| МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  РОССИЙСКОЙ ФЕДЕРАЦИИ  Федеральное государственное автономное образовательное учреждение  высшего образования  «ТЮМЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»  ШКОЛА КОМПЬЮТЕРНЫХ НАУК  Кафедра программного обеспечения | | | |
| --- | --- | --- | --- |
|  | | РЕКОМЕНДОВАНО К ЗАЩИТЕ В ГЭК  Заведующий кафедрой  К.т.н., доцент  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ М. С. Воробьева  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2024 г. | |
| **ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА**  бакалаврская работа  РАЗРАБОТКА ВЕБ-ПРИЛОЖЕНИЯ ДЛЯ ГЕНЕРАЦИИ ТЕСТОВЫХ ЗАДАНИЙ ПО МАТЕРИАЛАМ ЛЕКЦИЙ | | | |
| Математическое обеспечение и администрирование информационных систем  Профиль «Технологии программирования и анализа больших данных» | | | |
| Выполнили работу  (групповой проект) студенты 4 курса  очной формы обучения |  | | Демухаметов Павел Насипович  Зарембо Яков Андреевич |
| Руководитель  Старший преподаватель кафедры ПО |  | | Аврискин Михаил Владимирович |
| Руководитель Доцент, к.т.н., зав. кафедрой ПО |  | |  |
| Тюмень  2024 год | | | |

ОГЛАВЛЕНИЕ

[ВВЕДЕНИЕ 5](#_dcr1ypb1g96n)

[ГЛАВА 1. АНАЛИЗ МЕТОДОВ ДЛЯ АВТОМАТИЧЕСКОЙ ГЕНЕРАЦИИ ТЕСТОВ 8](#_reicfj9qr57k)

[1.1 ТЕСТОВЫЕ ЗАДАНИЯ 8](#_g537iy1pn8ur)

[1.2. ПОСТАНОВКА ЗАДАЧИ И ТРЕБОВАНИЯ К ПРОЕКТУ 9](#_i73cexq7b5ld)

[1.3. ГЕНЕРАЦИЯ ВОПРОСОВ ПО ТЕКСТУ 9](#_lhgwgvam45kj)

[1.3.1. ОПИСАНИЕ МОДЕЛЕЙ ГЕНЕРАЦИИ ВОПРОСОВ 9](#_vz9r4gsidgax)

[1.3.2. МЕТРИКИ И ДАТАСЕТЫ ДЛЯ ОЦЕНКИ МОДЕЛЕЙ ГЕНЕРАЦИИ ВОПРОСОВ 10](#_ndktvgysy21f)

[1.4. ГЕНЕРАЦИЯ ДИСТРАКТОРОВ 11](#_9nau7088b29x)

[1.4.1. ОПИСАНИЕ МЕТОДОВ ГЕНЕРАЦИИ ДИСТРАКТОРОВ 11](#_mnytzwnkyeqf)

[1.4.2. МЕТРИКИ И ДАТАСЕТЫ ДЛЯ ОЦЕНКИ МЕТОДОВ ГЕНЕРАЦИИ ДИСТРАКТОРОВ И СТАТИЧЕСКИХ ЭМБЕДДИНГ-МОДЕЛЕЙ 11](#_hlfmqyd9xc5h)

[ГЛАВА 2. РЕАЛИЗАЦИЯ И СРАВНЕНИЕ ПОДХОДОВ ДЛЯ АВТОМАТИЧЕСКОЙ ГЕНЕРАЦИИ ТЕСТОВ 13](#_3tekurykw7l)

[2.1. СРАВНЕНИЕ МОДЕЛЕЙ ГЕНЕРАЦИИ ВОПРОСОВ 13](#_72zklgjnaxha)

[2.2. СРАВНЕНИЕ МЕТОДОВ И МОДЕЛЕЙ ГЕНЕРАЦИИ ДИСТРАКТОРОВ И СТАТИЧЕСКИХ ЭМБЕДДИНГ-МОДЕЛЕЙ 14](#_b2ibbhtszxd)

[2.3. ПРОЦЕСС ГЕНЕРАЦИИ ТЕСТОВЫХ ЗАДАНИЙ 15](#_ge1l0g9oel25)

[ГЛАВА 3. РАЗРАБОТКА ВЕБ-ПРИЛОЖЕНИЯ 17](#_du5fyoqxp3g)

[3.1. ТЕХНИЧЕСКОЕ ЗАДАНИЕ 17](#_nx6gja9lyuga)

[3.1.1. ТРЕБОВАНИЯ К ФУНКЦИОНАЛУ 17](#_tbwm6a8cga93)

[3.2. АРХИТЕКТУРА ПРИЛОЖЕНИЯ. 18](#_ouuqldoaojru)

[3.3. РАЗРАБОТКА БАЗЫ ДАННЫХ ДЛЯ ХРАНЕНИЯ ПОЛЬЗОВАТЕЛЕЙ И ФАЙЛОВОГО ХРАНИЛИЩА 19](#_2yt7la6b3jea)

[3.4. РАЗРАБОТКА ПОЛЬЗОВАТЕЛЬСКОЙ ЧАСТИ ВЕБ-ПРИЛОЖЕНИЯ 20](#_44wdr95wflvg)

[3.5. РАЗРАБОТКА МОДУЛЯ ИЗВЛЕЧЕНИЯ ТЕКСТА ИЗ МАТЕРИАЛОВ ЛЕКЦИЙ 21](#_mx58udgfad1i)

[3.6. РАЗРАБОТКА МОДУЛЯ ИЗВЛЕЧЕНИЯ КЛЮЧЕВЫХ СЛОВ ИЗ ТЕКСТА МАТЕРИАЛОВ ЛЕКЦИЙ 22](#_lqak2b28dh9m)

[3.7. РАЗРАБОТКА МОДУЛЯ ГЕНЕРАЦИИ ВОПРОСОВ ПО ЧАСТЯМ ТЕКСТА. 22](#_bg18schyu4tu)

[3.8. ОПИСАНИЕ ПОЛЬЗОВАТЕЛЬСКОГО ИНТЕРФЕЙСА 24](#_jkjds5c3sp1c)

[3.9.ТЕСТИРОВАНИЕ 28](#_1znhzs3vz3f8)

[БИБЛИОГРАФИЧЕСКИЙ СПИСОК 31](#_wbdyd7xqx15d)

[ПРИЛОЖЕНИЕ 1. ПРОГРАММНЫЙ КОД МОДУЛЯ ИЗВЛЕЧЕНИЯ ТЕКСТА ИЗ МАТЕРИАЛОВ ЛЕКЦИЙ 32](#_xknqdicxfo5j)

[ПРИЛОЖЕНИЕ 2. ПРОГРАММНЫЙ КОД МОДУЛЯ ИЗВЛЕЧЕНИЯ КЛЮЧЕВЫХ СЛОВ 34](#_afhraigbmwzl)

[ПРИЛОЖЕНИЕ 3. ПРОГРАММНЫЙ КОД МОДУЛЯ ГЕНЕРАЦИИ ВОПРОСОВ ТЕКСТУ 35](#_w09c9g9vk4az)

## ВВЕДЕНИЕ

Тестирование остаётся одним из наиболее распространенных методов оценки знаний студентов. Однако процесс составления тестов представляет собой трудоёмкую задачу, требующую значительных временных затрат со стороны преподавателей. Основной сложностью является необходимость охвата всех ключевых аспектов учебного материала при разработке тестовых вопросов, а также создание правильных и логичных дистракторов - неверных вариантов ответов.

Для более понимания текущих проблем в области тестирования было проведено анкетирование среди 10 преподавателей Тюменского государственного университета, результаты представлены на рисунках 1-4. Все респонденты отметили, что активно используют тестирование в обучении. Преподаватели также выразили мнение, что проведение тестирования играет важную роль в образовательном процессе. По их мнению, тесты способствуют объективной оценке усвоения материала студентами и могут служить инструментом для повышения мотивации обучающихся, побуждая их более внимательно воспринимать и изучать лекционный материал.

## Points scored

Рис. 1. Результаты анкетирования

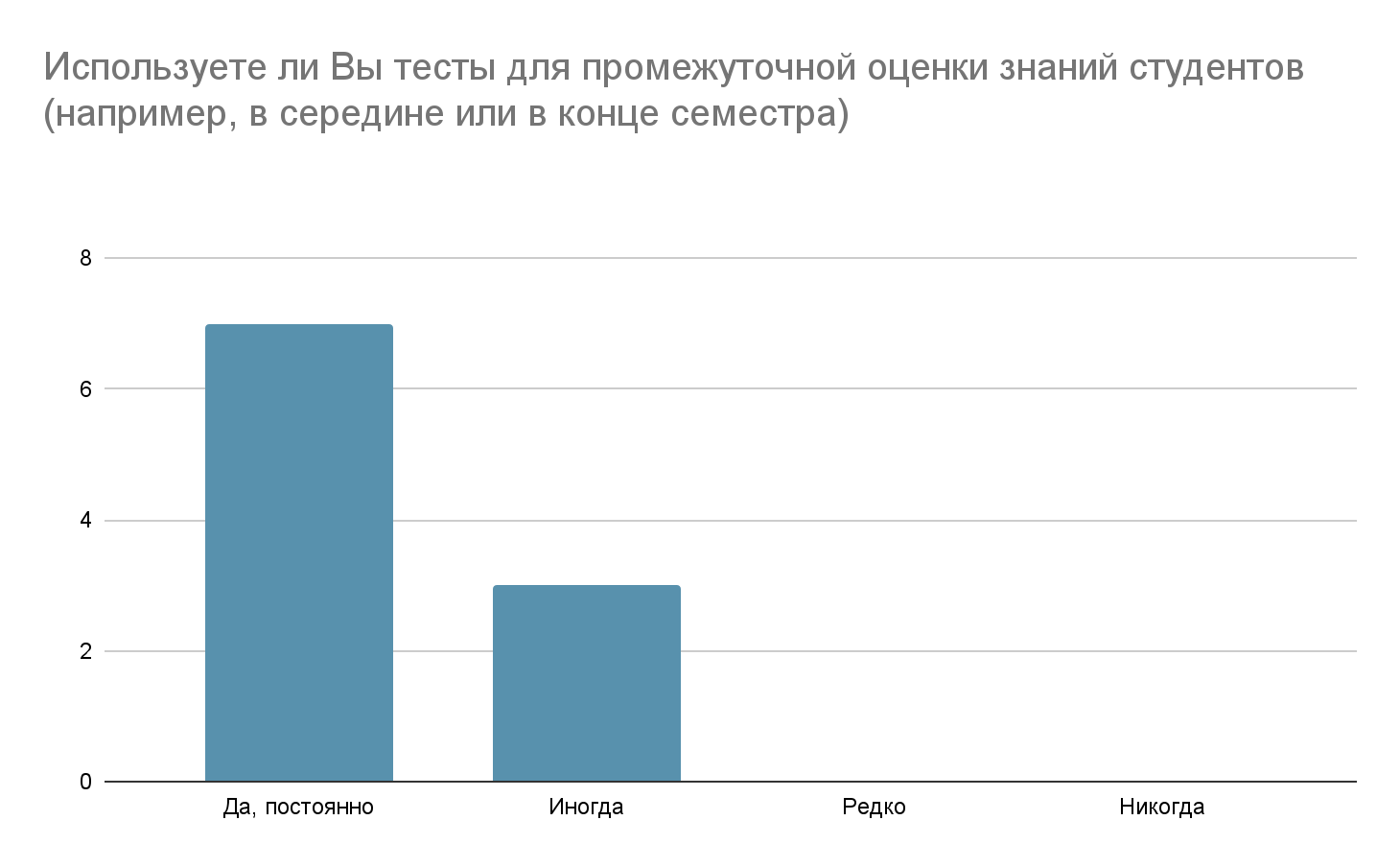


Рис. 2. Результаты анкетирования

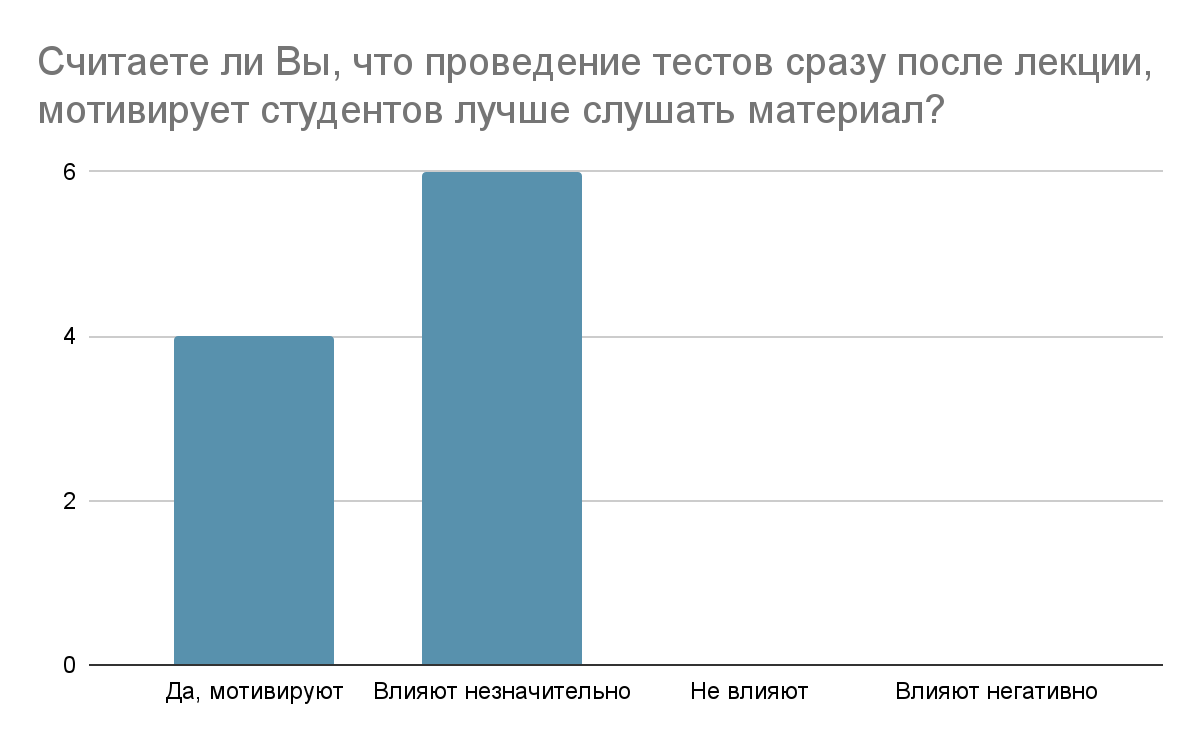


Рис. 3. Результаты анкетирования

## Points scored

Рис. 4. Результаты анкетирования

Кроме того, преподаватели отметили, что создание тестов вручную занимает в среднем от 1 до 2 часов. Девять из десяти опрошенных преподавателей подтвердили, что процесс создания тестов является трудоёмким, требует значительных временных и интеллектуальных затрат. Автоматическая генерация тестов могла бы снять эту нагрузку с преподавателей, освобождая их от необходимости составлять тест вручную.

В связи с этим существует потребность в автоматизации процесса создания тестов, что позволит упростить задачу преподавателей в составлении тестов и снизить время на подготовку тестов.

## ГЛАВА 1. АНАЛИЗ МЕТОДОВ ДЛЯ АВТОМАТИЧЕСКОЙ ГЕНЕРАЦИИ ТЕСТОВ

## 1.1 ТЕСТОВЫЕ ЗАДАНИЯ

Тестирование - это процесс, направленный на оценку знаний, навыков и компетенций субъекта.

Мы выяснили, какие типы вопросов наиболее часто используются преподавателями. Результаты показали, что наибольшей популярностью пользуются следующие виды вопросов: вопросы с выбором одного правильного ответа, вопросы с выбором нескольких правильных ответов и вопросы с открытым ответом.



Рис. 5. Результаты анкетирования

В рамках данной работы рассматриваются два основных типа тестов: тесты с открытым ответом и тесты с выбором одного правильного варианта ответа. Выбор этих типов вопросов обусловлен их популярностью и возможностью трансформации вопросов с выбором ответа в вопросы с открытым ответом путем удаления предложенных вариантов.

В тестах с открытым ответом от тестируемого требуется самостоятельно сформулировать ответ на вопрос.

В тестах с выбором одного ответа тестируемому предлагается несколько вариантов ответа, где только один правильный, а остальные варианты являются дистракторами (неправильные варианты ответа).

## 1.2. ПОСТАНОВКА ЗАДАЧИ И ТРЕБОВАНИЯ К ПРОЕКТУ

Цель проекта: разработать веб-приложение для автоматической генерации тестовых заданий по материалам лекций, что позволит упростить ускорить процесс создания тестов.

Для достижения цели проекта были определены следующие задачи:

1. Выбор и оценка языковой модели для генерации вопросов.
2. Выбор и оценка модели для генерации вариантов ответа на вопрос.
3. Разработка архитектуры веб-приложения.
4. Разработка структуры базы данных для хранения пользователей и созданных ими тестов
5. Создание клиентской части приложения.
6. Создание серверной части приложения.
7. Тестирование.

## 1.3. ГЕНЕРАЦИЯ ВОПРОСОВ ПО ТЕКСТУ

## 1.3.1. ОПИСАНИЕ МОДЕЛЕЙ ГЕНЕРАЦИИ ВОПРОСОВ

Была рассмотрена генерация вопросов по неструктурированным данным в формате текста с использованием seq2seq-моделей, основанных на трансформерах [1]. Всего рассмотрено 9 моделей с Hugging Face, представляющих из себя модификации mBart, mT5, FRED T5 и ruT5. Некоторые из этих моделей также способны отвечать на вопросы и генерировать вопросы с заранее известным ответом. Также был рассмотрен GigaChat.

## 1.3.2. МЕТРИКИ И ДАТАСЕТЫ ДЛЯ ОЦЕНКИ МОДЕЛЕЙ ГЕНЕРАЦИИ ВОПРОСОВ

Для оценки моделей генерации вопросов мы использовали датасет facebook/belebele [2].

Таблица 1

Поля датасета facebook/belebele

| **Название столбца** | **Описание** |
| --- | --- |
| link | Ссылка на источник текста |
| question\_number | Номер вопроса для текста |
| flores\_passage | Текст |
| question | Вопрос |
| mc\_answer{i} | i-ый ответ на вопрос |
| correct\_answer\_num | Номер верного ответа |
| dialect | Язык и письмо |
| ds | Дата обращения к источнику |

Для оценки генерации вопросов нам нужны:

Таблица 2

Поля из датасета facebook/belebele rus\_Cyrl,

нужные для оценки генерации вопросов, и их статистика

| **Название столбца** | **Медиана (длины)** | **Отклонение (длины)** | **Максимум (длины)** |
| --- | --- | --- | --- |
| flores\_passage | 498 | 180 | 1527 |
| question | 81 | 30 | 196 |
| correct\_answer | 23 | 19 | 132 |

Столбец correct\_answer формируется из столбцов mc\_answer{i} и номера correct\_answer\_num. Он нужен для проверки моделей, которые генерируют вопросы с учётом заранее известного ответа. (Статистика указана как за все mc\_answer{i}).

При оценке результатов производится токенизация, удаление знаков препинания и стоп-слов, приведение к леммам/стемам и нижнему регистру.

Используемые метрики:

, где

,

, где

,

,

где R — это множество слов (или токенов) в эталонном тексте, G — это множество слов в сгенерированном тексте, а sim(g,c) — это косинусное сходство между векторами слов r и g.

## 1.4. ГЕНЕРАЦИЯ ДИСТРАКТОРОВ

## 1.4.1. ОПИСАНИЕ МЕТОДОВ ГЕНЕРАЦИИ ДИСТРАКТОРОВ

Для оценки использовался метод извлечения близких по эмбеддингам слов с помощью статических эмбеддинг-моделей, таких как Word2Vec и GloVe [3].

## 1.4.2. МЕТРИКИ И ДАТАСЕТЫ ДЛЯ ОЦЕНКИ МЕТОДОВ ГЕНЕРАЦИИ ДИСТРАКТОРОВ И СТАТИЧЕСКИХ ЭМБЕДДИНГ-МОДЕЛЕЙ

, где

,

,

N – порог количества лучших значений, рассматриваемых при расчёте.

, где

,

,

N – порог количества лучших значений, рассматриваемых при расчёте.

Также рассматривается процент ответов, найденных в словаре, для статических моделей эмбеддингов.

## ГЛАВА 2. РЕАЛИЗАЦИЯ И СРАВНЕНИЕ ПОДХОДОВ ДЛЯ АВТОМАТИЧЕСКОЙ ГЕНЕРАЦИИ ТЕСТОВ

## 2.1. СРАВНЕНИЕ МОДЕЛЕЙ ГЕНЕРАЦИИ ВОПРОСОВ

Таблица 3

Сравнение моделей генерации вопросов на датасете facebook/belebele rus\_Cyrl 900 строк

| **Модель** | **ROUGE-L** | **BERTScore** | **Время, сек** |
| --- | --- | --- | --- |
| **hivaze/AAQG-QA-QG-FRED-T5-large (AAQG)** | **24.976** | **73.675** | **1025.806** |
| GigaChat[[1]](#footnote-0) | 17.265 | 69.776 | 2042.723 |
| **ai-forever/ruT5-base (SberQuad)** | **15.783** | **69.181** | **669.094** |
| lmqg/mbart-large-cc25-ruquad-qg (<hl>) | 15.075 | 67.809 | 767.226 |
| doc2query/msmarco-russian-mt5-base-v1 | 15.015 | 69.427 | 597.159 |
| GigaChat | 14.944 | 60.395 | 2042.723 |
| nbroad/mt5-base-qgen | 14.166 | 68.665 | 577.413 |
| hivaze/AAQG-QA-QG-FRED-T5-large (QG) | 14.105 | 68.046 | 1126.046 |
| lmqg/mbart-large-cc25-ruquad-qg | 13.777 | 67.900 | 768.575 |
| lmqg/mt5-base-ruquad-qg | 13.526 | 68.132 | 641.309 |
| lmqg/mt5-base-ruquad-qg (<hl>) | 13.004 | 67.809 | 606.463 |

Лучший результат достигла модель, которая заранее знает о верном ответе. Это может быть следствием того, что наличие верного ответа сокращает количество возможных вопросов, которые можно задать по тексту, и вероятность задать вопрос верный для эксперта, но не совпадающий с эталоном из датасета, ниже. На практике была выбрана модель на третьем месте из-за проблем, далее описанных в разделе 2.3.

Одним из недостатков GigaChat было игнорирование требования по генерации ответа в формате JSON, на это давалось 5 попыток. Это привело к падению значений метрик, а также большему времени, затраченному на генерацию. Это могло бы быть скорректировано изменением промпта: можно написать сам шаблон JSON-файла, либо написать пример другого структурированного формата ответа. Также можно разбить генерацию вопросов и ответов, но это будет дороже по количеству токенов.

Использованный промпт для GigaChat:

“Придумай вопрос по следующему тексту, а также правильный ответ исходя из этого текста и 3 неправильных ответа. Ты должен обязательно ответить в формате JSON с полями question, correct\_answer, incorrect\_answers. Сам текст: “

Остальные модели использовали свои промпты, указанные на их страницах, но большинство требуют лишь текст, по которому будут генерироваться вопросы.

## 2.2. СРАВНЕНИЕ МЕТОДОВ И МОДЕЛЕЙ ГЕНЕРАЦИИ ДИСТРАКТОРОВ И СТАТИЧЕСКИХ ЭМБЕДДИНГ-МОДЕЛЕЙ

Таблица 4

Сравнение генерации дистракторов на датасете facebook/belebele rus\_Cyrl 900 строк

| **Модель** | **ROUGE1@10** | **F1@10** | **Процент ответов в словаре** | **Время, сек** |
| --- | --- | --- | --- | --- |
| **navec\_hudlit\_v1\_12B\_500K\_300d\_100q** | **5.73** | **13.2** | **16.9** | **57** |
| navec\_news\_v1\_1B\_250K\_300d\_100q | 4.81 | 10.4 | 16.2 | 49 |
| ruwikiruscorpora\_upos\_cbow\_300\_10\_2021 | 4.59 | 11.9 | 15.4 | 49 |

На практике была выбрана модель ruwikiruscorpora, поскольку она содержит только леммы, а также разметку частей речи. Лучше всего данный метод себя показывает, когда генерация производится из названия, имени или числа. В будущем планируется оценить GigaChat.

## 2.3. ПРОЦЕСС ГЕНЕРАЦИИ ТЕСТОВЫХ ЗАДАНИЙ

Изначально был придуман следующий конвейер генерации тестов:

1. Разбиение текста на тематические части
2. Извлечение ключевых слов в каждой тематической части
3. Извлечение предложений, в которых содержится ключевое слово
4. Генерация вопроса языковой моделью с заранее известным верным ответом (ключевым словом)
5. Генерация дистракторов с помощью статической эмбеддинг-модели

На практике было выявлено, что наш способ для поиска потенциальных верных ответов и предложений для этих ответов приводит к тому, что модель либо игнорирует этот ответ, либо генерирует тривиальный вопрос, потому что в найденных предложениях эти слова являются второстепенными членами, вокруг которых нет утверждений, которые модель могла бы использовать для составления вопроса.

Одно из возможных решений этой проблемы – фильтрация предложений по синтаксису. Если ключевое слово находится глубже определённого порога в синтаксическом дереве предложения, то можно отсекать такие предложения.

Также вместо ключевых слов можно использовать словарь терминов и ввод терминов от пользователя.

Было решено, что будут также генерироваться ответы на вопрос. Ответы генерируются той же моделью, что генерирует и вопрос.

## ГЛАВА 3. РАЗРАБОТКА ВЕБ-ПРИЛОЖЕНИЯ

## 3.1. ТЕХНИЧЕСКОЕ ЗАДАНИЕ

## 3.1.1. ТРЕБОВАНИЯ К ФУНКЦИОНАЛУ

Разрабатываемое приложение для генерации тестовых заданий должно обеспечивать возможность выполнения перечисленных ниже функций:

1. регистрация пользователя приложения.
2. осуществление контроля уровня прав доступа: неавторизованные пользователи не могут сохранять и просматривать свои созданные тесты;
3. Операции с созданием тестов
   1. Загрузка материалов: поддержка текстовых данных и файлов следующих форматов:

* Текстовые файлы: txt, docx, pdf
* Медиафайлы: видео (mp4, avi, mov, mkv) и аудио (mp3, wav, flac, aac, ogg)
  1. Генерация тестов
     1. Выбор количества вопросов:
* Открытые вопросы
* Вопросы с выбором одного правильного варианта ответа
  + 1. Генерация вопросов на основе:
* Всех переданных материалов
* Отдельных ключевых слов и связанных с ними предложений
  + 1. Работа с ключевыми словами:
* Автоматическое извлечение ключевых слов
* Возможность создания собственных ключевых слов для генерации вопросов
  1. Редактирование тестов
     1. Редактировать вопросы (добавлять новые, изменять текст существующих, удалять)
     2. Редактировать варианты ответов (добавлять, изменять, удалять)

1. Выгрузка созданного теста в формате GIFT.
2. Возможность повторного скачивания сохраненного теста в формате GIFT для зарегистрированного пользователя.

## 3.2. АРХИТЕКТУРА ПРИЛОЖЕНИЯ.

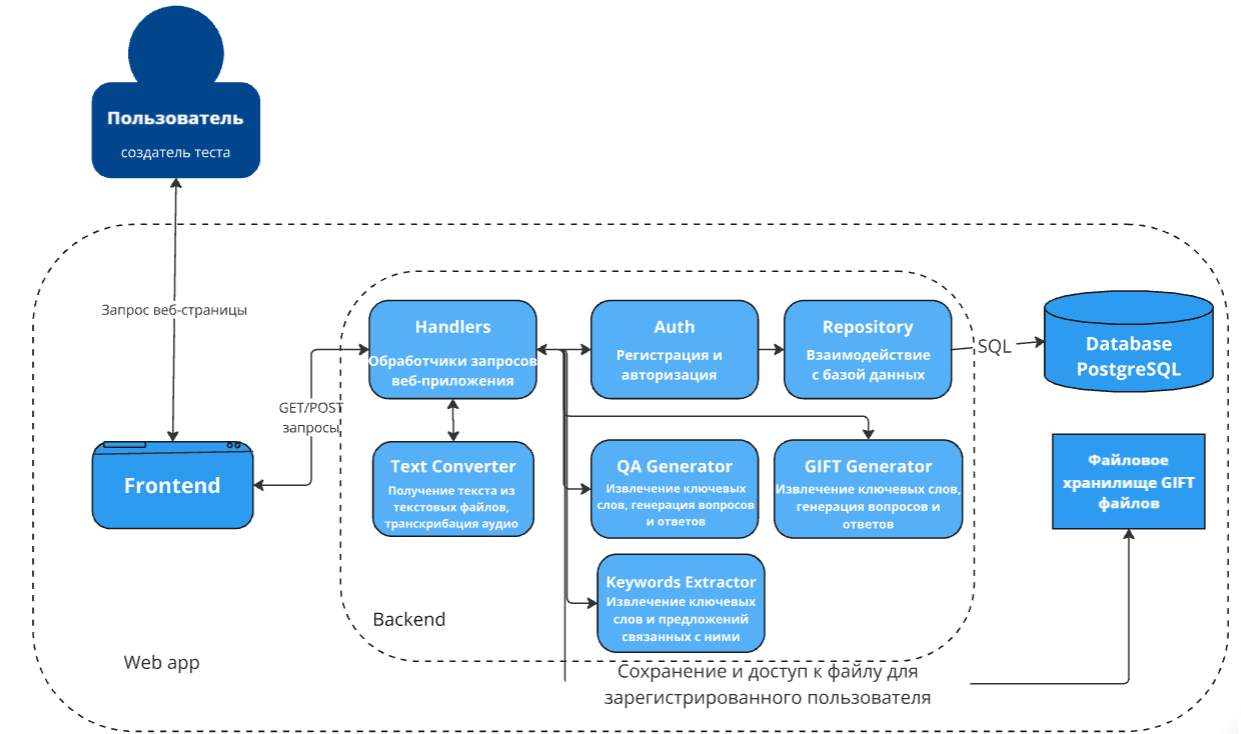


Рис. 6. Схема архитектуры веб-приложения

Приложение состоит из 3 частей:

* Клиентская часть: React, Tailwind CSS
* Серверная часть: Python, FastApi, PKE, Transformer, PyTorch
* База данных: PostgreSQL

Серверная часть веб-приложения реализована в виде монолита, что позволяет избежать издержек, связанных с передачей больших объемов данных между микросервисами. Центральным элементом серверной части является модуль Handlers — обработчик входящих GET/POST-запросов от клиентской части (Frontend). Он распределяет запросы между модулями

Модуль Auth отвечает за регистрацию и аутентификацию пользователей, обеспечивая контроль доступа и защиту данных. Взаимодействие с базой данных осуществляется через модуль Repository, который абстрагирует доступ к хранилищу данных.

Для обработки входящих материалов используется Text Converter. Этот модуль принимает текстовые и файлы, извлекает текст из документов, а также выполняет транскрибацию аудио- и видеофайлов, преобразуя их в текстовый формат. Далее, на основе полученного текста, модуль Keywords Extractor извлекает ключевые слова и соответствующие им предложения для разбиения текста.

Модуль QA Generator генерирует вопросы и ответы на основе предложений, содержащих ключевые слова. После генерации вопросов и ответов результаты передаются в GIFT Generator, который формирует тесты в стандартизированном формате GIFT. Для зарегистрированных пользователей, сгенерированные файлы сохраняются в файловом хранилище.

## 3.3. РАЗРАБОТКА БАЗЫ ДАННЫХ ДЛЯ ХРАНЕНИЯ ПОЛЬЗОВАТЕЛЕЙ И ФАЙЛОВОГО ХРАНИЛИЩА

Для базы данных приложения используется одна сущность: User. Это позволяет зарегистрированным пользователям сохранять созданные тесты в формате gift. Имя файла для сохранения формируется на основе следующих данных:

* user\_id — уникальный идентификатор текущего пользователя в базе данных, извлекаемый из JWT токена,
* test\_name — название теста, очищенное от специальных символов.

Итоговое имя файла будет иметь вид: user\_id\_название\_теста.gift. Все созданные тесты сохраняются в файле gift\_files.

Описание полей таблицы User приведено в таблице 5.

Таблица 5

Описание таблицы «User» в базе данных

| **Название** | **Тип данных** | **Key** | **Nullable** | **Unique** |
| --- | --- | --- | --- | --- |
| id | integer | PK | – | + |
| username | varchar(128) | – | – | – |
| email | varchar(128) | – | – | + |
| hashed\_password | varchar | – | – | – |
| created\_at | timestamptz |  |  |  |

## 3.4. РАЗРАБОТКА ПОЛЬЗОВАТЕЛЬСКОЙ ЧАСТИ ВЕБ-ПРИЛОЖЕНИЯ

Для разработки frontend части используется React и Tailwind CSS. Согласно требованиям к функциональным требованиям были разработаны страницы:

Главная страница

* Для незарегистрированных пользователей: отображает кнопки авторизации, входа и создания теста
* Для зарегистрированных пользователей: отображает кнопки создания теста и каталога созданных тестов

Страница регистрации и авторизации

* Пользователи могут зарегистрироваться или войти в систему, вводя свои данные (логин, электронная почта, пароль). Страница включает валидацию введенных данных и обработку ошибок (например, уже зарегистрированный пользователь или неправильные данные).
* Для авторизованных пользователей доступ к функционалу скачивания сохранённых тестов

Страница загрузки материалов

* Поддерживает загрузку различных типов материалов, включая текстовые файлы (txt, docx, pdf) и медиафайлы (видео форматов mp4, avi, mov, mkv и аудио форматов mp3, wav, flac, aac, ogg).
* Уведомления об успешной загрузке и ошибках (например, неподдерживаемые форматы файлов).

Страница генерации тестов

* Позволяет выбрать материалы для включения в тест, настроить количество вопросов каждого типа (открытые вопросы и вопросы с одним правильным ответом).
* Возможность выбора ключевых слов для генерации вопросов, с автоматическим извлечением ключевых слов из загруженных материалов и возможностью добавления собственных.

Страница редактирования теста и скачивания

* Пользователи могут редактировать вопросы и ответы уже созданных теста, добавлять новые вопросы, изменять текст существующих или удалять ненужные. Также есть возможность скачать готовый тест в формате GIFT.

Страница списка созданных тестов

* Отображает список всех тестов, созданных пользователем, с возможностью скачивания тестов в формате GIFT.

## 3.5. РАЗРАБОТКА МОДУЛЯ ИЗВЛЕЧЕНИЯ ТЕКСТА ИЗ МАТЕРИАЛОВ ЛЕКЦИЙ

Разработан модуль Text Converter (Приложение 1), обеспечивающий извлечение текста как из текстовых файлов, так и из аудиозаписей.

Для перевода аудио в текст происходит:

1. Извлечение аудио из видеоАудио извлекается из видеозаписей с помощью библиотеки ffmpeg. Для последующей обработки аудиоданные конвертируются в формат WAV с моно-каналом и частотой дискретизации 16 кГц. Это обеспечивает совместимость с моделью обработки речи и упрощает её применение.
2. Конвертация аудиофайловАудиофайлы любых форматов, таких как MP3 или AAC, также могут быть преобразованы в WAV с требуемыми параметрами. Это позволяет работать с аудио, не привязанным к видео
3. Транскрипция речиДля транскрипции используется модель whisper-large-v3-turbo. Аудио делится на сегменты фиксированной длины (чанки), что позволяет обрабатывать длительные записи. Каждый чанк передается в модель для преобразования речи в текст, а затем объединяется в единую текстовую транскрипцию.

Для извлечения текста из документов реализованы функции:

* PDF-файлы. Используется библиотека PyPDF2, которая позволяет извлекать текст постранично, игнорируя отсутствующие текстовые элементы.
* DOCX-файлы. Библиотека python-docx обеспечивает извлечение текста из всех абзацев документов Microsoft Word.
* TXT-файлы. Простые текстовые файлы обрабатываются с использованием стандартных методов чтения Python с поддержкой кодировки UTF-8.

## 3.6. РАЗРАБОТКА МОДУЛЯ ИЗВЛЕЧЕНИЯ КЛЮЧЕВЫХ СЛОВ ИЗ ТЕКСТА МАТЕРИАЛОВ ЛЕКЦИЙ

Для извлечение ключевых слов, модуль Keywords Extractor (Приложение 2) используется библиотека rutermextract, лучше всего показавшая себя среди методов без обучения [4][5]. Далее текст проходит процесс лемматизации найденных ключевых слов. Этот этап реализуется с использованием библиотеки Natasha. В процессе слова приводятся к их базовой форме, что позволяет сопоставлять ключевые слова с их вариантами использования в тексте

Заключительным этапом является поиск предложений, содержащих ключевые слова. Для этого текст разбивается на отдельные предложения, каждое из которых лемматизируется. Затем осуществляется поиск совпадений между лемматизированным ключевым словом и словами в предложении. Результатом работы является список предложений, связанных с конкретным ключевым словом, что позволяет выделить контекст его употребления.

## 3.7. РАЗРАБОТКА МОДУЛЯ ГЕНЕРАЦИИ ВОПРОСОВ ПО ЧАСТЯМ ТЕКСТА.

Модуль QA Generator (Приложение 3) предназначен для автоматизированной генерации тестовых заданий, включая формирование вопросов, ответов и дистракторов (неправильных вариантов ответа).

Для предобработки текста используется библиотека Natasha и NewsEmbedding, они отвечают за:

* Токенизацию текста
* Лемматизацию и фильтрацию слов, включая удаление стоп-слов и пунктуации, что улучшает качество анализа.
* Формирование векторных представлений текста, где для каждого предложения вычисляется усреднённый вектор на основе эмбеддингов слов.

Для генерации вопросов и ответов используются предобученные модели T5 из библиотеки transformers:

* Модель генерации вопросов принимает текстовый контекст и создаёт набор вопросов.
* Модель генерации ответов формирует корректный ответ на основе текста и вопроса, учитывая контекст и языковые особенности.
* Формирование дистракторов

Дистракторы (неправильные варианты ответа) генерируются через взаимодействие с GigaChat API. Этот процесс включает:

* Формирование запроса, включающего текст и правильный ответ.
* Парсинг ответа от API, который может содержать варианты в разных форматах (нумерация, разделение точкой с запятой и т.д.).
* Проверка корректности дистракторов и дополнение их, если изначально получено недостаточное количество.

Для повышения качества вопросов и ответов используется проверка их схожести. Косинусное расстояние между векторными представлениями текста помогает избежать дублирования и избыточности. Это также позволяет улучшить точность при выборе уникальных вопросов.

Модуль работает только с текстом. На данный момент, в тексте вопроса невозможно корректно отобразить сложные формулы или картинки.

## 3.8. ОПИСАНИЕ ПОЛЬЗОВАТЕЛЬСКОГО ИНТЕРФЕЙСА

**Для всех пользователей:**

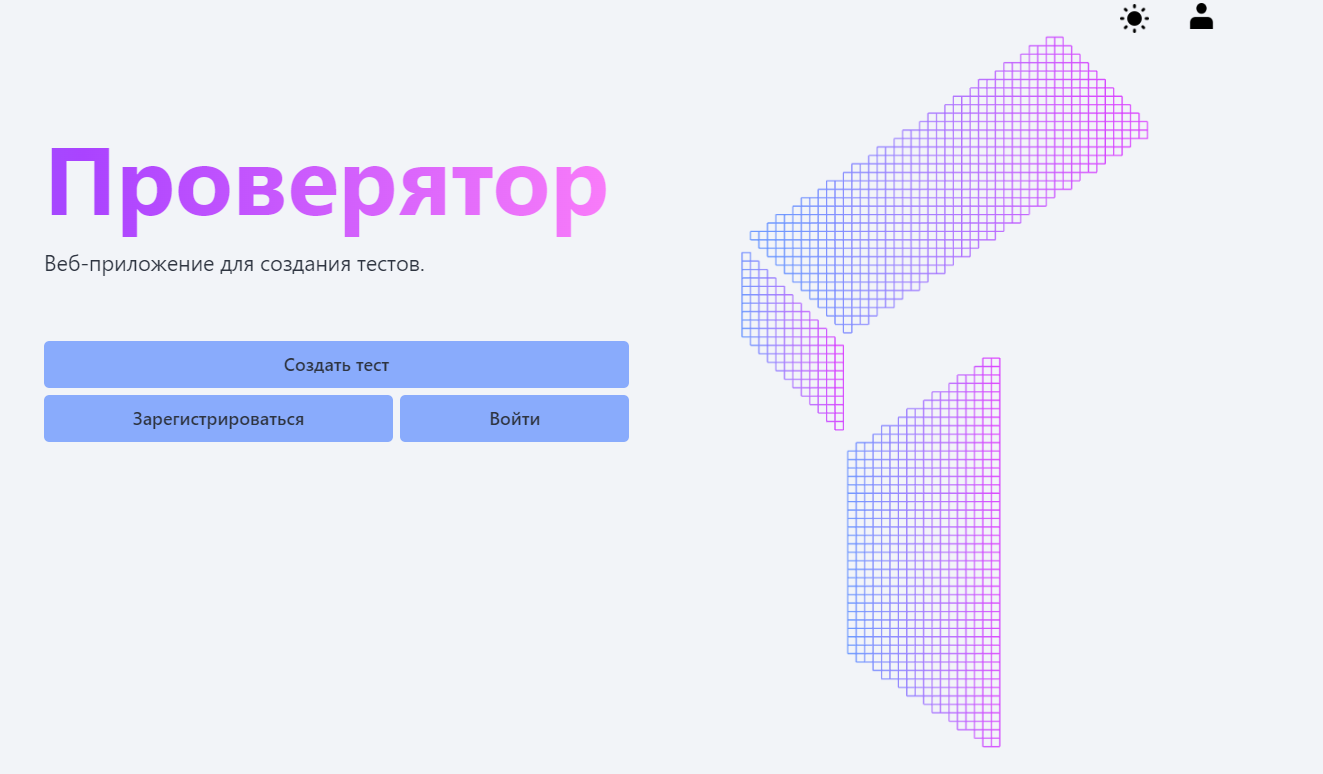
При входе на сайт пользователю открывается приветственная страница (Рис. 7):

Рис. 7. Приветственная страница

на которой доступны следующие действия:

1. **Регистрация**На странице регистрации можно создать новый аккаунт (Рис. 8).

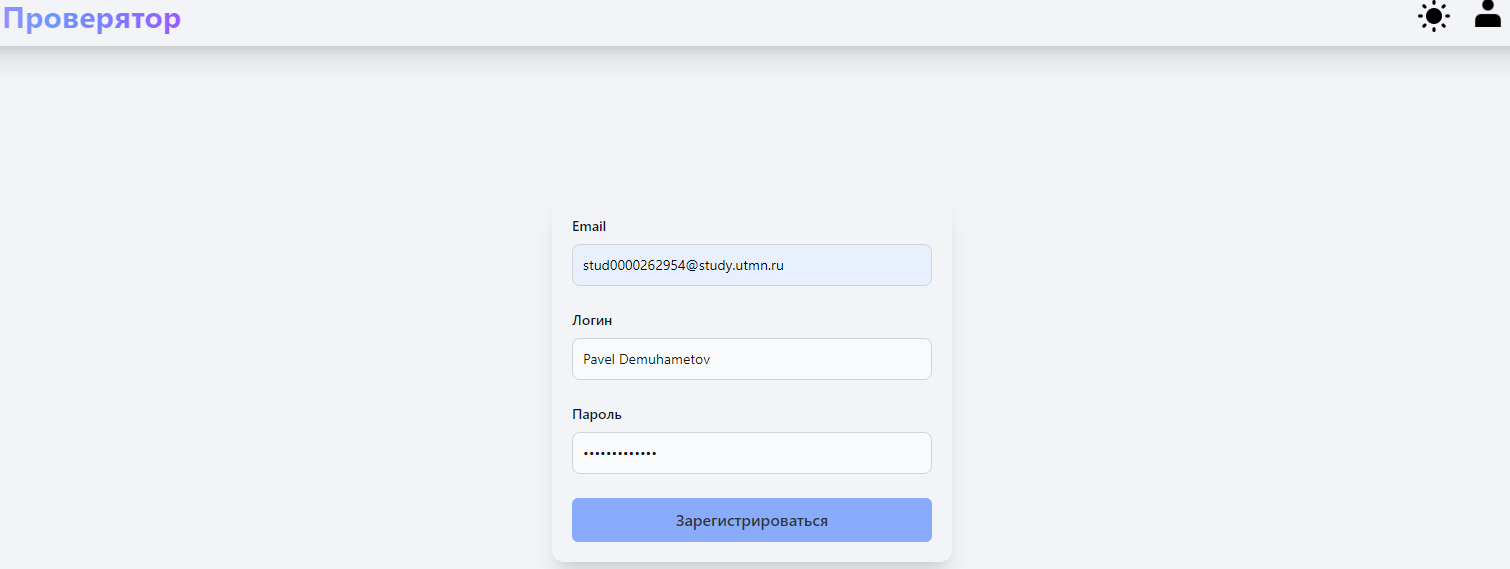


Рис. 8. Страница регистрации

1. **Вход**Если у пользователя уже есть аккаунт, он может войти, перейдя на страницу входа (Рис. 9).

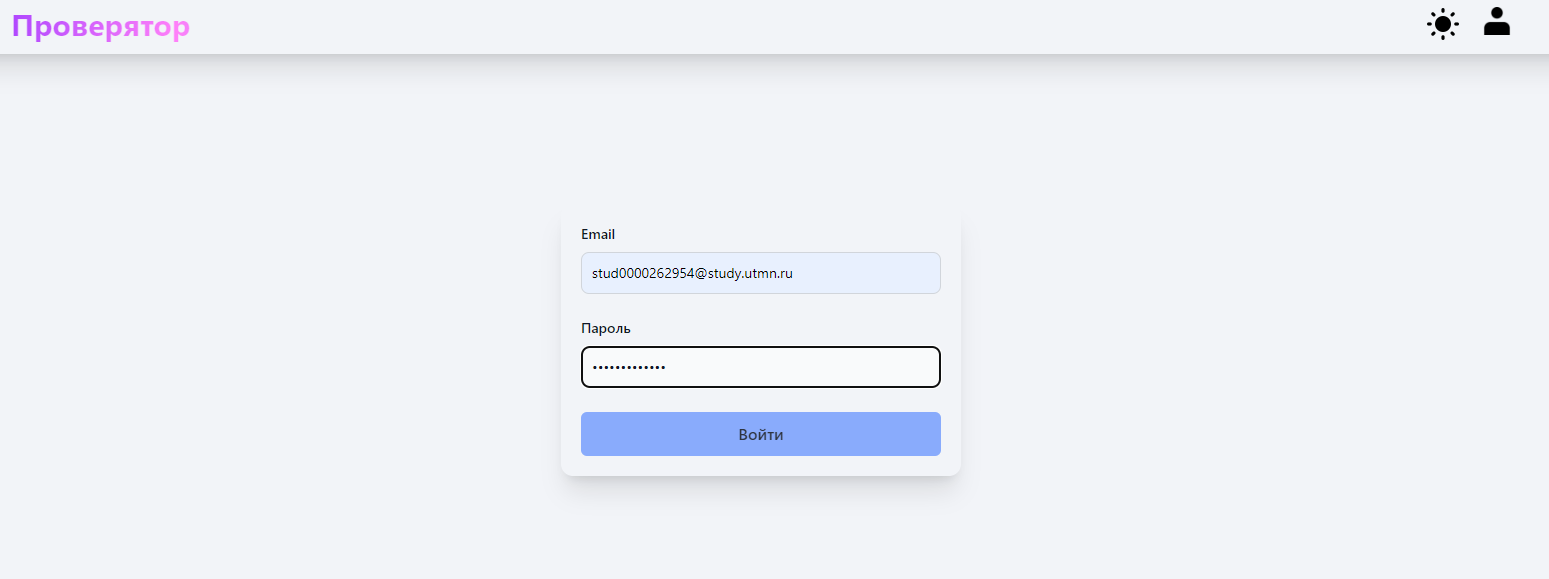


Рис. 9. Страница входа

1. **Создание теста:**

Первый этап создания - страница **загрузки материалов** лекции (Рис. 10), необходимо:

* Ввести материалы лекции.
* Загрузить файл в форматах: txt, docx, pdf, mp, avi, .mov, .mkv, mp3, wav, flac, aac, ogg
* Нажать на кнопку загрузки.

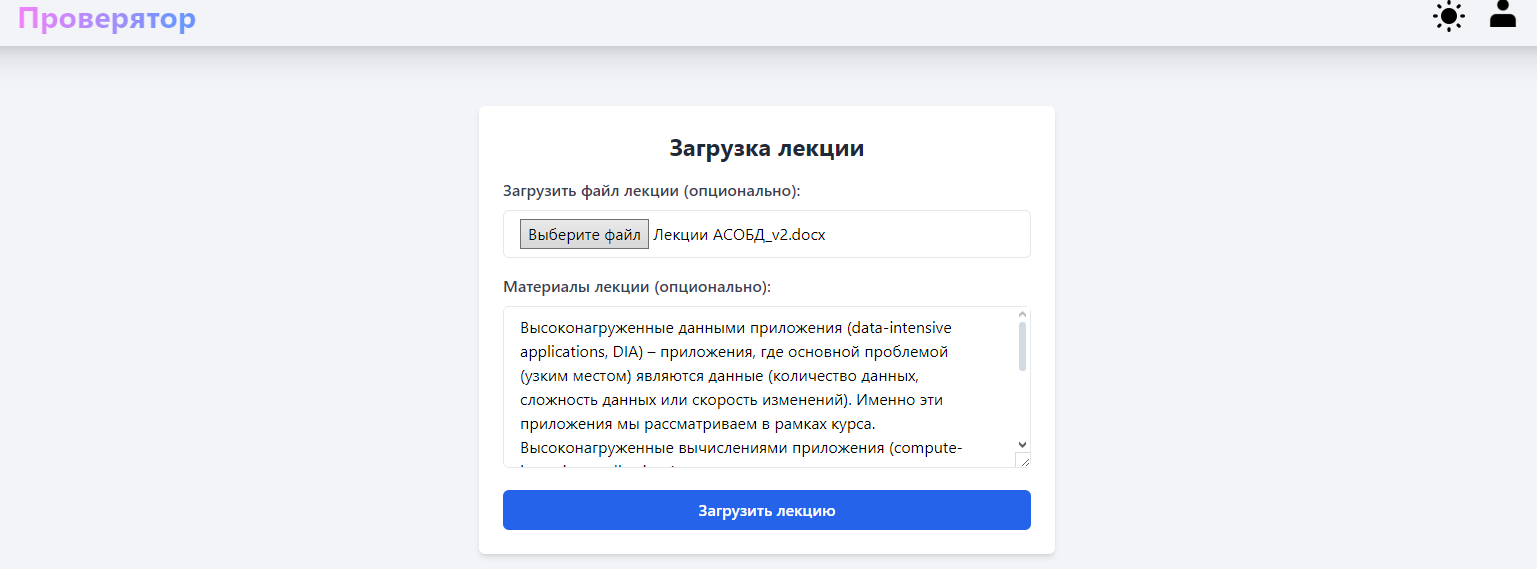


Рис. 10. Страница загрузки материалов

После загрузки лекции, страница **создания теста** (рисунки 11-12)

* Пользователь может указать количество вопросов с открытым ответом и количество вопросов с выбором одного правильного варианта для генерации по всем материалам.
* Пользователь может просмотреть ключевые слова и связанные с ними предложения, выбрать, какие ключевые слова использовать для теста как ответы, а также добавить свои собственные ответы.
* Для каждой темы можно задать количество вопросов с открытым ответом и с несколькими вариантами ответов.
* После настройки параметров, необходимо нажать кнопку для создания теста.

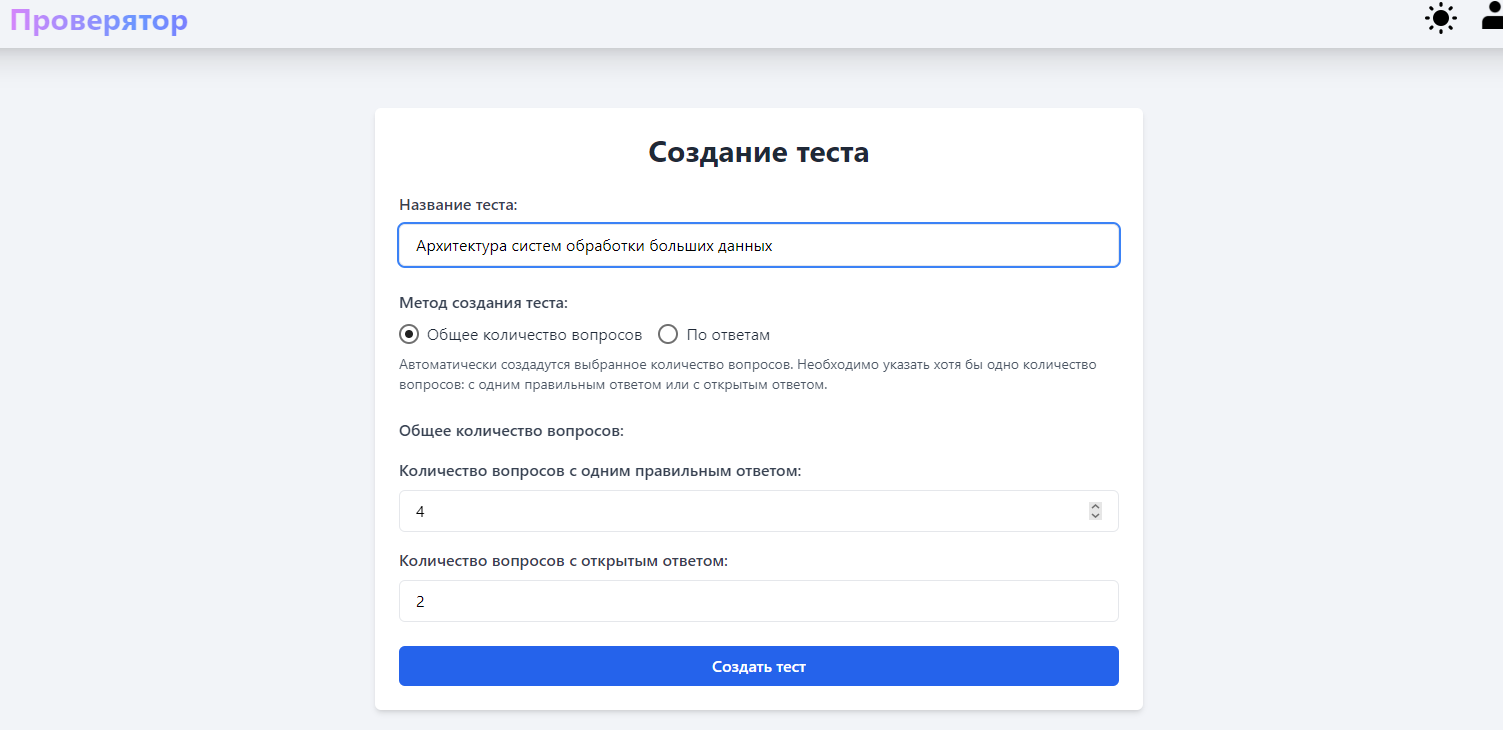


Рис. 11. Страница создания теста (по всем материалам)

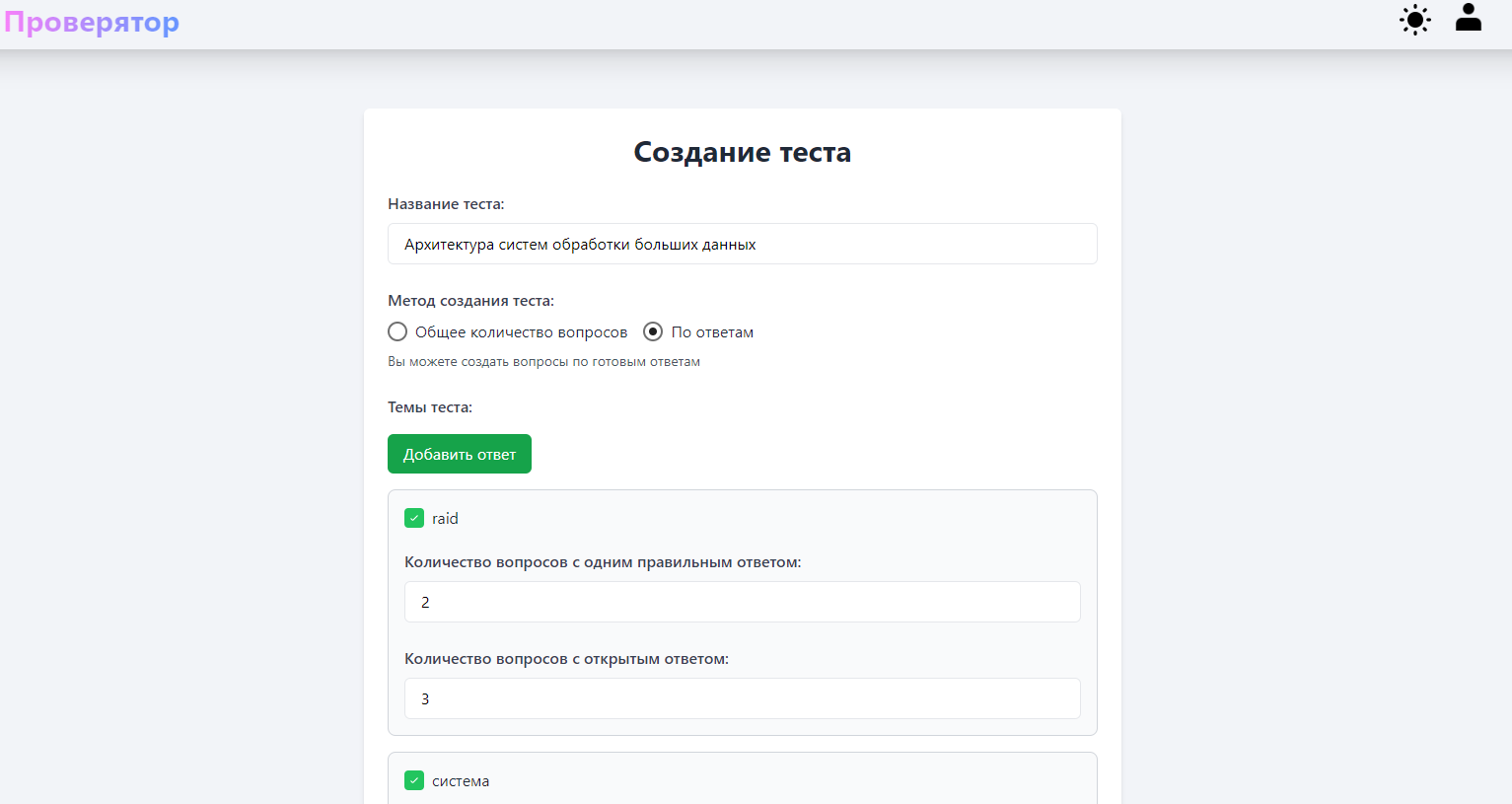


Рис. 11. Страница создания теста (по ответам)

После создания теста, открывается **страница редактирования** теста (Рис. 12):

* Изменить название теста.
* Редактировать вопросы (добавлять новые, изменять текст существующих, удалять).
* Редактировать варианты ответов (добавлять, изменять, удалять).

Отредактированный тест можно скачать в формате GIFT.

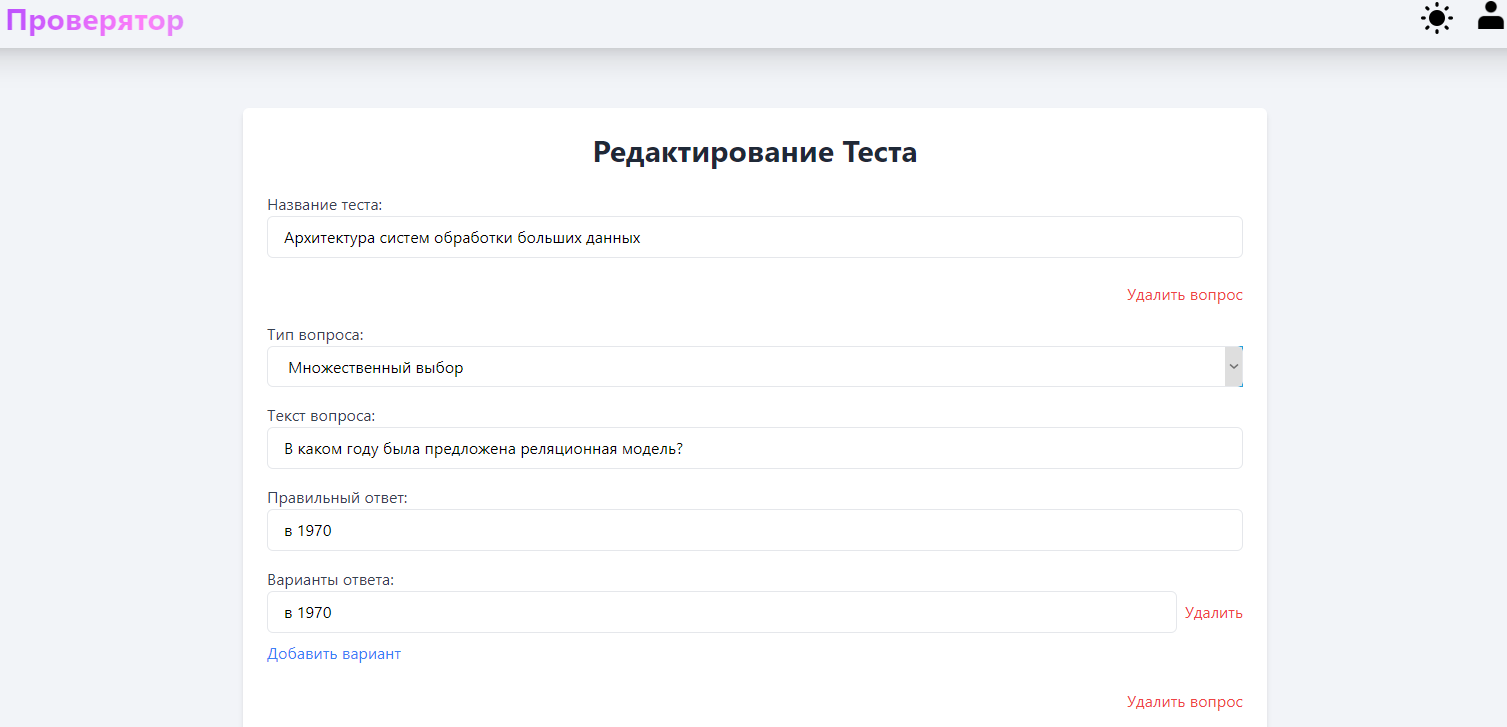


Рис. 12. Страница редактирования теста

**Для зарегистрированных пользователей:**

* Просмотр списка всех созданных тестов и их загрузка в формате GIFT (Рис. 13).

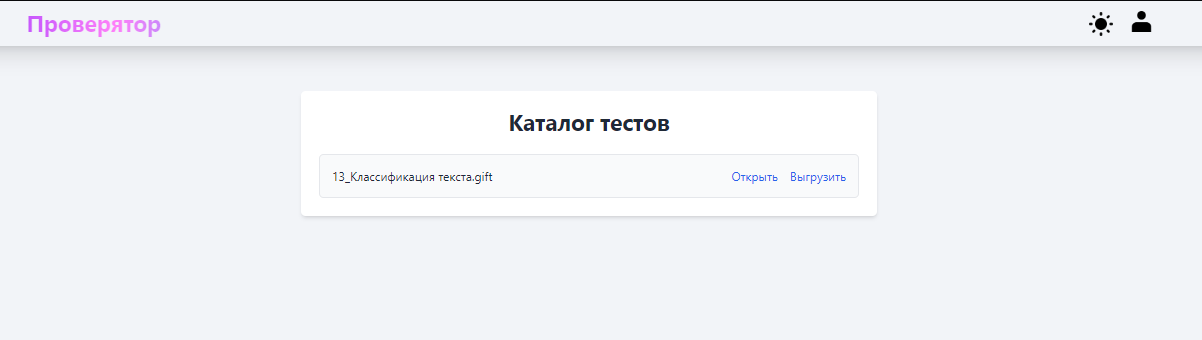


Рис. 13. Каталог созданных тестов

## 3.9.ТЕСТИРОВАНИЕ

Проведено ручное тестирование основных эндпоинтов с использованием Postman. Тестирование включало проверку корректности работы API. Были протестированы следующие эндпоинты:

* Эндпоинт загрузки материалов:

POST ${API\_URL}/upload/

Цель: Проверка функциональности загрузки лекционных материалов (файлы и текст).

Формат ожидаемых данных:

file: файл, загружается через поле file

materials: текст, загружается через поле materials (строка).

Результаты:

* Загрузка только файла:

Ответ 200 OK с подтверждением успешной загрузки файла.

* Загрузка только текстовых материалов:

Ответ 200 OK с подтверждением успешной загрузки текстовых материалов.

* Загрузка файла и текстовых материалов:

Ответ 200 OK с подтверждением загрузки как файла, так и текстовых материалов.

* Отсутствие файла и текстовых материалов:

Ответ 400 Bad Request с ошибкой: "Пожалуйста, загрузите файл лекции или введите материалы лекции."

* Некорректный файл (например, .exe):

Ответ 400 Bad Request

Выводы:

Эндпоинт корректно обрабатывает все предусмотренные сценарии загрузки данных, включая загрузку файлов и текстовых материалов. Возвращает ошибки при неправильных входных данных или отсутствии авторизации.

* Эндпоинт создания теста:

POST ${API\_URL}/create/test/

Цель: Генерация теста на основе материалов лекции или тем.

Формат ожидаемых данных:

method: строка, всегда "byThemes" или "general", указывает, создаётся тест по конкретным темам или общий тест.

title: строка, название теста.

lectureMaterials: строка, текстовые материалы для создания вопросов.

multipleChoiceCount: целое число, количество вопросов с одним правильным ответом.

openAnswerCount: целое число, количество открытых вопросов.

themes: список объектов, каждая тема должна содержать:

keyword: строка, ключевое слово темы.

sentences: список строк, предложения, относящиеся к теме.

multipleChoiceCount: целое число, количество MC вопросов для данной темы (только для `method: "byThemes").

openAnswerCount: целое число, количество открытых вопросов для данной темы (только для `method: "byThemes").

Результаты:

* Успешное создание теста:

Ответ 200 OK с сгенерированными вопросами, вариантами ответов и списком тем.

* Ошибка при отсутствии вопросов:

Ответ 400 Bad Request с сообщением: "Необходимо указать хотя бы одно количество вопросов."

* Ошибка при невозможности сгенерировать требуемое количество вопросов:

Ответ 400 Bad Request с сообщением о недостаточном количестве сгенерированных вопросов.

ЗАКЛЮЧЕНИЕ

В результате реализации проекта нами был разработан работоспособный прототип (MVP) сервиса, способного автоматически создавать тесты на основе материалов лекций. Этот инструмент позволяет преподавателям загружать имеющиеся материалы и генерировать тестовые задания. В процессе работы мы выявили проблемную область — использование ключевых слов. Ключевые слова недостаточно подходят для формирования ответов на вопросы, они лишь служат для разбивки текста. Вместо этого лучше использовать извлечение специализированных терминов. Это позволит предоставлять модели генерации вопросов предложения, содержащие только специализированную информацию, тем самым снижая долю нерелевантных данных при создании вопросов.

В перспективе планируется:

* Собрать датасет специализированных терминов и обучить модель для их извлечения из текста.
* Сохранять материалы лекций для пользователей и добавить возможность генерации тестов по нескольким материалам и темам.
* Выбрать, основываясь на сравнении [6], и дообучить модель генерации вопросов и ответов на датасете SberQuAD [7].

## БИБЛИОГРАФИЧЕСКИЙ СПИСОК

[1] Shasha Guo, Lizi Liao, Cuiping Li, Tat-Seng Chua. A Survey on Neural Question Generation: Methods, Applications, and Prospects. // arxiv : сайт. URL: <https://arxiv.org/pdf/2402.18267> (дата обращения: 18.12.2024).

[2] Lucas Bandarkar, Davis Liang, Benjamin Muller et al. The BELEBELE Benchmark: a Parallel Reading Comprehension Dataset in 122 Language Variants. // arxiv : сайт. URL: <https://arxiv.org/pdf/2308.16884> (дата обращения 18.12.2024).

[3] Elaf Alhazmi, Quan Z. Sheng, Wei Emma Zhang et al. Distractor Generation in Multiple-Choice Tasks: A Survey of Methods, Datasets, and Evaluation. // arxiv : сайт. URL: <https://arxiv.org/pdf/2402.01512> (дата обращения: 18.12.2024).

[4] Anna Glazkova, Dmitry Morozov. Exploring Fine-tuned Generative Models for Keyphrase Selection: A Case Study for Russian. // researchgate : сайт. – URL: <https://www.researchgate.net/publication/384085167_Exploring_Fine-tuned_Generative_Models_for_Keyphrase_Selection_A_Case_Study_for_Russian> (дата обращения: 11.12.2024).

[5] Anna Glazkova, Dmitry Morozov, Timur Garipov. Key Algorithms for Keyphrase Generation: Instruction-Based LLMs for Russian Scientific Keyphrases. // researchgate : сайт. – URL: <https://www.researchgate.net/publication/385177757_Key_Algorithms_for_Keyphrase_Generation_Instruction-Based_LLMs_for_Russian_Scientific_Keyphrases> (дата обращения: 11.12.2024).

[6] Dmitry Zmitrovich, Alexander Abramov, Andrey Kalmykov et al. A Family of Pretrained Transformer Language Models for Russian. // arxiv : сайт. – URL: <https://arxiv.org/pdf/2309.10931v3> (дата обращения: 18.12.2024)

[7] Pavel Efimov, Andrey Chertok, Leonid Boytsov, Pavel Braslavski. SberQuAD – Russian Reading Comprehension Dataset: Description and Analysis. // arxiv : сайт. – URL: <https://arxiv.org/pdf/1912.09723> (дата обращения: 18.12.2024).

## ПРИЛОЖЕНИЕ 1. ПРОГРАММНЫЙ КОД МОДУЛЯ ИЗВЛЕЧЕНИЯ ТЕКСТА ИЗ МАТЕРИАЛОВ ЛЕКЦИЙ

import logging

import os

from typing import Optional

from docx import Document

from PyPDF2 import PdfReader

import ffmpeg

from pydub import AudioSegment

import torch

from transformers import WhisperProcessor, WhisperForConditionalGeneration

import librosa

AudioSegment.converter = "D:\\ffmpeg-master-latest-win64-gpl\\bin\\ffmpeg.exe"

logger = logging.getLogger(\_\_name\_\_)

logging.basicConfig(level=logging.INFO)

def extract\_text\_from\_pdf(file\_path: str) -> str:

try:

reader = PdfReader(file\_path)

text = ""

for page in reader.pages:

text += page.extract\_text() or ""

return text

except Exception as e:

logger.error(f"Ошибка при извлечении текста из PDF: {e}")

raise e

def extract\_text\_from\_docx(file\_path: str) -> str:

try:

doc = Document(file\_path)

full\_text = []

for para in doc.paragraphs:

full\_text.append(para.text)

return '\n'.join(full\_text)

except Exception as e:

logger.error(f"Ошибка при извлечении текста из DOCX: {e}")

raise e

def extract\_text\_from\_txt(file\_path: str) -> str:

try:

with open(file\_path, 'r', encoding='utf-8') as f:

return f.read()

except Exception as e:

logger.error(f"Ошибка при извлечении текста из TXT: {e}")

raise e

class AudioTranscription:

def \_\_init\_\_(self, model\_name: str = "openai/whisper-large-v3-turbo"):

try:

logger.info(f"Загрузка модели Whisper: {model\_name}")

self.processor = WhisperProcessor.from\_pretrained(model\_name)

self.model = WhisperForConditionalGeneration.from\_pretrained(model\_name)

self.device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")

self.model.to(self.device)

logger.info("Модель Whisper успешно загружена.")

except Exception as e:

logger.error(f"Ошибка при загрузке модели Whisper: {e}")

raise e

def extract\_audio\_from\_video(self, video\_path: str, audio\_path: str) -> None:

try:

logger.info(f"Извлечение аудио из видео: {video\_path} -> {audio\_path}")

(

ffmpeg

.input(video\_path)

.output(audio\_path, format='wav', acodec='pcm\_s16le', ac=1, ar='16k')

.overwrite\_output()

.run(quiet=True)

)

logger.info(f"Аудио успешно извлечено и сохранено в {audio\_path}")

except ffmpeg.Error as e:

logger.error(f"Ошибка при извлечении аудио из видео: {e.stderr.decode()}")

raise e

def convert\_audio\_to\_wav(self, input\_path: str, output\_path: str) -> None:

try:

logger.info(f"Конвертация аудио: {input\_path} -> {output\_path}")

audio = AudioSegment.from\_file(input\_path)

audio = audio.set\_frame\_rate(16000).set\_channels(1)

audio.export(output\_path, format="wav")

logger.info(f"Аудио успешно конвертировано и сохранено в {output\_path}")

except Exception as e:

logger.error(f"Ошибка при конвертации аудио: {e}")

raise e

def transcribe(self, audio\_path: str) -> str:

try:

logger.info(f"Транскрипция аудио файла: {audio\_path}")

audio, sampling\_rate = librosa.load(audio\_path, sr=16000)

chunk\_length\_s = 30

stride\_length\_s = 5

chunk\_length = chunk\_length\_s \* sampling\_rate

stride\_length = stride\_length\_s \* sampling\_rate

total\_length = len(audio)

chunks = []

start = 0

while start < total\_length:

end = start + chunk\_length

chunk = audio[start:end]

chunks.append(chunk)

start += chunk\_length - stride\_length

logger.info(f"Разделено на {len(chunks)} чанков по {chunk\_length\_s} секунд.")

transcriptions = []

for idx, chunk in enumerate(chunks):

logger.info(f"Транскрипция чанка {idx + 1}/{len(chunks)}")

try:

inputs = self.processor(chunk, sampling\_rate=16000, return\_tensors="pt", language='ru', task="transcribe")

input\_features = inputs.input\_features.to(self.device)

attention\_mask = inputs.attention\_mask.to(self.device) if 'attention\_mask' in inputs else None

if attention\_mask is not None:

generated\_ids = self.model.generate(input\_features, attention\_mask=attention\_mask)

else:

generated\_ids = self.model.generate(input\_features)

transcription = self.processor.batch\_decode(generated\_ids, skip\_special\_tokens=True)[0]

transcriptions.append(transcription)

except Exception as e:

logger.error(f"Ошибка при транскрипции чанка {idx + 1}: {e}")

continue

full\_transcription = " ".join(transcriptions)

logger.info("Транскрипция успешно завершена.")

return full\_transcription

except Exception as e:

logger.error(f"Ошибка при транскрипции аудио: {e}")

raise e

## ПРИЛОЖЕНИЕ 2. ПРОГРАММНЫЙ КОД МОДУЛЯ ИЗВЛЕЧЕНИЯ КЛЮЧЕВЫХ СЛОВ

from rutermextract import TermExtractor

from natasha import Doc, Segmenter, NewsEmbedding, MorphVocab, NewsMorphTagger

import nltk

from nltk.corpus import stopwords

from typing import List, Optional

nltk.download('stopwords')

class KeywordsExtractor:

def \_\_init\_\_(self, top\_n: int = 10):

self.term\_extractor = TermExtractor()

self.stop\_words = set(stopwords.words('russian'))

self.segmenter = Segmenter()

self.emb = NewsEmbedding()

self.morph\_vocab = MorphVocab()

self.morph\_tagger = NewsMorphTagger(self.emb)

self.top\_n = top\_n

def extract\_keywords(self, text: str, top\_n: Optional[int] = None) -> List[str]:

top\_n = top\_n or self.top\_n

keywords = self.term\_extractor(text)[:top\_n]

return [keyword.normalized for keyword in keywords]

def tokenize\_lemmatize(self, text: str) -> str:

doc = Doc(text)

doc.segment(self.segmenter)

doc.tag\_morph(self.morph\_tagger)

tokens = []

for token in doc.tokens:

if token.pos in {'PUNCT', 'SYM'}:

continue

token.lemmatize(self.morph\_vocab)

if token.lemma:

tokens.append(token.lemma.lower())

else:

tokens.append(token.text.lower())

return ' '.join(tokens)

def extract\_sentences\_with\_keyword(self, text: str, keyword\_lemmatized: str) -> List[str]:

sentences\_with\_keyword = []

doc = Doc(text)

doc.segment(self.segmenter)

doc.tag\_morph(self.morph\_tagger)

for sent in doc.sents:

lemmatized\_sent = self.tokenize\_lemmatize(sent.text)

if keyword\_lemmatized in lemmatized\_sent:

sentences\_with\_keyword.append(sent.text)

return sentences\_with\_keyword

## ПРИЛОЖЕНИЕ 3. ПРОГРАММНЫЙ КОД МОДУЛЯ ГЕНЕРАЦИИ ВОПРОСОВ ТЕКСТУ

from gensim.models import KeyedVectors

from natasha import Doc, Segmenter, NewsEmbedding

import nltk

from nltk.corpus import stopwords

import torch

from transformers import AutoTokenizer, T5ForConditionalGeneration

from typing import List, Optional, Dict

from scipy.spatial.distance import cosine

import numpy as np

import logging

import re

import requests

import json

from dotenv import load\_dotenv

import os

nltk.download('stopwords')

logging.basicConfig(level=logging.INFO)

logger = logging.getLogger(\_\_name\_\_)

load\_dotenv()

oauth\_url = 'https://ngw.devices.sberbank.ru:9443/api/v2/oauth'

models\_url = 'https://gigachat.devices.sberbank.ru/api/v1/models'

completions\_url = 'https://gigachat.devices.sberbank.ru/api/v1/chat/completions'

uid = os.getenv('UID')

client\_base64 = os.getenv('CLIENT\_BASE64')

prompt = (

'На основе предоставленного текста и правильного ответа, составь 3 неправильных варианта ответа (дистракторы), не надо добавлять ещё какой-то текст или нумеровать неправильные варианты. Твой ответ должен содеражать только 3 неправильных варианта ответа '

'разделённых точкой с запятой. Текст: {text}. '

'Правильный ответ: {correct\_answer}'

)

def get\_access\_token():

headers = {

'Content-Type': 'application/x-www-form-urlencoded',

'Accept': 'application/json',

'RqUID': uid,

'Authorization': 'Basic ' + client\_base64

}

payload = 'scope=GIGACHAT\_API\_PERS'

response = requests.post(oauth\_url, headers=headers, data=payload, verify=".\\russian\_trusted\_root\_ca.cer")

return response.json().get('access\_token')

def parse\_distractors(response\_content: str) -> List[str]:

distractors = []

response\_content = re.sub(r'Неправильные варианты ответа[:\-]?\s\*', '', response\_content, flags=re.IGNORECASE)

response\_content = re.sub(r'3 неправильных варианта ответа[:\-]?\s\*', '', response\_content, flags=re.IGNORECASE)

numbered\_distractors = re.findall(r'\d+[.)]\s\*([^;]+)', response\_content)

if numbered\_distractors and len(numbered\_distractors) >= 3:

distractors = [distractor.strip() for distractor in numbered\_distractors[:3]]

logger.info("Дистракторы успешно распознаны с нумерацией.")

return distractors

split\_distractors = [distractor.strip() for distractor in response\_content.split(';') if distractor.strip()]

if len(split\_distractors) >= 3:

distractors = split\_distractors[:3]

logger.info("Дистракторы успешно распознаны без нумерации.")

return distractors

additional\_split = re.split(r'\d+[.)]\s\*', response\_content)

additional\_split = [distractor.strip() for distractor in additional\_split if distractor.strip()]

if len(additional\_split) >= 3:

distractors = additional\_split[:3]

logger.info("Дистракторы успешно распознаны при разделении по нумерации без точек с запятой.")

return distractors

sentences = re.split(r'[.!?]\s\*', response\_content)

sentences = [sentence.strip() for sentence in sentences if sentence.strip()]

if len(sentences) >= 3:

distractors = sentences[:3]

logger.warning("Дистракторы распознаны по предложениям, так как другие методы не сработали.")

return distractors

logger.error("Не удалось распознать дистракторы из ответа.")

return distractors

def generate\_distractors(access\_token: str, text: str, correct\_answer: str) -> List[str]:

if not access\_token:

logger.error("Отсутствует токен доступа OAuth.")

return []

formatted\_prompt = prompt.format(text=text, correct\_answer=correct\_answer)

payload = json.dumps({

"model": "GigaChat",

"messages": [

{

"role": "user",

"content": formatted\_prompt

}

],

"stream": False,

"repetition\_penalty": 1

})

headers = {

'Content-Type': 'application/json',

'Accept': 'application/json',

'Authorization': 'Bearer ' + access\_token

}

try:

response = requests.post(

completions\_url,

headers=headers,

data=payload,

verify=".\\russian\_trusted\_root\_ca.cer"

)

response.raise\_for\_status()

response\_json = response.json()

logger.info("Ответ от GigaChat получен успешно.")

except requests.exceptions.RequestException as e:

logger.error(f"Ошибка при обращении к GigaChat API: {e}")

return []

if "choices" not in response\_json or not response\_json["choices"]:

logger.error("Ответ от GigaChat не содержит необходимых данных.")

return []

response\_content = response\_json["choices"][0]["message"]["content"]

logger.debug(f"Содержимое ответа от GigaChat: {response\_content}")

distractors = parse\_distractors(response\_content)

if len(distractors) < 3:

additional\_distractors = [distractor.strip() for distractor in response\_content.split(';') if distractor.strip()]

for distractor in additional\_distractors:

if distractor not in distractors:

distractors.append(distractor)

if len(distractors) == 3:

break

if len(distractors) < 3:

return []

logger.info(f"Дистракторы успешно сгенерированы: {distractors}")

return distractors

class QAGenerator:

def \_\_init\_\_(

self,

question\_model\_path: str = 'question\_generation\_model',

answer\_model\_path: str = 'final\_model',

tokenizer\_model: str = "ai-forever/ruT5-base",

answer\_doublication: int = 25,

distractor\_spread: int = 10,

fractions: List[float] = [1/3, 2/3, 1.0]

):

self.device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")

self.tokenizer = AutoTokenizer.from\_pretrained(tokenizer\_model)

self.question\_model = T5ForConditionalGeneration.from\_pretrained(question\_model\_path).to(self.device)

self.answer\_model = T5ForConditionalGeneration.from\_pretrained(answer\_model\_path).to(self.device)

self.access\_token = get\_access\_token()

self.answer\_doublication = answer\_doublication

self.distractor\_spread = distractor\_spread

self.fractions = fractions

self.segmenter = Segmenter()

self.stop\_words = set(stopwords.words('russian'))

self.punct = {',', '?', '!', '.', ';', ':', '...', '(', ')', '\'', '\"', '«', '»'}

self.emb\_model = self.\_load\_embedding\_model()

def \_load\_embedding\_model(self) -> KeyedVectors:

news\_emb = NewsEmbedding()

return self.\_as\_gensim(news\_emb)

@staticmethod

def \_as\_gensim(navec\_model) -> KeyedVectors:

model = KeyedVectors(vector\_size=navec\_model.pq.dim)

weights = navec\_model.pq.unpack()

model.add\_vectors(navec\_model.vocab.words, weights)

return model

def get\_sentence\_embedding(self, text: str) -> Optional[np.ndarray]:

tokens = self.tokenize(text)

filtered\_tokens = self.model\_words\_filter(tokens)

if not filtered\_tokens:

return None

vectors = [self.emb\_model[token] for token in filtered\_tokens if token in self.emb\_model]

if not vectors:

return None

embedding = np.mean(vectors, axis=0)

return embedding

def is\_similar(self, existing\_embeddings: List[np.ndarray], new\_embedding: np.ndarray, threshold: float = 0.8) -> bool:

for emb in existing\_embeddings:

similarity = 1 - cosine(emb, new\_embedding)

if similarity >= threshold:

return True

return False

def tokenize(self, text: str) -> List[str]:

doc = Doc(text)

doc.segment(self.segmenter)

return [token.text.lower() for token in doc.tokens]

def model\_words\_filter(self, tokens: List[str]) -> List[str]:

return [token for token in tokens if token in self.emb\_model and token not in self.stop\_words and token not in self.punct]

def gen\_distractors(self, text: str, answer: str):

result = generate\_distractors(self.access\_token, text, answer)

if isinstance(result, list):

return result

else:

return []

def gen\_answer(self, text: str, question: str) -> str:

input\_text = f"{question} {text}"

input\_ids = self.tokenizer(

input\_text,

return\_tensors="pt",

max\_length=512,

truncation=True,

padding="max\_length",

).input\_ids.to(self.device)

answer\_ids = self.answer\_model.generate(

input\_ids, max\_length=128, num\_beams=5, early\_stopping=True

)

correct\_answer = self.tokenizer.decode(

answer\_ids[0], skip\_special\_tokens=True

)

return correct\_answer

def gen\_questions(self, text: str, num\_questions: int, question\_type: str = 'mc') -> List[str]:

input\_ids = self.tokenizer(

text,

return\_tensors="pt",

max\_length=512,

truncation=True,

padding="max\_length",

).input\_ids.to(self.device)

question\_ids = self.question\_model.generate(

input\_ids,

max\_length=64,

num\_beams=12,

num\_return\_sequences=num\_questions,

temperature=1.2,

top\_p=0.9,

early\_stopping=True,

)

questions = [

self.tokenizer.decode(q, skip\_special\_tokens=True) for q in question\_ids

]

return questions

def generate\_qa\_pairs(self, text: str, num\_questions: int = 10, question\_type: str = 'mc') -> List[Dict[str, Optional[List[str]]]]:

if question\_type not in ['mc', 'open']:

raise ValueError(f"Неизвестный тип вопроса: {question\_type}")

questions = self.gen\_questions(text, num\_questions, question\_type)

qa\_pairs = []

for quest in questions:

answer = self.gen\_answer(text, quest)

if question\_type == 'mc':

distractors = self.gen\_distractors(text, answer)

else:

distractors = None

qa\_pair = {

"type": question\_type,

"Вопрос": quest,

"Ответ": answer,

"Дистракторы": distractors

}

qa\_pairs.append(qa\_pair)

return qa\_pairs

1. Без учёта ответов с неверным форматом. [↑](#footnote-ref-0)