F1_expert - QA модель

Александр Кудрявцев

Июнь 2025

Abstract

F1_expert — это проект, целью которого является создание экспертной системы для ответа на вопросы о гонках в классе Формула-1 на русском языке. https://github.com/alexander-1976/F1_expert.

1 Введение

Проект F1_expert представляет собой экспертную систему для ответа на вопросы о Формуле 1 на русском языке.

Проект F1_expert направлен на создание системы, способной отвечать на вопросы о Формуле 1, используя данные из открытых источников, таких как Википедия. Основной задачей является разработка модели, которая может извлекать точные ответы из текстового контекста, а также предоставлять пользователю возможность улучшать модель через обратную связь. В условиях ограниченных вычислительных ресурсов (отсутствие устойчивого доступа к GPU) проект был реализован с использованием облегченных моделей, оптимизированных для работы на CPU.

В статье подробно описаны этапы сборки датасета, метрики качества (SQuAD), механизм формирования ответов (сходный с Retrieval-Augmented Generation, RAG), а также плюсы, минусы и возможные улучшения проекта.

1.1 Команда

Подготовил проект и составил отчет: **Александр Кудрявцев**.

2 Related Work

В данном разделе рассмотрены альтернативные подходы к реализации QA-проекта F1_expert.

2.1 Использование генеративных моделей (вместо извлечения ответов)

Вместо извлечения ответов из контекста (как в текущем проекте с использованием модели rubert-tiny2), можно использовать генеративные модели, такие

как T5, BART или их производные, которые генерируют ответы, не ограничиваясь подстроками контекста. Этот подход соответствует классическому методу Retrieval-Augmented Generation (RAG).

Преимущества:

- **Гибкость:** Генеративные модели могут переформулировать ответы, синтезировать информацию из нескольких контекстов и отвечать на вопросы, ответы на которые не содержатся дословно в тексте.
- **Естественность:** Ответы звучат более естественно, так как модель может генерировать текст, а не только извлекать подстроки.
- **Широкое применение:** Подходит для сложных вопросов, требующих анализа и обобщения.

Недостатки:

- **Высокие требования к ресурсам:** Генеративные модели, такие как `t5-large` или `bart-large`, требуют значительных вычислительных ресурсов, что делает их труднопримени мыми на CPU.
- **Риск галлюцинаций:** Модель может генерировать вымышленные или некорректные ответы, особенно если контекст недостаточно информативен.
- **Сложность обучения:** Требуется больше данных и более сложная настройка гиперпараметров для достижения хорошего качества.

Ссылки:

- Оригинальная статья о RAG: Lewis, P., Perez, E., Piktus, A., et al. (2020). Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. *arXiv preprint arXiv:2005.11401.* `https://arxiv.org/abs/2005.11401`

2.2 Использование больших языковых моделей (LLM) без поиска контекста

Описание: Вместо поиска контекста с помощью FAISS и извлечения ответов можно использовать большие языковые модели (LLM), такие как `Grok` от xAI, `LLaMA` или их производные, которые содержат обширные знания, включая информацию о Формуле 1, и могут отвечать на вопросы без явного поиска контекста.

Преимущества:

- **Простота реализации:** Не требуется этап поиска контекста, так как модель уже содержит знания, полученные во время предобучения.
- **Высокое качество:** Большие LLM, такие как `Grok`, могут давать точные и естественные ответы, особенно на общие вопросы о Формуле 1.

- **Гибкость:** Модель может отвечать на вопросы, не ограничиваясь конкретным корпусом данных.

Недостатки:

- **Высокие требования к ресурсам:** Большие LLM требуют значительных вычислительных ресурсов (обычно GPU) и большого объема памяти, что делает их труднопримениимыми на CPU.
- **Зависимость от предобучения:** Модель может не знать актуальной информации (например, результатов гонок после декабря 2023 года) и не может быть легко обновлена без дообучения.
- **Риск галлюцинаций:** Модель может генерировать вымышленные или некорректные ответы, особенно на специфические вопросы, не покрытые её предобучением.

Ссылки:

- Оригинальная статья о LLaMA: Touvron, H., Lavril, T., Izacard, G., et al. (2023). LLaMA: Open and Efficient Foundation Language Models. *arXiv preprint arXiv:2302.13971*. `https://arxiv.org/abs/2302.13971`

2.3 Использование готовых QA систем (например, Haystack)

Описание: Вместо реализации QA системы с нуля можно использовать готовые фреймворки, такие как Haystack, которые предоставляют инструменты для поиска контекста (retrieval) и извлечения/генерации ответов (reader/generator). Haystack поддерживает интеграцию с различными моделями и индексами, включая FAISS.

Преимущества:

- **Простота реализации:** Haystack предоставляет готовые компоненты для поиска контекста и извлечения/генерации ответов, что сокращает время разработки.
- **Гибкость:** Поддерживает как извлечение ответов, так и генерацию, а также интеграцию с различными моделями (например, BERT, T5, BART).
- **Оптимизация:** Haystack оптимизирован для работы с большими корпусами данных и поддерживает GPU для ускорения.

Недостатки:

- **Сложность настройки:** Требуется настройка компонентов (retriever, reader, pipeline), что может быть сложным для новичков.
- **Зависимость от ресурсов:** Для достижения высокого качества может потребоваться GPU, особенно при использовании генеративных моделей.

- **Ограниченная кастомизация:** Некоторые специфические требования проекта могут быть трудными для реализации в рамках готового фреймворка.

Ссылки:

- Документация Haystack: `https://haystack.deepset.ai/`

2.4 Использование Knowledge Graph QA

Описание: Вместо работы с текстовыми данными можно построить граф знаний (Knowledge Graph) о Формуле 1, где сущности (гонщики, команды, трассы) и их отношения (например, "гонщик выступает за команду") представлены в структурированном виде. QA система может использовать этот граф для ответа на вопросы, преобразуя их в запросы к графу (например, SPARQL).

Преимущества:

- **Точность:** Ответы основаны на структурированных данных, что снижает риск ошибок и галлюцинаций.
- **Эффективность:** Поиск в графе знаний быстрее, чем в текстовом корпусе, особенно для фактологических вопросов.
- **Объяснимость:** Ответы можно объяснить, показав путь в графе (например, "Льюис Хэмилтон $\rightarrow$ выступает за $\rightarrow$ Mercedes").

Недостатки:

- **Сложность построения графа:** Требуется извлечение сущностей и отношений из текстов, что может быть трудоемким процессом (например, с использованием NER и RE).
- **Ограниченная применимость:** Подходит только для фактологических вопросов, а не для вопросов, требующих анализа или переформулировки.
- **Зависимость от данных:** Качество графа зависит от полноты и актуальности исходных данных.

Ссылки:

- Описание Knowledge Graph QA: Huang, X., Zhang, J., Li, D., Li, P. (2021). Knowledge Graph Question Answering: A Survey. *arXiv preprint arXiv:2106.13407*. `https://arxiv.org/abs/2106.13407`

2.5 Использование API внешних QA систем

Описание: Вместо разработки собственной QA системы можно использовать API внешних систем, таких как Google Search API, Wolfram Alpha или специализированные QA сервисы, которые могут предоставлять ответы на вопросы о Формуле 1.

Преимущества:

- **Простота реализации:** Не требуется обучение модели или обработка данных, достаточно интегрировать API.
- **Актуальность:** Внешние сервисы могут предоставлять актуальную информацию, включая последние результаты гонок.
- **Высокое качество:** Профессиональные сервисы часто дают точные и естественные ответы.

Недостатки:

- **Зависимость от внешних сервисов:** Требуется стабильное интернет-соединение и оплата API (если сервис платный).
- **Ограниченная кастомизация:** Невозможно настроить модель под специфические требования проекта.
- **Конфиденциальность:** Передача данных внешним сервисам может быть проблемой с точки зрения конфиденциальности.

Ссылки:

- Google Search API: `https://developers.google.com/custom-search/v1/overview`
- Wolfram Alpha API: `https://products.wolframalpha.com/api/`

3 Описание Модели. Использованные модели и подходы

- Использована модель `cointegrated/rubert-tiny2` — облегченная версия BERT, разработанная для русского языка. Выбрана из-за низких требований к вычислительным ресурсам и возможности работы на CPU.
- Реализована токенизация данных с учетом длинных контекстов (использован параметр `max_length=384` и `doc_stride=128`), что позволяет обрабатывать большие тексты, разбивая их на перекрывающиеся фрагменты.
- Обучение проведено с использованием библиотеки `transformers` и `Trainer` с настройками для минимизации потребления ресурсов (малый размер батча, накопление градиентов, ранняя остановка).
- Добавлена оценка качества модели с использованием метрик SQuAD (`exact_match` и `f1`).

3.1 Генерация вопросов и ответов

- генерации пар вопрос-ответ на основе корпуса текстов использована модель

  `cointegrated/rut5-base-multitask` — облегченная версия модели T5, оптимизированная для работы на CPU.

3.2 Реализация поиска контекста

- Для поиска релевантного контекста использована модель

  `sentence-transformers/distiluse-base-multilingual-cased-v2` — облегченная модель для создания эмбеддингов, поддерживающая многоязычные тексты, включая русский язык.

- Реализован быстрый поиск с помощью FAISS (`IndexIVFFlat`), оптимизированный для работы на CPU.

3.3 Подходы

- **Сбор данных**: Использован рекурсивный обход категорий Википедии с многопоточной обработкой для повышения скорости выгрузки.
- **Обработка данных**: Реализована очистка текста от вики-разметки с помощью `mwparserfromhell`, а также фильтрация дубликатов и некорректных данных.
- **Обучение модели**: Использован подход fine-tuning с использованием `transformers.Trainer`, с настройками для минимизации потребления памяти (малый размер батча, накопление градиентов, ранняя остановка).
- **Поиск контекста**: Применен метод поиска ближайших соседей с использованием FAISS для быстрого поиска релевантного контекста.
- **Дообучение**: Реализован инкрементальный подход к дообучению, где модель обновляется только на примерах с низкой оценкой пользователя.

4 Датасет

Проект состоит из нескольких этапов, каждый из которых реализован в виде отдельного модуля:

1. Сбор данных из Википедии.
2. Формирование корпуса текстов.
3. Генерация датасета пар вопрос-ответ.
4. Обучение QA модели.

5. Реализация поиска контекста.
6. Пользовательский интерфейс.
7. Дообучение модели.

4.1 Сбор данных из Википедии

- **Источник**: Данные собирались из Википедии на русском языке с использованием библиотеки `wikipedia-api`.
- **Категории**: Были выбраны 21 категория, охватывающая различные аспекты Формулы 1, такие как Гран-при, сезоны, команды, гонщики, трассы, аварии, история и т.д. Примеры категорий: `Гран-при Формулы-1`, `Гонщики Формулы-1`, `Автодромы Формулы-1`.
- **Рекурсивный обход**: Реализован рекурсивный обход подкатегорий с глубиной до 2 уровней, чтобы охватить как основные статьи, так и связанные материалы. Например, категория `Гран-при Формулы-1` может включать подкатегорию `Гран-при Монако`, которая, в свою очередь, содержит статьи о конкретных гонках.
- **Обработка ошибок**: Для каждой статьи реализован механизм повторных попыток (до 5 попыток с экспоненциальной задержкой) в случае ошибок (например, сетевых сбоев). Неудачные попытки логировались в файл `failed_articles.json` с указанием причины (например, "статья не найдена" или "максимальное количество попыток исчерпано").
- **Сохранение**: Каждая статья сохранялась в отдельный текстовый файл в папке `data/raw` с названием, очищенным от недопустимых символов (например, `Гран-при Монако.txt`).

4.2 Формирование корпуса текстов

- **Источник**: Сырые данные из папки `data/raw`, содержащие текстовые файлы статей.
- **Очистка текста**: Для каждой статьи использовалась библиотека `mwparserfromhell` для удаления вики-разметки (шаблонов, ссылок, таблиц и т.д.), оставляя только читаемый текст. Если очистка с помощью `mwparserfromhell` не удавалась (например, из-за сложной разметки), сохранялся исходный текст статьи.
- **Метаданные**: Для каждого файла собирались метаданные (дата создания, дата модификации, длина текста), что позволяло отслеживать изменения и анализировать объем данных.
- **Сохранение**: Корпус сохранялся в трех форматах:

- **CSV** (`f1_corpus.csv`): Для удобства анализа и обработки с помощью pandas. Каждая строка содержала поля `title`, `text`, `length`, `file`, `created`, `modified`.
- **JSONL** (`f1_corpus.jsonl`): Для использования в задачах машинного обучения, где каждая строка представляла JSON-объект с теми же полями.
- **TXT** (`f1_corpus.txt`): Для удобства чтения человеком, где каждая статья отделена заголовком с названием (например, `### Гран-при Монако`).

- **Результат**: Полученный корпус представлял собой структурированный набор текстов, готовый для дальнейшей обработки, например, для генерации датасета или анализа.

4.3 Генерация датасета пар вопрос-ответ

- **Источник**: Для генерации датасета использовался сформированный корпус текстов (`f1_corpus.jsonl`), содержащий статьи о Формуле 1.
- **Модель генерации**: Использована модель `cointegrated/rut5-base-multitask` — облегченная версия T5, оптимизированная для работы на CPU. Эта модель была выбрана из-за её способности выполнять задачи генерации текста (вопросов и ответов) с минимальными вычислительными ресурсами. Модель поддерживает мультитаскинг, что позволяет использовать её как для генерации вопросов, так и для извлечения ответов с помощью специальных промптов.
- **Процесс генерации**:
  1. **Очистка текста**: Каждый текст из корпуса очищался от лишних символов, HTML-тегов и невидимых символов с помощью регулярных выражений. Это обеспечивало чистоту входных данных для модели. Например, текст "Льюис Хэмилтон — британский автогонщик, семикратный чемпион мира" очищался от возможных артефактов вики-разметки, таких как `[[Льюис Хэмилтон]]`.
  2. **Ограничение длины**: Для каждого текста использовался только фрагмент длиной до 1024 символов, чтобы уложиться в ограничения модели и снизить вычислительную нагрузку.
  3. **Генерация вопросов**: Для каждого текста модель генерировала вопрос с помощью промпта `ask | {context}`. Например, для текста "Льюис Хэмилтон — британский автогонщик, семикратный чемпион мира" модель могла сгенерировать вопрос "Кто такой Льюис Хэмилтон?" или "Сколько раз Льюис Хэмилтон становился чемпионом мира?". Промпт `ask` указывал модели, что нужно сгенерировать вопрос, основанный на контексте.

4. **Генерация ответов**: Для сгенерированного вопроса модель извлекала ответ из контекста с помощью промпта `comprehend | {context}. Вопрос: {question}?`. Например, для вопроса "Кто такой Льюис Хэмилтон?" модель могла извлечь ответ "британский автогонщик", а для вопроса "Сколько раз Льюис Хэмилтон становился чемпионом мира?" — "семикратный".
5. **Фильтрация**: Пары вопрос-ответ фильтровались по строгим критериям, чтобы обеспечить качество датасета:
   - Вопрос и ответ не должны быть пустыми.
   - Вопрос не должен дословно содержаться в контексте (чтобы избежать тривиальных вопросов, таких как "Что такое Формула-1?" из текста, где уже есть это определение).
   - Ответ должен дословно содержаться в контексте (чтобы гарантировать корректность и возможность использования в QA модели, которая извлекает ответы из текста).
   - Контекст должен быть достаточно длинным (не менее 50 слов), чтобы исключить короткие и неинформативные тексты, которые могут привести к генерации бессмысленных вопросов.
6. **Сохранение**: Отфильтрованные пары вопрос-ответ сохранялись в формате JSONL (`f1_qa_dataset.jsonl`), где каждая строка представляла JSON-объект с полями `title`, `context`, `question`, `answer`. Например:

```
{"title": "Льюис Хэмилтон",
"context": "Льюис Хэмилтон - британский автогонщик
           , семикратный чемпион мира.",
"question": "Кто такой Льюис Хэмилтон?",
"answer": "британский автогонщик"}
```

- **Ограничения**: Для тестирования и оптимизации ресурсов на этапе генерации датасета было обработано только 50 записей из корпуса для оценки необходимых вычислительных ресурсов. Далее был обработан весь корпус статей (текстов).
- **Результат**: Полученный датасет представляет собой набор из 2440 пар вопрос-ответ, где каждый вопрос основан на реальном контексте из статей о Формуле 1, а ответы извлечены из этого контекста, что делает его подходящим для обучения QA модели. Датасет содержит разнообразные вопросы, охватывающие факты, биографии, события и другие аспекты Формулы 1, что позволяет модели обучаться на реальных примерах.

5 Метрики качества

Для оценки качества QA модели использовались метрики, принятые в задаче вопросно-ответной системы, а именно метрики из бенчмарка SQuAD (Stanford Question Answering Dataset). Эти метрики широко применяются в задачах извлечения ответов из текста и позволяют объективно оценить производительность модели.

5.1 Метрики SQuAD

- **Exact Match (EM)**: Эта метрика измеряет долю предсказанных ответов, которые точно совпадают с истинными ответами (с учетом регистра и пробелов). Например, если истинный ответ — "Льюис Хэмилтон", а предсказанный — "Льюис Хэмилтон", то EM = 1. Если предсказанный ответ — "Хэмилтон", то EM = 0. Эта метрика строгая и требует полного совпадения, что делает её полезной для оценки точности модели.

- **F1 Score**: Эта метрика измеряет гармоническое среднее между точностью (precision) и полнотой (recall) на уровне токенов. Она более мягкая, чем EM, так как учитывает частичные совпадения. Например, если истинный ответ — "Льюис Хэмилтон", а предсказанный — "Хэмилтон", то F1 будет больше 0, так как часть ответа совпадает. F1 Score вычисляется следующим образом:

  - Точность (precision) = доля правильных токенов в предсказанном ответе относительно всех токенов в предсказанном ответе.
  - Полнота (recall) = доля правильных токенов в предсказанном ответе относительно всех токенов в истинном ответе.
  - $\text{F1} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$.

- **Использование в проекте**: Метрики SQuAD были реализованы с помощью библиотеки `evaluate`, которая предоставляет готовую реализацию `squad`. Эти метрики использовались для оценки качества модели на тренировочном и тестовом наборах данных, а также для выбора лучшей модели во время обучения (параметр `metric_for_best_model="f1"` в `TrainingArguments`).

5.2 Преимущества метрик SQuAD

- **Объективность**: Метрики SQuAD являются стандартом в задачах QA и позволяют сравнивать производительность модели с другими моделями, протестированными на SQuAD.

- **Учет частичных совпадений**: F1 Score позволяет оценить качество модели даже в случаях, когда ответ не полностью совпадает с истинным, что важно для реальных приложений.

- **Простота интерпретации**: EM и F1 легко интерпретировать, что делает их удобными для анализа и отладки.

5.3 Ограничения метрик SQuAD

- **Строгость EM**: Метрика EM может быть слишком строгой, особенно для длинных ответов, где небольшие различия (например, в пунктуации) приводят к нулевому значению.
- **Зависимость от контекста**: Метрики SQuAD предполагают, что истинный ответ всегда содержится в контексте, что может не соответствовать реальным сценариям, где ответ может быть сформулирован иначе.
- **Отсутствие оценки семантической точности**: Метрики SQuAD не учитывают семантическую близость ответов, а только точное совпадение токенов. Например, если истинный ответ — "Льюис Хэмилтон", а предсказанный — "Хэмилтон", F1 будет ненулевым, но если предсказанный ответ — "британский гонщик", метрики не учтут, что это тоже может быть семантически правильным.

6 Механизм формирования ответов (RAG-подход)

Модель в проекте `F1_expert` формирует ответы на вопросы пользователей, извлекая их из контекста, который определяется по подходящим эмбеддингам. Этот подход схож с Retrieval-Augmented Generation (RAG), хотя в данном проекте используется упрощенная версия, адаптированная для работы на CPU.

6.1 Описание механизма

- **Шаг 1: Поиск контекста (Retrieval)**
  - Для каждого вопроса пользователя модель должна найти релевантный контекст, из которого будет извлечен ответ. Для этого используется подход, схожий с retrieval-компонентой RAG:
    - Все контексты из тренировочного набора данных (тексты статей о Формуле 1) преобразуются в эмбеддинги с помощью модели `sentence-transformers/distiluse-base-multilingual-cased-v2`. Эта модель создает компактные векторные представления текстов, которые учитывают семантическую близость.
    - Эмбеддинги контекстов индексируются с помощью FAISS (`IndexIVFFlat`), что позволяет быстро находить ближайшие контексты по косинусному расстоянию или L2-метрике.
    - Для заданного вопроса пользователя создается эмбеддинг вопроса с помощью той же модели `distiluse-base-multilingual-cased-v2`.

* FAISS выполняет поиск ближайших контекстов (по умолчанию 10 ближайших), используя индекс. Это позволяет быстро определить, какие тексты наиболее релевантны вопросу.
* Дополнительно применяется фильтрация по ключевым словам (`filter_context_by_keywords`), чтобы убедиться, что контекст содержит слова, связанные с вопросом. Например, если вопрос — "Кто такой Льюис Хэмилтон?", контекст должен содержать слова "Льюис" или "Хэмилтон".

– **Результат**: На этом шаге определяется релевантный контекст, который будет использован для извлечения ответа. Если подходящий контекст не найден, возвращается сообщение "Контекст не найден".

- **Шаг 2: Извлечение ответа (Generation)**

  – После определения релевантного контекста модель `cointegrated/rubert-tiny2` используется для извлечения ответа из этого контекста. Это не генерация текста в классическом смысле (как в RAG, где используется генеративная модель), а извлечение подстроки из контекста, которая наиболее вероятно является ответом.

    * Вопрос и контекст токенизируются с помощью `BertTokenizerFast` и передаются в модель `rubert-tiny2`.
    * Модель предсказывает позиции начала и конца ответа в токенизированном контексте (используя `start_logits` и `end_logits`).
    * Предсказанные позиции преобразуются обратно в текст с помощью декодирования токенов, что дает окончательный ответ.

  – **Результат**: Пользователь получает ответ, который является подстрокой релевантного контекста. Например, для вопроса "Кто такой Льюис Хэмилтон?" и контекста "Льюис Хэмилтон — британский автогонщик, семикратный чемпион мира" модель извлечет ответ "британский автогонщик".

6.2 Сходство с RAG

- **Что такое RAG?** Retrieval-Augmented Generation (RAG) — это подход, который сочетает поиск релевантного контекста (retrieval) с генерацией ответа (generation). В классическом RAG:

  – Retrieval-компонента (обычно Dense Passage Retrieval, DPR) находит релевантные документы или фрагменты текста с помощью эмбеддингов и индекса (например, FAISS).

  – Generation-компонента (обычно генеративная модель, такая как BART или T5) генерирует ответ, используя найденный контекст.

- **Сходство с RAG в проекте**: В проекте `F1_expert` используется подход, схожий с RAG, но с некоторыми отличиями:

  - **Retrieval**: Как и в RAG, используется поиск релевантного контекста с помощью эмбеддингов (`distiluse-base-multilingual-cased-v2`) и FAISS. Это позволяет модели работать с большими объемами данных, не загружая весь корпус в память.
  - **Generation**: В отличие от классического RAG, где ответ генерируется, в данном проекте ответ извлекается из контекста с помощью модели `rubert-tiny2`. Это упрощает задачу и снижает вычислительные требования, что важно для работы на CPU.

- **Отличия от классического RAG**:
  - В классическом RAG генеративная модель может создавать ответы, которые не дословно содержатся в контексте, а являются переформулировкой или синтезом информации. В данном проекте ответы всегда являются подстроками контекста, что ограничивает гибкость, но упрощает реализацию и снижает требования к ресурсам.
  - Классический RAG обычно требует GPU для работы с большими генеративными моделями, тогда как в данном проекте используются облегченные модели, оптимизированные для CPU.

6.3 Преимущества подхода

- **Эффективность**: Использование FAISS для поиска контекста позволяет быстро находить релевантные тексты даже на CPU, что делает проект масштабируемым для больших корпусов.
- **Точность**: Извлечение ответов из контекста (а не генерация) гарантирует, что ответы основаны на фактической информации, содержащейся в данных, что снижает риск "галлюцинаций" (вымышленных ответов).
- **Простота**: Упрощенный подход (извлечение вместо генерации) снижает вычислительные требования и упрощает реализацию, что важно в условиях ограниченных ресурсов.

6.4 Ограничения подхода

- **Ограниченная гибкость**: Поскольку ответы извлекаются из контекста, модель не может переформулировать ответы или синтезировать информацию из нескольких источников. Например, если вопрос требует ответа, который не содержится дословно в одном контексте, модель не сможет ответить корректно.
- **Зависимость от качества поиска**: Точность ответов зависит от качества поиска контекста. Если FAISS не находит релевантный контекст (например, из-за недостаточной настройки параметров или низкого качества эмбеддингов), модель не сможет дать правильный ответ.

- **Отсутствие генеративных возможностей**: В отличие от классического RAG, модель не может генерировать ответы, которые не содержатся в контексте, что ограничивает её применимость для более сложных вопросов.

7 Результаты

7.1 Плюсы проекта

- **Оптимизация для CPU**: Проект полностью реализован для работы на CPU, что делает его доступным для пользователей без GPU. Использованы облегченные модели

  (`rut5-base-multitask`, `rubert-tiny2`,

  `distiluse-base-multilingual-cased-v2`),

  которые обеспечивают приемлемую производительность при ограниченных ресурсах.

- **Автоматизация**: Все этапы (сбор данных, формирование корпуса, генерация датасета, обучение модели) автоматизированы и могут быть запущены одним скриптом (`main.py`).

- **Дообучение**: Реализован механизм дообучения модели на основе обратной связи пользователя, что позволяет улучшать качество ответов без необходимости полной перестройки модели.

- **Быстрый поиск контекста**: Использование FAISS для поиска релевантного контекста обеспечивает высокую скорость инференса даже на CPU.

- **Открытость**: Проект подготовлен для размещения на GitHub с подробной документацией, что делает его доступным для сообщества и упрощает совместную разработку.

7.2 Минусы проекта

- **Ограничения облегченных моделей**: Использование облегченных моделей (`rut5-base-multitask`, `rubert-tiny2`) приводит к снижению качества по сравнению с более крупными моделями (например, `roberta-large` или `t5-large`). Это может проявляться в менее точных ответах, особенно на сложные вопросы.

- **Производительность на CPU**: Несмотря на оптимизацию, работа на CPU значительно медленнее, чем на GPU, особенно на этапах генерации датасета и обучения модели. Это может быть проблемой при обработке больших объемов данных.

- **Качество сгенерированного датасета**: Генерация пар вопрос-ответ с помощью `rut5-base-multitask` может приводить к созданию некорректных или неестественных вопросов и ответов, что требует дополнительной фильтрации и ручной проверки.
- **Ограниченная глубина поиска контекста**: Использование FAISS с `IndexIVFFlat` может приводить к потере точности поиска, особенно если количество кластеров (`n_clusters`) не оптимально настроено для размера датасета.
- **Отсутствие тестирования**: В текущей версии проекта отсутствуют автоматические тесты, что может затруднить проверку корректности работы при внесении изменений.
- **Зависимость от Википедии**: Качество данных зависит от актуальности и полноты статей в Википедии. Устаревшие или неполные статьи могут снижать качество корпуса и, как следствие, модели.

7.3 Особенности реализации под CPU

В связи с отсутствием устойчивого доступа к GPU проект был специально оптимизирован для работы на CPU. Вот ключевые аспекты этой оптимизации:

- **Выбор облегченных моделей**: Вместо ресурсоемких моделей, таких как `bert-base` или `t5-large`, были выбраны облегченные версии: `cointegrated/rubert-tiny2`, `cointegrated/rut5-base-multitask`, и `sentence-transformers/distiluse-base-multilingual-cased-v2`.

  Эти модели имеют меньшее количество параметров и требуют меньше памяти, что делает их подходящими для работы на CPU.
- **Настройка обучения**: Использованы параметры обучения, минимизирующие потребление памяти: малый размер батча (`per_device_train_batch_size=4`), накопление градиентов (`gradient_accumulation_steps=8`), и ранняя остановка (`EarlyStoppingCallback` с `early_stopping_patience=2`). Уменьшена максимальная длина последовательности (`max_length=384`) для снижения вычислительной нагрузки.
- **Оптимизация FAISS**: Использован `faiss-cpu` вместо `faiss-gpu`, а также индекс `IndexIVFFlat` для быстрого поиска контекста с минимальными требованиями к памяти.
- **Многопоточность**: Для ускорения обработки данных на CPU использована многопоточность (например, в `wiki_scraper` с `ThreadPoolExecutor` и в `dataset.map` с `num_proc=4`).
- **Ограничение объема данных**: Реализована обрезка длинных контекстов (`max_context_chars=2000`) для предотвращения перегрузки памяти при генерации датасета и обучении модели.

7.4 Возможные улучшения

- **Использование GPU**: При появлении доступа к GPU можно заменить облегченные модели на более крупные (например, `deepset/roberta-base-squad2` для QA или `t5-large` для генерации вопросов) и использовать `faiss-gpu` для ускорения поиска контекста.
- **Улучшение качества датасета**: Добавить ручную проверку или crowdsourcing для фильтрации сгенерированных пар вопрос-ответ. Использовать более сложные модели для генерации вопросов и ответов, если ресурсы позволят.
- **Оптимизация FAISS**: Настроить параметры FAISS (например, `n_clusters`, `nprobe`) для повышения точности поиска контекста.
- **Добавление тестов**: Реализовать автоматические тесты с использованием `pytest` для проверки корректности работы каждого модуля.
- **Интеграция с веб-интерфейсом**: Заменить CLI-интерфейс на веб-интерфейс с использованием библиотек, таких как `Flask` или `FastAPI`, для более удобного взаимодействия с пользователем.
- **Расширение данных**: Добавить данные из других источников (например, официальных сайтов Формулы 1, новостных порталов) для повышения полноты корпуса.

8 Заключение

Проект `F1_expert` представляет собой полноценную систему для ответа на вопросы о Формуле 1, оптимизированную для работы на CPU. Использование облегченных моделей и тщательная настройка параметров позволили реализовать проект в условиях ограниченных вычислительных ресурсов. Несмотря на некоторые ограничения, проект демонстрирует хорошую функциональность и может быть расширен при появлении доступа к GPU или другим ресурсам. Код подготовлен для размещения на GitHub с модульной структурой, документацией и списком зависимостей, что делает его доступным для сообщества и упрощает дальнейшую разработку.

References

[1] Lewis P. et al. (2020). Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. *Advances in Neural Information Processing Systems*, 33, 9459-9474.

[2] Vaswani A. et al. (2017). Attention Is All You Need. *Advances in Neural Information Processing Systems*, 30.

[3] Reimers N., Gurevych I. (2020). Making Monolingual Sentence Embeddings Multilingual Using Knowledge Distillation. *Proceedings of EMNLP 2020*.

[4] Johnson J., Douze M., Jégou H. (2019). Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, 7(3), 535-547.

[5] Rajpurkar P. et al. (2016). SQuAD: 100,000+ Questions for Machine Comprehension of Text. *Proceedings of EMNLP 2016*.

[6] Sanh V. et al. (2019). DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. *NeurIPS EMC2 Workshop*.

[7] Tay Y. et al. (2020). Efficient Transformers: A Survey. *ACM Computing Surveys*, 55(6), 1-28.

[8] Alberti C. et al. (2019). Synthetic QA Corpora Generation with Roundtrip Consistency. *Proceedings of ACL 2019*.

[9] Kuratov Y., Arkhipov M. (2019). Adaptation of Deep Bidirectional Multilingual Transformers for Russian Language. *Computational Linguistics and Intellectual Technologies*, 18(1), 333-339.

[10] Xue L. et al. (2021). mT5: A Massively Multilingual Pre-trained Text-to-Text Transformer. *Proceedings of NAACL-HLT 2021*.

[11] Shen Y. et al. (2021). Active Learning for Sequence Tagging with Deep Pre-trained Models and Bayesian Uncertainty Estimates. *Proceedings of EACL 2021*.