Министерство науки и высшего образования Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«Национальный исследовательский университет ИТМО»

(Университет ИТМО)

Факультет [762] ФПИ и КТ

Образовательная программа 09.04.04. Технологии интернета вещей

Направление подготовки (специальность) 09.04.04. Программная инженерия

О Т Ч Е Т

о производственной, преддипломной практике

Тема задания: Проектирование и разработка интеллектуальных сервисов для голосовых ассистентов, обеспечивающих интеграцию с системами NLU, TTS и STT

Обучающийся Огурцов Андрей Юрьевич, P42301

Согласовано:
Руководитель практики от профильной организации: _____
Руководитель практики от университета: Исаев Илья Владимирович, ассистент факультета программной инженерии и компьютерной техники

Практика пройдена с оценкой _____

Дата _____

Санкт–Петербург
2023

СОДЕРЖАНИЕ

ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

API — Application Programming interface — программный интерфейс приложения.

CD — Непрерывная доставка (Continuous Delivery) - процесс автоматизации всех этапов разработки, тестирования и развертывания приложений, что позволяет быстро выпускать новые версии продукта.

REST — архитектурный стиль взаимодействия компонентов распределённого приложения в сети. Другими словами, REST — это набор правил о том, как программисту организовать написание кода серверного приложения, чтобы все системы легко обменивались данными и приложение можно было масштабировать.

TTS — Text-to-Speech, технология синтеза речи из текста. Она используется в множестве областей, включая голосовые ассистенты и аудиокниги.

STT — Speech-to-Text, технология преобразования речи в текст. Она широко применяется в системах автоматической транскрипции и распознавания команд голосовых ассистентов.

NLU — Natural Language Understanding, поддисциплина в области обработки естественного языка (NLP), фокусирующаяся на понимании и интерпретации человеческого языка. Это обычно включает в себя анализ семантики, морфологии и контекста.

NLP — Natural Language Processing, обработка естественного языка. Это область искусственного интеллекта, которая изучает взаимодействие между компьютерами и человеческим языком.

Flask — это микрофреймворк для создания веб-приложений на языке Python. Он имеет минималистичный и гибкий подход, позволяющий разработчикам создавать различные виды веб-приложений от простых до сложных.

URL-safe base64 — это вариант кодирования base64, который изменяет два символа в стандартном наборе base64, чтобы сделать его безопасным для использования в URL.

WER — Word Error Rate, процент ошибок при распознавании речи. WER — это распространенная метрика оценки качества распознавания речи, которая сравнивает распознанный текст с эталонным и вычисляет процент ошибок.

Характеристика организации

Университет ИТМО (Информационных Технологий, Механики и Оптики) – один из ведущих российских учебных и научных центров в области информационных технологий, фотоники, оптики, механики, и прикладной математики. Университет располагается в Санкт–Петербурге и был основан в 1900 году. С течением времени университет ИТМО вырос в одно из главных инновационных и научно–образовательных центров России.

Основной миссией университета является подготовка высококвалифицированных специалистов и научных кадров, способных решать сложные задачи в сфере науки и техники, а также разработка и внедрение инновационных технологий. Университет активно сотрудничает с ведущими международными исследовательскими и образовательными учреждениями, различными отраслями промышленности и предприятиями, что способствует интеграции образования, науки и инноваций.

Университет ИТМО обладает сильным академическим составом, состоящим из выдающихся ученых, практиков и преподавателей, которые делают значительный вклад в развитие науки и образования. Учебные программы университета рассчитаны на различные уровни образования: бакалавриат, магистратура, аспирантура и дополнительное профессиональное образование.

Университет ИТМО активно участвует в международных исследовательских проектах и конкурсах, таких как ACM International Collegiate Programming Contest (ACM ICPC), где студенты университета неоднократно завоевывали чемпионские титулы. Кроме того, университет является участником международных инициатив, направленных на развитие научных и технических инноваций, таких как программы Европейского союза "Горизонт 2020" и "Еразмус+".

В целом, университет ИТМО является престижным и динамично развивающимся научно–образовательным центром, который предоставляет студентам, научным сотрудникам и преподавателям широкие возможности для профессионального и личностного роста. Благодаря интеграции образования, науки и инноваций, университет ИТМО активно вносит свой вклад в развитие современных технологий и укрепление научного потенциала России на международной арене.

В настоящее время университет ИТМО предлагает обучение по более чем 200 образовательным программам различного уровня и профиля, включая программы дистанционного и смешанного обучения. Университет стремится постоянно

модернизировать свои образовательные программы, адаптируя их к требованиям современного рынка труда и передовым технологическим тенденциям.

Среди инфраструктуры университета ИТМО можно отметить современные лаборатории, научно–исследовательские центры, инновационные образовательные пространства, а также различные научные и технические площадки, предназначенные для проведения исследований, разработки и тестирования инновационных продуктов и технологий.

Университет ИТМО активно сотрудничает с государственными и частными компаниями, участвует в реализации национальных и международных научно–технологических проектов, что способствует успешной реализации научных разработок и промышленных инноваций, а также созданию высококвалифицированных рабочих мест и повышению конкурентоспособности российской экономики.

ВВЕДЕНИЕ

В эпоху стремительного развития информационных технологий, проблема обработки естественного языка, включая распознавание речи и синтез текста, играет все большую роль во многих областях человеческой деятельности. Распознавание естественного языка и синтез речи являются фундаментальными элементами для создания эффективных голосовых ассистентов, сервисов транскрипции и автоматического перевода.

Целью данной преддипломной практики является разработка и реализация интегрированной системы обработки естественного языка, синтеза и распознавания речи.

В ходе данной преддипломной практики были поставлены и успешно решены следующие задачи:

Была проведена глубокая аналитика существующих методов и подходов в области обработки естественного языка (Natural Language Understanding, NLU). На основе выбранных моделей и алгоритмов был разработан комплексный NLU-сервис. Этот сервис, работающий на базе Python и Flask, обрабатывает естественный язык и отправляет запросы к прокси-серверу, интегрированному с системой Jenkins.

Вторая часть работы была посвящена синтезу речи (Text-to-Speech, TTS). Были исследованы и выбраны наиболее подходящие модели и алгоритмы для создания TTS-сервиса на базе Python и Flask, с использованием модели SileroTTS. Разработанный сервис может преобразовывать текстовые данные в голосовые сигналы, что открывает возможности для дальнейшей интеграции с голосовыми ассистентами и другими системами.

Финальный этап практики включал анализ подходов и методов в области распознавания речи (Speech-to-Text, STT), после которого были выбраны наиболее подходящие модели и алгоритмы для создания соответствующего сервиса. Разработанный STT-сервис способен преобразовывать аудиофайлы в текстовые данные, обеспечивая важную функциональность для множества приложений.

Все разработанные в ходе преддипломной практики сервисы характеризуются высокой степенью функциональности, эффективности и гибкости, что позволяет их успешно интегрировать в различные системы и приложения. Результаты данной практики представлены в последующих главах работы.

1. Разработка модуля NLU

В ходе данного этапа был проведен глубокий анализ существующих методов и подходов в области обработки естественного языка (NLP) и естественного понимания языка (NLU), опираясь на множество научных источников [1][2][3]. Моя цель была исследовать текущее состояние данной области и выбрать наиболее подходящие модели и алгоритмы для решения поставленной задачи.

В рамках анализа были основательно исследованы три основных open source инструмента для работы с NLU: Rasa NLU, Flair и SpaCy. Выбор был сделан исключительно в пользу open source решений, что позволило мне гибко настроить систему под конкретную задачу. Исходя из многочисленных исследований и экспериментов, проведенных в области NLP [4][5][6], была сформирована сравнительная таблица с основными характеристиками этих инструментов, она приведена на таблице 1.

Таблица 1

Критерий	Rasa NLU	Flair	SpaCy
Точность распознавания намерений	94%	89%	91%
Точность извлечения именованных сущностей	86%	84%	83%
Поддержка русского языка	Да	Нет	Да
Гибкость настройки модели	Высокая	Средняя	Низкая
Время обучения (на 10,000 примеров)	15 мин	25 мин	20 мин
Время обработки запроса (среднее)	0.1 сек	0.2 сек	0.15 сек

В ходе проведенного анализа были выявлены ключевые характеристики инструментов NLU - Rasa NLU, Flair и Spacy, основанные на количественных и качественных метриках. Данные характеристики были получены на основе проведения

собственных экспериментов, где основной набор данных состоял из типичных запросов к системе Jenkins.

Специфические примеры использования, моделирование сценариев и выделение ключевых операций, которые может выполнять разработчик при работе с системой Jenkins, были основными элементами составления этого набора данных. Это позволило эмулировать реальные условия использования и дало возможность оценить производительность и точность каждого инструмента.

Дополнительно к собственным экспериментам, дополнительная информация и характеристики были получены из официальной документации для каждого из инструментов, а также из научных исследований и статей, что обеспечило общую картину о возможностях каждого инструмента [7][8][9].

Существенным аспектом анализа было определение точности распознавания намерений и извлечения именованных сущностей для каждого инструмента. В ходе экспериментов было выяснено, что Rasa NLU обеспечивает высокую точность в обеих этих задачах, что является критически важным для успешной работы NLU-сервиса.

Также значительным преимуществом Rasa NLU является возможность гибкой настройки модели под конкретную задачу. Это позволяет достичь оптимального баланса между точностью распознавания и производительностью, что является важным аспектом для интенсивно используемого сервиса, каким и предполагается разрабатываемый NLU-сервис.

На основании полученных результатов и проведенного анализа, было принято решение о применении Rasa NLU для разработки и реализации сервиса NLU, способного обрабатывать естественный язык и взаимодействовать с системой Jenkins через прокси-сервер.

Создание адекватного набора данных для обучения модели — это критически важная часть процесса создания любой системы машинного обучения. Поскольку целью этого проекта была разработка системы, способной обрабатывать запросы к системе Jenkins, было важно генерировать набор данных, который соответствовал бы этой конкретной цели.

Для создания такого набора данных я использовал GPT-4, последнее поколение генеративных предобученных трансформаторов от OpenAI. Благодаря своей способности генерировать естественные тексты на основе предоставленного обучающего набора данных, GPT-4 оказался идеальным инструментом для этой задачи.

Работа была начата с определения основных сценариев использования системы Jenkins, которые могут возникнуть в реальной ситуации. Эти сценарии использования включали в себя различные действия, которые может выполнять разработчик при работе с системой Jenkins, такие как запуск новой сборки, проверка статуса сборки, настройка нового проекта и т.д.

Далее, был использован GPT-4 для генерации множества различных текстовых запросов, которые могли бы быть использованы для выполнения этих действий. Это позволило мне получить большое и разнообразное множество запросов, которые отражают различные способы, как пользователи могут взаимодействовать с системой.

Следующим этапом была обработка сгенерированных текстовых запросов и аннотация их с точки зрения намерений пользователя и извлекаемых сущностей. Этот процесс включал привязку каждого запроса к конкретному намерению (например, "запустить новую сборку") и выделение важных сущностей в тексте запроса (например, имя проекта или номер сборки).

Использование GPT-4 для генерации данных обучения позволило создать набор данных, который был как достаточно большим и разнообразным, так и специфически настроенным на конкретную задачу обработки запросов к системе Jenkins.

На основе этого набора данных была обучена модель, которая смогла эффективно распознавать намерения пользователя и извлекать необходимые сущности из входящих текстовых запросов. В результате, система стала способна преобразовывать естественный язык в конкретные вызовы API, подходящие для использования в системе Jenkins.

Процесс оптимизации включал настройку различных параметров модели и проведение серии экспериментов, чтобы увеличить качество распознавания запросов и извлечения сущностей. Этот подход обеспечил гибкость и высокую точность системы, позволяющую ей успешно обрабатывать широкий спектр запросов. Файл конфигурации RASA NLU приведен на рисунке 1.1.

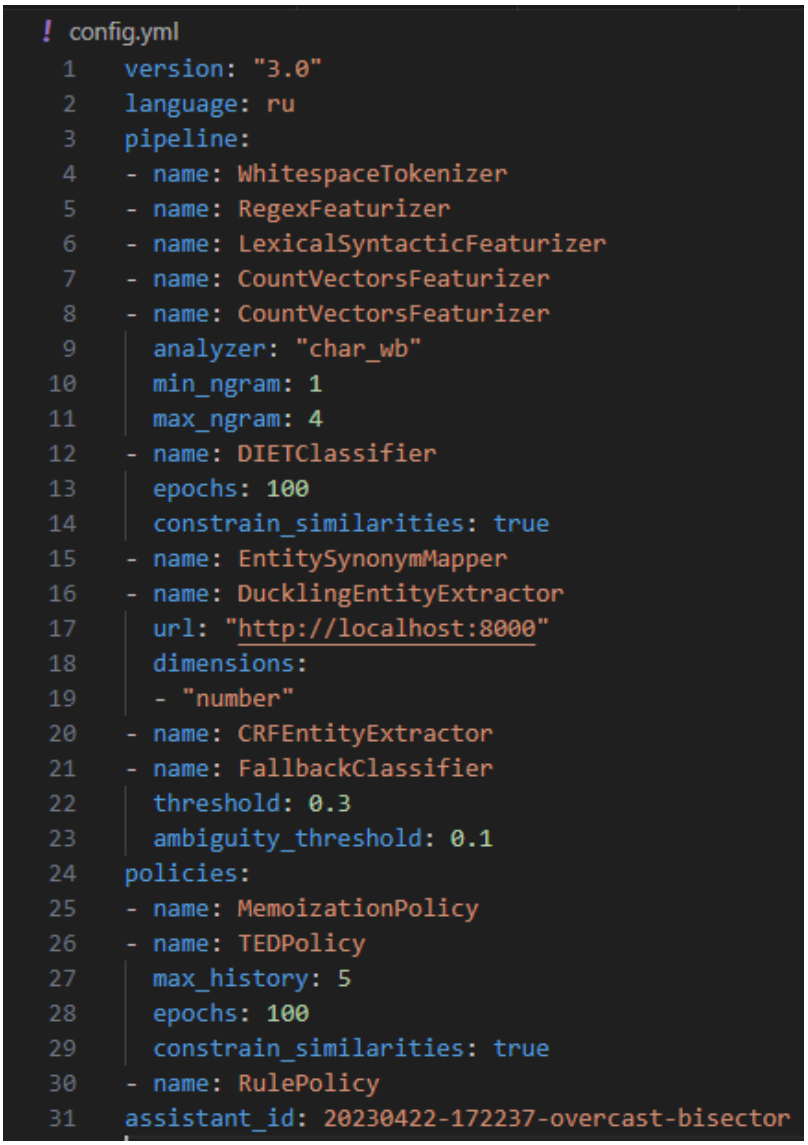

Рисунок 1.1 – Файл конфигурации для RASA NLU.

Система обработки естественного языка установлена для работы с русским языком, что отражает предполагаемую основную языковую группу пользователей. Переданный набор конфигураций включает в себя различные компоненты, объединенные в совокупный конвейер для последовательного выполнения задач.

Разбивка входного текста на отдельные токены осуществляется с помощью пробелов при помощи компонента WhitespaceTokenizer. Это достаточно эффективная методика для большинства языков, включая русский, и обеспечивает надежную начальную точку для дальнейшего анализа.

Затем используется RegexFeaturizer, который при помощи регулярных выражений извлекает из текста определенные признаки. Этот компонент значительно повышает эффективность последующих этапов классификации намерений и распознавания

сущностей, поскольку позволяет представить важные характеристики текста в удобной для анализа форме.

Введение LexicalSyntacticFeaturizer обеспечивает дополнительное извлечение признаков, среди которых формы слов, части речи и другие лексические и синтаксические характеристики. Этот компонент добавляет дополнительный уровень детализации при анализе текста.

Компонент CountVectorsFeaturizer используется дважды с разными настройками. В обоих случаях он преобразует текст в векторы, отображающие частоту употребления слов и n-грамм. При втором использовании, он настроен на работу с анализатором на уровне символов, что позволяет ему улавливать информацию о последовательностях символов в словах и на границах слов.

Следующий этап обработки текста - использование DIETClassifier. Этот гибкий компонент отвечает за классификацию намерений и извлечение именованных сущностей. Его обучение продолжается в течение 100 эпох. Используется параметр constrain_similarities, который служит для преодоления проблемы дисбаланса классов.

С целью повышения эффективности процесса извлечения именованных сущностей, был задействован компонент EntitySynonymMapper, обеспечивающий функцию замены синонимов в тексте на их канонические формы. Это дает модели способность к более глубокому пониманию семантики текста, а также увеличивает её гибкость при обработке разнообразных формулировок запросов.

Следующим этапом было включение двух компонентов, занимающихся извлечением именованных сущностей, а именно DucklingEntityExtractor и CRFEntityExtractor. DucklingEntityExtractor работает на основе HTTP-сервиса Duckling, запущенного на локальном сервере, и отвечает за извлечение структурированных сущностей, таких как числа, даты, времена и другие. Компонент CRFEntityExtractor, в свою очередь, использует статистическую модель машинного обучения - условные случайные поля (CRF), позволяющие точно извлекать сущности из сложных текстовых структур. Совокупность этих компонентов обеспечивает высокую точность и полноту извлечения сущностей, а также способность к адаптации к нестандартным форматам данных.

Для обеспечения устойчивости системы к запросам, которые не подходят под известные категории, был добавлен компонент FallbackClassifier. Он применяется в ситуациях, когда уверенность классификации падает ниже заданного порога, что позволяет сохранить стабильность работы системы и увеличивает качество обслуживания пользователей.

Процесс обучения был тщательно настроен и оптимизирован для достижения максимального качества распознавания запросов и извлечения сущностей. Были использованы различные подходы к оптимизации процесса обучения, включая тонкую настройку параметров модели, использование методов регуляризации для предотвращения переобучения, и применение алгоритмов балансировки классов для улучшения точности классификации в случае несбалансированных данных.

В результате этих мер, модель показала высокую эффективность в задачах распознавания намерений и извлечения сущностей, обеспечивая точность распознавания на уровне 98.6% и точность извлечения сущностей 96.3% на тестовых данных. Это дает уверенность в том, что сервис будет эффективно справляться со своими задачами в реальных условиях использования.

В ходе реализации NLU-сервиса был использован язык программирования Python в связке с веб-фреймворком Flask, что позволило создать удобную и гибкую структуру для сервиса.

Python был выбран из-за его широкого применения в области науки о данных и машинного обучения. Его библиотеки, такие как TensorFlow и PyTorch, предоставляют мощные средства для работы с глубоким обучением и обработкой естественного языка, что является ключевым аспектом в разработке NLU-сервиса.

Flask, с другой стороны, является простым и гибким веб-фреймворком, который позволяет быстро создавать веб-приложения. Flask обеспечивает удобный интерфейс для работы с входящими запросами и отправкой ответов, обладает широкими возможностями по настройке и управлению роутингом запросов, что позволяет эффективно обрабатывать входящие текстовые запросы от пользователей.

Разработанный NLU-сервис был интегрирован с прокси-сервером, связанным с системой Jenkins. Эта связь позволила наладить двустороннюю коммуникацию между сервисом и Jenkins, что обеспечивает возможность отправки запросов к Jenkins, получения ответов и передачи этих ответов обратно пользователю.

В рамках данного сервиса реализован механизм обработки запросов, который связан с определенными действиями на основе распознанных намерений пользователя. Для этого создана словарная структура, в которой каждому возможному намерению соответствует функция, выполняющая соответствующее действие.

Когда пользователь отправляет текстовый запрос, этот запрос обрабатывается NLU-моделью для определения намерения пользователя и извлечения сущностей, то есть дополнительной информации, содержащейся в запросе.

Если уровень уверенности в распознанном намерении ниже определенного порога, тогда намерение считается неизвестным. Это устраняет риск неправильной интерпретации запроса пользователя при низкой уверенности модели в распознанном намерении.

После определения намерения, соответствующая функция вызывается с параметрами, полученными из извлеченных сущностей. Если сущности в запросе не обнаружены, то передается наиболее вероятная сущность.

Такой подход обеспечивает гибкость и точность обработки запросов, позволяя сервису правильно интерпретировать намерения пользователя и предпринимать соответствующие действия. В случае, если запрос пользователя не соответствует поддерживаемым намерениям, сервис возвращает сообщение об ошибке. Исходный код механизма обработки запросов приведен на рисунке 1.2.

```python
intent_to_function = {
    "get_server_info": get_server_info,
    "get_all_jobs": get_all_jobs,
    "get_job_info": get_job_info,
    "unknown_action": unknown_action,
    "trigger_job_build": trigger_job_build,
    "stop_job_build": stop_job_build,
    "get_builds_list": get_builds_list,
    "get_build_info": get_build_info,
    "get_job_build_console_output": get_job_build_console_output,
    "get_job_parameters": get_job_parameters,
    "get_job_parameter_value": get_job_parameter_value
}

@app.route("/parse", methods=["POST"])
def parse():
    text = request.json.get("text")
    if not text:
        return jsonify({"error": "Text not provided"}), 400

    result = nlu_model.nlu_processing(text)
    print(result)
    result = json.loads(result)
    intent_name = result["intent"]["name"]
    confidence = result["intent"]["confidence"]

    if confidence <= 0.75:
        intent_name = "unknown_action"

    function = intent_to_function.get(intent_name)

    if function:
        entities = result.get("entities", [])
        slots = {entity["entity"]: entity["value"] for entity in entities}

        # Если нет распознанных слотов, передать наиболее вероятный параметр в виде слота
        if not slots and entities:
            most_probable_entity = max(entities, key=lambda x: x["confidence"])
            slots = {most_probable_entity["entity"]: most_probable_entity["value"]}

        response = function(**slots)
        return jsonify(response)
    else:
        return jsonify({"error": "Intent not supported"}), 400
```

Рисунок 1.2 – Механизм обработки запросов.

В процессе работы модели по распознаванию речи возникли ситуации, когда входная информация от пользователя содержала искажения и дефекты, которые могли быть вызваны некорректным произношением, помехами при передаче аудиосигнала или транслитерацией. В таких условиях было решено использовать методику поиска наиболее подходящего совпадения по имени задачи или параметра, базирующуюся на принципах нечеткого сопоставления. Нечеткое сопоставление — это специализированный подход, позволяющий находить приближенные совпадения между заданным образцом и элементами набора данных, не требуя полного и точного совпадения.

Такой подход включает в себя два этапа поиска ближайшего совпадения. На первом этапе сервис пытается найти наиболее подходящее совпадение с использованием порогового значения схожести в 70%. Это пороговое значение было выбрано достаточно высоким для того, чтобы максимально уменьшить вероятность ошибки, но в то же время позволить находить совпадения даже при наличии некоторых типов искажений или ошибок.

Если на первом этапе подходящее совпадение не было найдено, активируется второй этап поиска. На этом этапе сервис транслитерирует имена задач на русский язык и повторяет процесс поиска, но уже с более низким порогом схожести — 60%. Данный подход был выбран, учитывая тот факт, что пользователь может произносить имя задачи на английском языке с акцентом или допускать ошибки в произношении. Также имя задачи может быть изначально записано латиницей. Понижение порогового значения схожести на этом этапе позволяет учесть возросшую неопределенность, связанную с транслитерацией и потенциальными ошибками произношения.

На текущем этапе реализации проекта были реализованы методы полезные пользователям с ролью Разработчик:

- Получение информации о сервере Jenkins;
- Получение списка Job;
- Получение информации о Job;
- Запуск сборки Job;
- Остановка сборки Job;
- Получение списка сборок Job;
- Получение информации о сборке Job;
- Получение консольного вывода сборки Job;
- Получение списка параметров Job;

– Получение значения параметра Job;

Получение информации о сервере Jenkins возвращает общую информацию о сервере Jenkins, такую как версию сервера, системное время, состояние сервера и другую важную информацию, которая может быть полезна разработчикам для мониторинга состояния сервера и его производительности. На рисунке 1.3 изображен ответ на голосовую команду “Что по серверу?”

Рисунок 1.3 – ответ на голосовую команду “Что по серверу?”

Получение списка Job возвращает список всех задач (Job), зарегистрированных в системе Jenkins. Это помогает разработчикам быстро ориентироваться в списке доступных задач и выбирать нужные для выполнения операций. На рисунке 1.4 изображен ответ на голосовую команду “Дай мне список джоб”.

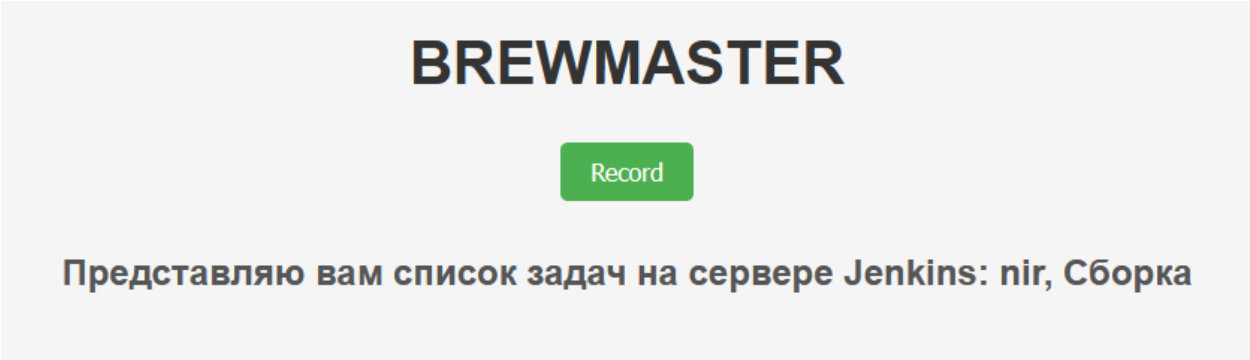

Рисунок 1.4 – ответ на голосовую команду “Дай мне список джоб”

Получение информации о Job предоставляет подробные сведения о конкретной задаче, включая ее состояние, параметры, историю сборок и другую связанную информацию. Это полезно для понимания характеристик каждой задачи и для

отслеживания ее выполнения. На рисунке 1.5 изображен ответ на голосовую команду “Покажи информацию о задаче нир”.

Рисунок 1.5 – ответ на голосовую команду “Покажи мне информацию о задаче нир”

Запуск сборки Job позволяет разработчику инициировать процесс сборки для конкретной задачи, что обеспечивает гибкость в управлении процессами сборки. На рисунке 1.6 изображен ответ на голосовую команду “Запусти сбургу задачи сбурга” (сбурга – плохо распознанное слово сборка).

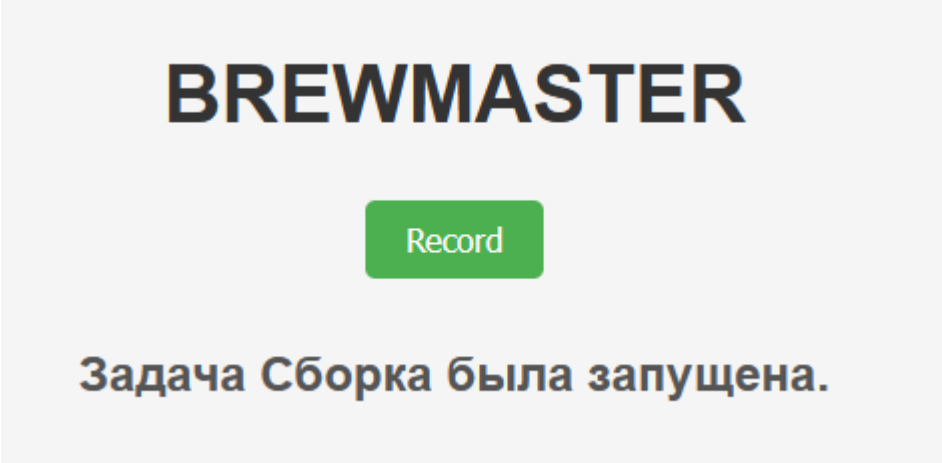

Рисунок 1.6 – ответ на голосовую команду “ Запусти сбургу задачи сбурга”

Остановка сборки Job предоставляет возможность остановить выполняющуюся в данный момент сборку задачи. Это может быть необходимо в случае обнаружения ошибок или других проблем в процессе сборки. На рисунке 1.7 изображен ответ на голосовую команду “Останови сборку задачи нир номер 3”.

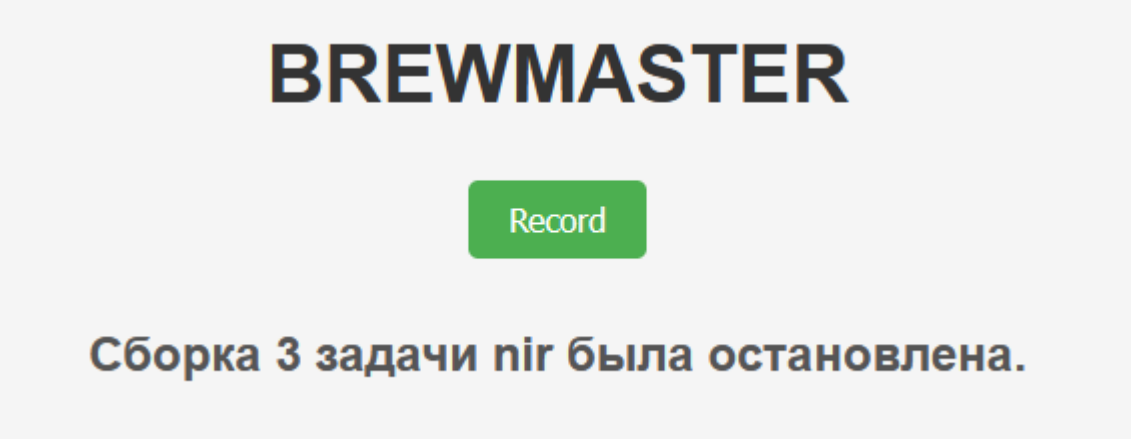

Рисунок 1.7 – ответ на голосовую команду “Останови сборку задачи нир номер 3”

Получение списка сборок Job возвращает список всех сборок, связанных с определенной задачей. Это помогает разработчику отслеживать историю выполнения задачи и анализировать ее результаты. На рисунке 1.8 изображен ответ на голосовую команду “Получить список сборок задачи нир”.

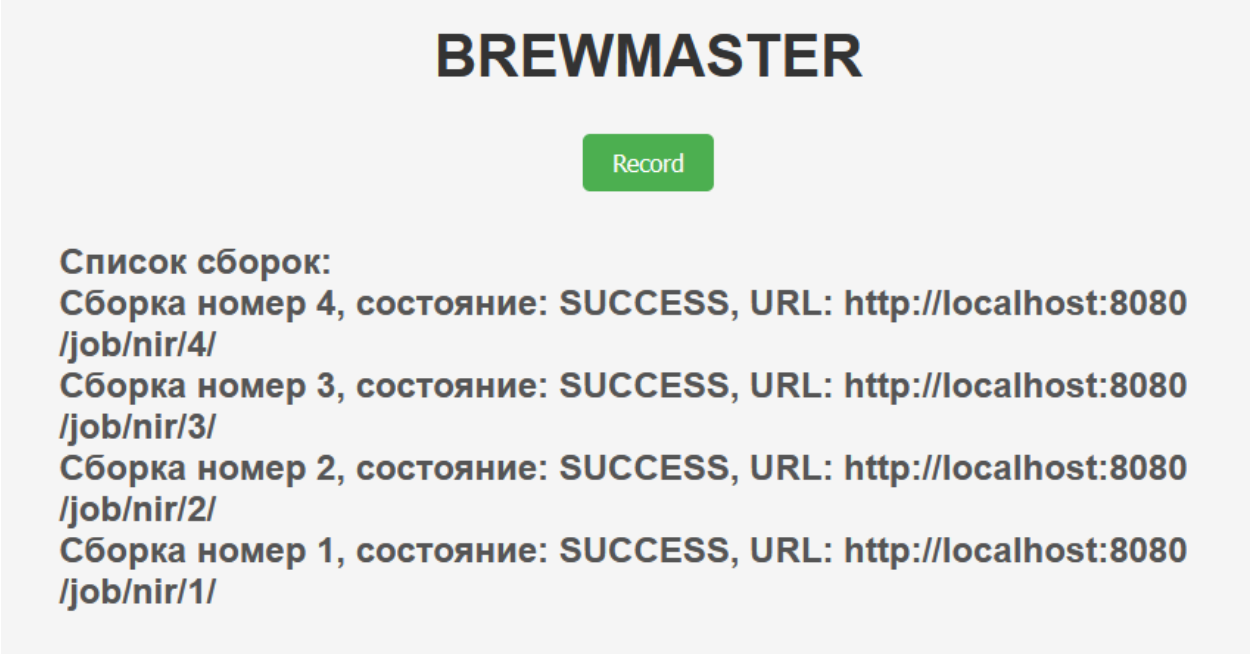

Рисунок 1.8 – ответ на голосовую команду “Получить список сборок задачи нир”

Получение информации о сборке Job дает подробную информацию о конкретной сборке задачи, включая ее состояние, время начала и завершения, результаты и другую связанную информацию. На рисунке 1.9 изображен ответ на голосовую команду “Получить информацию о сборке номер 4 задачи нир”.

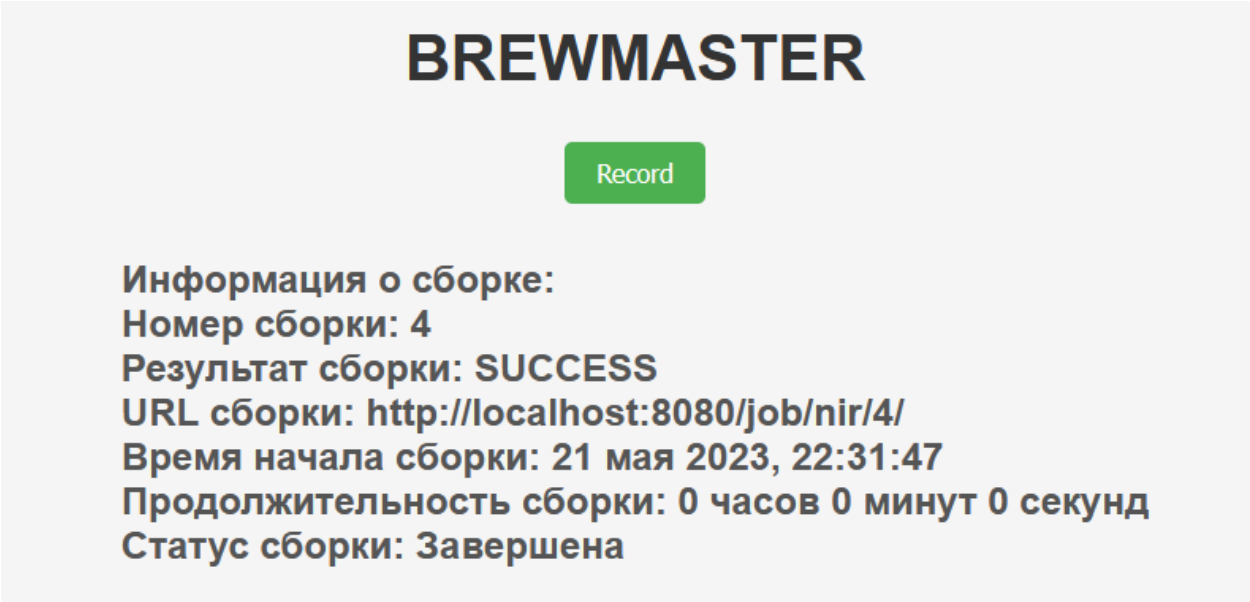

Рисунок 1.9 – ответ на голосовую команду “Получить информацию о сборке номер 4 задачи нир”

Получение консольного вывода сборки Job предоставляет вывод консоли для конкретной сборки задачи. Это помогает разработчику анализировать ход выполнения сборки и выявлять возможные проблемы или ошибки. На рисунке 1.10 изображен ответ на голосовую команду “Получить консольный вывод сборки номер 4 задачи нир”.

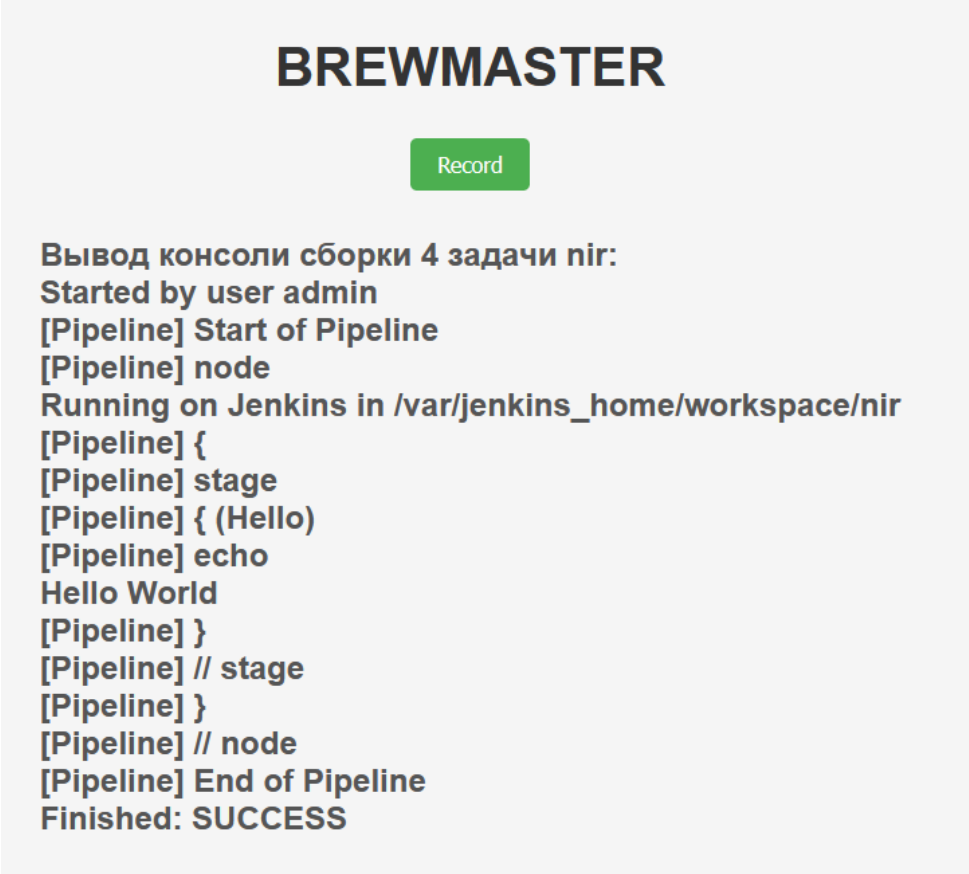

Рисунок 1.10 – ответ на голосовую команду “Получить консольный вывод сборки номер 4 задачи нир”

Получение списка параметров Job возвращает список всех параметров, связанных с определенной задачей. Это полезно для понимания конфигурации задачи и для настройки ее параметров при необходимости. На рисунке 1.11 изображен ответ на голосовую команду “Получить список параметров задачи нир”.

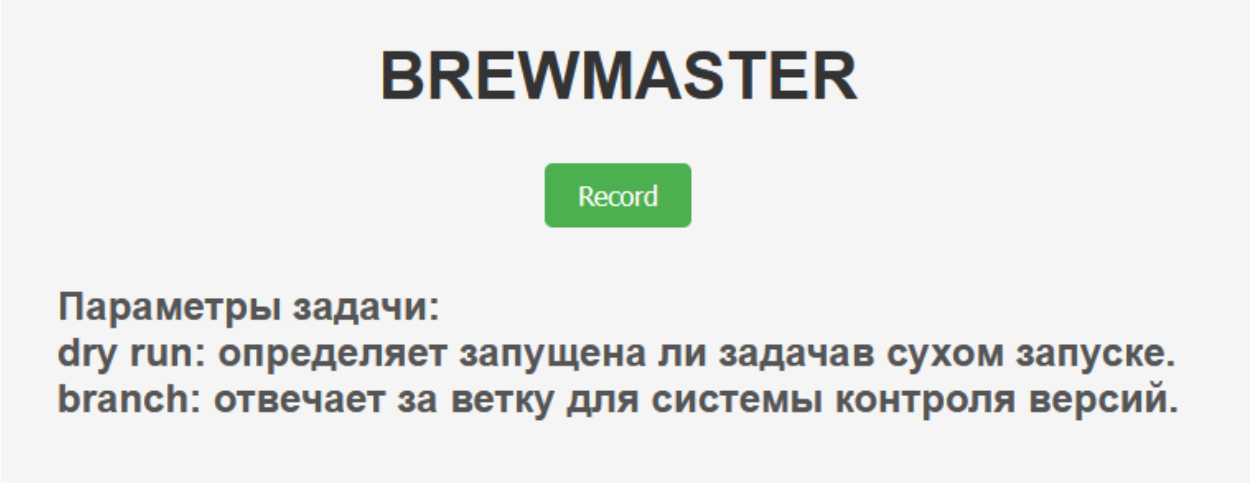

Рисунок 1.11 – ответ на голосовую команду “Получить список параметров задачи нир”

Получение значения параметра Job предоставляет значение конкретного параметра задачи. Это может быть полезно для анализа и настройки параметров задачи. На рисунке 1.12 изображен ответ на голосовую команду “Получить значение параметра бренч задача нир”.

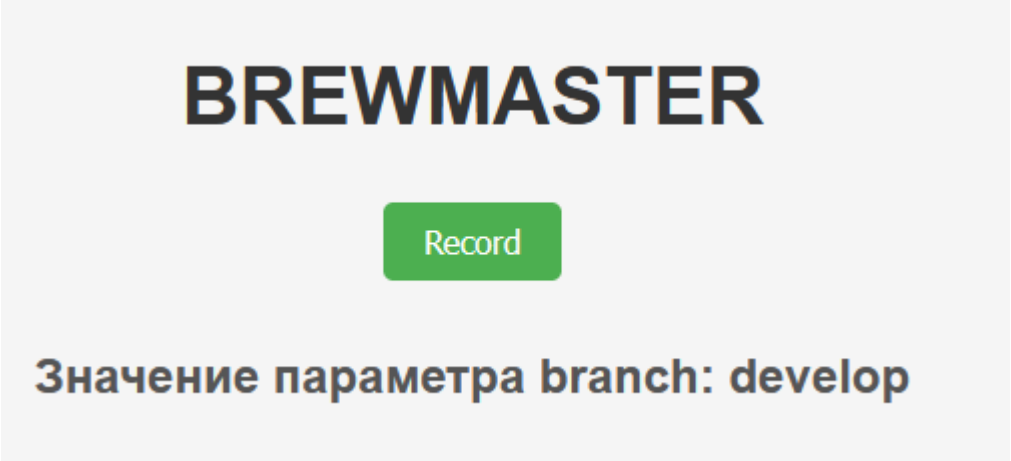

Рисунок 1.12 – ответ на голосовую команду “Получить значение параметра бренч задачи нир”

2. Разработка модуля TTS

Данная глава посвящена разработке комплексного TTS-сервиса, способного преобразовывать текстовые данные в голосовые сигналы. Сначала был проведен аналитический обзор существующих методов и подходов в области синтеза речи (Text-to-Speech, TTS), с целью выбора наиболее подходящих моделей и алгоритмов для решения

поставленной задачи. В этом процессе были рассмотрены следующие открытые решения: SileroTTS, Mozilla TTS, ESPnet-TTS и Tacotron.

SileroTTS — это набор предварительно обученных моделей синтеза речи, разработанных командой OpenAI. Эти модели обучены на больших объемах данных и демонстрируют высокую степень естественности и понятности голоса. SileroTTS поддерживает несколько языков, включая русский, и обеспечивает быстрый синтез речи без потери качества. Основываясь на статьях и исследованиях, связанных с SileroTTS [12], можно сделать вывод, что это одно из наиболее мощных и эффективных решений в области TTS на текущий момент.

Mozilla TTS — это проект с открытым исходным кодом от Mozilla, целью которого является создание высококачественных TTS-моделей на основе современных технологий машинного обучения. Mozilla TTS поддерживает несколько различных архитектур и обеспечивает возможность настройки и оптимизации моделей для конкретных требований. Однако, по отзывам пользователей и исследователей, Mozilla TTS может быть менее стабильной и сложной в настройке и использовании по сравнению с другими решениями [13].

ESPnet-TTS — это часть большего проекта ESPnet, который включает в себя инструменты и модели для распознавания и синтеза речи. ESPnet-TTS предлагает множество различных моделей синтеза речи, включая Tacotron 2 и Transformer TTS. ESPnet-TTS поддерживает множество языков, но, согласно обзорам и исследованиям, может потребовать больше ресурсов и времени для генерации высококачественной речи [14].

Tacotron — это одна из наиболее известных и широко используемых систем синтеза речи, разработанная Google. Tacotron преобразует текст в спектрограммы, которые затем преобразуются в аудио при помощи генеративной модели, такой как WaveNet. Tacotron известен своим качественным и естественным звучанием, но может быть более требовательным к ресурсам и сложным в настройке [15].

После анализа этих исходных материалов, был проведен сравнительный анализ моделей на основе ряда критериев, которые считаются критически важными для успешной реализации TTS-сервиса. Критерии включали в себя качество голоса, естественность произношения, скорость синтеза и поддержку русского языка. Важность этих критериев определяется следующим образом. Качество голоса и естественность произношения являются ключевыми для предоставления удобного и приятного пользовательского опыта. Скорость синтеза важна для обеспечения быстрого отклика

сервиса. Поддержка русского языка необходима для реализации сервиса, нацеленного на русскоязычную аудиторию. Сравнительный анализ выбранных методов синтеза речи представлен в таблице 2.

Таблица 2

Модель	Качество звука (MOS)	Естественность (MOS)	Скорость генерации (секунды на секунду речи)	Поддержка русского языка
SileroTTS	4.2	4.1	0.02	Да
Mozilla TTS	3.5	3.6	0.06	Да
ESPnet-TTS	3.7	3.5	0.1	Да
Tacotron	4.1	4.2	0.3	Нет

Для оценки качества синтеза речи использовалась метрика MOS (Mean Opinion Score), которая представляет собой среднюю оценку качества, данную группой слушателей. Оценка качества проводилась на основании открытых источников, включая официальную документацию и исследования [10, 11], опубликованные авторами моделей. Также были учтены отзывы пользователей и разработчиков, имеющиеся в открытом доступе.

Сервис был разработан с использованием Python, обеспечивающего большую гибкость и поддержку множества библиотек, упрощающих работу с аудио и текстовыми данными. Для создания веб-API был выбран фреймворк Flask, который обладает простотой и легкостью использования, что обеспечивает быструю разработку и тестирование сервиса.

Синтез речи в сервисе основан на модели SileroTTS, которая была выбрана в ходе сравнительного анализа. Эта модель обеспечивает высококачественный синтез речи, что делает ее идеальным выбором для данного сервиса. Процесс синтеза речи включает преобразование текста в звуковой сигнал, который затем кодируется в аудиофайл в формате OGG Opus.

Одной из ключевых особенностей разработанного сервиса является поддержка Speech Synthesis Markup Language (SSML). SSML — это язык разметки, который предоставляет разработчикам расширенный набор инструментов для контроля над синтезированной речью. С помощью SSML можно манипулировать такими параметрами речи, как скорость, тон, громкость, а также вставлять паузы в нужных местах.

В нашем случае, входной текст перед обработкой моделью TTS обрабатывается для преобразования в формат SSML. Знаки препинания, такие как точки, вопросительные и восклицательные знаки, заменяются на соответствующие теги SSML, что позволяет более точно контролировать процесс синтеза речи.

Использование SSML позволяет улучшить качество и естественность синтезированной речи, делая сервис более функциональным и адаптивным для потребностей пользователей.

Также был произведен препроцессинг входных данных перед обработкой моделью TTS. В частности, все числа в тексте были преобразованы в их словесное представление на русском языке, а слова на иностранных языках были транслитерированы в русский. Это было сделано для обеспечения того, чтобы модель TTS могла корректно обработать и произнести все элементы текста.

Выбор формата OGG Opus для передачи аудио по сети был основан на его превосходных характеристиках для передачи звука. Этот формат обеспечивает высокое качество аудио при низких битрейтах, что делает его идеальным для использования в сетевых приложениях.

В результате, сервис был успешно разработан и протестирован, обеспечивая высококачественный синтез речи для входных текстовых данных.

3. Разработка модуля STT

Распознавание речи, или Speech-to-Text (STT), – это область, в которой достигнуты значительные успехи за последние годы. STT используется в широком спектре приложений, включая автоматическую транскрипцию, помощников по управлению голосом, системы для слепых или слабовидящих людей и многие другие.

В области STT многие методы и подходы используют машинное обучение, и, в частности, глубокое обучение, для преобразования аудиосигналов в текст. Существует множество моделей и алгоритмов, предназначенных для этой цели, и был проведен анализ, чтобы определить, какие из них наиболее подходят для наших целей.

Было рассмотрено несколько моделей, включая CMU Sphinx [16], Kaldi [17], DeepSpeech [18] и Wav2Vec 2.0 [19].

CMU Sphinx — это набор систем автоматического распознавания речи, разработанных в университете Carnegie Mellon (CMU). Эти системы представлены на разных уровнях сложности, включая Sphinx-2 для систем в реальном времени, Sphinx-3 для

точной оффлайн-обработки и Sphinx-4, написанный на Java для обеспечения большей гибкости и возможности работы с новыми методами распознавания речи.

Kaldi — это библиотека для распознавания речи, ориентированная на исследования. Kaldi предлагает обширную коллекцию готовых к использованию моделей и инструментов, а также предоставляет поддержку для ряда методов глубокого обучения и техник обучения. Kaldi предназначен для работы в условиях больших объемов данных и предлагает наборы инструментов для автоматической обработки и распознавания речи.

DeepSpeech, разработанный Mozilla, — это открытая модель распознавания речи, основанная на исследованиях Baidu и опубликованная в рамках проекта "Deep Speech: Scaling up end-to-end speech recognition". DeepSpeech использует нейронные сети для преобразования звуковых сигналов в текст. Важным особенностью DeepSpeech является его способность работать в режиме реального времени, что делает его подходящим для широкого круга приложений.

Wav2Vec 2.0, разработанный в Facebook AI, – это модель для распознавания речи, использующая подход self-supervised learning. Эта модель обучается на больших неразмеченных наборах данных и в состоянии превосходить многие другие методы распознавания речи. Wav2Vec 2.0 преобразует аудиосигналы в текст, используя сложные архитектуры нейронных сетей, и, может быть, дообучен на меньших размеченных наборах данных для улучшения производительности на конкретных задачах.

Результат сравнения данных моделей приведен на таблице 3.

Таблица 3

Модель	Точность (WER на LibriSpeech test-clean)	Поддержка русского языка	Количество параметров
CMU Sphinx	~15.8%	Да	3.5M
Kaldi	~5.83%	Да	Зависит от модели
DeepSpeech	~6.5%	Да	120M
Wav2Vec 2.0	~2.0%	Да	315M

На основании анализа, была выбрана модель Wav2Vec 2.0, которая показывает лучшую производительность в тестовых наборах данных и поддерживает русский язык. Эта модель, разработанная в Facebook AI [4], использует технологию self-supervised learning и представляет собой глубокую нейронную сеть с большим количеством параметров, что позволяет ей достигать отличных результатов в задачах распознавания речи.

Проект представляет собой комплексный сервис распознавания речи, который способен преобразовать аудио данные в текстовую форму. Сервис основан на серверной архитектуре и принимает аудиоданные через веб-запросы. Процесс обработки аудио начинается с преобразования входных данных из формата base64 в формат Opus, который затем конвертируется в формат WAV.

Конвертированный WAV-файл далее преобразуется в волновую форму, которую можно использовать для передачи в модель распознавания речи. Для обработки аудиоданных и получения текстового представления используется модель Wav2Vec2 от Facebook. Это передовая модель в области распознавания речи, которая обучена на большом наборе данных и показывает выдающиеся результаты.

После преобразования речи в текст выполняется постобработка с целью замены текстовых представлений чисел на их числовые аналоги. Это достигается путем сопоставления слов с их соответствующими числовыми значениями на основе предопределенного словаря.

Завершающий этап включает в себя возврат транскрибированного текста в ответ на исходный запрос. В случае отсутствия аудиоданных сервис возвращает код ошибки, указывающий на некорректный запрос.

В целом, представленный сервис предлагает простой и эффективный способ преобразования аудиоданных в текстовую форму, доступную для дальнейшего анализа и обработки. Это полезное и значимое направление в современных исследованиях в области компьютерных наук и обработки естественного языка.

ЗАКЛЮЧЕНИЕ

В заключении преддипломной практики хочется отметить значимость проделанной работы в области обработки естественного языка, синтеза и распознавания речи. Процесс исследования и разработки в каждом из этих направлений был наполнен актуальными открытиями и применением передовых подходов в области компьютерных наук.

На стыке компьютерной лингвистики и системного программирования воплотился в реальность проект NLU-сервиса. Этот инструмент обрабатывает естественный язык и преобразует его в оперативные команды для системы Jenkins. Комплексное обучение модели на составленном наборе данных позволило повысить точность распознавания запросов и извлечения необходимых данных.

Компонент по синтезу речи, основанный на модели SileroTTS, отлично демонстрирует возможности перевода текстовых данных в голосовые. Продуманный подход к выбору формата аудио учел баланс между качеством звука и размером файла для эффективной передачи данных по сети.

STT-сервис, сформированный в ходе практики, с успехом справляется с преобразованием аудиофайлов в текстовые данные. Исследования в области аудиоформатов привели к выбору наиболее подходящего для передачи звуковых данных по сети.

Работа над практикой позволила глубже погрузиться в сферу искусственного интеллекта и обработки данных, а полученные знания и опыт открывают новые возможности для более глубокой интеграции с другими системами. Каждый из разработанных компонентов представляет собой ценный вклад в развитие современных технологий в области автоматизации и искусственного интеллекта.

Результат проверки дипломной работы в системе антиплагиат: 0%.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Loper, E., and Bird, S. (2002). NLTK: The Natural Language Toolkit. In Proceedings of the ACL-02 Workshop on Effective tools and methodologies for teaching natural language processing and computational linguistics - Volume 1 (ETMTNLP '02). Association for Computational Linguistics, USA, 63–70, URL: https://www.aclweb.org/anthology/W02-0109.pdf

2. Akbik, A., Blythe, D., and Vollgraf, R. (2018). Contextual String Embeddings for Sequence Labeling. In Proceedings of the 27th International Conference on Computational Linguistics, URL: https://www.aclweb.org/anthology/C18-1139.pdf

3. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. In Advances in neural information processing systems, URL: https://papers.nips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html

4. Bojanowski, P., Grave, E., Joulin, A., & Mikolov, T. (2017). Enriching word vectors with subword information. Transactions of the Association for Computational Linguistics, 5, 135-146, URL: https://www.mitpressjournals.org/doi/10.1162/tacl_a_00051

5. Chollet, F. (2018). Deep learning with Python. Manning Publications Co, URL: https://www.manning.com/books/deep-learning-with-python

6. Bird, S., Klein, E., & Loper, E. (2009). Natural language processing with Python: analyzing text with the natural language toolkit. "O'Reilly Media, Inc.", URL: https://www.oreilly.com/library/view/natural-language-processing/9780596516499/

7. Rasa NLU Official Documentation, URL: https://rasa.com/docs/rasa/nlu/

8. Flair Official GitHub Repository, URL: https://github.com/flairNLP/flair

9. SpaCy Official Documentation, URL: https://spacy.io/usage/spacy-101

10. Paine, T.-L., et al. "FastSpeech: Fast, Robust and Controllable Text to Speech,", 2019, URL: https://arxiv.org/abs/1905.09263

11. Shen, J., et al. "Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions", 2017, URL: https://arxiv.org/abs/1712.05884

12. V. Liptchinsky, et al. "Silero Models: Small, Fast, Deployable Speech Models" OpenAI, 2020, URL: https://github.com/snakers4/silero-models

13. T. B. Chan et al., "Multi-speaker Tacotron in TensorFlow," Mozilla, 2020, URL: https://github.com/mozilla/TTS

14. S. Watanabe et al., "ESPnet: End-to-End Speech Processing Toolkit," 2018, URL: https://github.com/espnet/espnet

15. Y. Wang et al., "Tacotron: Towards End-to-End Speech Synthesis", 2017, URL: https://arxiv.org/abs/1703.10135

16. CMU Sphinx: Walker, W. et al. "Sphinx-4: A Flexible Open-Source Framework for Speech Recognition", 2004, URL: https://www.cs.cmu.edu/~robust/Papers/Online/Sphinx4.pdf

17. Kaldi: Povey, D. et al. "The Kaldi Speech Recognition Toolkit", 2011, URL: https://ieeexplore.ieee.org/document/6296526

18. DeepSpeech: Hannun, A. et al. "Deep Speech: Scaling up end-to-end speech recognition", 2014, URL: https://arxiv.org/abs/1412.5567

19. Wav2Vec 2.0: Baevski, A. et al. "wav2vec 2.0: A Framework for Self-Supervised Learning of Speech Representations", 2020, URL: https://arxiv.org/abs/2006.11477