

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет

вычислительной техники

Кафедра

«Вычислительная техника»

Направление подготовки: 09.04.01 «Информатика и вычислительная техника»

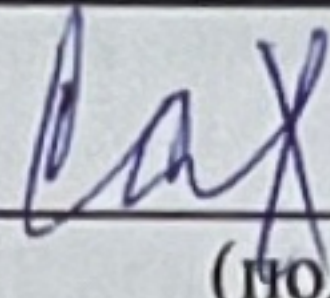
Магистерская программа: «Прикладной искусственный интеллект»

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

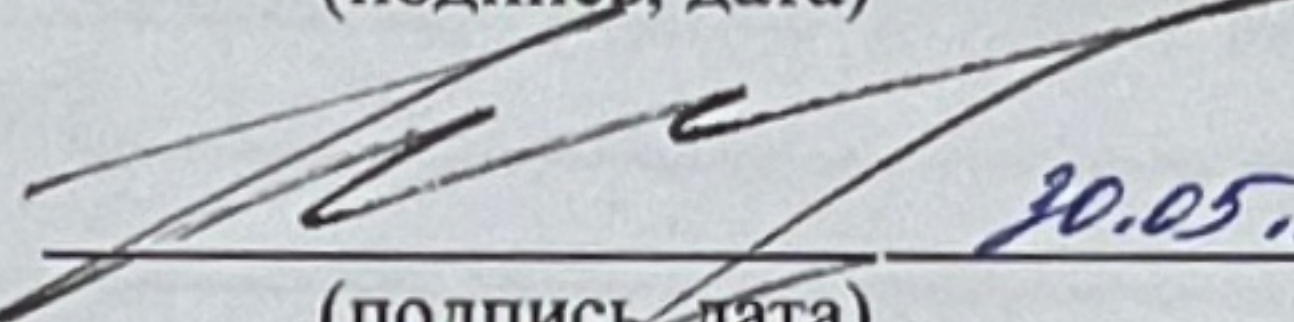
на тему

РАЗРАБОТКА ГОЛОСОВОГО ПОМОЩНИКА ДЛЯ
ОЗВУЧИВАНИЯ ТЕКСТА С ИСПОЛЬЗОВАНИЕМ
НЕЙРОННЫХ СЕТЕЙ

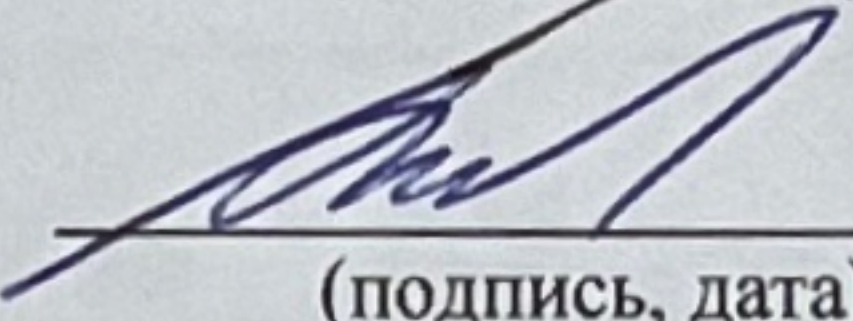
Студент

 30.05.2025 Сохацкий Дмитрий Юрьевич
(подпись, дата) (ФИО полностью)

Руководитель

 30.05.2025 Кузнецова О.Ю.
(подпись, дата) (фамилия, инициалы)

Нормоконтролёр

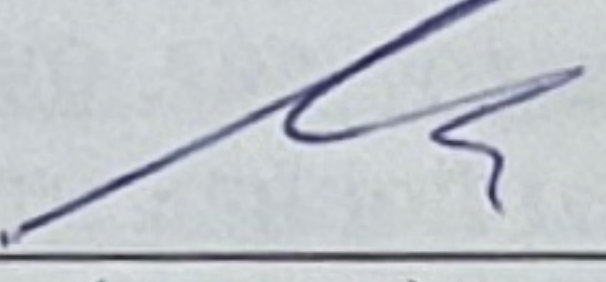
 30.05.2025 Бычков А.С.
(подпись, дата) (фамилия, инициалы)

Рецензент

зав. каф. Программирования ИИ, 30.05.2025 Мартыненко А.И.
(подпись, дата) (фамилия, инициалы)

Работа допущена к защите (протокол зас. кафедры от 28.05.2025 г. № 16)

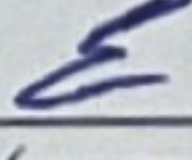
Заведующий кафедрой

 М.А. Митрохин
(подпись) (фамилия, инициалы)

Работа защищена с отметкой отлично (протокол заседания ГЭК от

5 июня 2025 № 3)

Секретарь ГЭК

 Гурин Е.И.
(подпись) (фамилия, инициалы)

Пенза 2025

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет

вычислительной техники

Кафедра

«Вычислительная техника»

«Утверждаю»

Заведующий кафедрой

М.А. Митрохин

« 29 » 09 2023 г.

ЗАДАНИЕ
НА МАГИСТЕРСКУЮ ДИССЕРТАЦИЮ

1. Студент _____ гр. 23ВВИм1 факультета ВТ

Сохацкий Дмитрий Юрьевич

2. Тема работы «Разработка голосового помощника для озвучивания текста с использованием нейронных сетей»

Тема утверждена приказом ПГУ № 986/0 от " 29 " 09 2023 г.

3. Руководитель магистерской диссертации: к.т.н. доцент кафедры «ИВС»
Кузнецова Ольга Юрьевна

4. Задание на работу

Разрабатываемое приложение предназначено для создания аудиоматериалов из текста с использованием нейронных сетей. Предметом данной работы является совершенствование технологий с использованием нейронных сетей для синтеза голосов и озвучивания текстов. Функциональные требования. Программа должна:

Осуществлять синтез речи

Распознавать ударения в словах и знаки препинания

Иметь возможность настройки высоты, скорости и громкости речи

Генерировать аудиофайл в форматах .wav и .mp3

Иметь возможность обучения голосовых моделей

5. Перечень подлежащих разработке вопросов

Анализ программных продуктов для синтеза речи

Анализ методов и технологий для преобразования текста в речь

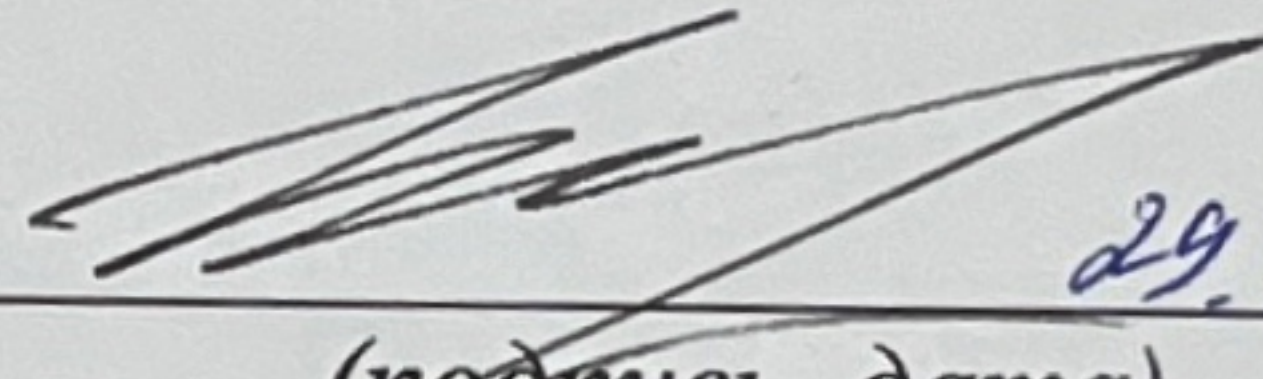
Исследование и тестирование существующих моделей для обучения и синтеза речи

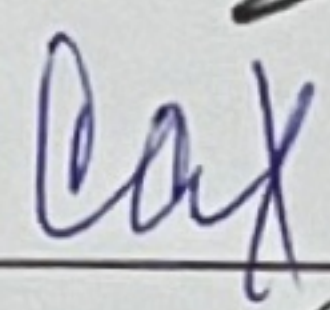
Реализация выбранных методов и технологий для преобразования текста в речь и обучения

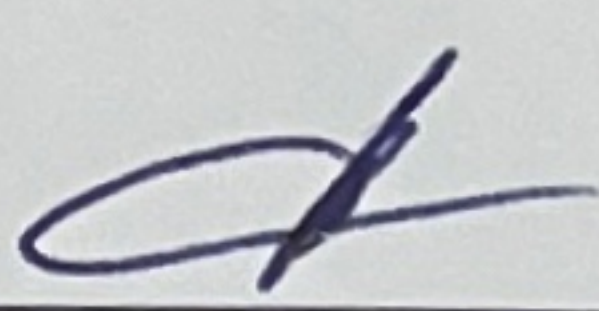
6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Предпроектное обследование	10 %	20.12.2023	
2	Анализ предметной области	25 %	25.03.2024	
3	Выбор средств разработки	10 %	10.05.2024	
4	Разработка приложения	25%	25.11.2024	
5	Тестирование ПО	15 %	15.03.2025	
6	Оформление ПЗ	15 %	30.05.2025	
7	Нормоконтроль		31.05.2025	

Дата выдачи задания " 29 " 09 20 23 г.

Руководитель магистерской диссертации  29.09.2023
(подпись, дата)

Задание к исполнению принял студент  29.09.2023
(подпись, дата)

Работу к защите допустить
Декан факультета  04.06.2025
(подпись, дата)

РЕФЕРАТ

Пояснительная записка содержит 103 листа, 33 рисунка, 5 таблиц, 26 источников.

НЕЙРОСЕТИ, ОБРАБОТКА ТЕКСТА, VISUAL STUDIO CODE, ОЗВУЧИВАТЕЛЬ, PYTHON, VUEJS, TEXT-TO-SPEECH

Целью магистерской диссертации является разработка голосового помощника для озвучивания текста с использованием обученной модели голоса.

В результате проделанной работы проведен анализ предметной области, выбрана среда разработки, проведен анализ аналогичных продуктов, проектирование и разработка приложения.

В результате выполнения работы разработано приложение голосового помощника для озвучивания текста с использованием нейронных сетей.

Эффективность разработки заключается в минимизации временных и трудовых затрат на создание аудиоматериалов.

При разработке приложения использовался дистрибутив языков программирования на языках Python и JS с использованием фреймворка VueJS с приложением Visual Studio Code.

					ПГУ 1.09.04.01.11.001 ПЗ			
Изм.	Лист	№ докум.	Подпись	Дата	Разработка голосового помощника для озвучивания текстов с использованием нейронных сетей	Лит.	Лист	Листов
Разраб.		Сохацкий Д.Ю.		30.05.25				
Провер.		Кузнецова О.Ю.		30.05.25			2	103
Н. Контр		Бычков А.С.		30.05.25		ФВТ, каф. ВТ		
Утверд.		Митрохин М.А.		30.05.25		гр. 23ВВИм1		

СОДЕРЖАНИЕ

ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ	5
ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ	6
ВВЕДЕНИЕ	7
1 Актуальность	9
2 Предпроектное исследование	10
2.1 Сравнение аналогичных решений	10
2.2 Обоснование использования выбранных средств разработки	14
3 Анализ предметной области	19
3.1 Анализ архитектур нейронных сетей	19
3.1.1 Рекуррентные нейронные сети	19
3.1.2 Сети долгой краткосрочной памяти	20
3.1.3 Gated Recurrent Units (GRU)	23
3.2 Анализ архитектур нейронных сетей для синтеза речи из текста	24
3.3 Анализ вокодеров	30
3.3.1 WaveGlow	30
3.3.2 HiFi-GAN	32
3.4 Обоснование выбранных технологий для разработки приложения ...	34
4 Техническое задание	37
4.1 Основание для разработки	37
4.2 Назначение разработки	37
4.3 Требования к программе	37
4.3.1 Требования к функциональным характеристикам	37
4.3.2 Требования к надежности	37
4.3.3 Требования к составу и параметрам технических средств	38
4.3.4 Требования к информационной и программной совместимости ..	38
4.4 Требования к программной документации	38
4.5 Стадии и этапы разработки	39
4.6 Порядок контроля и приемки	39
5 Разработка программного обеспечения	40

5.1 Диаграмма вариантов использования	40
5.2 Диаграмма развертывания	41
5.3 Разработка структуры приложения	41
6 Описание программы	44
6.1 Общие сведения	44
6.2 Функциональное назначение	44
6.3 Используемые технические средства	44
6.4 Вызов и загрузка	44
6.5 Входные данные	45
6.6 Выходные данные	45
7 Программа и методика испытаний	46
7.1 Объект испытаний	46
7.2 Цель испытаний	46
7.3 Средства и порядок испытаний	46
7.4 Методы испытаний	46
7.5 Результаты испытаний	49
8.1 Назначение применения	52
8.2 Условия применения	52
8.3 Описание задачи	52
8.4 Инструкция развертывания	56
ЗАКЛЮЧЕНИЕ	57
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ.	58
ПРИЛОЖЕНИЕ А Конфигурации предобученных моделей	61
ПРИЛОЖЕНИЕ Б Листинг программы	67
ПРИЛОЖЕНИЕ В Презентация	94

ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ

В настоящей магистерской диссертации применяются следующие термины и определения

UML – Унифицированный язык моделирования. Это графический язык, который с помощью диаграмм и схем описывает разнообразные процессы и структуры

TTS – Технология, которая преобразует текст в устную речь. Она позволяет компьютерам и другим устройствам передавать информацию пользователю интерфейса голосом, озвучивая текстовые данные

JS – Мультипарадигменный язык программирования

RNN – Тип нейронных сетей, специально разработанный для обработки последовательных данных

LSTM – Специализированная архитектура рекуррентных нейронных сетей, разработанная для обработки последовательностей данных, где важен контекст

GRU – Тип архитектуры рекуррентной нейронной сети. Она используется в глубоком обучении для обработки последовательных данных

GAN – Генеративно-сопоставительная сеть, алгоритм машинного обучения

IDE (от англ. Integrated Development Environment) – интегрированная среда разработки, набор ПО для создания кода

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ

В настоящей магистерской диссертации применяются следующие сокращения и обозначения

UML – унифицированный язык моделирования (Unified modeling language)

TTS – текст-в-речь (Text-to-speech)

JS – JavaScript

RNN – рекуррентная нейронная сеть (Recurrent neural network)

LSTM – долгая краткосрочная память (Long short-term memory)

GRU – управляемые рекуррентные блоки (Gated recurrent unit)

GAN – генеративно-сопоставительная сеть (Generative adversarial network)

IDE – интегрированная среда разработки (Integrated development environment)

ВВЕДЕНИЕ

В современном мире технологии нейросетей становятся все более популярными и востребованными. Они позволяют создавать аудио-контент из письменного текста, делая его доступным для людей с ограниченными возможностями. Кроме того, они могут синтезировать человеческую речь в информационных, образовательных, развлекательных и рекламных целях.

Одним из методов голосового озвучивания является синтез речи с использованием нейросетей, которые способны обрабатывать и интерпретировать текст с удивительной точностью и естественностью. Однако, существующие голосовые помощники для озвучивания текста имеют определенные ограничения и недостатки в области озвучивания текста, что ограничивает их потенциал в использовании.

Таким образом актуальна разработка программного решения, позволяющего улучшить работу нейронных сетей в области озвучивания текста. Создание качественного озвучивателя может позволить сократить расходы и ускорить процесс создания различных аудио и видео, которые могут использоваться в информационных, образовательных, развлекательных и рекламных целях.

Целью данной работы является оптимизация процессов для создания медиа материалов в различных сферах деятельности.

Объектом данной работы является программное средство для озвучивания текста с использованием нейронных сетей.

Предметом данной работы является совершенствование технологий с использованием нейронных сетей для синтеза голосов и озвучивания текстов.

Для достижения цели будут использованы следующие методы: анализ аналогов озвучивателей текста, а также существующих методов и технологий, разработку и обучение нейронной сети, разработку средств, обеспечивающих работу нейронной сети, тестирование и сравнение результатов с другими голосовыми помощниками.

Полученные результаты могут иметь значительное значение для улучшения качества озвучивателей текста и их применения в различных областях жизнедеятельности.

По теме исследования опубликована 1 научная работа.

1 Актуальность

Актуальность данной темы обусловлена постоянно возрастающим количеством источников информации и медиа ресурсов в различных сферах деятельности, включая медицину, образование, бизнес и повседневную жизнь. Зачастую информацию более удобно воспринимать, прослушивая данные ресурсы, не сильно отвлекаясь от более приоритетных дел. Искусственный интеллект может позволить заметно расширить возможности потребления информации, а также существенно сократить расходы на создание медиа-контента и сократить сроки создания работ, связанных с озвучиванием текста.

Существующие голосовые помощники имеют определенные ограничения в качестве озвучивания текста, что снижает их эффективность и удобство использования. Улучшение качества и доступности озвучивания текста с помощью нейросетей может значительно повысить привлекательность и полезность голосовых помощников для конечных пользователей.

Кроме того, разработка более естественного и качественного голосового помощника может способствовать улучшению доступности информации для людей с ограниченными возможностями, такими как люди с нарушениями зрения или чтения.

Таким образом, разработка голосового помощника с использованием нейросетей представляет собой актуальную задачу, имеющую потенциально значительное влияние на различные области человеческой деятельности.

2 Предпроектное исследование

2.1 Сравнение аналогичных решений

В качестве аналогичных решения для исследования были выбраны голосовые озвучиватели текста Zvukogram, NaturalReaders, Cybervoice и Genny.

Zvukogram – это сервис для озвучки текста с применением искусственного интеллекта. Он имеет свой собственный API и позволяет взаимодействовать через свой интерфейс (рисунок 1).

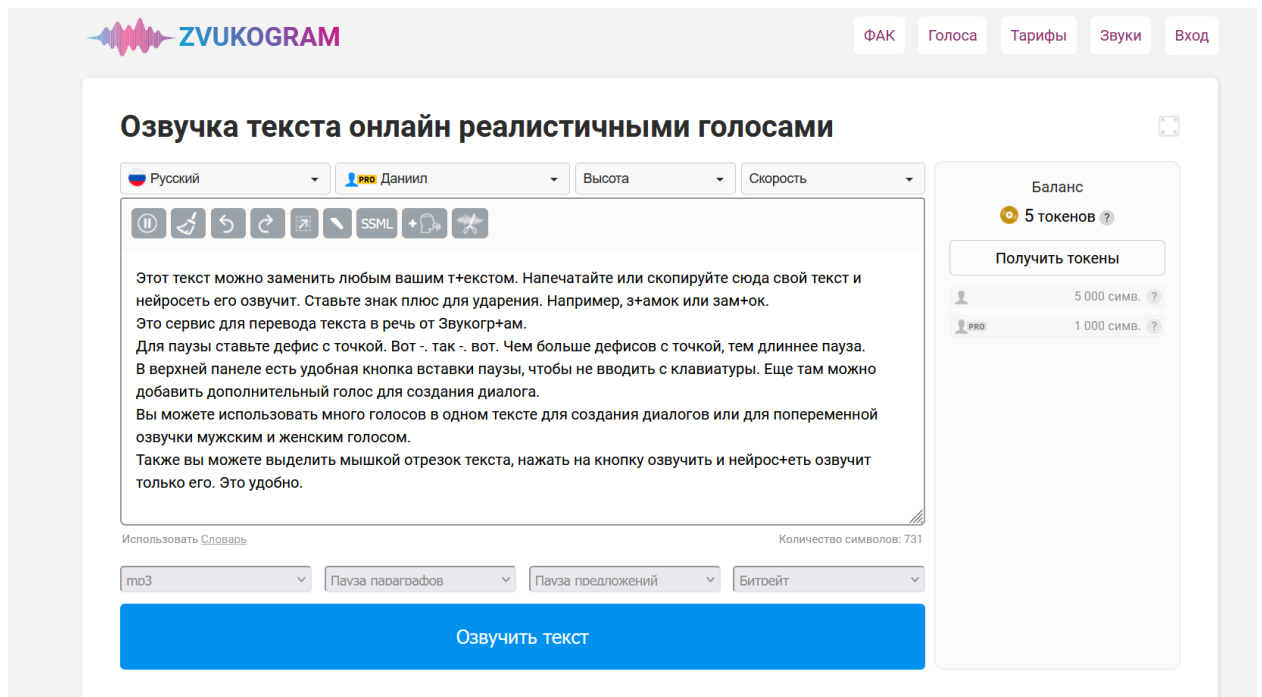


Рисунок 1 – Интерфейс сервиса Zvukogram

Zvukogram позволяет настроить тембр и скорость чтения текста. Кроме того, доступно озвучивание на разных языках и разными голосами (в том числе мужские и женские). Их разнообразие ограничено стоимостью в специальных токенах и премиум подписками. Также можно настроить паузы при переходе на следующий параграф и в конце предложений. Возможно добавление ударений путем добавления знака «+» перед нужной гласной буквой. В личном кабинете доступен словарь в который можно добавлять слова с нужным ударением, после чего сервис будет запоминать их и использовать с требуемым ударением [1].

Бесплатно доступно только 5 пробных озвучек, после чего заканчиваются токены. В качестве примера был введен следующий текст:

«Пробный вариант озвучивания. Студент: Сохацкий Дмитрий Юрьевич, группа 23ВВИм2.

Neural TTS — относительно новая технология, которая преобразует текст в речь с помощью нейросети. Это позволяет добиться более реалистичной имитации человеческого голоса: он звучит естественно, хорошо передает простые эмоции и не допускает ошибок в ударениях. Тест окончен.»

После чего в течении 10 секунд был получен результат в виде файла формата mp3.

Следует отметить, что голос в большинстве случаев правильно расставляет ударения и паузы и также адекватно озвучивает текст как на русском, так и на английском языке. Все слова также разборчивы и понятны. Однако можно без проблем заметить, что данную речь произносит не человек из-за многочисленных голосовых искажений.

NaturalReader — это англоязычный сервис для умного озвучивания текста, также имеющий свой интерфейс, представленный на рисунке 2.

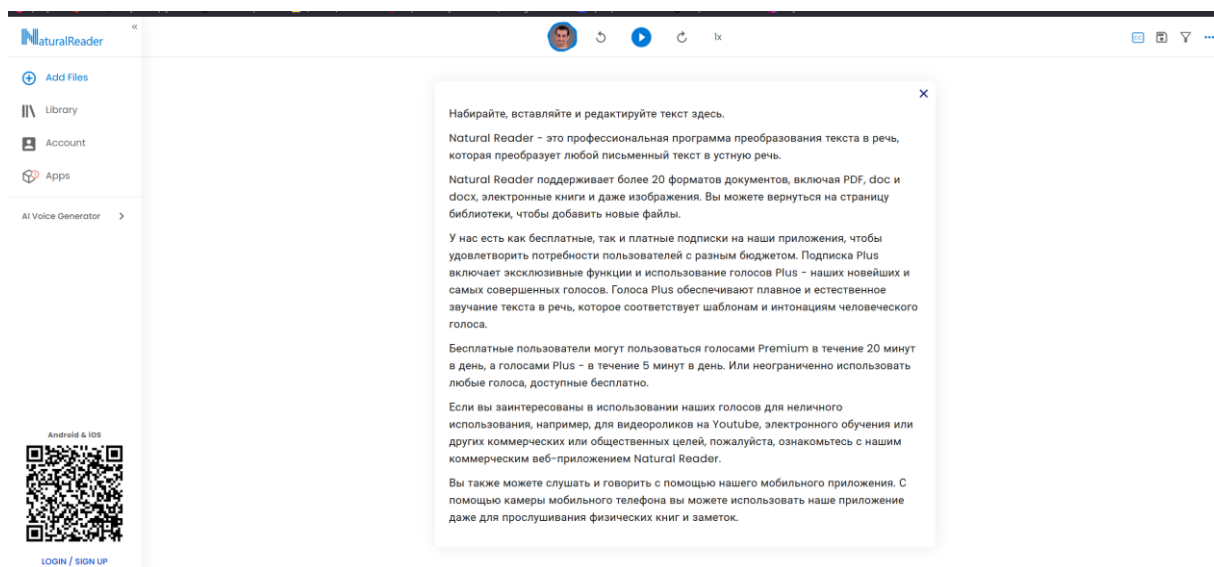


Рисунок 2 – Интерфейс сервиса NaturalReader

NaturalReader достаточно ограничен в плане разнообразия голосов при бесплатном использовании. Так, для озвучки на русском языке доступен только 1 женский и 1 мужской голос. Возможно изменение скорости чтения, а

также чтение отдельных фрагментов текста. Скачивание аудиофайла доступно только при подписке [2].

В ходе тестирования на приведенном выше примере в течении 5 секунд была сгенерирована озвучка с использованием мужского голоса на русском языке. Слова звучат разборчиво, а паузы от знаков препинания более выдержаны по сравнению с прошлым аналогом. Однако качество озвучивания желает оставлять лучшего. Полученный голос совершенно не передает ощущения человеческой речи, а также ударения расставлены зачастую неверно. Вероятно, сказывается ориентированность на иностранных пользователей.

Genny – это также англоязычный сервис от компании LOVO, для пользования которым требуется зарегистрироваться на сайте разработчика. Сервис также подписывает несколько тарифных планов, делающих работу более комфортной и качественной.

Данный сервис предполагает создание двух вариантов проектов: простое озвучивание коротких текстовых речей или создание аудио/видео медиа-содержимого. Для тестирования будет использован первый вариант. Интерфейс представлен на рисунке 3 [3].

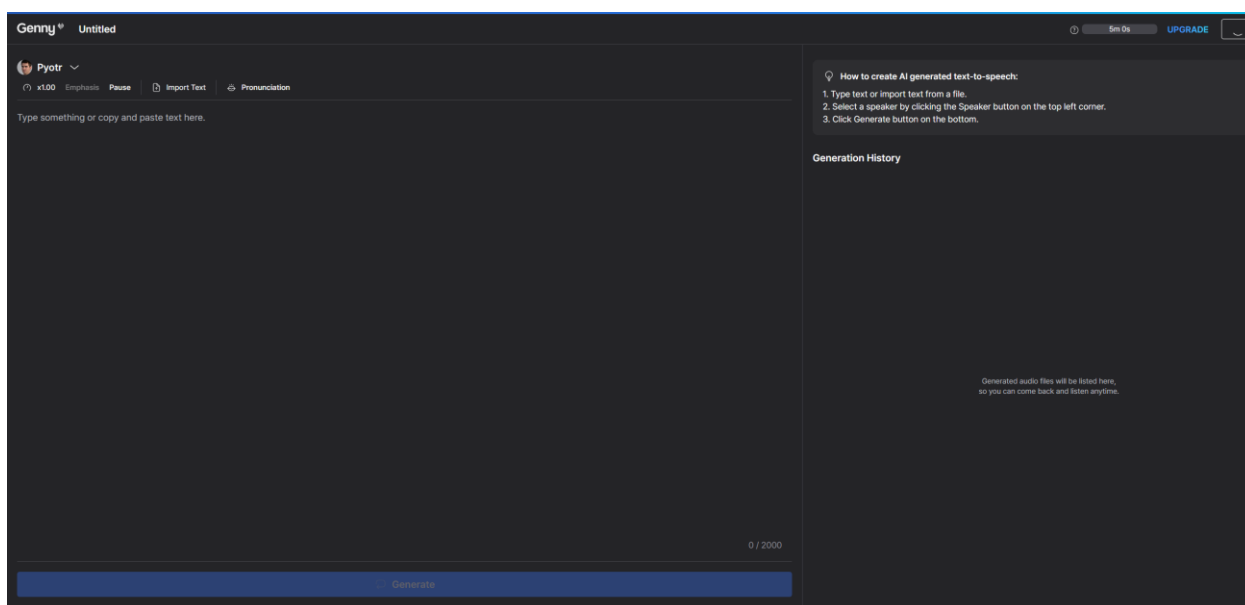


Рисунок 3 – Интерфейс сервиса Genny

Бесплатный тариф позволяет выбрать достаточно большое количество голосов, как женских, так и мужских. За каждым голосом также закреплен язык, на котором будет осуществляться озвучка. Также возможно импортировать текст из текстового файла.

В ходе тестирования была использована модель голоса «Pyotr» на русском языке. В течении 17 секунд была сгенерирована речь длиной 28 секунд. Данная речь звучит качественно, все слова разборчивы, а ударения в большинстве случаев расставлены правильно. Кроме того, искажения голоса минимальны и достаточно похожа на человеческую. Можно отметить также использование соответствующих интонаций при использовании знаков препинания. Недостатком можно выделить слишком затянутые паузы между предложениями и отсутствие регулировки данного аспекта.

CyberVoice (SteosVoice) – это решение для озвучивания вводимого текста путем использования бота в мессенджере Telegram. Также данный сервис предоставляет API для разработчиков, которые хотят создать звуковое сопровождение в своих проектах, что позволяет их приложениям общаться с пользователями.

Telegram бот данного решения бесплатен и позволяет генерировать озвучку 2 000 символов каждый день. Кроме того, допускается использовать более 300 различных голосов, в том числе известных личностей и персонажей медиа-культуры, с указанием языка озвучивания. Однако данного бота запрещено использовать в коммерческих целях.

Из настроек доступен выбор голоса озвучивания, а также скорость чтения и высота. После настройки нужно ввести текст в сообщении к данному боту, вследствие чего в ответном сообщении будет выслан аудиофайл [4].

Данный сервис практически мгновенно генерирует аудиофайл. Полученный результат звучит достаточно натурально из-за использования интонаций, однако заметны немногочисленные искажения при попытках поставить ударение. Также очень плохо озвучивается текст на языке, не

предполагаемый выбранным голосом. Качество звучания коррелируется в зависимости от выбранного голоса.

2.2 Обоснование использования выбранных средств разработки

В ходе анализа аналогичных продуктов было определено, что каждое решение обладает преимуществами и недостатками. Наилучшим решением в плане качества звучания является решение Genny, однако имеет низкую производительность и доступность. Наиболее же быстрым и доступным решением оказалось CyberVoice (SteosVoice) благодаря использованию бота. Zvukogram имеет самые гибкие настройки для генерации речи, а NaturalReader позволяет динамически озвучивать отдельные части текста в любой момент времени.

В общем виде экспертные оценки (от 0 до 10) критериев аналогичных решений для озвучки текста представлены в таблице 1.

Таблица 1 – Экспертные оценки

Нейросеть	Качество	Скорость	Языки	Доступность	Кол-во голосов
Zvukogram	9	4	9	5	8
Genny	8	1	2	3	8
Cybervoice	5	7	7	8	10
Natural Readers	4	8	5	8	1

Для разработки было решено реализовать веб-приложение голосового озвучивателя текстов с использованием следующих инструментов: средство разработки UML-процессов StarUML, редактор кода Visual Studio Code с использованием языков программирования Python и JavaScript на базе фреймворка VueJS 3 и инструментов для сборки веб-проектов Vite. Для создания календарного плана было использовано приложение ProjectLibre.

Основными преимуществами разработки веб-приложения являются:

- отсутствие нужды установки приложения на устройство,
- более низкие системные требования и мультиплатформенность,
- отсутствие нужды загружать обновления, так как все обновления устанавливаются на сервере,

– более удобная разработка без нужды перекомпилировать код.

Для реализации проектов и приложений существует большое количество интегрированных платформ разработки (IDE) и редакторов кода. Для реализации проекта голосового озвучивателя текстов необходимо выбрать наиболее удобную из перечня самых востребованных.

Сравнительная характеристика интегрированных средств разработки Visual Studio Code, Anaconda Spyder IDE [5] и Notepad++ [6] представлена в таблице 2.

Таблица 2 – Сравнительная характеристика современных IDE

Критерий	Visual Studio Code	Notepad++	Anaconda Spyder IDE
JavaScript	+	+	-
Python	+	+	+
Наличие расширений	+	+	+
Наличие отладчика	+	-	+
Наличие терминала	+	-	+
Компания разработчик	Microsoft	Eclipse Foundation	Apache Software Foundation, Oracle

Исходя из сравнительной характеристики IDE, можно сделать вывод о том, что наиболее средство для разработки на Windows является Microsoft Visual Studio Code, так как имеет все необходимые инструменты для разработки на языках программирования JavaScript и Python, а также для комфортного процесса написания кода, отладки и тестирования.

Visual Studio Code представляет собой модифицированный текстовый редактор для разработчиков. Данный инструмент поддерживает множество языков программирования, в том числе JavaScript и Python, а также предоставляет большие возможности для написания, отладки и тестирования кода. В Visual Studio Code можно модернизировать среду разработки под собственные нужды, используя расширения и дополнительные инструменты. Данный продукт позволяет реализовывать различные задачи от простого редактирования файлов до разработки сложных веб- и мобильных приложений.

Кроме того, можно добавить поддержку дополнительных языков и фреймворков через расширения. Также редактор позволяет создавать проекты и структуры файлов, делать отладку и запускать код на определенных языках программирования.

Visual Studio Code обеспечивает комфортную работу для разработчиков благодаря многим возможностям. Так он поддерживает подсветку синтаксиса и автодополнение кода при помощи встроенного инструмента IntelliSense. Также имеет встроенный терминал, позволяющий выполнять команды, не закрывая редактор кода и интеграцию с Git: прямо из редактора можно управлять репозиториями, делать коммиты и просматривать историю изменений. Интерфейс можно изменять под свои предпочтения, выбирая темы, горячие клавиши и другие настройки. Кроме того, поддерживаются многочисленные расширения и интеграции с различными инструментами, такими как Docker, Kubernetes и другие [7].

Python – бесплатный объектно-ориентированный язык программирования с открытым исходным кодом. Он включает в себя модули, исключения, динамическую типизацию, динамические типы данных очень высокого уровня и классы, а также поддерживает несколько парадигм программирования помимо объектно-ориентированного, например, процедурное и функциональное.

Python – язык общего назначения высокого уровня, который можно использовать для создания самых различных программных решений. Он имеет понятный синтаксис и прост для изучения – его можно начать применять сразу в процессе обучения. Для Python регулярно появляются новые функции, сообщество пользователей создает сторонние библиотеки. Также он совместим с любой операционной системой и поддерживает автоматическую сборку мусора [8].

JavaScript (аббр. JS) – мультипарадигменный язык программирования. Поддерживает объектно-ориентированный и функциональный подходы для

программирования. Является реализацией спецификации ECMAScript (стандарт ECMA-262).

JavaScript обычно используется как встраиваемый язык для программного доступа к объектам приложений. Наиболее широкое применение находит в браузерах как язык сценариев для придания интерактивности веб-страницам [9].

Также для упрощения работы был использован фреймворк VueJS в паре с инструментом Vite.

Vue.js – это прогрессивный фреймворк для разработки пользовательских интерфейсов и одностраничных веб-приложений на языке JavaScript. Он решает задачи уровня представления (view) и упрощает работу с библиотеками. Vue.js можно внедрять постепенно, этим он отличается от других фреймворков. Его основными особенностями являются:

- реактивность. Vue.js – это реактивный фреймворк. Представление (view) изменяется по мере изменения модели,
- простота. Начать работать с фреймворком можно без базовых знаний в веб-разработке. Низкий порог вхождения — причина популярности у начинающих разработчиков,
- небольшой вес. Фреймворк занимает около 20 кБ, поэтому реализованные на нем проекты быстрее загружаются и лучше ранжируются поисковыми роботами,
- быстрая разработка. Шаблоны, удобная документация, широкое сообщество энтузиастов позволяют решить множество проблем, возникающую при создании проектов на Vue.js,
- доступность. Vue.js распространяется по лицензии MIT. Его можно свободно использовать и в коммерческих, и в личных целях [10].

Vite – это современный инструмент сборки для фронтенд-разработки со встроенным сервером. Он обеспечивает быструю и удобную разработку современных веб-проектов. Данный инструмент имеет встроенный сервер с режимом контроля изменений. При изменении одного файла в проекте Vite сам

обновляет представление в браузере и отображает изменения в реальном времени. Также поддерживает работу с ES-модулями. Это устраняет необходимость пересборки всего приложения при изменении кода. Интеграция с популярными фреймворками, благодаря чему позволяет легко адаптировать существующие проекты и начинать новые с минимальной настройкой [11].

Исходя из анализа программных продуктов, можно сделать вывод о целесообразности использования Visual Studio Code для разработки на языках Python и JavaScript с использованием фреймворка VueJS и инструментом для сборки Vite, а также StarUML и ProjectLibre для проектирования и разработки программного обеспечения голосового помощника для озвучивания текста с использованием нейросетей.

3 Анализ предметной области

3.1 Анализ архитектур нейронных сетей

Речь представляется последовательностью во времени некоторых векторов (амплитуд звукового сигнала или коэффициентов спектра), значит, для обработки ее необходимо применять методы, работающие с последовательностями. Для обработки последовательностей используется два типа нейронных сетей - это рекуррентные сети и сети на основе механизма внимания, которые часто используются вместе [13].

3.1.1 Рекуррентные нейронные сети

Рекуррентные нейронные сети (RNN) представляют собой специализированный тип нейросетевых архитектур, созданных для эффективной работы с последовательностями данных. Их ключевая особенность заключается в механизме сохранения и использования информации о предшествующих элементах последовательности при обработке новых данных (рисунок 4).

Благодаря системе памяти, RNN могут учитывать контекст и временные зависимости в таких данных как текстовые сообщения, аудиозаписи или финансовые временные ряды. Эта архитектура демонстрирует гибкость при работе с последовательностями произвольной длины, что особенно важно при анализе предложений разного размера или сигналов с переменной продолжительностью.

Обучение RNN осуществляется через алгоритм обратного распространения ошибки, что позволяет сети постепенно повышать точность своих прогнозов и адаптироваться к особенностям входных данных.

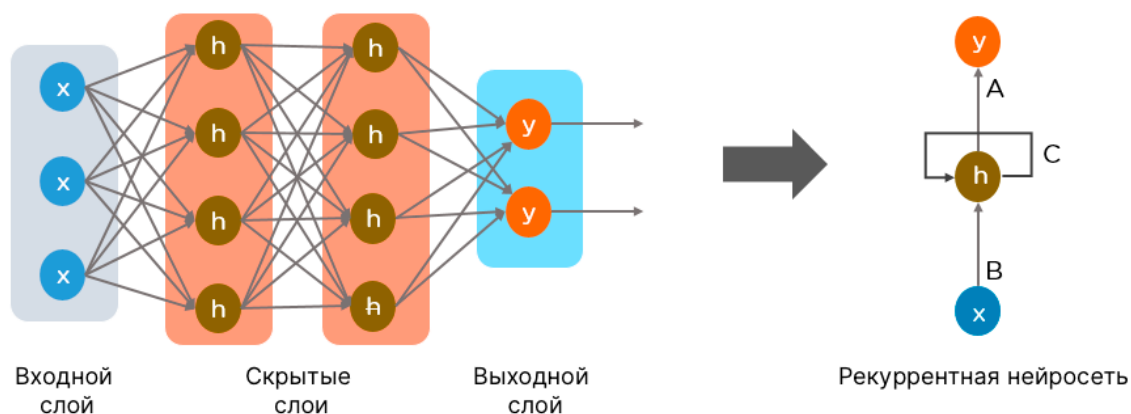


Рисунок 4 – Рекуррентная нейронная сеть

RNN могут для распознавания и синтеза речи. Они могут анализировать звуковые волны и преобразовывать их в текст или наоборот. А также эффективно работают с данными, где длительные зависимости между элементами играют роль.

Однако, данная архитектура имеет проблему затухающего градиента и неспособность эффективно учить долгосрочные зависимости. Для преодоления этих ограничений были разработаны более сложные архитектуры, такие как LSTM (Long Short-Term Memory) и GRU (Gated Recurrent Unit), которые имеют дополнительные механизмы для борьбы с этими проблемами [14].

3.1.2 Сети долгой краткосрочной памяти

Сети долгой краткосрочной памяти (Long Short-Term Memory, LSTM) являются одним из типов рекуррентных нейронных сетей. Данный вид нейронных сетей разработан для работы с последовательными данными, такими как тексты, временные ряды, аудио-сигналы и другие.

Основным отличием LSTM от стандартных RNN является способность LSTM сохранять и передавать информацию на протяжении длительных периодов времени без проблем, связанных с затуханием или взрывом градиентов. Это достигается благодаря внутренней структуре LSTM, которая включает в себя следующие компоненты (рисунок 5):

а) ячейка памяти (memory cell): сохраняет информацию и контролирует, как эта информация обновляется и передается на следующий этап. Это позволяет модели удерживать важные сведения на протяжении длительного времени,

б) ворота забывания (forget gates). Они управляют продолжительностью памяти в ячейке, определяя, какую информацию следует забыть, а какую сохранить. Для этого применяется сигмоидная функция активации ко входным данным. Она возвращает значения в диапазоне от 0 до 1. Эти значения определяются как степень «забывания» информации,

в) входные ворота (input gates). Данный компонент определяет, какая часть новой информации должна быть сохранена в долгосрочной памяти ячейки LSTM и определяют, какая информация достаточно важна для сохранения, они используют аналогичную сигмоидную функцию активации для определения важности входных данных. Сигмоид возвращает значения между 0 и 1, где 1 означает полное сохранение информации, а 0 — полное отклонение. Одновременно, входные ворота применяют гиперболический тангенс к тем же входным данным для создания вектора кандидатов на добавление в ячейку памяти,

г) выходные ворота (output gates) определяют, какая информация из текущего состояния ячейки будет передана в выходной сигнал сети. Выходные ворота также используют сигмоидную функцию активации σ , как и другие ворота в LSTM,

д) скрытое состояние (hidden state). Это выходной результат LSTM, который может быть передан на следующий временной шаг или использоваться для конечного предсказания.

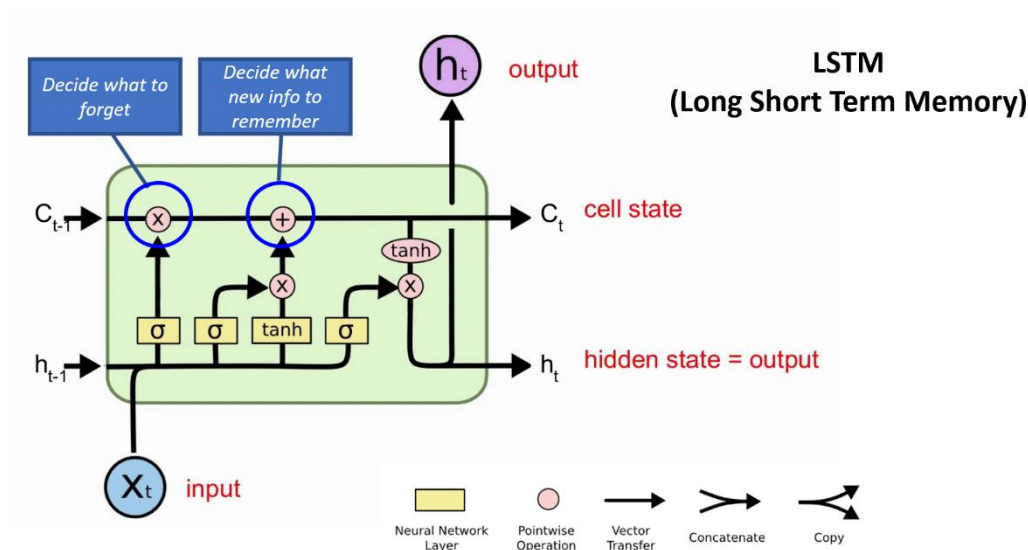


Рисунок 5 - Сети долгой краткосрочной памяти

Выход сигмоидной функции (пункт б) получается следующим образом:

$$f_i = \sigma(W_{xx} * x_t + W_{hh} * h_{t-1} + b_i)$$

где σ – сигмоидная функция;

W_{xx} и W_{hh} – весовые матрицы для входных данных и предыдущего состояния соответственно;

b_i – вектор смещения (bias);

f_i – выход ворот забывания.

LSTM широко используются для различных задач, связанных с обработкой речи, таких как машинный перевод, генерация текстов, анализ тональности. Соответственно они могут быть использованы для разработки систем распознавания и синтеза речи.

LSTM имеют ряд преимуществ перед обычными RNN, но также могут иметь вычислительную сложность и требовать большого объема данных для обучения. В последние годы появились и другие архитектуры рекуррентных сетей, такие как Gated Recurrent Units (GRU) [15].

3.1.3 Gated Recurrent Units (GRU)

GRU, или Gated Recurrent Unit, является одним из видов рекуррентных нейронных сетей (RNN), аналогичным LSTM (Long Short-Term Memory). Данная архитектура была предназначена для упрощения и ускорения обучения по сравнению с LSTM, сохраняя при этом большую часть ее эффективности.

В отличие от LSTM, в GRU нет отдельного долгосрочного состояния ячейки. Состояние ячейки в GRU является комбинацией прошлого состояния и новых входных данных, модулируемых через обновляющие (update gates) и сбрасывающие ворота (reset gates). Это состояние обновляется на каждом шаге и переносит информацию по всей сети.

Ворота обновления в GRU контролируют, насколько новая информация будет влиять на состояние ячейки. Они определяют, какую часть прошлого состояния следует сохранить и какую часть текущего ввода следует учесть. Ворота сброса помогают модели решить, какая информация из прошлого необходима для текущего выхода. Если ворота сброса близки к нулю, то это означает, что прошлая информация не будет учитываться, что позволяет модели отбросить нерелевантные данные.

В GRU нет отдельной ячейки памяти, поэтому она обновляет скрытое состояние напрямую. Ворота обновления помогают модели решить, какая часть предыдущего скрытого состояния должна быть сохранена. Это делается путем комбинирования прошлого состояния и новой информации.

Ворота сброса определяют, какая часть прошлого состояния важна для текущего ввода, а ворота обновления используют эту информацию для формирования нового скрытого состояния. Это позволяет GRU более гибко регулировать поток информации (рисунок 6).

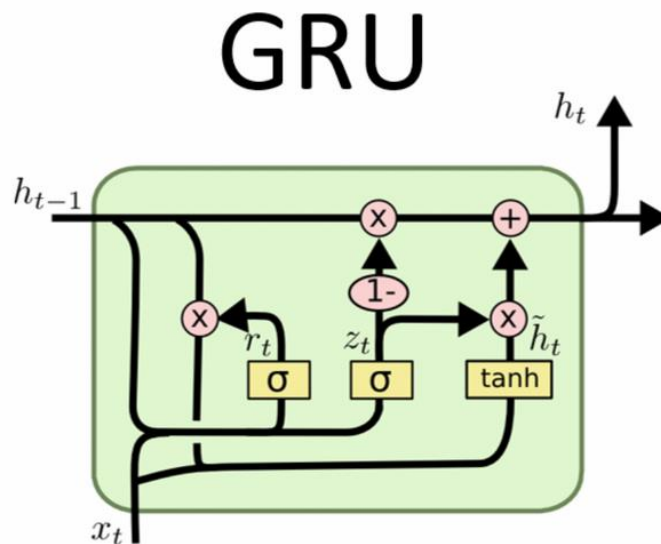


Рисунок 6 – Gated Recurrent Unit

GRU выполняет серию математических операций для обновления своего состояния, включая использование сигмоидной функции для ворот и гиперболического тангенса для создания новых кандидатов на состояние. Эти операции помогают GRU динамически регулировать поток информации [16].

3.2 Анализ архитектур нейронных сетей для синтеза речи из текста

При синтезе звука речи из текста входящими данными является текст. Он может представлять собой символы алфавита, специальные фонетические значки или даже некоторые управляющие символы, описывающие, например, эмоциональную окраску произносимой речи. Обычно такие символы переводят за счет обучения в векторное представление, тогда входом для системы синтеза будет последовательность таких векторов. Результатом является звуковая волна, описывающая синтезированную речь [17].

Для синтеза речи использовали два основных подхода:

- 1) конкатенативный синтез. Основан на соединении фрагментов записанной речи в нужном порядке для создания целых слов и предложений,
- 2) параметрический синтез. Задача синтеза решается в два этапа: акустическая модель и вокодер. Сначала по тексту предсказываются некоторые параметры речи – чаще всего спектрограмма. Затем она превращается в звук с помощью другой модели – вокодера [18].

Данный процесс включает несколько этапов. Сначала текст приводится в нормализованный вид. В нем определяются ударения, аббревиатуры, числа. Затем с помощью энкодера текст преобразуется в числовое представление, сохраняющее смысл и контекст подаваемой информации. Далее с помощью специального компонента модели синтеза речи осуществляется прогнозирование времени звучания фонем в тексте, сколько кадров требуется для каждого символа, и повторяет это представление нужное количество раз (рисунок 7) [19].

Этот модуль нужен только для неавторегрессионных моделей (FastSpeech, Vits), а для авторегрессионных (Tacotron 2, Valle) – не нужен. В Tacotron 2 длительность фонем часто контролируется напрямую через механизм внимания – модель решает, когда переходить от одной фонемы к другой, на основе контекста.

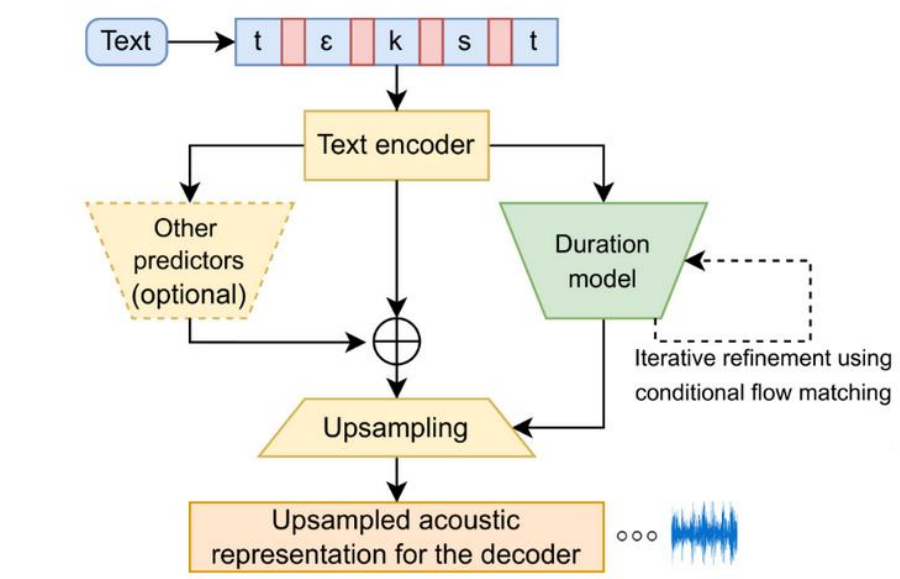


Рисунок 7 – Преобразования текста до попадания в декодер

Далее осуществляется генерация визуального представления звука с помощью декодера.

Декодер – это часть нейросети, которая преобразует эмбединги в мел-спектрограмму. Мел-спектрограмма – это визуальное представление звукового сигнала, отображающее его частоты во времени (рисунок 8). Она служит основой для последующего преобразования в аудиосигнал.

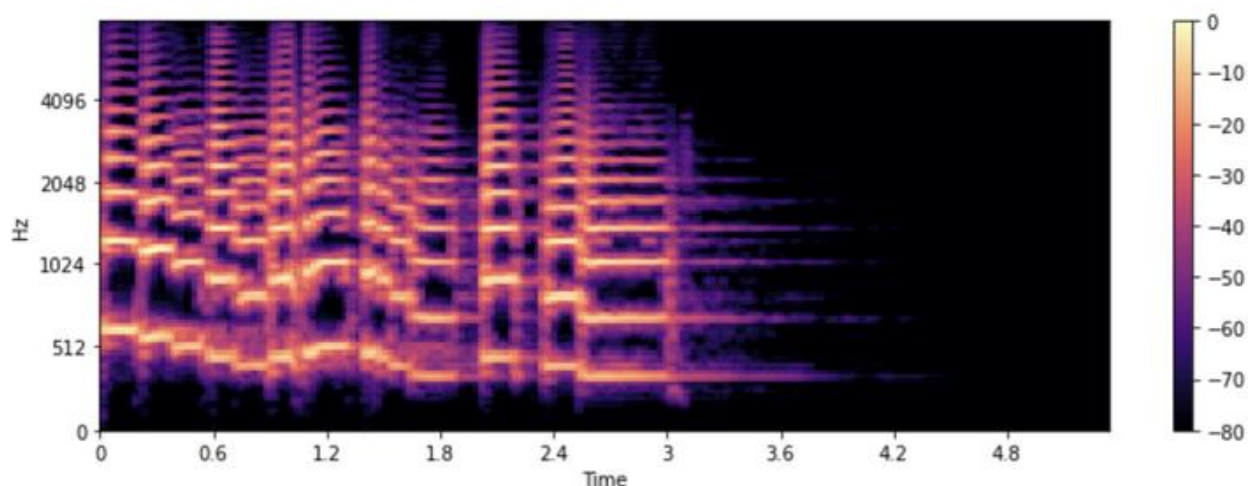


Рисунок 8 – Мел-спектрограмма библиотеки librosa

Финальным шагом является преобразование мел-спектрограммы в звуковой сигнал с помощью вокодера, представляющего собой специальный алгоритм [18].

Основными моделями для синтеза речи сейчас являются Tacotron, VITS и FastSpeech2.

3.2.1 Tacotron

Tacotron2 представляет собой последовательную модель преобразования "текст-в-спектрограмму", построенную по принципу encoder-decoder. Входные текстовые токены сначала обрабатываются кодировщиком, формирующим их скрытое представление, которое затем декодер преобразует в мел-спектрограмму. Особенностью модели является PostNet - дополнительная нейросетевая структура, выполняющая постобработку и улучшение сгенерированной спектрограммы.

На начальном этапе обработки входных данных модель использует слой векторного представления (Embedding), который преобразует последовательность числовых идентификаторов символов в плотные 512-мерные векторные представления. Эти эмбединги затем проходят через каскад из трех одномерных сверточных слоев (по 512 фильтров размером 5 каждый). Выбор именно такого размера фильтра (5) обусловлен оптимальным охватом контекста - центрального символа и двух соседних с каждой стороны.

Каждая сверточная операция сопровождается пакетной нормализацией и активационной функцией ReLU.

Обработанные сверточными слоями данные затем поступают на двунаправленную LSTM-структуру (256 нейронов в каждом направлении), где информация анализируется в прямом и обратном порядке, а результаты обоих проходов объединяются для формирования итогового представления.

Декодер имеет рекуррентную архитектуру, то есть на каждом последующем шаге используются выходные данные с предыдущего шага (рисунок 9) [20].

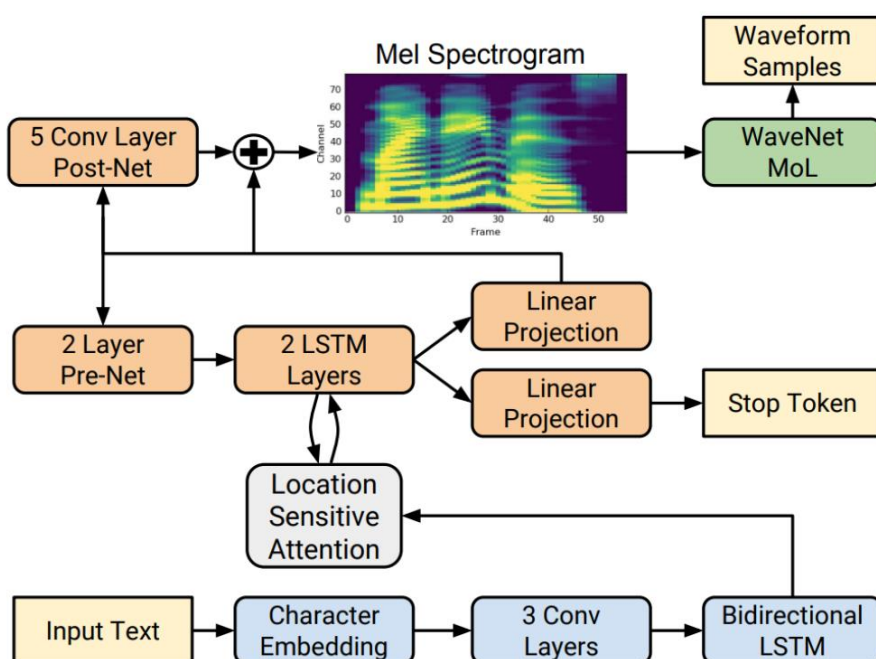


Рисунок 9 – Архитектура Tacotron

3.2.2 VITS

Модель VITS (Variational Inference with adversarial learning for end-to-end Text-to-Speech) представляет собой модель для синтеза речи, использующую комбинацию вариационных автоэнкодеров (VAE), нормализующих потоков и состязательного обучения (GAN) для прямого преобразования текста в речь без промежуточных этапов.

Основным компонентом VITS является вариационный автоэнкодер, состоящий из трех модулей: постериорного кодировщика, декодера и априорного распределения. Для обработки текста используется

трансформерный кодировщик, который преобразует входные последовательности в скрытые представления. Затем потоковый модуль (flow-based module), состоящий из нескольких связующих слоёв, предсказывает акустические признаки на основе мел-спектрограмм (рисунок 10).

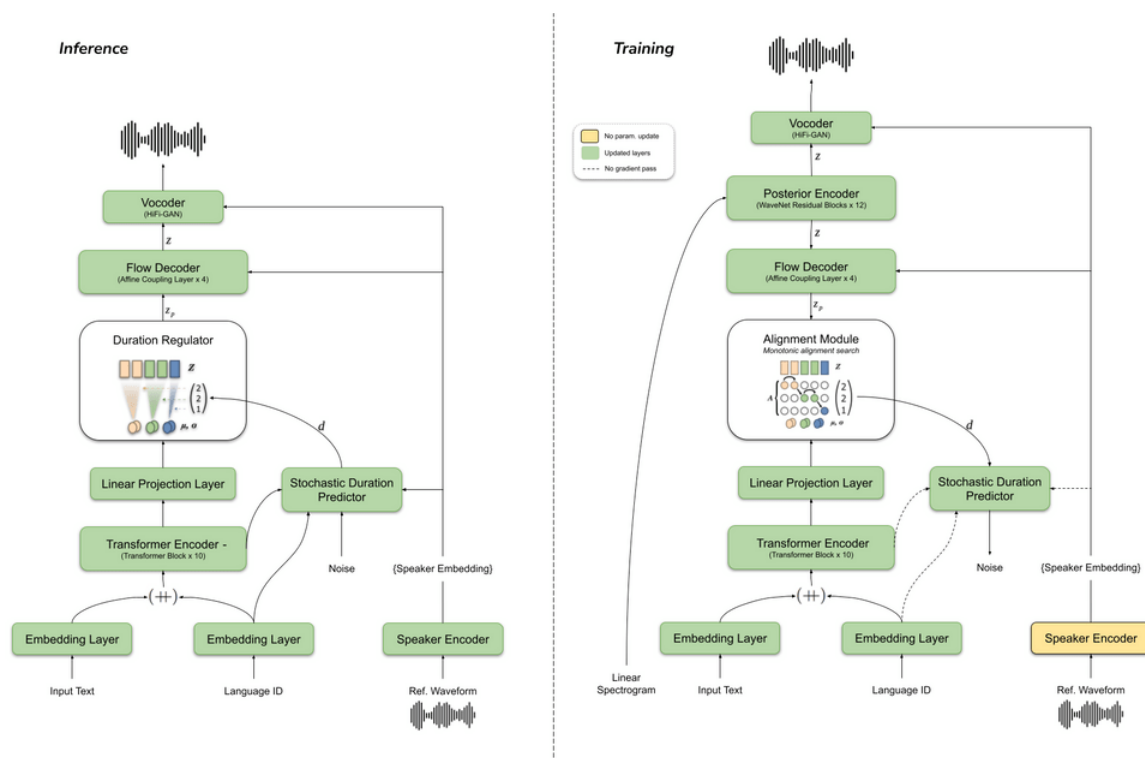


Рисунок 10 – Синтез речи и обучение в архитектуре VITS

Одна из ключевых особенностей VITS – стохастический предсказатель длительности, позволяющий голосовой модели генерировать речь с разной ритмикой и интонацией из одного и того же текста. Это делает синтезированную речь более естественной.

Модель обучается энд-ту-энд с использованием комбинации двух типов потерь: вариационной нижней границы (ELBO) и состязательных потерь (GAN). Нормализующие потоки применяются к априорному распределению, что позволяет улучшить его выразительность и точность.

На этапе синтеза текст сначала кодируется в скрытое представление, после чего предсказатель длительности определяет продолжительность каждой фонемы. Затем потоковый модуль и HiFi-GAN декодер преобразуют эти данные в итоговую звуковую волну. Из-за стохастичности предсказания

длительностей для обеспечения воспроизведения результатов используется фиксированный сид (seed) [21].

3.2.3 FastSpeech2

FastSpeech2 – это неавторегрессионная модель преобразования текста в речь (TTS), разработанная для синтеза речи быстрее и с более высоким качеством по сравнению с предыдущими моделями. Она устраняет ограничения FastSpeech, используя эталонные данные для обучения и включая дополнительную информацию о вариациях речи, такую как высота тона, энергия и точная длительность.

Данная модель обучается напрямую на эталонных мел-спектрограммах, высоте тона, энергии и точной длительности, извлечённых из звуковых сигналов. Модель использует упрощённый процесс обучения без необходимости в обучающей модели и дистилляции.

FastSpeech2 учитывает детальную информацию о вариациях речи, чтобы лучше решать проблему «один-ко-многим» (one-to-many mapping).

Архитектура включает кодировщик (encoder), адаптер вариаций (variance adaptor) с предсказателями длительности, высоты тона и энергии, декодер мел-спектрограмм (рисунок 11).

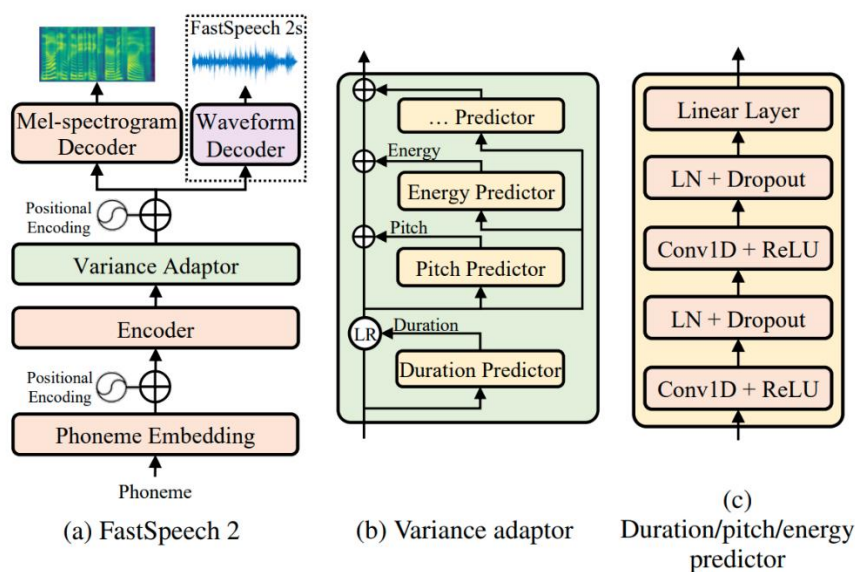


Рисунок 11 – Архитектура FastSpeech2

FastSpeech 2s добавляет декодер звуковых волн, который генерирует речь напрямую из текста. Адаптер вариаций использует непрерывное вейвлет-преобразование для предсказания высоты тона, а в декодере звука применяется состязательное обучение (adversarial training) для восстановления фазы.

Его ключевыми особенностями является использование фонемного кодирования, адаптации вариаций (с предсказанием длительности, высоты тона и энергии) и декодирование мел-спектрограмм/звука для генерации высококачественной речи.

Звуковые данные конвертируются в мел-спектрограммы с определёнными размерами кадра и шага, а высота тона и энергия извлекаются из аудиосигнала для обучения предсказателей вариаций [22].

3.3 Анализ вокодеров

Вокодер представляет собой алгоритм, который преобразует мел-спектрограмму, полученную от декодера, в итоговый звуковой сигнал. Это последний шаг в процессе синтеза речи.

Основными методами преобразования являются

- flow-based-модели. Например, модель WaveGlow использует flow-based (потокковое) программирование для преобразования мел-спектрограмм в аудиосигналы с высоким качеством,
- generative Adversarial Networks (GAN). Архитектуры, такие как MelGAN и HiFi-GAN, используют GAN-фреймворк, в котором одна модель генерирует звук, а другая оценивает его качество [18].

3.3.1 WaveGlow

WaveGlow – это генеративный вокодер, способный генерировать качественные речевые волновые формы из мел-спектрограмм. WaveGlow получил это имя, поскольку сочетает в себе идеи из Glow (потокковая генеративная модель, созданная OpenAI в 2018 году) и WaveNet (другой

вокодер) для того, чтобы обеспечить быстрое, эффективное и высокое качество звучания.

WaveGlow реализован с использованием единой сети, обученной только с использованием единственной функции потерь: максимизации вероятности обучения данных, что делает процедуру обучения простой и стабильной. Архитектура WaveGlow представлена на рисунке 12:

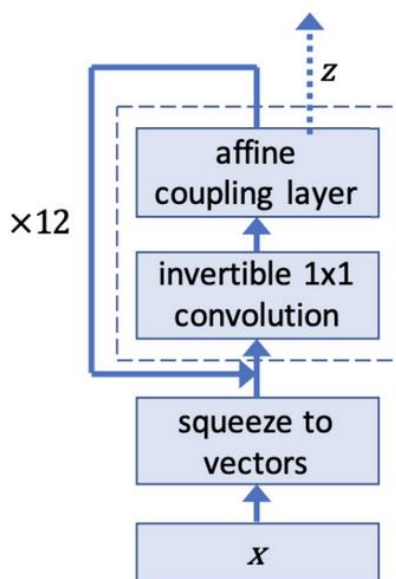


Рисунок 12 – Архитектура WaveGlow

Для прямого прохода (forward pass) эта модель берет группы из 8 аудиосэмплов в виде векторов и выполняет операцию "сжатия" (squeeze).

Таким образом, если на вход подается тензор размера $s \times s \times c$, оператор сжатия берет блоки размером $2 \times 2 \times c$ и преобразует их в блоки $1 \times 1 \times 4c$. Это преобразование легко обратить с помощью изменения формы (reshape). Затем модель обрабатывает входные данные через несколько "шагов потока" (steps of flow). Каждый шаг потока включает:

- обратимую свертку 1×1 (invertible 1×1 convolution),
- аффинный слой связи (affine coupling layer).

Оба этих компонента будут рассмотрены подробнее далее.

Модель обучается путем прямой минимизации следующей функции потерь

$$\log p(x) = \frac{z(x)^T * z(x)}{2 * \sigma^2} + \sum_{k=0}^{\#conv} \log \det(|W_k|) + \sum_{j=0}^{\#coupling} \log s_j(x, melspectrogram) ,$$

где $z(x)$ – это латентное представление входных данных x ;

σ^2 – предполагаемая дисперсия гауссовского распределения;

W_k – матрица весов k -го слоя свёрточной сети;

$\det(|W_k|)$ – определитель модуля матрицы W_k ;

$\#conv$ – количество свёрточных слоёв;

$s_j(x, melspectrogram)$ – функция масштабирования;

$\#coupling$ – количество связывающих слоёв.

Первое слагаемое соответствует логарифму правдоподобия сферического гауссовского распределения с нулевым средним (подробности см. в статье Glow). Остальные слагаемые учитывают изменение переменных в аффинных слоях связи и обратимых свертках соответственно [23].

3.3.2 HiFi-GAN

HiFi-GAN – это современная генеративная модель, разработанная для синтеза высококачественного аудио. Она использует архитектуру генератора и дискриминатора. Модель особенно полезна при создании синтезированной речи, музыки и других звуковых сигналов.

Генератор – это полносверточная нейронная сеть (FCN), которая принимает на вход мел-спектрограмму и увеличивает её разрешение с помощью транспонированных сверток (transposed convolutions), пока длина выходной последовательности не совпадёт с исходным аудиосигналом. Далее после каждой транспонированной свертки применяется модуль Multi-Receptive Field Fusion (MRF), который объединяет признаки с разными рецептивными полями.

Архитектура дискриминаторов включает компоненты:

– многопериодный дискриминатор (MPD). Он состоит из нескольких суб-дискриминаторов, каждый из которых анализирует определённую

частотную полосу входного аудиосигнала и позволяет оценивать периодические структуры в звуке.

– Многомасштабный дискриминатор (MSD). Работает с аудио на разных уровнях разрешения (исходный сигнал и даунсэмплированные версии) и помогает улавливать долгосрочные зависимости и естественность звучания.

Помимо стандартного состязательного обучения (GAN loss), используются:

- Feature Matching Loss – согласование признаков между реальным и сгенерированным аудио,
- Mel-Spectrogram Loss – минимизация разницы между мел-спектрограммами целевого и синтезированного сигнала.

Эти лоссы стабилизируют обучение и улучшают качество синтеза (рисунок 13) [24].

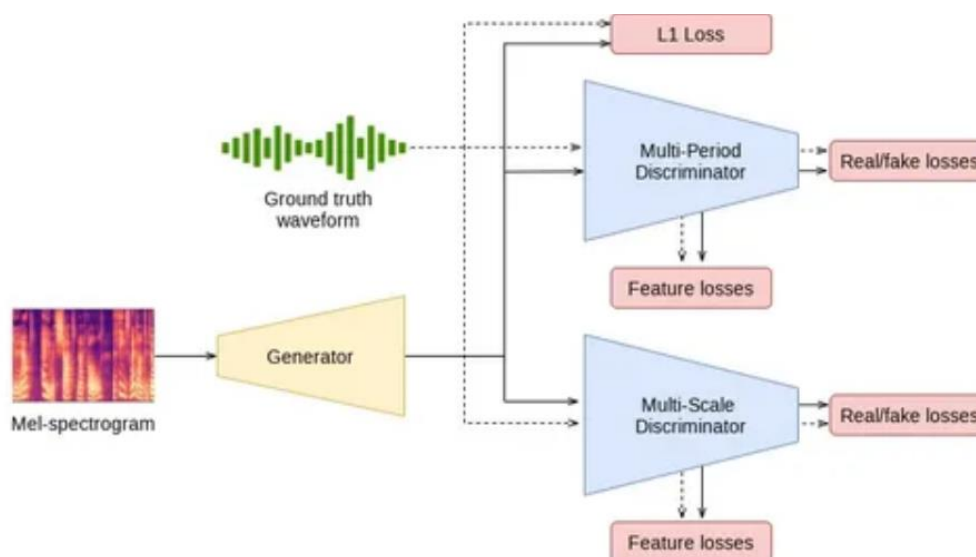


Рисунок 13 – Архитектура HiFi-GAN

HiFi-GAN выделяется высоким качеством синтеза звука, обеспечивая чистый и естественный аудиосигнал с минимальными шумами и искажениями. Эта модель применяется как для генерации речи, так и для создания музыкальных композиций. Благодаря оптимизированной архитектуре, HiFi-GAN выполняет синтез с высокой скоростью, что делает её удобной для практического использования. Кроме того, модель поддерживает

различные аудиоформаты, включая WAV и MP3, что расширяет её совместимость с разными системами и приложениями. [25].

3.4 Обоснование выбранных технологий для разработки приложения

Проведя анализ моделей для синтеза речи, были сформированы следующие выводы, представленные в таблице 3.

Таблица 3 – Анализ архитектур для синтеза речи

Параметр	Tacotron 2	FastSpeech 2	VITS
Тип генерации	Авторегрессионная	Неавторегрессионная	Неавторегрессионная
Скорость синтеза	Медленная	Быстрая	Быстрая
Вокодер	Требуется (WaveNet)	Требуется (HiFi-GAN)	Встроенный (HiFi-GAN)
Качество	Высокое	Высокое	Высокое
Проблемы	Пропуски слов, медленный синтез	Зависит от выравнивания длительностей	Сложность обучения
Многоязычность	Ограниченная	Поддерживается	Поддерживается

Учитывая ограниченные вычислительные ресурсы, целесообразным будет использование архитектуры VITS или FastSpeech2, однако FastSpeech2 не имеет полностью русскоязычных предобученных моделей, поэтому в качестве примера сравнили качество озвучивания русскоязычной модели (рисунок 14) и мультязычной модели (рисунок 15) с использованием библиотеки transformers. В качестве примера был озвучен текст «Ночь, улица, фонарь, аптека, Бессмысленный тусклый свет. Живи ещё хоть четверть века - Все будет так. Исхода нет.» и выведен в виде спектрограмм.

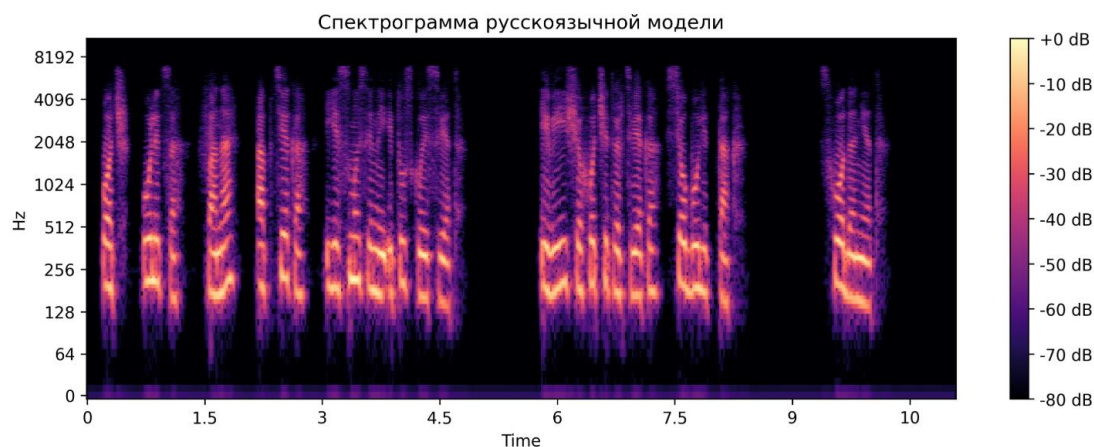


Рисунок 14 – Спектрограмма русскоязычной модели

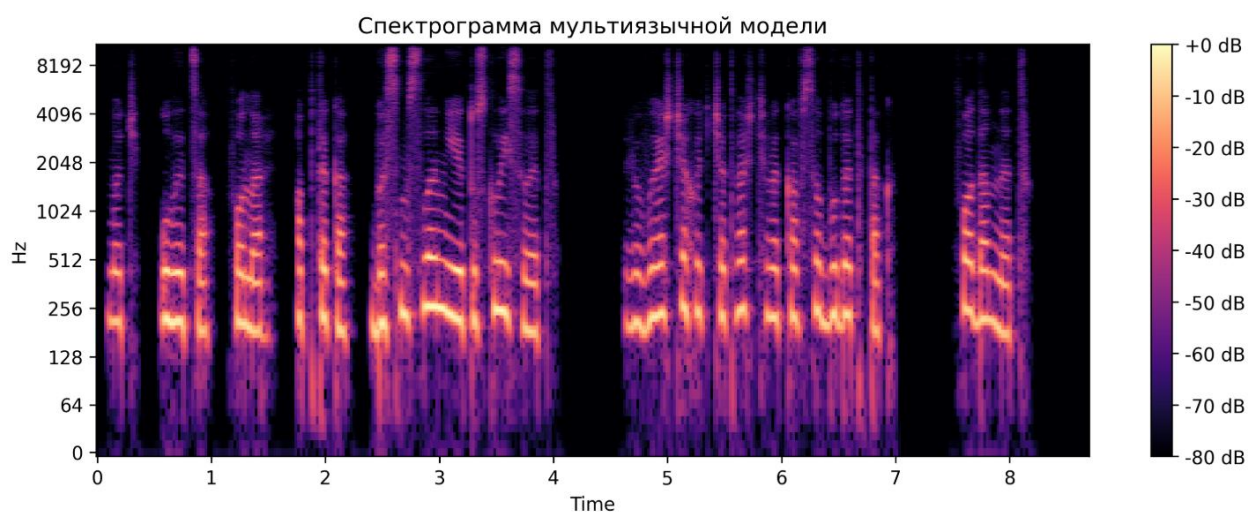


Рисунок 15 – Спектрограмма мультязычной модели

Судя по полученным результатам, мультязычная модель имеет больше артефактов и шумов, которые можно судить по резким перепадам в верхней и нижней части диапазона.

Таким образом целесообразно использование модели VITS, поскольку позволяет быстро и качественно синтезировать речь. Кроме того, данное решение имеет встроенный вокодер и предобученные модели на русском языке.

Для упрощения разработки голосового помощника для озвучивания текстов с использованием нейронных сетей за основу была взята библиотека transformers.

Transformers – это Python-библиотека, предназначенная для обучения современных моделей в сфере NLP (Natural Language Processing). В её основе

лежит концепция использования предобученных трансформерных архитектур. В данной библиотеке доступны различные варианты таких моделей, отличающиеся структурой, размером и конфигурацией, в том числе и основанные на архитектуре VITS. Для работы с трансформерами входные данные должны быть предварительно преобразованы в токенизированный вид [26].

Для обучения модели должен быть подготовлен набор данных, состоящий из аудиофайлов формата .wav и метаданных в формате файлов .txt или .csv, содержащих наименование файла и транскрипции на русском языке.

Чтобы упростить работу бекенд и фронтенд частей веб-приложения между собой будет использован фреймворк FastAPI.

4 Техническое задание

4.1 Основание для разработки

Разработка голосового помощника для озвучивания текста с использованием нейронных сетей ведется на основании задания, выданного руководителем дипломного проекта кандидатом технических наук, доцентом кафедры ИВС Кузнецовой О.Ю. и утвержденного заведующим кафедрой ВТ Митрохиным М.А.

4.2 Назначение разработки

Программное обеспечение должно быть предназначено для озвучивания текстов различными моделями голосов и упрощения создания медиа-контента.

4.3 Требования к программе

4.3.1 Требования к функциональным характеристикам

Программное обеспечение должно реализовывать следующие функции:

- выбор модели голоса,
- настройка высоты, громкости и скорости воспроизведения,
- генерация аудиодорожки или аудиофайла с озвучкой в форматах .mp3 или .wav,
- ввод текста,
- распознавание ударений в словах и знаков препинаний,
- синтезирование речи.

Программное обеспечение должно иметь интуитивно понятный интерфейс, обеспечивать комфортную работу для пользователей, не имеющих опыта работы с персональным компьютером.

4.3.2 Требования к надежности

Программное обеспечение должно работать без перебоев и предоставлять отчет об ошибках. Сбои при работе программного обеспечения не должны приводить к экономическому ущербу.

При вводе входной информации должен осуществляться её контроль на допустимость вводимых значений: данные, имеющие формат даты, строки,

числа и т.д., должны заноситься в систему только если они соответствуют формату, ненормативная лексика также не должна озвучиваться. В случае ввода недопустимых данных должно выводиться сообщение об ошибке ввода.

При выполнении операций по изменению и удалению данных должна быть обеспечена целостность базы данных.

Программное обеспечение должно обеспечивать защиту данных от несанкционированного доступа на основе процедур аутентификации пользователей и разграничения прав доступа к данным.

4.3.3 Требования к составу и параметрам технических средств

Программное обеспечение должно быть предназначено для использования как на локальном компьютере, так и в сети. Рабочие станции должны иметь минимальные технические характеристики, соответствующие минимальным требованиям для работы современных браузеров.

К рабочей машине должны быть подключены стандартные мышь и клавиатура.

4.3.4 Требования к информационной и программной совместимости

Разработка программного обеспечения должна осуществляться в среде разработки Microsoft Visual Studio Code.

Разработанное приложение должно работать под управлением операционной системы Windows 10 и выше.

4.4 Требования к программной документации

Программное обеспечение должно сопровождаться пояснительной запиской, включающей следующую программную документацию:

- техническое задание,
- описание программы,
- программа и методика испытаний,
- описание применения.

Структура и содержание программной документации соответствовать требованиям стандартам Единой системы программной документации (ЕСПД).

4.5 Стадии и этапы разработки

Разработка программного обеспечения должна осуществляться в соответствии со следующими этапами:

- предпроектное исследование,
- разработка и утверждение технического задания,
- разработка технического проекта,
- разработка и отладка программного обеспечения,
- испытания программного обеспечения,
- внедрение программного обеспечения.

4.6 Порядок контроля и приемки

Для проверки правильности работы программного обеспечения должны быть разработаны тестовые примеры, в которых должны быть учтены все возможные сочетания входных данных, отражающие работу всех его частей. Должны быть протестированы действия программного обеспечения на возможные реакции пользователя: запуск программы, заполнение полей ввода, выбор пункта меню, нажатие кнопок.

Приёмка программного обеспечения должна осуществляться путём анализа работоспособности и правильного выполнения всех возложенных на него функциональных возможностей при использовании в качестве входных данных, разработанных тестовых наборов, правильном выполнении возложенных на него функций, наличия пояснительной записки, включающей полный комплект программной документации.

5 Разработка программного обеспечения

5.1 Диаграмма вариантов использования

При проектировании программного продукта необходимо определить функциональные возможности, взаимодействующие лица и взаимосвязи между ними. StarUML может позволить создать все необходимые UML-диаграммы, для определения структуры и функционала будущего приложения.

Диаграмма вариантов использования отображает функционал будущего продукта и лица, которые будут взаимодействовать с ним.

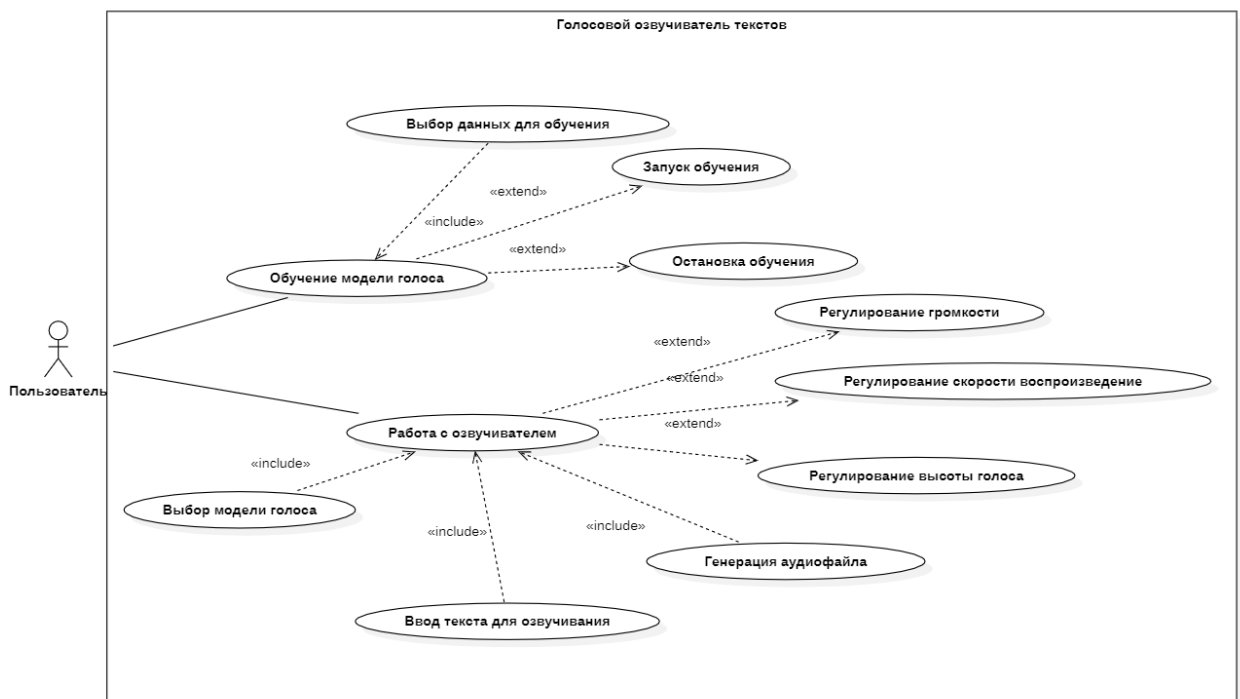


Рисунок 16 – Диаграмма вариантов использования

Исходя из данной диаграммы, пользователь будет иметь два функциональных блока. Первый блок связан с обучением модели и позволяет выбрать данные для обучения, а также запускать и останавливать обучение. Второй блок связан с озвучиванием текста уже обученной моделью. Здесь у пользователя имеется возможность настраивать параметры, генерировать файл, выбирать модель голоса и вводить текст для дальнейшего озвучивания.

5.2 Диаграмма развертывания

Диаграмма развёртывания используется для того, чтобы визуализировать аппаратные узлы системы, каналы связи между ними и размещение программных файлов. В данном случае демонстрируется архитектура, состоящая из двух серверов, для отображения внешней оболочки веб-страницы и внутренней части, и взаимодействие файлов в них.

Благодаря описанным UML-диаграммам вариантов использования и развёртывания описана структура и функционал разрабатываемой системы.

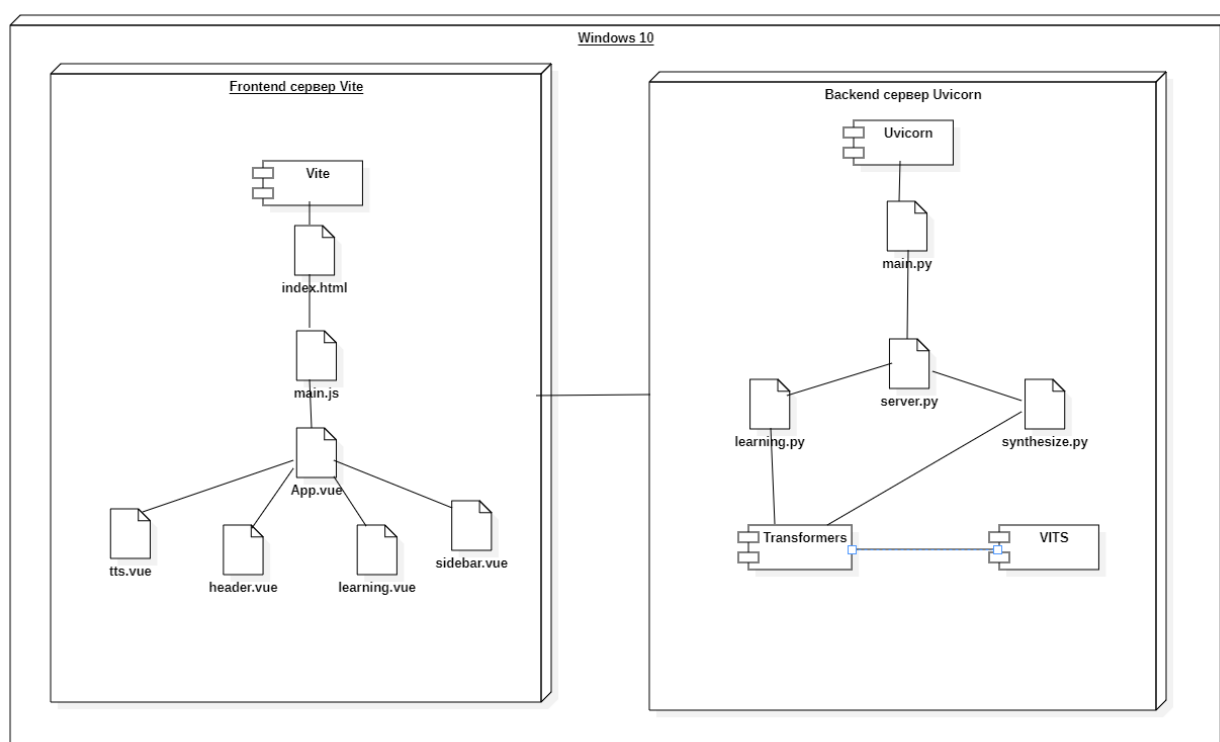


Рисунок 17 – Диаграмма развертывания

5.3 Разработка структуры приложения

На основе функциональной модели системы определены основные функции, которые должно выполнять программное обеспечение. Описание функций программного обеспечения представлено в таблице 4.

Таблица 4 – Функции системы

Уровень	Наименование функции	Описание функции
1	Открытие приложения	Осуществить подключение к веб-приложению через браузер

Продолжение таблицы 4

Уровень	Наименование функции	Описание функции
2	Взаимодействие с функционалом обучения модели	Взаимодействие с функционалом обучения голосовой модели для синтеза речи
2.1	Загрузка файлов для обучения	Выбрать каталог с аудиофайлами для обучения
2.2	Обучение модели	Осуществить обучение модели на основе загруженных файлов
3	Взаимодействие с функционалом синтеза речи из текста	Осуществить изменение данных о грузе
3.1	Настройка параметров	Выбрать громкость, высоту и скорость воспроизведения, а также формат выводимого аудиофайла
3.2	Выбор модели	Осуществить выбор голосовой модели для озвучивания
3.3	Генерация аудиофайла с озвучкой текста	Осуществить формирование аудиофайла исходя из введенного текста и параметров

На основе определенных функций разработана схема «функции-модули», представленная на рисунке 18.

Список функций и модулей представлен в таблице 5.



Рисунок 18 – Схема функции-модули

Таблица 5 – Списки функций и модулей

Функция	Модуль
Обучение нейросетевой модели голоса	learning
Синтез голоса из текста	tts
Загрузка аудиофайлов	loadAudio

6 Описание программы

6.1 Общие сведения

Программное обеспечение предназначено для автоматизации бизнес-процессов транспортной компании.

Программное обеспечение написано на языках программирования Python и JavaScript с применением фреймворка Vue.js и реализовано в среде разработки Visual Studio Code.

6.2 Функциональное назначение

Программное обеспечение обеспечивает выполнение следующих функций:

- обучение модели голоса на основе загруженных аудиофайлов,
- синтез речи из текста,
- контроль ввода информации.

6.3 Используемые технические средства

Программное обеспечение должно быть разработано как клиент-серверное приложение.

В качестве сервера должен использоваться персональный компьютер с характеристиками не ниже следующих:

- процессор с тактовой частотой не ниже 2,8 ГГц,
- оперативная память должна быть не менее 16 Гб,
- объем свободного места на жестком диске должен быть не менее 8 Гбайтов.

В качестве рабочего места должен использоваться персональный компьютер с характеристиками, позволяющими открыть и работать в интернет-браузере.

6.4 Вызов и загрузка

Для запуска программного обеспечения необходимо запустить фронтенд и бекенд сервера localhost.

6.5 Входные данные

Входными данными являются:

- путь к файлам для обучения,
- путь к файлам голосовой модели,
- значение громкости,
- значение высоты голоса,
- значение скорости речи.

6.6 Выходные данные

Выходными данными бизнес-процесса «Взаимодействие пользователя с голосовым помощником» являются:

- обученная модель голоса,
- аудиофайл с озвученным текстом.

7 Программа и методика испытаний

7.1 Объект испытаний

Объектом испытаний является программное обеспечение, предназначенное для озвучивания текстов с использованием нейронной сети.

7.2 Цель испытаний

Целью испытаний является проверка правильности работы программного обеспечения, а также установления факта соответствия выполняемых функций требованию технического задания.

7.3 Средства и порядок испытаний

Проведение испытаний программного обеспечения должно осуществляться на персональных компьютерах с минимальными характеристиками, указанными в техническом задании.

Для проверки правильности работы программного обеспечения должно быть проведено тестирование его работы на основе тестовых наборов данных.

7.4 Методы испытаний

Проверка правильности работы программного обеспечения будет включать в себя проверку функционала, описанного в техническом задании. После открытия программы появляется окно для ввода данных (рисунок 19).

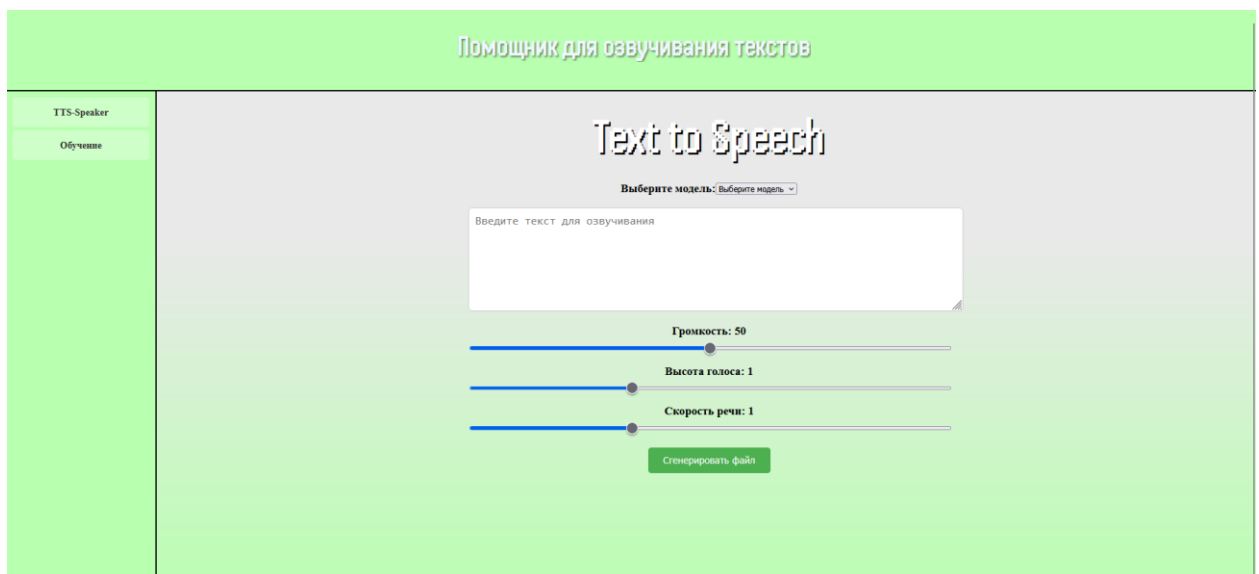


Рисунок 19 – Меню программы

Далее требуется выбрать модель голоса студента нажатием на кнопку «Выбрать модель», после чего должен открыться список, в котором нам нужно выбрать предобученную модель голоса.

С использованием тестового примера в виде текста стихотворения А.А. Блока «Ночь, улица, фонарь, аптека...» и файла модели голоса Joefox-VITS в формате .pth, а также конфигурации в формате JSON. Протестируем вывод аудио-дорожки в формате .wav, а также ввод текста и распознавание знаков препинаний в тексте. Требуется выбрать настройки: скорость – 1.0x, громкость – 50, высота – 1. (рисунок 20).

Затем с помощью тестового примера в виде текста: «Замок заперт на замок!» протестируем правильность расстановки ударений в зависимости от контекста предложения моделью голоса UtrobinMV VITS. Также для данного примера для проверки настройки скорости должна быть выбрана скорость 0.8x, для проверки настройки громкости – 30, для проверки высоты голоса, он должен быть понижен на 0.5 (рисунок 21).

По нажатию кнопки «Сгенерировать файл» должен быть сгенерирован аудиофайл в зависимости от выбранных настроек.

Если в ходе испытаний будет получен результат согласно ожиданиям, то это будет значить, что программа работает корректно.

The interface features a title "Text to Speech" in a stylized font. Below it is a dropdown menu labeled "Выберите модель:" with "Joefox VITS" selected. A green status bar indicates "Выбрана модель: models/joefox-vits". A text input area contains the Russian text: "Ночь, улица, фонарь, аптека, Бессмысленный и тусклый свет. Живи ещё хоть четверть века – Всё будет так. Исхода нет." Below the text are three sliders: "Громкость: 50", "Высота голоса: 1", and "Скорость речи: 1". A green button labeled "Сгенерировать файл" is at the bottom.

Text to Speech

Выберите модель: Joefox VITS

Выбрана модель: models/joefox-vits

Ночь, улица, фонарь, аптека,
Бессмысленный и тусклый свет.
Живи ещё хоть четверть века –
Всё будет так. Исхода нет.

Громкость: 50

Высота голоса: 1

Скорость речи: 1

Сгенерировать файл

Рисунок 20 – Входные данные для первого примера

The interface is similar to the first one. The dropdown menu shows "UtrobinMV VITS" selected. The status bar indicates "Выбрана модель: utrobinmv/tts_ru_free_hf_vits_low_multispeaker". The text input area contains: "Замок заперт на замок!". The sliders are set to "Громкость: 30", "Высота голоса: 0.5", and "Скорость речи: 0.8". The "Сгенерировать файл" button is at the bottom.

Text to Speech

Выберите модель: UtrobinMV VITS

Выбрана модель: utrobinmv/tts_ru_free_hf_vits_low_multispeaker

Замок заперт на замок!

Громкость: 30

Высота голоса: 0.5

Скорость речи: 0.8

Сгенерировать файл

Рисунок 21 – Входные данные для второго примера

7.5 Результаты испытаний

Для проверки работоспособности программы, необходимо ввести текст, а также выбрать модель голоса по нажатию на поле «Выбрать модель». Затем выбрать указанные выше значения параметров, после чего нажать на кнопку «Сгенерировать файл».

В ходе испытаний были получены результаты, представленные на рисунках 22-26.

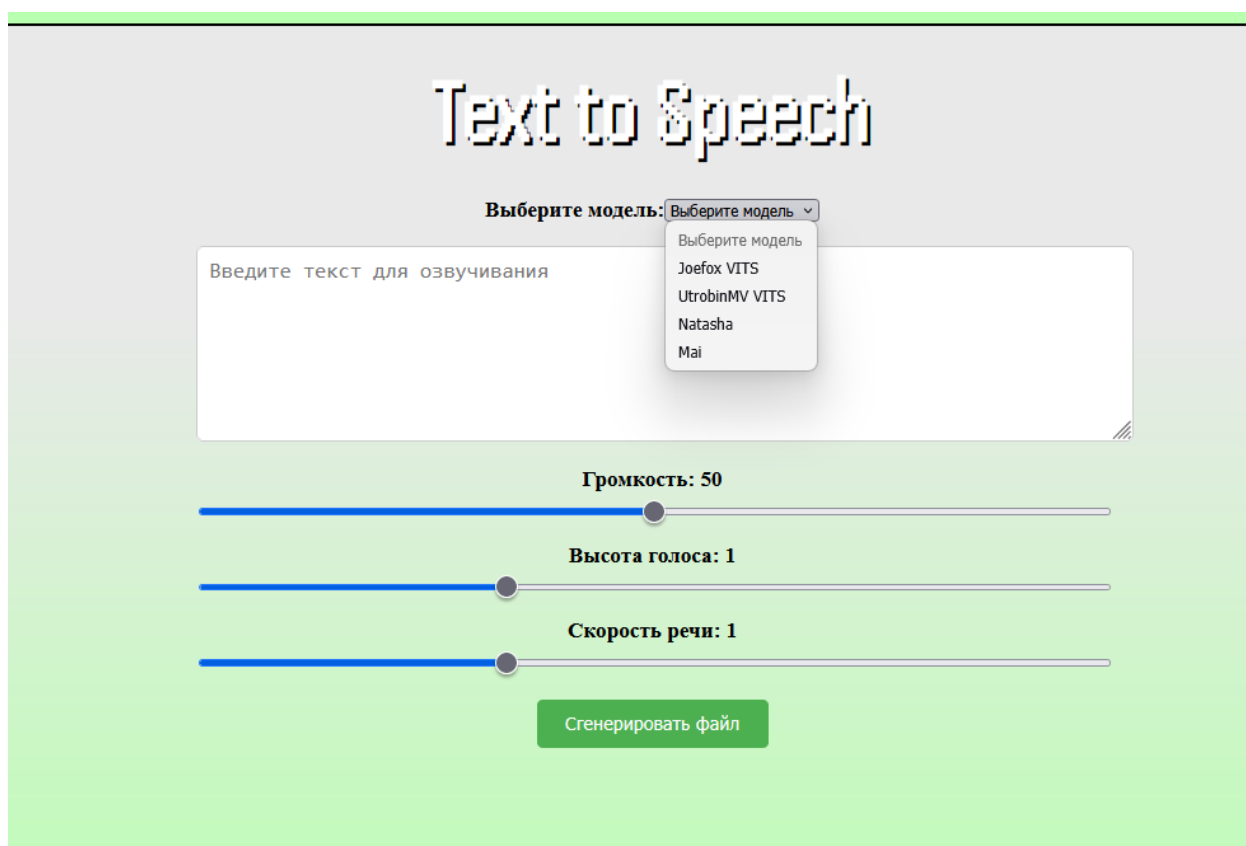


Рисунок 22 – Выбор голосовой модели

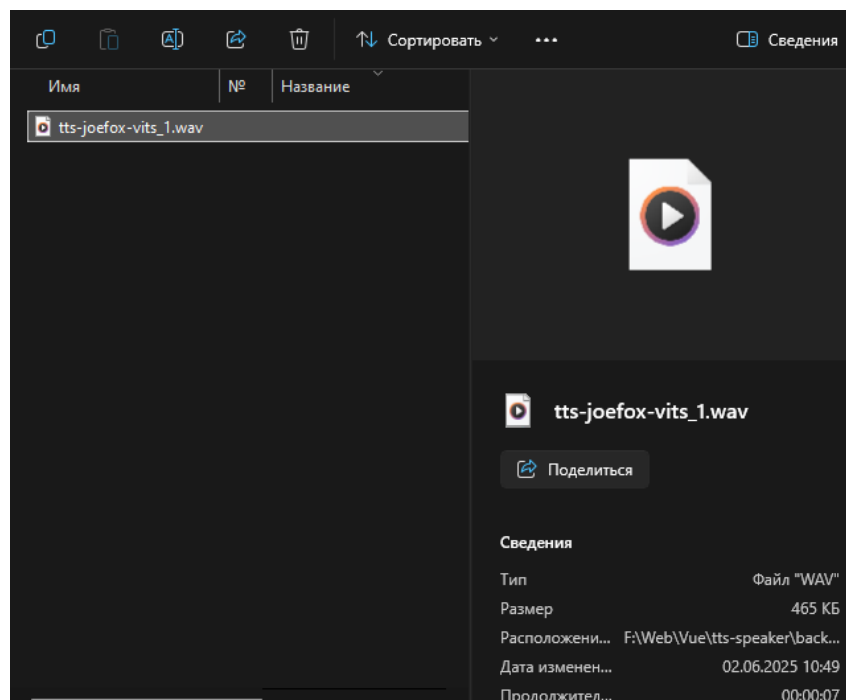


Рисунок 23 – Результат выполнения первого примера

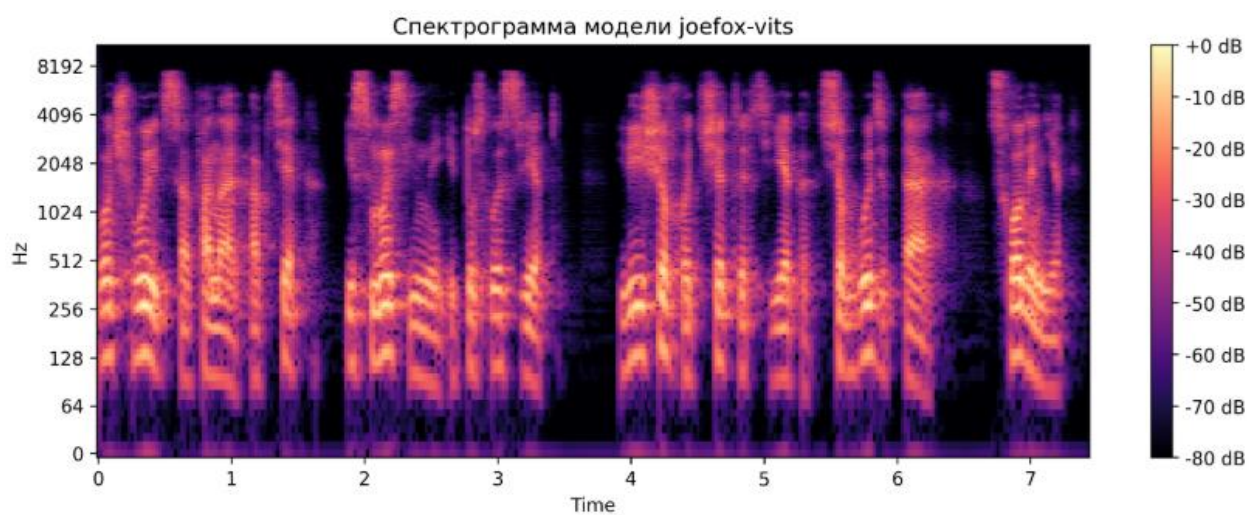


Рисунок 24 – Спектрограмма аудиофайла второго примера

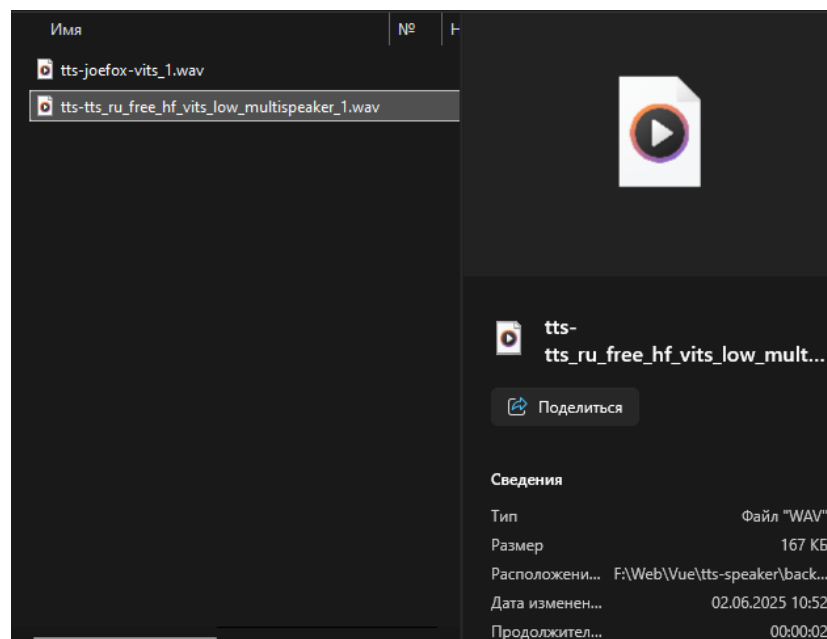


Рисунок 25 – Результат выполнения второго примера

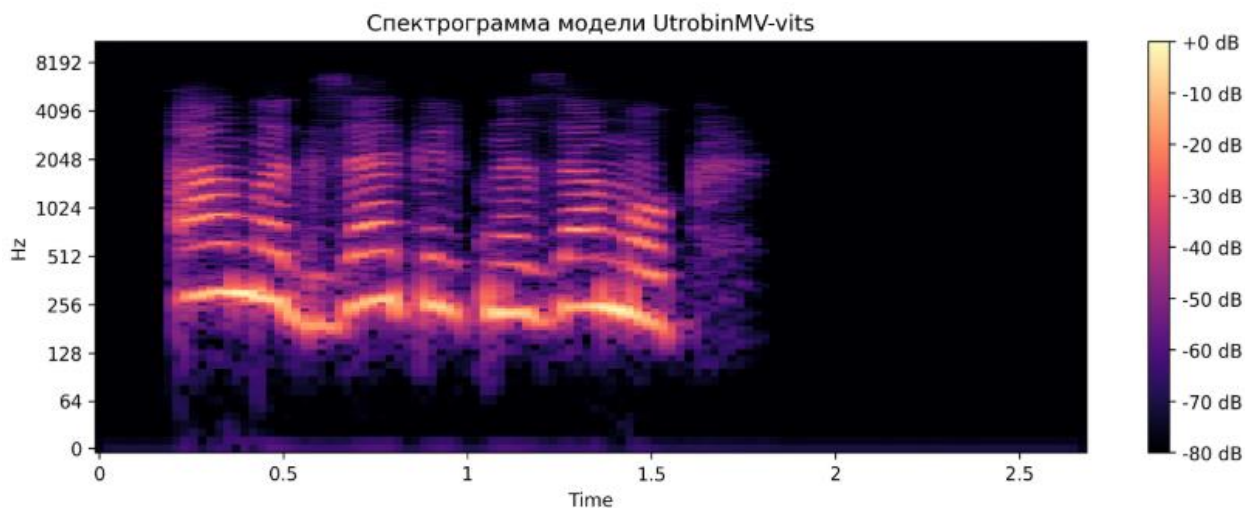


Рисунок 26 – Спектрограмма аудиофайла второго примера

Полученные результаты работы программы, а также полученные спектрограммы, позволяют сделать вывод о том, что программа работает корректно.

8 Описание применения

8.1 Назначение применения

Программное обеспечение предназначено для создания аудиоматериалов из текста с использованием нейронных сетей.

8.2 Условия применения

Программное обеспечение должно быть разработано как клиент-серверное приложение.

В качестве сервера должен использоваться персональный компьютер с характеристиками не ниже следующих:

- процессор с тактовой частотой не ниже 2,5 ГГц,
- оперативная память должна быть не менее 8Гб.

В качестве рабочего места должен использоваться персональный компьютер с характеристиками, удовлетворяющими минимальным требованиям современных браузеров.

8.3 Описание задачи

После запуска программного обеспечения открывается окно с боковой панелью, заголовком и рабочим пространством (рисунок 19).

Для того, чтобы открыть режим синтеза речи из текста пользователю нужно нажать на кнопку “tts-speaker” в левой панели окна (рисунок 27).

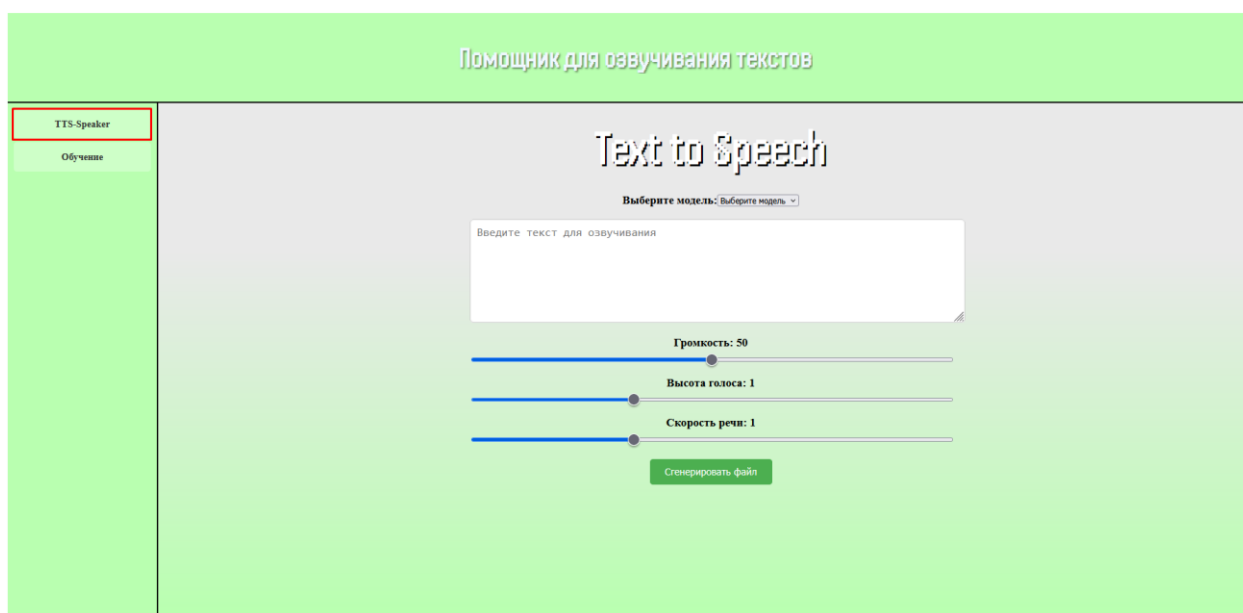


Рисунок 27 – Открытие окна для синтеза речи из текста

В результате открывается окно «Text to Speech» с полями ввода и ползунками для настройки параметров аудиозаписи. Для того, чтобы сгенерировать аудиофайл, пользователю нужно выбрать модель голоса нажатием на поле «Модель голоса» и выбрать модель, а также ввести текст в поле для ввода и нажать на кнопку «Сгенерировать файл» (рисунок 28).



Рисунок 28 – Необходимые поля для генерации аудиофайла

По желанию пользователь может применить параметры для изменения аудиозаписи. Для этого в области с ползунками нужно выбрать необходимые значения громкости, высоты голоса и скорости речи. По умолчанию данные значения составляют 50, 1, 1 соответственно (рисунок 29).



Рисунок 29 – Настройка параметров аудиозаписи

Для того, чтобы открыть панель для обучения модели голоса нужно нажать на кнопку «Обучение» в левой панели окна (рисунок 30).

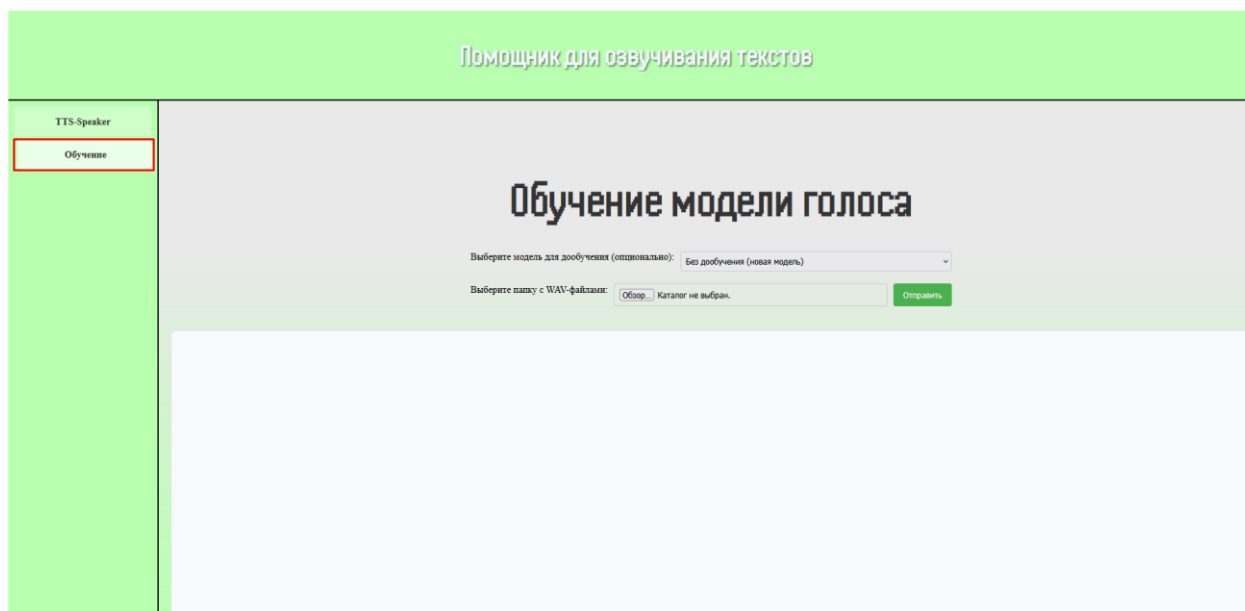


Рисунок 30 – Открытие окна обучения

Далее будет открыто окно для обучения модели голоса. В нем можно дообучить предобученную модель. Для этого нужно выбрать в поле модель для дообучения (рисунок 31).

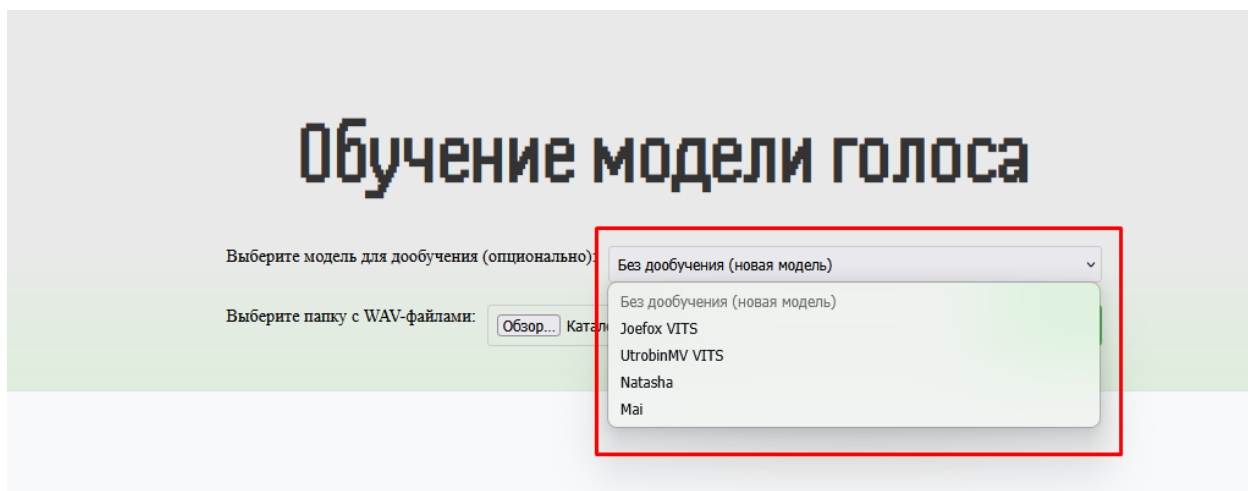


Рисунок 31 – Выбор предобученной модели

Чтобы обучить новую модель, в данном поле не нужно выбирать ничего. Затем нужно выбрать файлы для обучения нажатием на поле выбора (рисунок 32).

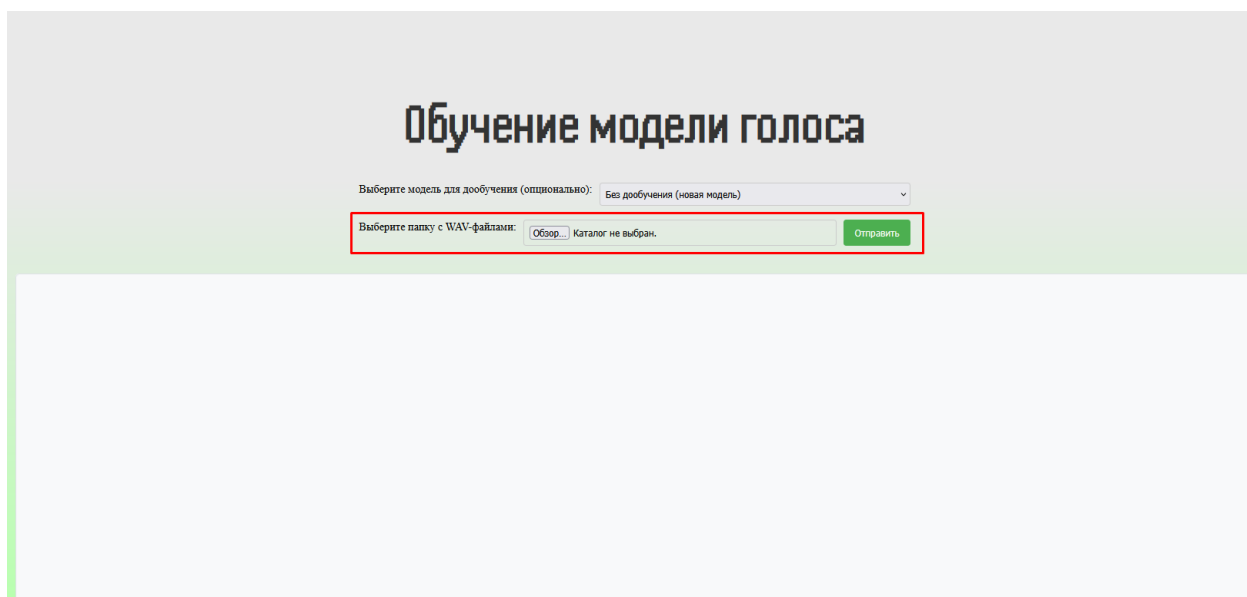


Рисунок 32 – Выбор загружаемых файлов

После чего нужно выбрать папку с .wav файлами, после чего они будут загружены. Затем появится список загруженных файлов и кнопка «Начать обучение», на которую нужно нажать для старта обучения (рисунок 33).

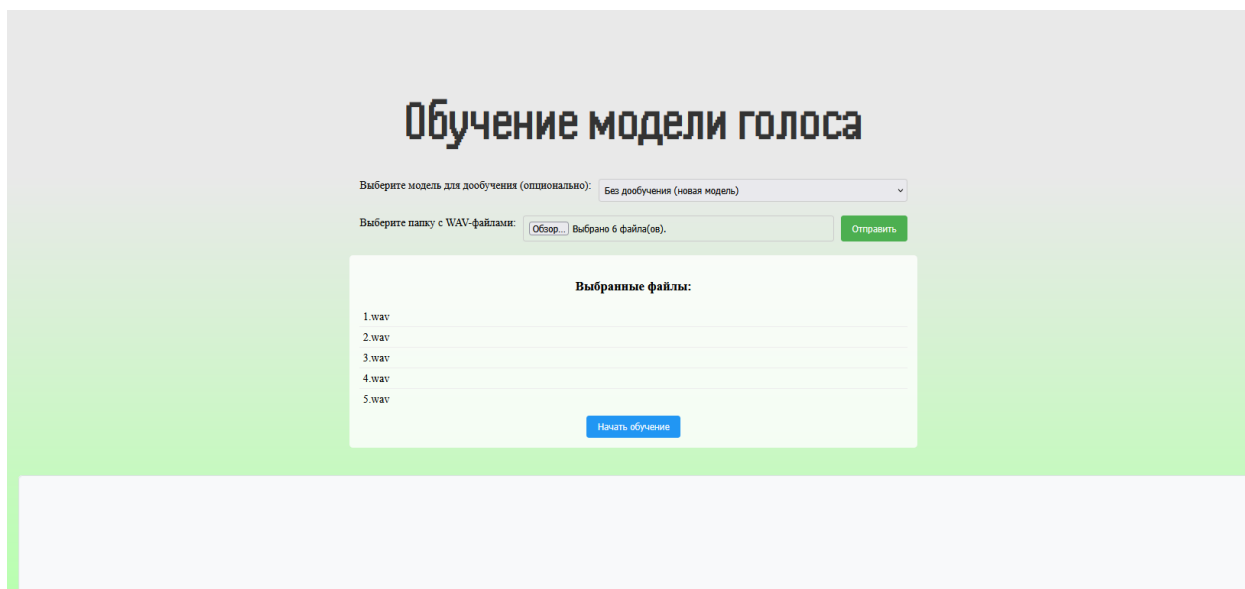


Рисунок 33 – Список загруженных файлов и начало обучения

8.4 Инструкция развертывания

Для запуска backend части приложения требуется установленный Python версии 3.8. Затем перейти в терминале в каталог backend проекта:

```
cd ./tts-speaker/backend
```

Далее запустить файл main.py, написав команду:

```
python -m uvicorn main:app --host 0.0.0.0 --port 8000
```

Для запуска frontend части приложения нужны установленный node.js, vite, Vue.js. Нужно перейти в терминале в каталог Frontend проекта:

```
cd ./tts-speaker/Frontend
```

Затем написать команду для запуска сервера vite:

```
npm run dev.
```

После чего будет возможно использование приложения по адресу <http://localhost:5173/>

ЗАКЛЮЧЕНИЕ

В результате выполнения работы изучена предметная область, была определена актуальность разработки голосового помощника для озвучивания текста с использованием нейросетей. Также был проведен анализ существующих решений, использующих аналогичные технологии и решающих соответствующие задачи. В ходе исследования было определено, что каждое решение обладает преимуществами и недостатками.

В ходе исследования было определено, что скорость обучения и доступные вычислительные ресурсы ограничены, GRU может быть предпочтительнее из-за своей более простой структуры и меньшего количества параметров по сравнению с LSTM. Для более простых или меньших наборов данных GRU может быть достаточной, поскольку она может достаточно эффективно обучаться на таких данных, не теряя в производительности. Также целесообразно использование модели VITS для синтеза речи и обучения, поскольку позволяет быстро и качественно синтезировать речь. Кроме того, данное решение имеет встроенный вокодер и предобученные модели на русском языке.

Разработка отечественного голосового помощника для озвучивания текста с использованием нейросетей позволит создавать аудиоматериалы на русском языке с лучшим качеством и доступностью, тем самым нивелировав недостатки аналогичных решений.

Проведенный анализ позволяет сделать вывод, что разработка проекта целесообразна.

Разработка программного обеспечения системы осуществлена с использованием среды программирования Visual Studio Code, с использованием языков Python и JS с применением фреймворка Vue.js.

Проведенные испытания показали, что программное обеспечение работает правильно и соответствует требованиям технического задания.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- 1 Звукограм – сервис для звуковых дизайнеров [Электронный ресурс] – URL: <https://zvukogram.com/> (Дата последнего обращения: 13.01.2024)
- 2 AI Voices – NaturalReader [Электронный ресурс] – URL: <https://www.naturalreaders.com/> Дата последнего обращения: 13.01.2025)
- 3 Genny | Home [Электронный ресурс] – URL: <https://genny.lovo.ai/> (Дата последнего обращения: 13.01.2025)
- 4 SteosVoice (formely CyberVoice) – Vocal cords of Artifitial Intelligence [Электронный ресурс] – URL: <https://cybervoice.io/ru/> (Дата последнего обращения: 14.01.2025)
- 5 Spyder | Anaconda.org [Электронный ресурс] – URL: <https://anaconda.org/anaconda/spyder> (Дата последнего обращения: 21.05.2025)
- 6 Notepad++: что это такое простыми словами, где применяется и что значит [Электронный ресурс] – URL: <https://blog.skillfactory.ru/glossary/notepad/> (Дата последнего обращения: 21.05.2025)
- 7 Что такое Visual Studio Code? [Электронный ресурс] – URL: <https://ru.hexlet.io/blog/posts/chto-takoe-visual-studio-code?ysclid=maz86jfh5b936881683> (Дата последнего обращения: 21.05.2025)
- 8 Язык программирования Python (Пайтон): что это такое, где и для чего используется... [Электронный ресурс] – URL: <https://gitverse.ru/blog/articles/development/125-yazyk-programmirovaniya-python-pajton-chto-eto-takoe-gde-i-dlya-chego-ispolzuet-sya-piton> (Дата последнего обращения: 21.05.2025)
- 9 JavaScript – Википедия [Электронный ресурс] – URL: <https://ru.wikipedia.org/wiki/JavaScript> (Дата последнего обращения: 21.05.2025)

10 Vue.js – что это: примеры проектов на фреймворке JavaScript [Электронный ресурс] – URL: <https://blog.skillfactory.ru/glossary/vue-js/> (Дата последнего обращения: 21.05.2025)

11 Vite – современный инструмент для фронтенд-разработчиков [Электронный ресурс] – URL: <https://result.school/roadmap/frontend/article/vite-sovremennyyj-instrument-dlya-frontend-razrabotchikov> (Дата последнего обращения: 21.05.2025)

12. Диаграмма Ганта: как с её помощью управляют проектами [Электронный ресурс] – URL: <https://kontur.ru/articles/6559> (Дата последнего обращения: 12.05.2025)

13 Разработка полноценной системы синтеза голоса с помощью нейронных сетей [Электронный ресурс] – URL: https://www.susu.ru/sites/default/files/files/Statya_na_konkurs.pdf (Дата последнего обращения: 20.06.2024).

14 Рекуррентные Нейронные Сети (RNN): Анализ последовательных данных [Электронный ресурс] – URL: <https://vc.ru/u/22269-aleksandr-shulepov/804299-rekurrentnye-neironnye-seti-rnn-analiz-posledovatelnyh-dannyh> (Дата последнего обращения: 20.06.2024)

15 Сети долгой краткосрочной памяти (LSTM) в работе – URL: <https://vc.ru/u/22269-aleksandr-shulepov/864489-seti-dolgoi-kratkosrochnoi-pamyati-lstm-v-rabote?ysclid=lxnj862tak549233988> (Дата последнего обращения: 20.06.2024)

16 LSTM и GRU / Хабр [Электронный ресурс] – URL: <https://habr.com/ru/companies/mvideo/articles/780774/> Дата последнего обращения: 20.06.2024)

17 Нейронные сети синтеза речи голосовых помощников и поющих автоматов [Электронный ресурс] – <https://cyberleninka.ru/article/n/neyronnye-seti-sinteza-rechi-golosovyh-pomoschnikov-i-poyuschiy-avtomatov/viewer> Дата последнего обращения: 27.12.2024)

18 Что такое технология TTS, как устроена и каких сферах используется синтез речи [Электронный ресурс] – URL:– <https://habr.com/ru/companies/skillfactory/articles/850118/> (Дата последнего обращения: 20.05.2025)

19 Should you use a probabilistic duration model in TTS? Probably! Especially for spontaneous speech [Электронный ресурс] – URL: <https://arxiv.org/html/2406.05401v1> (Дата последнего обращения: 20.05.2025)

20 Нейросетевой синтез речи с помощью архитектуры Tacotron 2 <https://habr.com/ru/companies/nix/articles/436312/> Дата последнего обращения: 27.12.2024)

21 Unveiling the Revolutionary VITS Model: Transforming Text-to-Speech Synthesis [Электронный ресурс] – URL: <https://blog.unrealspeech.com/unveiling-the-revolutionary-vits-model/> (Дата последнего обращения: 20.05.2025)

22 FastSpeech2 - Community Infrastructure to Strengthen AI for Audio Deepfake analysis (CISAAD) [Электронный ресурс] – URL: <https://cisaad.umbc.edu/fastspeech-2/> (Дата последнего обращения: 20.05.2025)

23 WaveGlow [Электронный ресурс] – URL: <https://anwarvic.github.io/speech-synthesis/WaveGlow> (Дата последнего обращения: 20.05.2025)

24 HiFi-GAN Explained | Papers With Code [Электронный ресурс] – URL: <https://paperswithcode.com/method/hifi-gan> (Дата последнего обращения: 20.05.2025)

25 Нейросеть HiFi-GAN: характеристики, описание, аналоги [Электронный ресурс] – URL: <https://aiabout.ru/apps/hifi-gan> Дата последнего обращения: 20.05.2025)

26 NLP модели на TensorFlow 2.0 [Электронный ресурс] – URL: <https://neurohive.io/ru/novosti/nlp-modeli-na-tensorflow-2-0-v-10-strok-koda/> (Дата последнего обращения: 27.05.2025)

ПРИЛОЖЕНИЕ А

(обязательное)

Разработка голосового помощника для озвучивания текста с использованием
нейронных сетей

Конфигурации предобученных моделей

```

#joefox-vits
{
  "_name_or_path": "/share/tts_vits_ru_hf",
  "activation_dropout": 0.1,
  "architectures": [
    "VitsModel"
  ],
  "attention_dropout": 0.1,
  "depth_separable_channels": 2,
  "depth_separable_num_layers": 3,
  "duration_predictor_dropout": 0.5,
  "duration_predictor_filter_channels": 256,
  "duration_predictor_flow_bins": 10,
  "duration_predictor_kernel_size": 3,
  "duration_predictor_num_flows": 4,
  "duration_predictor_tail_bound": 5.0,
  "ffn_dim": 384,
  "ffn_kernel_size": 3,
  "flow_size": 96,
  "hidden_act": "relu",
  "hidden_dropout": 0.1,
  "hidden_size": 96,
  "hop_length": 256,
  "initializer_range": 0.02,
  "layer_norm_eps": 1e-05,
  "layerdrop": 0.1,
  "leaky_relu_slope": 0.1,
  "model_type": "vits",
  "noise_scale": 0.667,
  "noise_scale_duration": 0.8,
  "num_attention_heads": 2,
  "num_hidden_layers": 6,
  "num_speakers": 6,
  "posterior_encoder_num_wavenet_layers": 16,

```

```

"prior_encoder_num_flows": 4,
"prior_encoder_num_wavenet_layers": 4,
"resblock_dilation_sizes": [
  [
    1,
    2
  ],
  [
    2,
    6
  ],
  [
    3,
    12
  ]
],
"resblock_kernel_sizes": [
  3,
  5,
  7
],
"sampling_rate": 16000,
"segment_size": 8192,
"speaker_embedding_size": 192,
"speaking_rate": 1.0,
"spectrogram_bins": 513,
"torch_dtype": "float32",
"transformers_version": "4.36.2",
"upsample_initial_channel": 512,
"upsample_kernel_sizes": [
  16,
  16,
  8
],

```

```

"upsample_rates": [
    8,
    8,
    4
],
"use_bias": true,
"use_stochastic_duration_prediction": true,
"vocab_size": 43,
"wavenet_dilation_rate": 1,
"wavenet_dropout": 0.0,
"wavenet_kernel_size": 5,
>window_size": 4
}

```

```

#natasha

```

```

{
    "train": {
        "log_interval": 200,
        "eval_interval": 1000,
        "seed": 1234,
        "epochs": 20000,
        "learning_rate": 1e-4,
        "betas": [0.8, 0.99],
        "eps": 1e-9,
        "batch_size": 16,
        "fp16_run": false,
        "lr_decay": 0.999875,
        "segment_size": 8192,
        "init_lr_ratio": 1,
        "warmup_epochs": 0,
        "c_mel": 45,
        "c_kl": 1.0
    },
    "data": {

```

```

"use_mel_posterior_encoder": true,

"training_files": "/media/kingspec/Datasets/Russian/Natasha/metadata_fullname_train.csv.cleaned",
"validation_files": "/media/kingspec/Datasets/Russian/Natasha/metadata_fullname_val.csv.cleaned",
    "text_cleaners": ["basic_cleaners"],
    "max_wav_value": 32768.0,
    "sampling_rate": 22050,
    "filter_length": 1024,
    "hop_length": 256,
    "win_length": 1024,
    "n_mel_channels": 80,
    "mel_fmin": 0.0,
    "mel_fmax": null,
    "add_blank": false,
    "n_speakers": 0,
    "cleaned_text": true
},
"model": {
    "use_mel_posterior_encoder": true,
    "use_transformer_flows": true,
    "transformer_flow_type": "pre_conv",
    "use_spk_conditioned_encoder": false,
    "use_noise_scaled_mas": true,
    "use_duration_discriminator": true,
    "inter_channels": 192,
    "hidden_channels": 192,
    "filter_channels": 768,
    "n_heads": 2,
    "n_layers": 10,
    "kernel_size": 5,
    "p_dropout": 0.1,
    "resblock": "1",

```

```
"resblock_kernel_sizes": [3,7,11],
"resblock_dilation_sizes": [[1,3,5], [1,3,5], [1,3,5]],
"upsample_rates": [8,8,2,2],
"upsample_initial_channel": 512,
"upsample_kernel_sizes": [16,16,4,4],
"n_layers_q": 3,
"use_spectral_norm": false,
"use_sdp": false
}
}
```

ПРИЛОЖЕНИЕ Б

(обязательное)

Разработка голосового помощника для озвучивания текста с использованием
нейронных сетей

Листинг программы

Файл App.vue

```
<template>
  <div id="app">
    <Header />
    <div class="workspace">
      <Sidebar @form-changed="updateForm"/>
      <component :is="currentForm" class="content"></component>
    </div>
  </div>
</template>

<script>
import { ref } from 'vue';
import TTS from './components/tts.vue';
import Sidebar from './components/sidebar.vue';
import Header from './components/header.vue';
import Learning from './components/learning.vue';
export default {
  components: {Learning, TTS, Sidebar, Header },
  data() {
    return {
      currentForm: 'TTS',
    },
  },
  methods: {
    updateForm(form) {
      this.currentForm = form
    }
  },
  setup() {
    const amount = ref(0); // Используем ref для реактивной переменной
    return {
      amount,
    };
  },
}
```

```
};  
</script>  
<style scoped>  
* {  
    margin: 0;  
    padding: 0;  
    box-sizing: border-box;  
}  
html, body {  
    height: 100%;  
    width: 100%;  
    overflow: hidden;  
}  
#app {  
    display: flex;  
    flex-direction: column;  
    height: 100vh;  
    width: 100vw;  
}  
.header {  
    background: rgb(185, 255, 176);  
    flex-shrink: 0;  
}  
.workspace {  
    display: flex;  
    flex: 1;  
    min-height: 0; /* Это важно для правильного растягивания вложенных элементов */  
    width: 100%;  
}  
.content {  
    flex: 1;  
    overflow: auto; /* Добавляем прокрутку если контент не помещается */  
}  
</style>
```

Файл tts.vue

```
<template>
<div class="tts-container">
  <h1>Text to Speech</h1>
  <div class="input-group">
    <label for="model-folder">Выберите модель:</label>
    <select
      id="model-folder"
      v-model="selectedModelPath"
      class="model-select">
      <option value="" disabled>Выберите модель</option>
      <option v-for="model in availableModels" :key="model.path" :value="model.path">
        {{ model.name }}
      </option>
    </select>
    <div v-if="selectedModelPath" class="selected-folder">
      Выбрана модель: {{ selectedModelPath }}
    </div>
  </div>
  <div class="text-input">
    <textarea v-model="text" placeholder="Введите текст для озвучивания"></textarea>
  </div>
  <!-- Параметры синтеза -->
  <div class="parameters">
    <div class="parameter">
      <label>Громкость: {{ volume }}</label>
      <input type="range" v-model.number="volume" min="0" max="100" />
    </div>
    <div class="parameter">
      <label>Высота голоса: {{ pitch }}</label>
      <input type="range" v-model.number="pitch" min="0.5" max="2" step="0.1" />
    </div>
    <div class="parameter">
      <label>Скорость речи: {{ speed }}</label>
    </div>
  </div>
</div>
```

```

    <input type="range" v-model.number="speed" min="0.5" max="2" step="0.1" />
  </div>
</div>
<button
  @click="synthesizeVoice"
  class="generate-btn"
  :disabled="isLoading" >
  <span v-if="!isLoading">Сгенерировать файл</span>
  <span v-else>Генерация...</span>
</button>
<audio
  v-if="audioUrl"
  controls
  class="audio-player"
  ref="audioPlayer">
  <source :src="audioUrl" type="audio/wav">
</audio>
</div>
</template>
<script>
export default {
  data() {
    return {
      text: "",
      selectedModelPath: "",
      availableModels: [
        { name: 'Joefox VITS', path: 'models/joefox-vits' },
        { name: 'UtrobinMV VITS', path: 'utrobinmv/tts_ru_free_hf_vits_low_multispeaker' },
        { name: 'Natasha', path: 'models/natasha-vits' },
        { name: 'Mai', path: 'models/mai-vits' }
        // Добавьте другие модели здесь
      ],
      volume: 50,
      pitch: 1,

```

```

    speed: 1,
    isLoading: false,
    audioUrl: null
  };
},
methods: {
  async synthesizeVoice() {
    if (!this.text.trim()) {
      alert("Введите текст для синтеза.");
      return;
    }
    if (!this.selectedModelPath) {
      alert("Выберите модель.");
      return;
    }
    this.isLoading = true;
    // Очистка предыдущего audioUrl для предотвращения кэширования
    this.audioUrl = null;
    try {
      const response = await fetch("http://127.0.0.1:8000/api/synthesize", {
        method: "POST",
        headers: {
          "Content-Type": "application/json",
        },
        body: JSON.stringify({
          text: this.text,
          volume: Number(this.volume), // Преобразуем в число
          pitch: Number(this.pitch), // Преобразуем в число
          speed: Number(this.speed), // Преобразуем в число
          speaker_id: 0,
          model_path: this.selectedModelPath
        }),
      });
      if (!response.ok) {

```

```

    const errorData = await response.json();
    throw new Error(errorData.detail || "Ошибка сервера");
  }
  const data = await response.json();
  if (data?.filepath) {
    // Добавляем временную метку к URL, чтобы избежать кэширования
    this.audioUrl = `http://127.0.0.1:8000${data.filepath}?t=${new Date().getTime()}`;
    this.$nextTick(() => {
      this.$refs.audioPlayer.load(); // Перегружаем аудиоэлемент
      this.$refs.audioPlayer.play();
    });
  }
} catch (err) {
  console.error("Ошибка синтеза:", err);
  alert(`Ошибка синтеза: ${err.message}`);
} finally {
  this.isLoading = false;
}
}
};
</script>
<style scoped>
.tts-container {
  flex: 1;
  display: flex;
  flex-direction: column;
  align-items: center;
  padding: 20px;
  gap: 20px;
  background: linear-gradient(0deg, rgb(185, 255, 176) 3%, rgb(233, 233, 233) 72%);
  height: calc(100vh - 150px);
  overflow-y: auto;
  box-sizing: border-box;

```

```

}
.tts-container h2 {
  font-size: 1.5rem;
  margin: 10px 0;
  text-shadow: 2px 2px black;
  color: #333;
}
h1 {
  text-shadow: 3px 3px black;
  margin: 10px 0;
  color: white;
}
.input-group, .text-input, .parameters {
  width: 100%;
  max-width: 800px;
}
.folder-input {
  width: 100%;
  padding: 8px;
  border: 2px dashed #ccc;
  border-radius: 5px;
  background-color: #f9f9f9;
  cursor: pointer;
}
.selected-folder {
  margin-top: 8px;
  padding: 8px;
  background-color: #e8f5e9;
  border-radius: 4px;
  font-size: 14px;
}
textarea {
  width: 100%;
  height: 150px;

```

```

padding: 10px;
font-size: 14pt;
border-radius: 5px;
border: 1px solid #ccc;
resize: vertical;
}
.parameters {
display: flex;
flex-direction: column;
gap: 15px;
}
.parameter {
display: flex;
flex-direction: column;
gap: 5px;
}
input[type="range"] {
width: 100%;
}
.generate-btn {
padding: 12px 25px;
background-color: #4CAF50;
color: white;
border: none;
border-radius: 5px;
cursor: pointer;
font-size: 16px;
transition: background-color 0.3s;
}
.generate-btn:disabled {
background-color: #cccccc;
cursor: not-allowed;
}
.generate-btn:hover:not(:disabled) {

```

```

background-color: #45a049;
}
.audio-player {
width: 100%;
max-width: 800px;
margin-top: 20px;
}
label {
color: rgb(0, 0, 0);
font-weight: bold;
font-size: larger;
}
@media (max-width: 768px) {
.tts-container {
padding: 15px;
}
textarea {
height: 120px;
font-size: 12pt;
}
}
</style>

```

Файл learning.vue

```

<template>
<div class="learning-container">
<h1>Обучение модели голоса</h1>
<div class="input-group">
<label for="model-select">Выберите модель для дообучения (опционально):</label>
<select
id="model-select"
v-model="selectedModelPath"
class="model-select"
>
<option value="" disabled>Без дообучения (новая модель)</option>

```

```

    <option v-for="model in availableModels" :key="model.path" :value="model.path">
      {{ model.name }}
    </option>
  </select>
</div>

  <div class="input-group">
    <label for="folder-input">Выберите папку с WAV-файлами:</label>
    <input
      type="file"
      id="folder-input"
      webkitdirectory
      directory
      multiple
      accept=".wav"
      @change="handleFolderSelect"
      class="folder-input"
    />
    <button @click="submitPath" class="action-btn">Отправить</button>
  </div>

  <div v-if="wavFiles.length > 0" class="files-section">
    <h3>Выбранные файлы:</h3>
    <ul class="files-list">
      <li v-for="(file, index) in wavFiles" :key="index">{{ file.name }}</li>
    </ul>
    <button
      @click="startTraining"
      :disabled="!wavFiles.length || isTraining"
      class="action-btn train-btn">
      {{ isTraining ? 'Обучение...' : 'Начать обучение' }}
    </button>
  </div>

  <p v-if="message" class="message">{{ message }}</p>
  <div class="log-container">
    <div v-for="(line, index) in log" :key="index" class="log-line">{{ line }}</div>
  </div>

```

```

    </div>
  </div>
</template>
<script>
export default {
  name: 'Learning',
  data() {
    return {
      wavFiles: [],
      selectedModelPath: "",
      availableModels: [
        { name: 'Joevox VITS', path: 'models/joevox-vits' },
        { name: 'UtrobinMV VITS', path: 'utrobinmv/tts_ru_free_hf_vits_low_multispeaker' },
        { name: 'Natasha', path: 'models/natasha-vits' },
        { name: 'Mai', path: 'models/mai-vits' }
      ],
      log: [],
      message: "",
      isTraining: false
    };
  },
  methods: {
    handleFolderSelect(event) {
      this.wavFiles = Array.from(event.target.files).filter(file => file.name.endsWith('.wav'));
      this.message = this.wavFiles.length > 0 ? " : 'Выберите папку с WAV-файлами.'";
    },
    async submitPath() {
      if (this.wavFiles.length === 0) {
        this.message = 'Выберите папку с WAV-файлами.';
        return;
      }
      this.log = ["Загрузка файлов..."];
      const formData = new FormData();
      this.wavFiles.forEach((file) => formData.append('wavs', file));
    }
  }
}

```

```

if (this.selectedModelPath) {
  formData.append('model_path', this.selectedModelPath);
}
try {
  const response = await fetch('http://127.0.0.1:8000/api/upload-wavs', {
    method: 'POST',
    body: formData
  });
  if (!response.ok) {
    const errorData = await response.json();
    throw new Error(errorData.detail || 'Ошибка загрузки файлов');
  }
  this.log.push('Файлы успешно загружены. Готово к обучению.');
  this.message = "";
} catch (err) {
  this.log.push(`Ошибка: ${err.message}`);
  this.message = `Ошибка загрузки: ${err.message}`;
}
},
async startTraining() {
  this.isTraining = true;
  this.log.push('Запуск обучения...');
  const eventSource = new EventSource('http://127.0.0.1:8000/api/train-stream');
  eventSource.onmessage = (event) => {
    if (event.data === '[DONE]') {
      this.log.push('Обучение завершено.');
      eventSource.close();
      this.isTraining = false;
    } else {
      this.log.push(event.data);
    }
  };
  eventSource.onerror = () => {
    this.log.push('Ошибка соединения с сервером.');
```

```

    this.isTraining = false;
    eventSource.close();
  };
  try {
    const response = await fetch('http://127.0.0.1:8000/api/start-training', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json'
      },
      body: JSON.stringify({
        model_path: this.selectedModelPath || null
      })
    });
    if (!response.ok) {
      const errorData = await response.json();
      throw new Error(errorData.detail || 'Ошибка запуска обучения');
    }
  } catch (err) {
    this.log.push(`Ошибка: ${err.message}`);
    this.isTraining = false;
  }
}
};
</script>
<style scoped>
.learning-container {
  flex: 1;
  display: flex;
  flex-direction: column;
  padding: 20px;
  gap: 20px;
  background: linear-gradient(0deg, rgb(185, 255, 176) 3%, rgb(233, 233, 233) 72%);
  height: 100%;

```

```

    overflow-y: auto;
    box-sizing: border-box;
}
h1 {
    text-align: center;
    margin-bottom: 20px;
    color: #333;
}
.input-group {
    display: flex;
    gap: 10px;
    width: 100%;
    max-width: 800px;
    margin: 0 auto;
}
.model-select, .folder-input {
    flex: 1;
    padding: 8px;
    border: 1px solid #ccc;
    border-radius: 4px;
}
.action-btn {
    padding: 8px 16px;
    background-color: #4CAF50;
    color: white;
    border: none;
    border-radius: 4px;
    cursor: pointer;
    transition: background-color 0.3s;
}
.action-btn:hover {
    background-color: #45a049;
}

```

```
.action-btn:disabled {
  background-color: #cccccc;
  cursor: not-allowed;
}

.train-btn {
  background-color: #2196F3;
}

.train-btn:hover {
  background-color: #0b7dda;
}

.files-section {
  width: 100%;
  max-width: 800px;
  margin: 0 auto;
  background: rgba(255, 255, 255, 0.8);
  padding: 15px;
  border-radius: 5px;
}

.files-list {
  max-height: 150px;
  overflow-y: auto;
  text-align: left;
  margin: 10px 0;
  padding: 0;
  list-style-type: none;
}

.files-list li {
  padding: 5px;
  border-bottom: 1px solid #eee;
}

.log-container {
  flex: 1;
  background: #f8f9fa;
  border: 1px solid #dee2e6;
```

```

border-radius: 5px;
padding: 15px;
overflow-y: auto;
font-family: monospace;
text-align: left;
margin-top: 20px;
}
.log-line {
margin-bottom: 5px;
white-space: pre-wrap;
}
.message {
color: #d9534f;
font-weight: bold;
}
@media (max-width: 768px) {
.input-group {
flex-direction: column;
}
.learning-container {
padding: 15px;
}
}
</style>

```

Файл main.py

```

from fastapi import FastAPI
from fastapi.middleware.cors import CORSMiddleware
from fastapi.staticfiles import StaticFiles
from server import router

app = FastAPI(title="API преобразования текста в речь")
# Mount the outputs directory to serve audio files
app.mount("/outputs", StaticFiles(directory="outputs"), name="outputs")
# Настройка CORS для обработки запросов с фронта
app.add_middleware(

```

```

CORSMiddleware,
allow_origins=["http://localhost:5173"], # Порт Vite по умолчанию
allow_credentials=True,
allow_methods=["*"],
allow_headers=["*"],
)
# Подключение маршрутов из server.py
app.include_router(router)
if __name__ == "__main__":
    import uvicorn
    print("Запуск сервера на http://127.0.0.1:8000")
    uvicorn.run(app, host="127.0.0.1", port=8000)

```

Файл server.py

```

from fastapi import APIRouter, HTTPException
from fastapi.responses import JSONResponse
from pydantic import BaseModel
from synthesize import TextToSpeechSynthesizer
from learning import VitsTrainer
from pathlib import Path
import uuid
import logging
import re

router = APIRouter()
# Configure logging
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)
# Initialize the synthesizer (will be loaded when model path is provided)
synthesizer = None

class SynthesizeRequest(BaseModel):
    text: str
    volume: float
    pitch: float
    speed: float
    speaker_id: int

```

```

    model_path: str
def get_next_file_index(model_name: str, output_dir: Path) -> int:
    pattern = re.compile(rf'tts-{re.escape(model_name)}_(\d+)\.wav")
    max_index = 0
    for file in output_dir.glob(f'tts-{model_name}_*.wav"):
        match = pattern.match(file.name)
        if match:
            index = int(match.group(1))
            max_index = max(max_index, index)
    return max_index + 1
@router.post("/api/synthesize")
async def synthesize(request: SynthesizeRequest):
    try:
        # Check if model path exists
        model_dir = Path(request.model_path)
        # if not model_dir.exists():
            # raise HTTPException(status_code=400, detail=f"Model path {request.model_path} does
not exist")
        # Initialize or reinitialize synthesizer with model path
        global synthesizer
        synthesizer = TextToSpeechSynthesizer(model_dir)
        # Generate output file path
        output_dir = Path("outputs")
        output_dir.mkdir(exist_ok=True)
        model_name = model_dir.name
        file_index = get_next_file_index(model_name, output_dir)
        output_filename = f'tts-{model_name}_{file_index}.wav"
        output_path = output_dir / output_filename
        # Synthesize speech
        synthesizer.synthesize(
            text=request.text,
            output_path=str(output_path),
            volume=request.volume/100, # Convert to 0-1 scale
            pitch=request.pitch,

```

```

        speed=request.speed,
        speaker_id=request.speaker_id
    )
    return JsonResponse(
        status_code=200,
        content={"filepath": f"/outputs/{output_filename}"}
    )
except Exception as e:
    logger.error(f"Synthesis error: {str(e)}")
    raise HTTPException(status_code=500, detail=str(e))
@router.post("/api/upload-wavs")
async def upload_wavs(wavs: list[UploadFile], model_path: str = Form(None)):
    global temp_wav_dir
    try:
        # Create temporary directory for WAV files
        temp_wav_dir = Path(f"temp_wavs/{uuid.uuid4()}")
        temp_wav_dir.mkdir(parents=True, exist_ok=True)
        # Save uploaded WAV files
        for wav in wavs:
            if not wav.filename.endswith('.wav'):
                raise HTTPException(status_code=400, detail="Only WAV files are allowed")
            file_path = temp_wav_dir / wav.filename
            with file_path.open("wb") as buffer:
                shutil.copyfileobj(wav.file, buffer)
        return JsonResponse(status_code=200, content={"message": "Files uploaded successfully"})
    except Exception as e:
        logger.error(f"Upload error: {str(e)}")
        raise HTTPException(status_code=500, detail=str(e))
@router.post("/api/start-training")
async def start_training(request: dict):
    global trainer, training_log, temp_wav_dir
    try:
        model_path = request.get("model_path")

```

```

if not temp_wav_dir or not temp_wav_dir.exists():
    raise HTTPException(status_code=400, detail="No WAV files uploaded")
training_log = [] # Reset logs
trainer = VitsTrainer(model_dir=model_path)
async def train():
    try:
        trainer.train(temp_wav_dir, stream_callback=lambda msg: training_log.append(msg))
        training_log.append("[DONE]")
    except Exception as e:
        training_log.append(f"Error: {str(e)}")
        raise
# Start training in a separate task
asyncio.create_task(train())
return JSONResponse(status_code=200, content={"message": "Training started"})
except Exception as e:
    logger.error(f"Training start error: {str(e)}")
    raise HTTPException(status_code=500, detail=str(e))
@router.get("/api/train-stream")
async def train_stream():
    async def stream_logs():
        global training_log
        while True:
            if training_log:
                for log in training_log[:]:
                    yield {"data": log}
                    training_log.remove(log)
                if log == "[DONE]":
                    break
            await asyncio.sleep(1)
            yield {"data": "[DONE]"}
    return EventSourceResponse(stream_logs())

```

Файл learning.py

```

import os
import uuid

```

```

from pathlib import Path
import librosa
import torch
from transformers import VitsModel, VitsTokenizer, Trainer, TrainingArguments
from datasets import Dataset, Audio
import logging
# Configure logging
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)
class VitsTrainer:
    def __init__(self, model_dir=None, output_dir="models/trained_model"):
        self.device = 'cuda' if torch.cuda.is_available() else 'cpu'
        self.output_dir = Path(output_dir)
        self.output_dir.mkdir(parents=True, exist_ok=True)

    try:
        if model_dir and Path(model_dir).exists():
            self.model = VitsModel.from_pretrained(model_dir).to(self.device)
            self.tokenizer = VitsTokenizer.from_pretrained(model_dir)
            logger.info(f"Loaded existing model from {model_dir}")
        else:
            # Use default VITS model for new training
            model_name = "utrobinmv/tts_ru_free_hf_vits_low_multispeaker"
            self.model = VitsModel.from_pretrained(model_name).to(self.device)
            self.tokenizer = VitsTokenizer.from_pretrained(model_name)
            logger.info("Initialized new model from default")
    except Exception as e:
        logger.error(f"Error loading model: {str(e)}")
        raise

    self.model.train()

    def prepare_dataset(self, wav_dir):
        """
        Prepare dataset from WAV files.

```

```

Args:
    wav_dir: Directory containing WAV files
    """
    wav_files = list(Path(wav_dir).glob("*.wav"))
    if not wav_files:
        raise ValueError("No WAV files found in the provided directory")
    data = {
        "audio": [],
        "text": [] # Placeholder for transcriptions (if available)
    }
    for wav_file in wav_files:
        audio, sr = librosa.load(wav_file, sr=self.model.config.sampling_rate)
        data["audio"].append({"path": str(wav_file), "array": audio, "sampling_rate": sr})
        # Note: Transcriptions are required for real training. This is a placeholder.
        # In a real scenario, you need a transcription file for each WAV.
        data["text"].append("Placeholder transcription")
    dataset = Dataset.from_dict(data)
    dataset = dataset.cast_column("audio",
Audio(sampling_rate=self.model.config.sampling_rate))
    return dataset

def train(self, wav_dir, stream_callback=None):
    """
    Train or fine-tune the VITS model.

    Args:
        wav_dir: Directory with WAV files
        stream_callback: Function to stream logs
    """
    try:
        dataset = self.prepare_dataset(wav_dir)
        if stream_callback:
            stream_callback("Dataset prepared with {} samples".format(len(dataset)))
        training_args = TrainingArguments(
            output_dir=str(self.output_dir),
            num_train_epochs=3, # Adjust based on dataset size

```

```

        per_device_train_batch_size=2,
        gradient_accumulation_steps=4,
        learning_rate=2e-5,
        save_steps=100,
        logging_steps=10,
        save_total_limit=2,
        fp16=self.device == 'cuda',
    )
    def data_collator(features):
        audio = [f["audio"]["array"] for f in features]
        input_ids = self.tokenizer([f["text"] for f in features], return_tensors="pt",
padding=True).input_ids
        return {
            "input_ids": input_ids.to(self.device),
            "audio": torch.tensor(audio, dtype=torch.float32).to(self.device)
        }
    trainer = Trainer(
        model=self.model,
        args=training_args,
        train_dataset=dataset,
        data_collator=data_collator
    )
    if stream_callback:
        stream_callback("Starting training...")
    trainer.train()
    if stream_callback:
        stream_callback("Training completed. Saving model...")
    self.model.save_pretrained(self.output_dir)
    self.tokenizer.save_pretrained(self.output_dir)
    if stream_callback:
        stream_callback(f"Model saved to {self.output_dir}")
except Exception as e:
    logger.error(f"Training error: {str(e)}")
    if stream_callback:

```

```
stream_callback(f"Error: {str(e)}")
```

```
raise
```

Файл synthesizer.py

```
from transformers import VitsModel, AutoTokenizer
```

```
from ruaccent import RUAccent
```

```
import torch
```

```
import scipy
```

```
import numpy as np
```

```
from pathlib import Path
```

```
import librosa
```

```
class TextToSpeechSynthesizer:
```

```
    def __init__(self, model_dir):
```

```
        """
```

```
        Initialize the TTS synthesizer with model files.
```

```
        Args:
```

```
        model_dir: Path to the directory containing model files
```

```
        """
```

```
        self.device = 'cuda' if torch.cuda.is_available() else 'cpu'
```

```
        try:
```

```
            # Load local model
```

```
            self.model = VitsModel.from_pretrained(model_dir).to(self.device)
```

```
            self.tokenizer = AutoTokenizer.from_pretrained(model_dir)
```

```
            print("Локальная модель успешно загружена")
```

```
        except Exception as e:
```

```
            print(f"Ошибка загрузки локальной модели: {e}")
```

```
            # Fallback to default model
```

```
            model_name = "utrobinmv/tts_ru_free_hf_vits_low_multispeaker"
```

```
            self.model = VitsModel.from_pretrained(model_name).to(self.device)
```

```
            self.tokenizer = AutoTokenizer.from_pretrained(model_name)
```

```
            self.model.eval()
```

```
            # Load accentizer
```

```
            self.accentizer = RUAccent()
```

```
            self.accentizer.load(omograph_model_size='turbo', use_dictionary=True,
```

```
device=self.device)
```

```

def synthesize(self, text, output_path, volume=1.0, pitch=1.0, speed=1.0, speaker_id=0):
    """
    Synthesize speech with given parameters.
    Args:
        text: Input text to synthesize
        output_path: Path to save the output WAV file
        volume: Volume adjustment (0.0-1.0)
        pitch: Pitch adjustment (0.5-2.0)
        speed: Speed adjustment (0.5-2.0)
        speaker_id: Speaker ID (0 for female, 1 for male)
    """
    try:
        # Apply accents to text
        text = self.accentizer.process_all(text)
        # Tokenize input text
        inputs = self.tokenizer(text, return_tensors="pt")
        # Synthesize speech
        with torch.no_grad():
            output = self.model(**inputs.to(self.device), speaker_id=speaker_id).waveform
            audio = output.detach().cpu().numpy()[0]
            audio = audio.astype(np.float32)
        # Apply audio effects
        # Volume adjustment
        audio = audio * volume
        # Pitch adjustment (using librosa)
        if pitch != 1.0:
            # Convert pitch to n_steps (empirical scaling)
            n_steps = float(pitch) * 4
            audio = librosa.effects.pitch_shift(
                y=audio,
                sr=self.model.config.sampling_rate,
                n_steps=n_steps
            )

```

```
# Speed adjustment
if speed != 1.0:
    audio = librosa.effects.time_stretch(
        y=audio,
        rate=1/float(speed)
    )
# Save to file
scipy.io.wavfile.write(output_path, rate=self.model.config.sampling_rate, data=audio)
except Exception as e:
    raise Exception(f"Failed to synthesize speech: {str(e)}")
```

ПРИЛОЖЕНИЕ В

(обязательное)

Разработка голосового помощника для озвучивания текста с использованием
нейронных сетей

Презентация

Разработка голосового помощника для озвучивания текста с использованием нейронных сетей

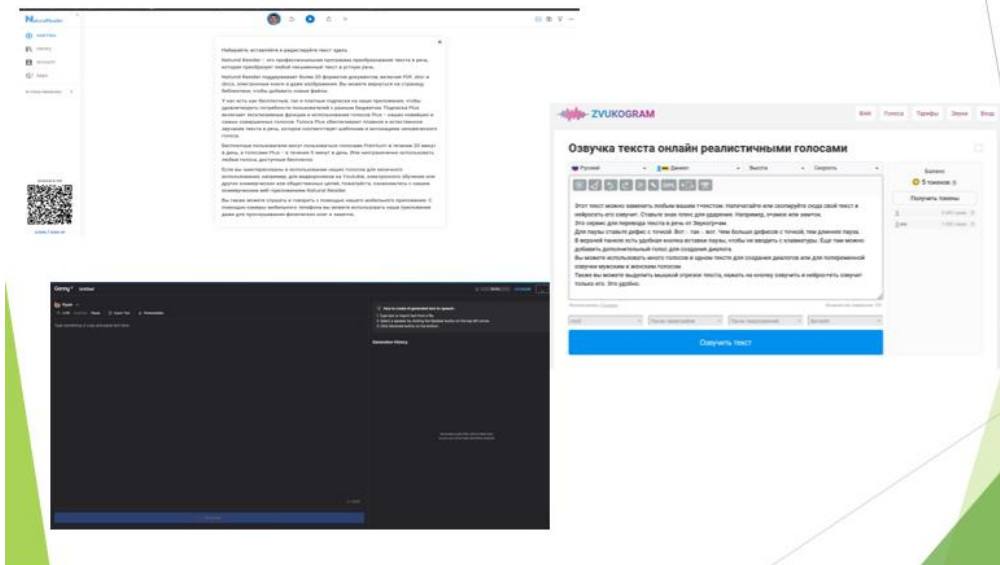
Выполнил: студент гр. 23ВВИМ1
Сохацкий Д.Ю.

Введение

Одним из методов голосового озвучивания является синтез речи с использованием нейросетей, которые способны обрабатывать и интерпретировать текст с удивительной точностью и естественностью. Данная технология сокращенно называется TTS (Text-to-speech)



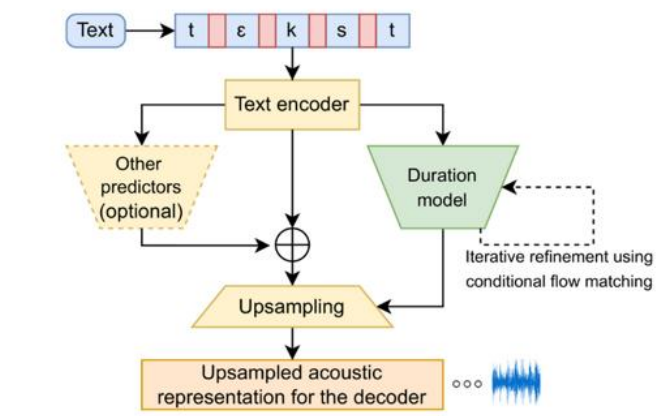
Аналоги



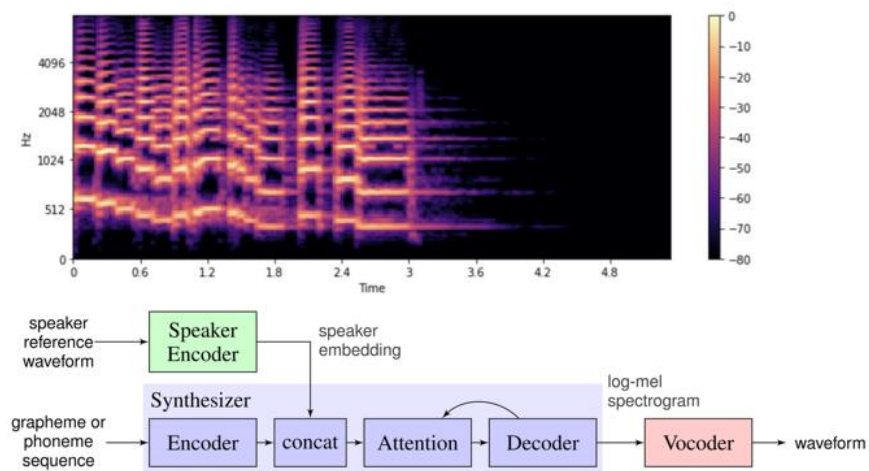
Аналоги

Нейросеть	Качество	Скорость	Языки	Доступность	Кол-во голосов
Zvukogram	9	4	9	5	8
Genny	8	1	2	3	8
Cybervoice	5	7	7	8	10
Natural Readers	4	8	5	8	1

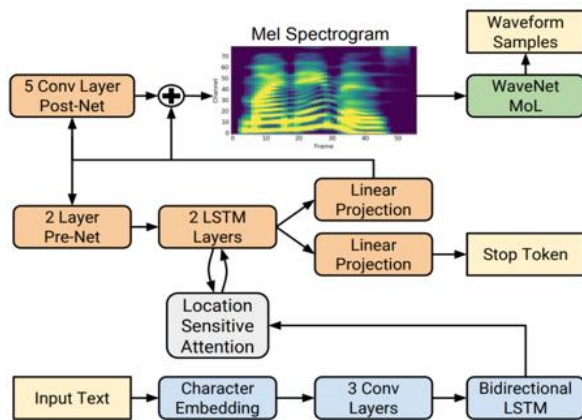
Принцип работы Text-to-speech



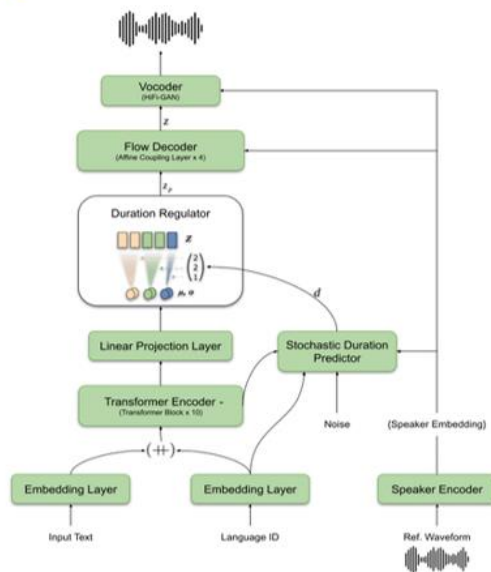
Принцип работы Text-to-speech



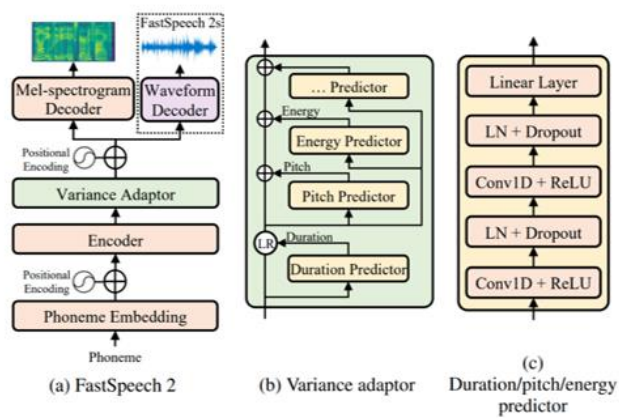
Tacotron



VITS



FastSpeech2



Выбор модели

Параметр	Tacotron 2	FastSpeech 2	VITS
Тип генерации	Авторегрессионная	Неавторегрессионная	Неавторегрессионная
Скорость синтеза	Медленная	Быстрая	Быстрая
Вокодер	Требуется (WaveNet)	Требуется (HiFi-GAN)	Встроенный (HiFi-GAN)
Качество	Высокое	Высокое	Высокое
Проблемы	Пропуски слов, медленный синтез	Зависит от выравнивания длительностей	Сложность обучения
Мультиязычность	Ограниченная	Поддерживается	Поддерживается

Выбор модели

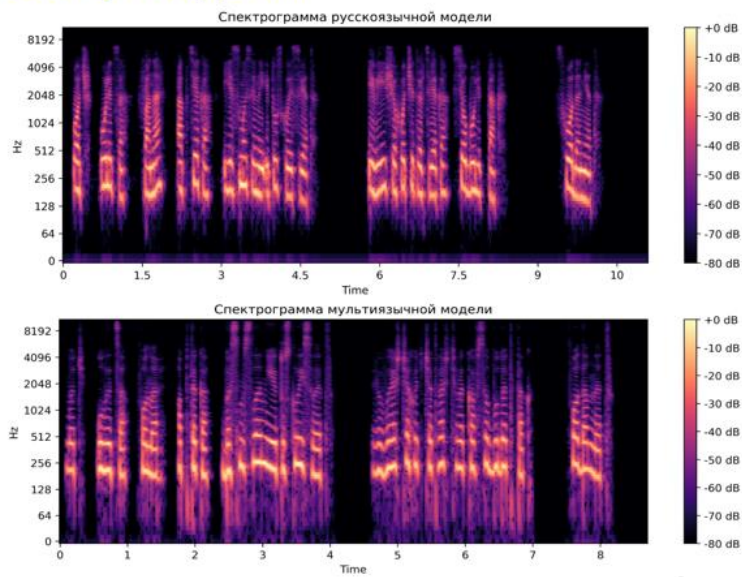


Диаграмма вариантов использования

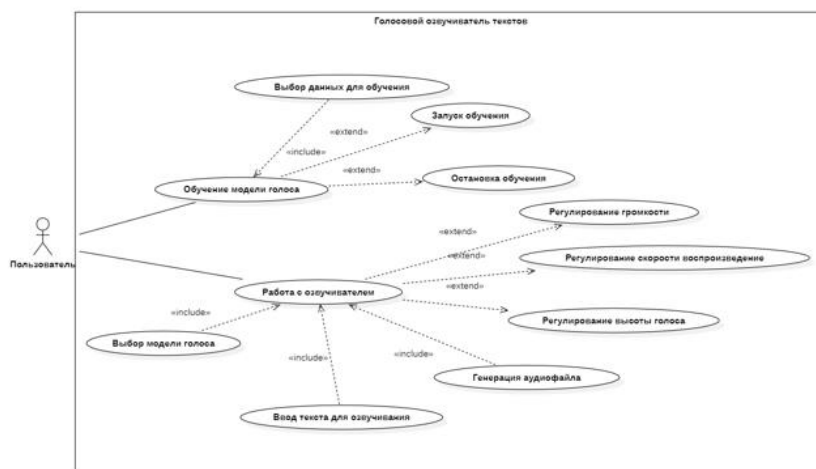
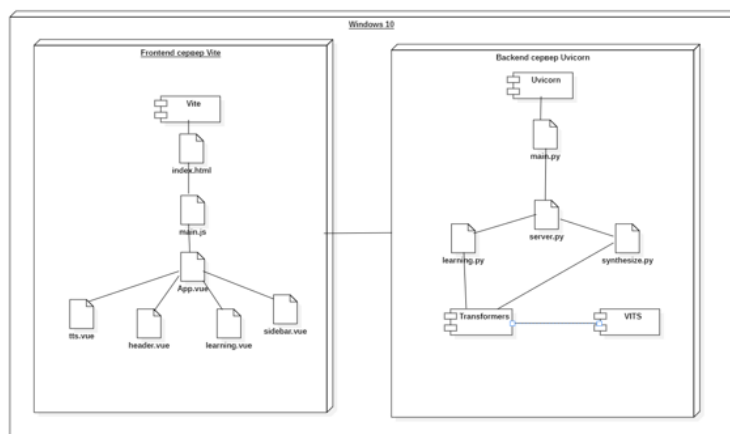


Диаграмма развертывания



Структура приложения



Экранные формы

The screenshot shows the 'Text to Speech' interface. At the top, a green header bar contains the text 'Помощник для озвучивания текстов'. Below this, a sidebar on the left has two tabs: 'TTS-Speaker' and 'Обучение', with 'Обучение' currently selected. The main content area has a title 'Text to Speech' and a dropdown menu labeled 'Выборите модель: Выберите модель...'. Below the menu is a text input field with the placeholder 'Введите текст для озвучивания'. Underneath the input field are three sliders: 'Громкость: 50', 'Высота голоса: 1', and 'Скорость речи: 1'. At the bottom of the sliders is a green button labeled 'Отправить файл'.

Экранные формы

The screenshot shows the 'Voice Model Training' interface. At the top, a green header bar contains the text 'Помощник для озвучивания текстов'. Below this, a sidebar on the left has two tabs: 'TTS-Speaker' and 'Обучение', with 'Обучение' currently selected. The main content area has a title 'Обучение модели голоса'. Below the title are two dropdown menus: 'Выборите модель для обучения (определённая)' and 'Как добавление (новая модель)'. Below these is a text input field with the placeholder 'Выборите модель с WAV файлами' and a green button labeled 'Отправить'. Below the input field is a section titled 'Выборите файлы:' with a list of files: '1.wav', '2.wav', '3.wav', '4.wav', and '5.wav'. At the bottom of the list is a blue button labeled 'Начать обучение'.

Заключение

Разработка отечественного голосового помощника для озвучивания текста с использованием нейронных сетей позволит создавать аудиоматериалы на русском языке с лучшим качеством и доступностью, тем самым нивелировав недостатки аналогичных решений.

Для этого целесообразно использование модели VITS для синтеза речи и обучения, поскольку позволяет быстро и качественно синтезировать речь. Кроме того, данное решение имеет встроенный вокодер и предобученные модели на русском языке.



СПАСИБО ЗА ВНИМАНИЕ