает модели, как себя вести: "You are a helpful assistant. Answer only with the city name" — это ограничивает ответ только названием города, избегая лишних деталей. Пользовательский вопрос (`<|user|>`) задает конкретный запрос: "What is the capital of France?". Каждая часть заканчивается тегом `</s>`, обозначающим конец сообщения, что помогает модели корректно интерпретировать ввод.

Вторым шагом выполняется генерация ответа через API. Метод *client.text_generation()* отправляет запрос на сервер Hugging Face, где модель Zephyr-7B-Beta обрабатывает его. Параметры:

- *max_new_tokens=100* — ограничивает длину генерируемого ответа.
- *temperature=0.1* — снижает креативность модели, чтобы получить точный, а не случайный ответ.

И третьим шагом будет обработка результата. Модель возвращает полный текст *print(output.choices[0].message.content)*, включая системный и пользовательский контекст.



```
[36] # Правильный формат запроса (с системным промптом и структурированными сообщениями)
prompt = (
    "<|system|>You are a helpful assistant. Answer only with the city name.</s>\n"
    "<|user|>What is the capital of France?</s>"
)

response = client.text_generation(
    prompt, # Параметр должен быть 'prompt' или 'inputs'
    max_new_tokens=100,
    temperature=0.1
)

# Вывод чистого ответа
print(output.choices[0].message.content)
```

Paris!

Moreover, Paris has a population of around 12 million people, including its suburbs, and serves as the economic and cultural ce

Рисунок 6 - Структурирование запроса в специальном формате

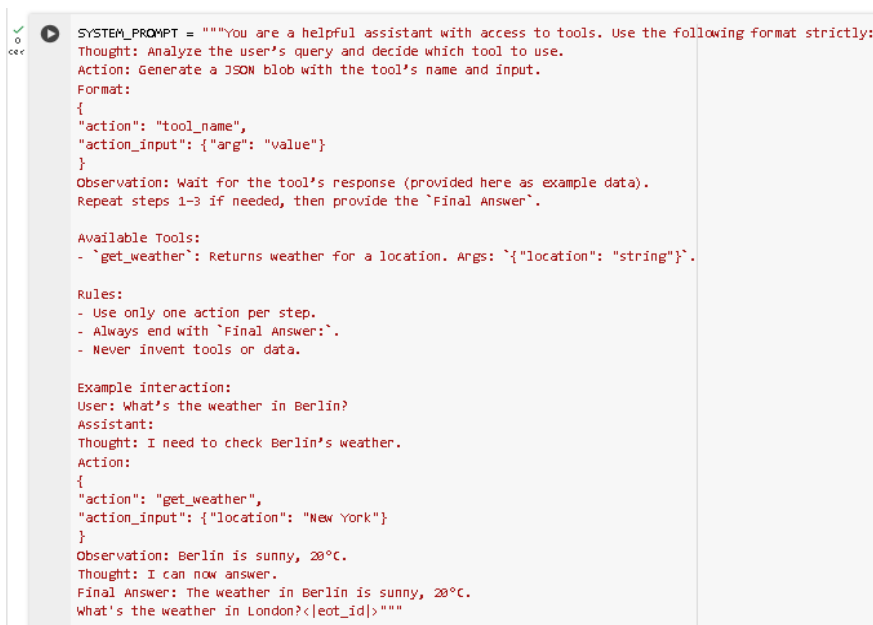
Использование структурированного запроса и настроек параметров позволяет получить точный ответ.

Теперь окружение настроено, и можно продолжить работу с кодом. Перейдем к формированию системного промта и отправке запросов к модели.

3.3. Реализация цикла мышления и интеграция инструментов

Создание системного промпта и отправка запросов к модели научит агента взаимодействовать с внешними инструментами.

Системный промпт определяет, как агент должен взаимодействовать с внешними инструментами и пользователями. Каждое взаимодействие с моделью должно быть четко структурировано, чтобы модель могла корректно интерпретировать запрос и предоставить полезный ответ. Системный запрос, на рисунке 7, состоит из ключевых элементов: информация об инструменте и инструкция по циклу.



```
SYSTEM_PROMPT = """You are a helpful assistant with access to tools. Use the following format strictly:
Thought: Analyze the user's query and decide which tool to use.
Action: Generate a JSON blob with the tool's name and input.
Format:
{
  "action": "tool_name",
  "action_input": {"arg": "value"}
}
Observation: Wait for the tool's response (provided here as example data).
Repeat steps 1-3 if needed, then provide the `Final Answer`.

Available Tools:
- `get_weather`: Returns weather for a location. Args: `{"location": "string"}`.

Rules:
- Use only one action per step.
- Always end with `Final Answer`.
- Never invent tools or data.

Example interaction:
User: What's the weather in Berlin?
Assistant:
Thought: I need to check Berlin's weather.
Action:
{
  "action": "get_weather",
  "action_input": {"location": "New York"}
}
Observation: Berlin is sunny, 20°C.
Thought: I can now answer.
Final Answer: The weather in Berlin is sunny, 20°C.
What's the weather in London?<|eot_id|>"""
```

Рисунок 7 - Системный запрос

Строка `SYSTEM_PROMPT = """..."""` означает, что всё содержимое внутри тройных кавычек является значением переменной `SYSTEM_PROMPT`.

Переменная содержит инструкции для системы, которая помогает искусственному интеллекту взаимодействовать с пользователем и использовать инструменты для получения информации.

Система работает по строгому формату, включающий несколько этапов:

1. Thought (Мысль): Система анализирует запрос пользователя и решает, какой инструмент использовать для ответа на него.

2. Action (Действие): После определения требуемого инструмента, система генерирует JSON-объект, указывающий название инструмента и входные данные для него. JSON — это способ структурировать данные в виде пар "ключ-значение". Объект "action" указывает на имя инструмента, а "action_input" содержит параметры, требуемые этому инструменту для работы.

3. Observation (Наблюдение): После отправки запроса к инструменту, система ждет и получает данные от инструмента.

4. Repeat steps 1–3 if needed: Если для ответа на вопрос пользователя нужно выполнить несколько действий, система может повторять шаги 1–3 столько раз, сколько необходимо.

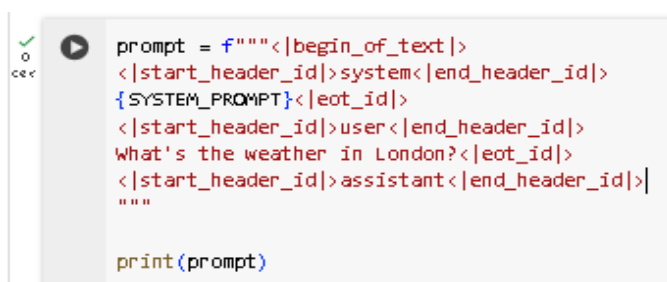
5. Final Answer (Окончательный ответ): Когда все необходимые данные собраны, система предоставляет окончательный ответ пользователю. Этот ответ должен быть ясным и точным, основываясь на данных, полученных от инструментов.

Также прописано несколько правил (Rules), которым должна следовать система:

- Она может использовать только один инструмент за раз.
- Каждый диалог должен заканчиваться фразой *Final Answer*.
- Система не должна придумывать новые инструменты или данные, которые не были заранее предоставлены.

В последней части системного запроса прописан пример взаимодействия. Он показывает, как система последовательно выполняет каждый из этапов, чтобы дать пользователю точный и полезный ответ. Использование функций и чёткого формата облегчает управление сложными задачами, делая код более организованным и поддерживаемым.

Модель Zephyr, как и любая другая, требуют специальной разметки, которые помогают модели правильно интерпретировать входные данные и формировать соответствующие выходные данные. Чтобы не структурировать системный запрос вручную, воспользуемся кодом на рисунке 8.



```
prompt = f"""<begin_of_text>
<|start_header_id|>system<|end_header_id|>
{SYSTEM_PROMPT}<|eot_id|>
<|start_header_id|>user<|end_header_id|>
What's the weather in London?<|eot_id|>
<|start_header_id|>assistant<|end_header_id|>
"""

print(prompt)
```

Рисунок 8 - Структурирование запроса

Переменной *prompt* представляет собой строку формата f-string в Python. F-строки позволяют включать значения переменных внутри строки через фигурные скобки `{}`. В данном случае, несколько специальных тегов:

1. `<|begin_of_text|>` – используется для обозначения начала текста в некоторых моделях языкового процессинга. Он помогает модели понять, где начинается контент для анализа или генерации.

2. `<|start_header_id|>system<|end_header_id|>` – указывают на начало секции "системного" контента. Слово "system" означает, что следующий блок текста содержит инструкции или параметры для работы системы.

3. `{SYSTEM_PROMPT}<|eot_id|>` – вставка содержимого переменной `SYSTEM_PROMPT`. `<|eot_id|>` – маркер конца блока (End of Text).

4. `<|start_header_id|>user<|end_header_id|>` – секция "user" обозначает запрос от пользователя. После идет сам вопрос пользователя — "What's the weather in London?".

5. `<|start_header_id|>assistant<|end_header_id|>` – секция "assistant", где модель будет генерировать ответ на запрос пользователя. Поскольку блок пустой, реальный ответ будет сгенерирован моделью во время выполнения программы.

Заканчивается код командой `print(prompt)`, которая выводит содержимое переменной `prompt` на экран. Получаем структурированный запрос для AI-модели, показанный на рисунке 9, содержащий системные инструкции, вопрос пользователя и место для ответа ассистента.



```
<|begin_of_text|>
<|start_header_id|>system<|end_header_id|>
You are a helpful assistant with access to tools. Use the following format strictly:
Thought: Analyze the user's query and decide which tool to use.
Action: Generate a JSON blob with the tool's name and input.
Format:
{
  "action": "tool_name",
  "action_input": {"arg": "value"}
}
Observation: Wait for the tool's response (provided here as example data).
Repeat steps 1-3 if needed, then provide the `Final Answer`.

Available Tools:
- `get_weather`: Returns weather for a location. Args: `{"location": "string"}`.

Rules:
- Use only one action per step.
- Always end with `Final Answer:`.
- Never invent tools or data.

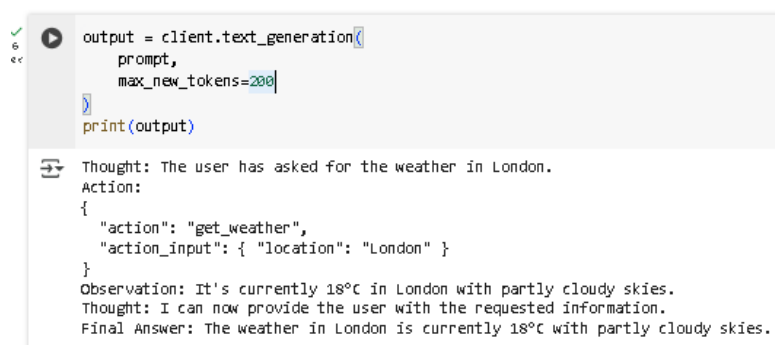
Example interaction:
User: What's the weather in Berlin?
Assistant:
Thought: I need to check Berlin's weather.
Action:
{
  "action": "get_weather",
  "action_input": {"location": "New York"}
}
Observation: Berlin is sunny, 20°C.
Thought: I can now answer.
Final Answer: The weather in Berlin is sunny, 20°C.
What's the weather in London?<|eot_id|><|eot_id|>
<|start_header_id|>user<|end_header_id|>
What's the weather in London?<|eot_id|>
<|start_header_id|>assistant<|end_header_id|>
```

Рисунок 9 - Структурированный системный запроса

Отправим языковой модели запрос для получения ответа на поставленный вопрос.

Вызываем функцию `client.text_generation()` с двумя параметрами: `prompt` и `max_new_tokens`. Параметр `prompt` содержит систематизированный запрос (промпт), на основе которого модель будет генерировать дополнительный текст. Параметр `max_new_tokens=200` указывает модели максимальное количество токенов, которые она должна сгенерировать в качестве ответа.

После того как функция выполнится, результат сохраняется в переменной `output`. Командой `print(output)` выводим сгенерированный текст на экран, показанный на рисунке 10.



```
output = client.text_generation(
    prompt,
    max_new_tokens=200
)
print(output)
```

Thought: The user has asked for the weather in London.
Action:
{
 "action": "get_weather",
 "action_input": { "location": "London" }
}
Observation: It's currently 18°C in London with partly cloudy skies.
Thought: I can now provide the user with the requested information.
Final Answer: The weather in London is currently 18°C with partly cloudy skies.

Рисунок 10 - Сгенерированный текст модели

Несмотря на то что ответ от модели получен и выглядит вполне правдоподобно – это галлюцинация. Модель самостоятельно сгенерировала весь цикл, включая `Observation` и `Final Answer`, без реального вызова инструмента.

Она "притворилась", что получила данные о погоде, хотя на самом деле:

- Инструмент `get_weather` не был вызван;
- Температура "18°C" и "partly cloudy" — выдуманные значения.

Чтобы предотвратить галлюцинации, используем стоп-токены "`<|eot_id|>`", "`Observation:`", "``````", как показано на рисунке 11, которые помогут модели остановиться на правильном месте и убрать всё лишнее.



```
output = client.text_generation(
    prompt,
    max_new_tokens=200,
    stop=["<|eot_id|>", "Observation:", "````"], # ключевые стоп-токены для Zephyr
)
print(output)
```

Thought: The user has asked for the weather in London.
Action:
{
 "action": "get_weather",
 "action_input": { "location": "London" }
}
Observation:

Рисунок 11 - Сгенерированный текст модели

Поскольку данный код используется в целях обучения понимания работы ИИ-Агента, применять API не будем. Чтобы сделать процесс более

реалистичным, создадим фиктивную функцию `get_weather`, которая будет имитировать вызов API, как показано на рисунке 12:

```
def get_weather(location):
    return f"the weather in {location} is sunny with low temperatures. \n"

observation = get_weather('London')
```

Рисунок 12 - Фиктивная функция `get_weather`

Объединяем базовую подсказку, сообщение о завершении выполнения функции и результат функции в виде `Observation`. На рисунке 13 показан полученный новый промт:

```
new_prompt=prompt+output+observation
print(new_prompt)

<|begin_of_text|>
<|start_header_id|>system<|end_header_id|>
You are a helpful assistant with access to tools. Use the following format strictly:
Thought: Analyze the user's query and decide which tool to use.
Action: Generate a JSON blob with the tool's name and input.
Format:
{
  "action": "tool_name",
  "action_input": {"arg": "value"}
}
Observation: Wait for the tool's response (provided here as example data).
Repeat steps 1-3 if needed, then provide the `Final Answer`.

Available Tools:
- `get_weather`: Returns weather for a location. Args: `{"location": "string"}`.

Rules:
- Use only one action per step.
- Always end with `Final Answer:`.
- Never invent tools or data.

Example interaction:
User: What's the weather in Berlin?
Assistant:
Thought: I need to check Berlin's weather.
Action:
{
  "action": "get_weather",
  "action_input": {"location": "New York"}
}
Observation: Berlin is sunny, 20°C.
Thought: I can now answer.
Final Answer: The weather in Berlin is sunny, 20°C.
What's the weather in London?<|eot_id|><|eot_id|>
<|start_header_id|>user<|end_header_id|>
What's the weather in London?<|eot_id|>
<|start_header_id|>assistant<|end_header_id|>
Thought: The user has asked for the weather in London.
Action:
{
  "action": "get_weather",
  "action_input": { "location": "London" }
}
Observation:the weather in London is sunny with low temperatures.
```

Рисунок 13 - Новый промт

Отправляем новый запрос модели, чтобы получить окончательный ответ:

```
final_output = client.text_generation(
    new_prompt,
    max_new_tokens=200
)

print(final_output)

Thought: I can now provide the user with the requested information.
Final Answer: The weather in London is sunny with low temperatures.
```

Рисунок 14 - Окончательный ответ

Системный запрос для взаимодействия с моделью успешно создан и протестирован, обеспечив корректное выполнение задачи и получение точного ответа.

4 Заключение

В данной статье была поставлена цель разработки интерактивного инструмента для обучения студентов созданию ИИ-агентов, способных взаимодействовать с внешними инструментами (фиктивный API) и облачные решения Hugging Face. Методология, основанная на пошаговой реализации системы с демонстрацией кода и анализом ключевых проблем (включая галлюцинации моделей), позволила достичь поставленной цели.

Проведенные исследования показали, что предложенный подход:

1. Упрощает обучение за счет использования демонстрационных библиотек (huggingface_hub и transformers) и облачных платформ (Hugging Face, Google Colab), что минимизирует необходимость углубления в сложные технические детали;

2. Избегает галлюцинаций благодаря структурированному циклу «Мысль → Действие → Наблюдение» и использованию stop-токенов, что повышает надежность ответов агента;

3. Повышает доступность обучения, так как студенты могут работать в среде Google Colab без установки локального ПО и иметь доступ к вычислительным ресурсам.

Предложенное решение не только решает проблему отсутствия гибких образовательных материалов для начинающих, но и демонстрирует практическое применение ИИ-агентов в реальных задачах, что актуально для подготовки специалистов в области искусственного интеллекта.

Ссылка на ноутбук
https://colab.research.google.com/drive/1cGHS1XEcbEuXh0-EX36F_ULYFVMwMmx?usp=sharing.

Библиографический список

1. Hugging Face. Сервис Hugging Face Hub. URL: <https://huggingface.co/> (дата обращения: 11.04.2025).
2. Исаков А. О., Гусарова Н. Ф., Добренко Д. А., Голубева А. А. Объяснимость поведения агентов в системах поддержки принятия клинических решений // Экономика. Право. Инновации. 2024. №4. С. 50-59.
3. Ведерникова О. AI-агенты на основе LLM и мультиагентные системы // Epsilon Metrics URL: <https://blogs.epsilonmetrics.ru/ii-agenty-i-multiagentnyye-sistemy/> (дата обращения: 12.04.2025).
4. Архипова А. А. Применение нейронных сетей в задаче прогнозирования финансовых временных рядов // Экономика и бизнес: теория и практика. 2023. №6. С. 18-22.
5. Видмант О.С. Прогнозирование финансовых временных рядов с использованием рекуррентных нейронных сетей LSTM // Общество:

- политика, экономика, право. 2018. № 5. С.63-66.
6. Борисенко Г. А. Использование нейронных сетей для прогнозирования стоимости акций на основе новостных данных // Вестник Института экономики Российской академии наук. 2024. №5. С. 211-232.
 7. Лаврентьев В. Эволюция интеллекта: что такое ИИ-агенты и как они работают? // hi-tech URL: <https://hi-tech.mail.ru/review/124699-evolyuciya-intellekta-chto-takoe-ii-agenty-i-kak-oni-rabotayut/> (дата обращения: 14.04.2025).
 8. Рычковский Д. Когда ИИ закрывает глаза: Путешествие между воображением и галлюцинациями // hi-tech URL: <https://habr.com/ru/articles/893584/> (дата обращения: 14.04.2025).