Содержание

[ПЕРЕЧЕНЬ УСЛОВНЫХ ОБОЗНАЧЕНИЙ 6](#_Toc199413076)

[ВВЕДЕНИЕ 8](#_Toc199413077)

[1. АНАЛИЗ ИСХОДНЫХ ДАННЫХ И ПРОЕКТИРОВАНИЕ 9](#_Toc199413078)

[1.1 Постановка и анализ задачи 9](#_Toc199413079)

[1.1.1 Анализ существующих подходов к распознаванию речи 9](#_Toc199413080)

[1.1.2 Метрика оценки качества распознавания речи 10](#_Toc199413081)

[1.1.3 Критерии выбора подхода для разрабатываемой системы 11](#_Toc199413082)

[1.2 Проектирование архитектуры системы 13](#_Toc199413083)

[1.2.1 Общая концепция архитектуры системы распознавания речи 13](#_Toc199413084)

[1.2.2 Проектирование модуля предобработки звукового сигнала 14](#_Toc199413085)

[1.2.2.1 Функциональные требования для модуля предобработки звукового сигнала 15](#_Toc199413086)

[1.2.2.2 Выбор метода представления звукового сигнала 15](#_Toc199413087)

[1.2.2.3 Проектирование функции буферизации и обработки аудиоданных 17](#_Toc199413088)

[1.2.3 Проектирование модуля распознавания речи 17](#_Toc199413089)

[1.2.3.1 Обоснование выбора архитектуры нейронной сети 18](#_Toc199413090)

[1.2.3.2 Проектирование компонентов нейронной модели 18](#_Toc199413091)

[1.2.3.3 Метод декодирования последовательностей 19](#_Toc199413092)

[1.2.4 Проектирование модуля пользовательского интерфейса 21](#_Toc199413093)

[1.2.4.1 Общая структура пользовательского интерфейса 21](#_Toc199413094)

[1.2.4.2 Проектирование механизма взаимодействия с пользователем 22](#_Toc199413095)

[1.2.4.3 Проектирование механизма обновления интерфейса 23](#_Toc199413096)

[1.3 Обоснование выбора средств разработки программного обеспечения 23](#_Toc199413097)

[1.3.1 Выбор языка программирования 23](#_Toc199413098)

[1.3.2 Библиотеки для обработки аудио и извлечения признаков 24](#_Toc199413099)

[1.3.3 Библиотека машинного обучения 25](#_Toc199413100)

[1.3.4 Средства разработки пользовательского интерфейса 26](#_Toc199413101)

[1.3.5 Вспомогательные библиотеки 27](#_Toc199413102)

[2. Разработка системы 29](#_Toc199413103)

[2.1 Архитектура разрабатываемой системы 29](#_Toc199413104)

[2.2 Разработка модуля предобработки звукового сигнала 29](#_Toc199413105)

[2.3 Разработка модуля распознавания речи 32](#_Toc199413106)

[2.3.1 Архитектура нейронной модели 32](#_Toc199413107)

[2.3.2 Обучение нейронной модели 39](#_Toc199413108)

[2.4 Разработка пользовательского интерфейса 46](#_Toc199413109)

[3. Тестирование программного обеспечения 51](#_Toc199413110)

[3.1 Проведение тестирования программы 51](#_Toc199413111)

[3.1.1 Тестирование модуля предобработки звукового сигнала 51](#_Toc199413112)

[3.1.2 Тестирования модуля графического интерфейса 54](#_Toc199413113)

[3.2 Анализ результатов тестирования 60](#_Toc199413114)

[3.2.1 Результаты тестирования модуля предобработки звукового сигнала 60](#_Toc199413115)

[3.2.2 Результаты тестирования модуля графического интерфейса 60](#_Toc199413116)

[3.2.3 Общие результаты тестирования 60](#_Toc199413117)

[ЗАКЛЮЧЕНИЕ 62](#_Toc199413118)

[Список использованных источников 64](#_Toc199413119)

[Приложение А. Код системы 66](#_Toc199413120)

[Приложение А.1. CTC\_train.py 66](#_Toc199413121)

[Приложение A.2. app.py 75](#_Toc199413122)

[Приложение Б. Демонстрационный материал 89](#_Toc199413123)

##### ПЕРЕЧЕНЬ УСЛОВНЫХ ОБОЗНАЧЕНИЙ

В тексте выпускной квалификационной работы введены специальные термины на русском и английском языках:

| Термин, сокращение | Полная форма |
| --- | --- |
| ВКР | Выпускная квалификационная работа |
| ASR | Automatic Speech Recognition (Автоматическое распознавание речи) |
| CNN | Convolutional neural network (Сверточная нейронная сеть) - тип нейронной сети прямого распространения, которая изучает признаки с помощью фильтра (ядра). |
| RNN | Recurrent neural network (Рекуррентная нейронная сеть) - представляет собой класс искусственных нейронных сетей, разработанных для обработки последовательных данных, таких как текст, речь и временные ряды. |
| LSTM | Long short-term memory (Долгая краткосрочная память) - является разновидностью RNN, направленной на смягчение проблемы исчезновения градиента, обычно встречаемой традиционными RNN. |
| biLSTM | Bidirectional LSTM (Двунаправленный LSTM) |
| CTC | Connectionist temporal classification - это тип вывода нейронной сети и связанная с ним функция оценки для обучения RNN, таких как сети LSTM, для решения задач последовательности, где время является переменным. |
| WER | Word Error Rate (Коэффициент ошибочных слов) - является распространенным показателем точности системы распознавания речи или машинного перевода. |
| Conv2D | Слой свертки по двумерным данных в PyTorch. |
| BatchNorm2d | Нормализация пакета данных в PyTorch. |

##### ВВЕДЕНИЕ

Автоматическое распознавание речи (Automatic Speech Recognition, ASR) является одной из фундаментальных технологий в области искусственного интеллекта и обработки естественного языка. Системы ASR обеспечивают преобразование звуковой речи человека в текстовый формат, что становится основой для многочисленных современных приложений и сервисов. В условиях активной цифровизации всех сфер жизни общества технологии распознавания речи приобретают особую значимость как эффективный механизм человеко-машинного взаимодействия.

Актуальность разработки систем автоматического распознавания речи обусловлена их широким применением в различных сферах: голосовое управление устройствами, цифровые голосовые ассистенты, системы автоматического перевода речи, технологии доступности для людей с ограниченными возможностями, транскрибация лекций и аудиозаписей, системы безопасности на основе голосовой биометрии. Особую важность имеют системы, не зависящие от говорящего, которые способны адаптироваться к различным дикторам независимо от их пола, возраста, акцента и других индивидуальных особенностей голоса.

В настоящее время развитие ASR-систем основывается на применении методов глубокого обучения, в частности, рекуррентно-сверточных нейронных архитектур, которые демонстрируют значительный прогресс в точности распознавания по сравнению с традиционными подходами. Современные исследования в этой области направлены на разработку эффективных архитектур нейронных сетей, способных обрабатывать акустические сигналы с высокой точностью даже в условиях сложной звуковой среды.

Целью данной работы является разработка системы автоматического распознавания русской речи на основе рекуррентно-сверточной нейронной архитектуры, позволяющей преобразовывать речевой сигнал в текст в режиме реального времени с высокой точностью распознавания. Система должна обеспечивать распознавание речи независимо от индивидуальных особенностей говорящего.

# АНАЛИЗ ИСХОДНЫХ ДАННЫХ И ПРОЕКТИРОВАНИЕ

## Постановка и анализ задачи

### Анализ существующих подходов к распознаванию речи

Развитие технологии распознавания речи прошло несколько этапов эволюции, начиная от простых статистических моделей до современных глубоких нейронных сетей. Основные подходы, существующие в настоящее время:

**Скрытые марковские модели (Hidden Markov Models, HMM)** исторически первый эффективный подход к распознаванию речи, основанный на статистическом моделировании. В основе HMM лежит вероятностная модель, где речь разбивается на фонемы, а переходы между ними описываются Марковской цепью. Состояние цепи генерирует акустические признаки. Несмотря на долгое доминирование, HMM имеют ограничения в моделировании долговременных зависимостей в речевом сигнале и ограниченную гибкость при работе с шумом и разнородными данными.

**Глубокие нейронные сети (Deep Neural Networks, DNN)** с развитием вычислительных мощностей стали основой современных систем распознавания речи. DNN способны автоматически извлекать множество признаков из сырых данных, что существенно повышает точность распознавания. Среди архитектур глубоких нейронных сетей, применяемых для ASR, выделяются:

* **Сверточные нейронные сети (CNN)** — эффективны для извлечения пространственных признаков из спектрограмм речи;
* **Рекуррентные нейронные сети (RNN)** — моделируют временные зависимости в последовательных данных;
* **Сети с долгой краткосрочной памятью (LSTM)** — разновидность RNN, способная к обучению долговременным зависимостям в последовательностях;
* **Двунаправленные LSTM (BiLSTM)** — учитывают контекст как из прошлого, так и из будущего, что является важной составляющей для распознавания речи.

**Гибридные модели** комбинируют преимущества различных подходов, например, CNN для извлечения признаков и RNN/LSTM для моделирования временных зависимостей. Такой тип модели, представленный в 2018 году Google DeepMind, достигнул ошибки 10,2 % по метрике WER [1].

**Трансформеры и модели с механизмом внимания** — архитектура Transformer была предложена в 2017 году как замена RNN (LSTM) и CNN для последовательных данных, полностью опирающаяся на механизм самовнимания (self-attention). В модели имеются два основных блока, называемых кодировщиком (Encoder), выполняющим функцию извлечения скрытых признаков из входных данных, и декодировщиком (Decoder), формирующим на основе этих признаков представление текста. Эти блоки и механизм self-attention позволяют эффективно учитывать как локальные, так и глобальные зависимости во входной последовательности. Эти архитектуры превышают точность моделей основанных на CNN-RNN, но при обучении требуют намного больших вычислительных мощностей и огромных объемов обучающих данных.

### Метрика оценки качества распознавания речи

Для объективной оценки качества работы системы распознавания речи используется метрика **коэффициента ошибок в словах (Word Error Rate, WER)** — основная метрика оценки качества ASR-систем, рассчитываемая как отношение суммы ошибок замены, вставки и удаления слов в предсказанном тексте к общему количеству слов в эталонной транскрибированной речи:

,

где S (Substitutions) — количество замен;

D (Deletions) — количество пропусков;

I (Insertions) — количество лишних вставок;

N — общее количество слов в эталонном тексте.

Пример расчета WER для целевой последовательности “Привет как дела”. Допустим система распознавания речи выдает следующий результат распознавания “Привед дела”, тогда число подстановок S = 1 из-за необходимости замены “Привед” на “Привет”, число вставок I = 1 из-за того, что слово “как” пропущено, при этом удалять ничего не нужно D = 0.

Тогда расчет .

Чем ниже WER, тем выше точность распознавания.

### Критерии выбора подхода для разрабатываемой системы

Для создания эффективной системы распознавания речи, не зависящей от говорящего, необходимо учитывать следующие требования:

1. Независимость от диктора — отсутствие дополнительного шага обучения или калибровки под голос конкретного человека. Система должна сразу “понимать” любого пользователя, сохраняя стабильную точность распознавания речи;
2. Высокая точность распознавания — минимизация ошибок распознавания, измеряемая метрикой коэффициента ошибок в словах (Word Error Rate, WER) тестовых корпусах не выше 15–20 %;
3. Работа в реальном времени — способность системы обрабатывать аудиопоток с минимальной задержкой до 5 секунд;
4. Распознавание речи на русском языке;
5. Кроссплатформенность — возможность использования на различных операционных системах (Windows, Linux).

Также при выборе подхода для реализации системы распознавания речи необходимо учитывать следующие факторы:

* Доступные данные для обучения — объем и качество обучающей выборки, наличие размеченных данных на русском языке;
* Вычислительные ресурсы — требования к аппаратному обеспечению для обучения модели и ее дальнейшего использования.

Эти требования и факторы определяют выбор архитектуры модели и инструментов разработки, а также задают метрики для оценки качества системы. На основе анализа требований и существующих подходов, для разрабатываемой системы выбран гибридный подход, сочетающий сверточные нейронные сети для извлечения акустических признаков и двунаправленные LSTM для моделирования временных зависимостей. Данный выбор обусловлен следующими преимуществами:

* Эффективное извлечение признаков — сверточные слои позволяют извлекать локальные акустические признаки из мел-спектрограмм с уменьшением размерности для последующей подачи в слои LSTM;
* Моделирование контекста — двунаправленный LSTM учитывает как предыдущий, так и последующий контекст, что имеет влияние при распознавании речи;
* Баланс между точностью и вычислительной эффективностью — гибридная архитектура обеспечивает хорошую точность распознавания при умеренных требованиях к вычислительным ресурсам.

Обучающий набор данных состоит из двух наборов данных:

* Golos [2] в основном состоит из записанных аудиофайлов, вручную аннотированных. Общая продолжительность аудио составляет около 1240 часов. Практически идеальная разметка;
* Open STT phone calls [3] состоит из аудио файлов звонков, собранных из открытых источников. Общая продолжительность аудио составляет около 811 часов. Некоторые типы аудио аннотируются автоматически и проверяются статистически с использованием эвристики. Качество разметки хуже, чем Golos.

## Проектирование архитектуры системы

### Общая концепция архитектуры системы распознавания речи

Основная цель проектируемой системы — обеспечить распознавание русской речи, не зависящей от говорящего, в реальном времени с последующим сохранением результатов в текстовом формате. Для достижения этой цели архитектура системы должна обеспечивать эффективное взаимодействие различных компонентов, ответственных за обработку звукового сигнала и преобразование его в текст с отображением в графическом интерфейсе.

Принципы, положенные в основу проектирования архитектуры системы:

* Модульность — разделение системы на независимые компоненты, каждый из которых выполняет определенные функции. Такой подход обеспечивает гибкость при разработке и возможность независимого тестирования отдельных модулей;
* Последовательность обработки данных — организация потока данных между модулями в логической последовательности их обработки;
* Кроссплатформенность — обеспечение работоспособности системы на различных операционных системах (Windows, Linux).

На основе этих принципов предлагается трехкомпонентная архитектура системы (Рисунок 1), где каждый из модулей отвечает за определённый этап обработки аудиосигнала.

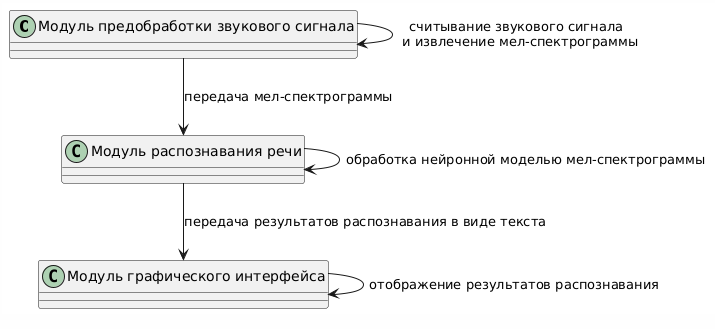


Рисунок 1 — Компоненты системы автоматического распознавания речи.

Модули системы взаимодействуют последовательно по принципу конвейера: выходные данные одного модуля являются входными для следующего. Обмен данными между модулями, работающими как потоки, осуществляется через потокобезопасные очереди: audio\_queue для передачи аудиоданных из потока записи в поток обработки и transcribation\_queue для передачи распознанного текста из потока обработки в поток графического интерфейса.

Такая архитектурная организация обеспечивает высокую модульность системы, что позволяет осуществлять независимую разработку и тестирование каждого компонента, а также упрощает дальнейшее развитие и масштабирование системы при необходимости.

### Проектирование модуля предобработки звукового сигнала

**Модуль предобработки звукового сигнала** — отвечает за получение и буферизацию аудиоданных с микрофона, а также за извлечение мел-спектрограммы. Основная функция данного модуля — преобразование амплитудных отсчётов в мел-спектрограмму, которая является более информативным представлением звуковых данных для последующего анализа. Использование мел-шкалы обусловлено тем, что она лучше соответствует особенностям человеческого восприятия звука: человеческое ухо более чувствительно к изменениям на низких частотах и менее чувствительно на высоких.

#### Функциональные требования для модуля предобработки звукового сигнала

Функции, которые должен выполнять модуль предобработки звукового сигнала:

1. Захват аудиосигнала с устройства записи (микрофона) в реальном времени;
2. Буферизация звукового сигнала для обработки блоками (фиксированное число амплитудных отсчетов);
3. Извлечение акустических признаков из звукового сигнала;
4. Передача обработанных данных модулю распознавания речи.

#### Выбор метода представления звукового сигнала

В качестве метода представления звукового сигнала для задачи распознавания речи выбрана мел-спектрограмма. Данный выбор обусловлен следующими факторами:

* **Биологическая обоснованность** — мел-шкала основана на особенностях восприятия звука человеческим ухом, которое более чувствительно к изменениям на низких частотах и менее чувствительно на высоких;
* **Информативность представления** — мел-спектрограмма позволяет получить компактное и информативное представление речи за счет неравномерного распределения частотных бинов;
* **Практическая эффективность** — исследования [4,5] показывают, что использование мел-спектрограмм в качестве входных данных для нейронных сетей обеспечивает высокую точность распознавания речи.

При проектировании модуля необходимо определить оптимальные параметры мел-спектрограммы. На основе анализа существующих решений и экспериментальных данных используются следующие параметры:

* Число мел-уровней: 80
* Размер окна: 25 мс (400 отсчетов при частоте дискретизации 16 кГц)
* Сдвиг окна: 10 мс (160 отсчетов при частоте дискретизации 16 кГц)
* Максимальная частота: 8 кГц

Выбор 80 мел-уровней объясняется тем, что увеличение этого параметра сверх определенного значения не приводит к значительному улучшению точности распознавания, однако увеличивает вычислительную сложность. Экспериментальные данные, представленные на рисунке 2, подтверждают оптимальность выбранного значения.

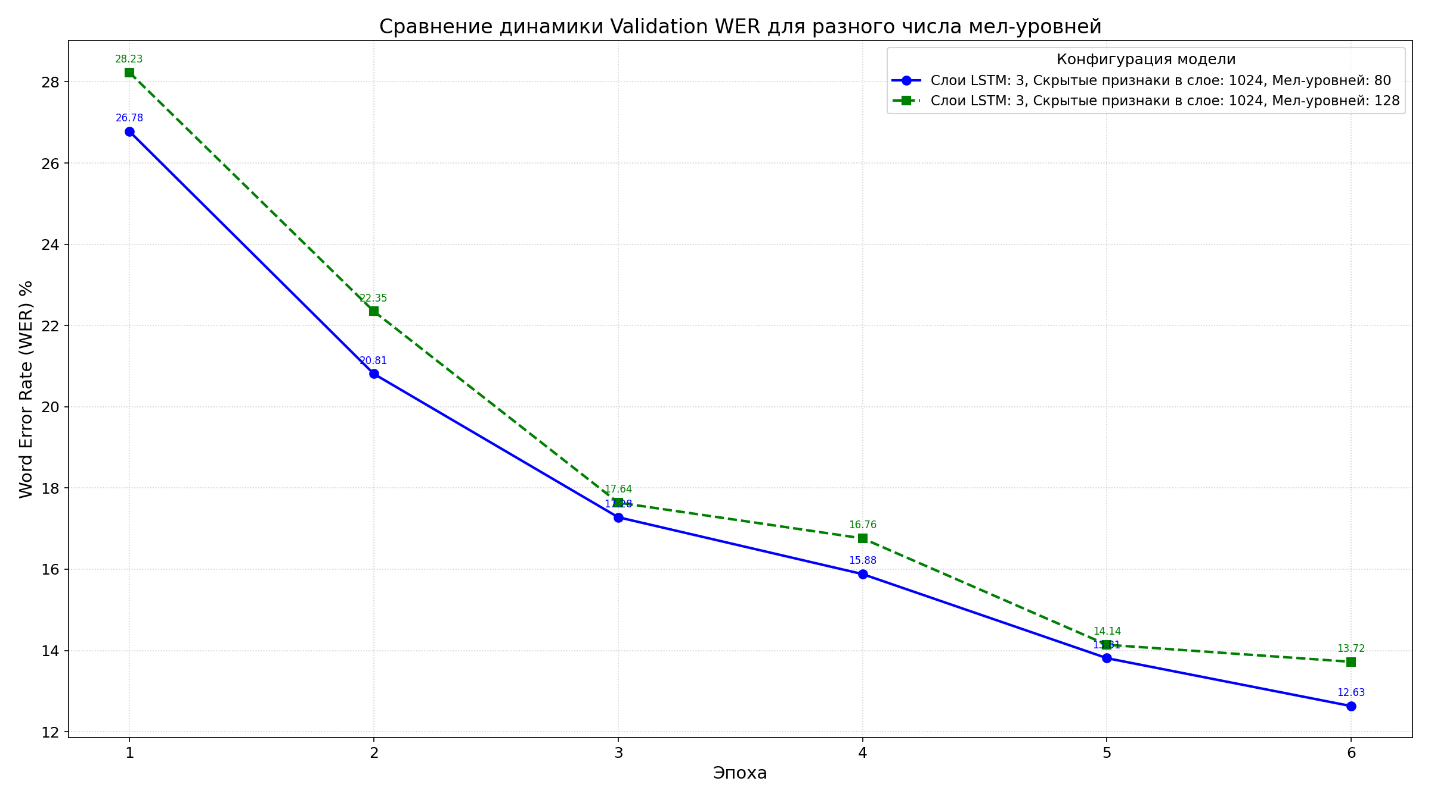


Рисунок 2 — Отображение влияния числа мел-уровней на точность распознавания.

Размер окна 25 мс является компромиссным решением между временным и частотным разрешением: слишком короткое окно не позволяет адекватно представить низкочастотные компоненты речи, а слишком длинное окно не обеспечивает необходимое временное разрешение для быстро меняющихся звуков.

#### Проектирование функции буферизации и обработки аудиоданных

Для обработки аудиосигнала в реальном времени необходима эффективная система буферизации, которая выполняет следующие функции:

1. Накопление входных отсчетов до достижения требуемого объема для обработки;
2. Предотвращение потери данных при неравномерности поступления отсчетов (если запись закончилась, прежде чем поступило достаточное число отсчетов для обработки).

Предлагается использовать поток, считывающий аудиосигнал и возвращающий каждый раз фиксированное число амплитудных отсчетов, помещаемых в очередь на обработку. Аудиосигнал должен представляться амплитудными отсчетами с частотой дискретизации 16 КГц, так как модель распознавания речи работает именно с такими данными. При этом запись является одноканальной, при наличии нескольких аудиоканалов выбирается один.

Для обеспечения потокобезопасности при многопоточной работе системы (Поток записи в буфер и поток считывания из буфера) используется буфер с механизмом блокировки. При накоплении определенного числа отсчетов в буфере они собираются в блок, из которого извлекается мел-спектрограмма с заданными в пункте 1.2.2.2 параметрами. Полученная спектрограмма отправляется дальше в модель для обработки.

### Проектирование модуля распознавания речи

**Модуль распознавания речи** — представляет собой ядро системы, включающее CNN-biLSTM модель для преобразования акустических признаков в текстовую форму. Модуль принимает на вход мел-спектрограмму и с помощью рекуррентно-сверточной архитектуры интерпретирует последовательность звуковых данных в последовательность токенов. Токены включают в себя русский алфавит, пробел и специальный символ “\_” для работы с алгоритмом CTC (Connectionist Temporal Classification). Полученная последовательность токенов преобразуется в текст при помощи алгоритма жадного декодирования.

#### Обоснование выбора архитектуры нейронной сети

При выборе архитектуры нейронной сети для задачи распознавания речи необходимо учитывать следующие требования:

1. Способность обрабатывать последовательности переменной длины;
2. Эффективность анализа как локальных, так и глобальных признаков в аудиосигнале;
3. Вычислительная эффективность для работы в реальном времени.

На основе анализа современных подходов к распознаванию речи предлагается использовать гибридную архитектуру, сочетающую сверточные и рекуррентные слои. Такая архитектура позволяет эффективно выделять как пространственные (частотные), так и временные закономерности в речевом сигнале. Используемая архитектура нейронной сети включает:

* **Сверточные слои** для выделения локальных признаков из мел-спектрограммы;
* **Рекуррентные слои** (LSTM) для моделирования временных зависимостей;
* **Полносвязный слой** для классификации выходных последовательностей.

#### Проектирование компонентов нейронной модели

**Сверточные слои** предназначены для извлечения локальных признаков из мел-спектрограммы. Предлагается использовать два последовательных сверточных слоя со следующими параметрами:

* Размер ядра: 3×3 (kernel size)
* Шаг: 2×2 (stride)
* Дополнение: 1×1 (padding)
* Количество выходных каналов: 32 (output channels)

После каждого сверточного слоя применяется функция активации ReLU и нормализация по пакету данных (Batch Normalization). Нормализация по пакету данных обладает следующими преимуществами:

* Ускоряет сходимость процесса обучения;
* Снижает зависимость от начальной инициализации весов и увеличивает способности обобщения модели при обучении;
* Действует как регуляризатор, предотвращая переобучение (позволяет уменьшить значение прореживания весов (dropout) или вовсе убрать его).

**Рекуррентные слои** предназначены для моделирования временных зависимостей в последовательности признаков. Предлагается использовать двунаправленные LSTM-слои (BiLSTM), которые обрабатывают последовательность в обоих направлениях, что позволяет учитывать как предыдущий, так и последующий контекст при распознавании каждого элемента последовательности.

Выбор LSTM обусловлен их способностью эффективно моделировать долговременные зависимости без проблемы исчезающих градиентов, характерной для простых рекуррентных сетей. Это особенно важно при распознавании речи, где контекст может определяться элементами, находящимися на расстоянии друг от друга во временном ряду.

**Выходной слой** представляет собой полносвязный слой с размерностью 2048, который отображается на 35 классов, соответствующих символам русского алфавита, пробелу и специальному символу для CTC-алгоритма “\_”.

#### Метод декодирования последовательностей

Для выхода нейронной модели предлагается использовать алгоритм Connectionist Temporal Classification (CTC), который позволяет обучать модель без необходимости точного временного выравнивания между входным сигналом и выходной последовательностью символов.

CTC-алгоритм вводит специальный символ “пустого перехода” (\_), который позволяет моделировать паузы между символами и повторения символов. После обучения модели с использованием CTC-функции потерь применяется жадное декодирование для получения текстовой последовательности. Алгоритм жадного декодирования на рисунке 3.

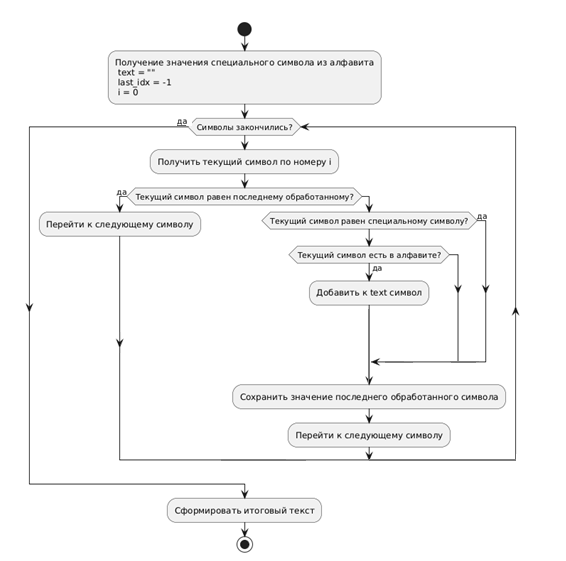


Рисунок 3 — Диаграмма жадного декодирования.

Этот алгоритм последовательно просматривает выходные вероятности модели, выбирая символ с наибольшей вероятностью в каждый момент времени, и применяет правила для объединения символов:

1. Удаление повторяющихся символов;

2. Удаление специального символа “пустого перехода”.

### Проектирование модуля пользовательского интерфейса

**Модуль пользовательского интерфейса** — обеспечивает взаимодействие пользователя с системой. Данный модуль отвечает за отображение распознанного текста внутри графического интерфейса в реальном времени, а также предоставляет возможности начала/конца записи аудио и сохранения текста в выбранную пользователем директорию по окончании записи. Интерфейс должен обеспечивать кроссплатформенность приложения на Windows и Linux.

Модуль пользовательского интерфейса обеспечивает взаимодействие пользователя с системой распознавания речи. Основные требования к этому модулю:

* Минимальное количество элементов управления;
* Наглядное отображение результатов распознавания;
* Возможность сохранения результатов в файл;
* Возможность запуска/остановки записи.

#### Общая структура пользовательского интерфейса

Графический интерфейс состоит из следующих элементов:

1. Верхняя панель с элементами управления содержит:

* Кнопка выбора директории для сохранения результатов;
* Текстовое поле, отображающее выбранный путь сохранения;
* Кнопка “Начать запись” / “Остановить запись”;
* Индикатор состояния записи.

1. Центральная область для отображения результатов распознавания:

* Многострочное текстовое поле с вертикальной полосой прокрутки.

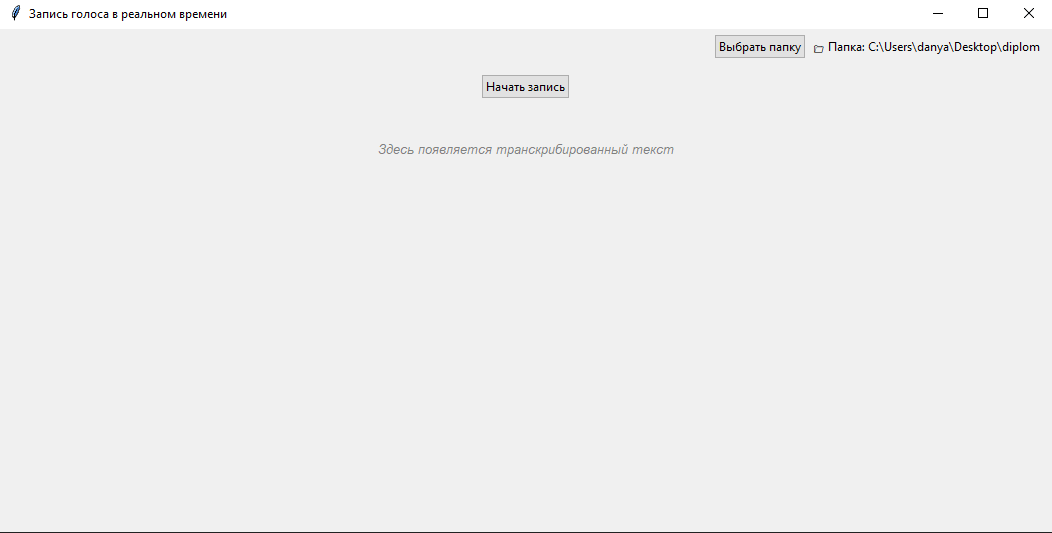


Рисунок 4 — Прототип графического интерфейса.

#### Проектирование механизма взаимодействия с пользователем

Взаимодействие пользователя с системой осуществляется через следующие элементы:

1. **Кнопка “Выбрать папку”** — открывает диалоговое окно выбора директории для сохранения текстовых результатов распознавания;
2. **Кнопка “Начать запись”** — инициирует процесс записи аудио с микрофона. При нажатии кнопка меняет свою надпись на “Остановить запись” и активирует индикатор состояния записи для наглядного отображения работы системы распознавания речи;
3. **Кнопка “Остановить запись”** — останавливает процесс записи, сохраняет результаты распознавания в выбранную директорию и возвращает кнопку в исходное состояние **“Начать запись” c деактивацией индикатора состояния записи**;
4. Текстовое поле результатов — отображает распознанный текст в реальном времени по мере его поступления от модуля распознавания речи.

#### Проектирование механизма обновления интерфейса

Для обеспечения производительной работы интерфейса при длительных операциях распознавания речи предлагается использовать многопоточную архитектуру:

1. **Главный поток** — отвечает за отображение интерфейса, обработку событий пользователя (Нажатие кнопок) и отображения результата распознавания во время записи, извлекая его из очереди транскрибированного текста;
2. **Поток записи** — выполняет считывание аудиоданных с микрофона и помещает их в очередь для обработки;
3. **Поток обработки** — извлекает данные из очереди, выполняет их предобработку и распознавание, после чего помещает результаты в очередь для транскрибированного текста.

Обновление графического интерфейса происходит с использованием механизма периодического опроса очереди результатов. Главный поток с определенным интервалом (100 мс) проверяет наличие новых результатов распознавания и обновляет текстовое поле при их наличии.

## Обоснование выбора средств разработки программного обеспечения

### Выбор языка программирования

В качестве языка программирования для системы автоматического распознавания речи был выбран Python. Данный выбор обоснован следующими факторами:

* **Обширная экосистема библиотек для работы с данными и машинного обучения**. Python предоставляет доступ к многочисленным библиотекам, специализированным для обработки аудиоданных (librosa, sounddevice), математических вычислений (NumPy, SciPy), машинного обучения (PyTorch, TensorFlow) и обработки данных (pandas). Это существенно сокращает время разработки и позволяет сконцентрироваться на решении основной задачи;
* **Скорость разработки**. Python является высокоуровневым языком с лаконичным синтаксисом, что обеспечивает высокую скорость написания и отладки кода. Динамическая типизация и интерпретируемость позволяют быстро тестировать отдельные компоненты системы;
* **Кроссплатформенность**. Разработанное на Python приложение может работать на различных операционных системах (Windows, Linux) без необходимости внесения существенных изменений в код, что обеспечивает широкий охват пользовательской аудитории;
* **Популярность в научной среде**. Python широко используется в научном сообществе для исследований в области машинного обучения. Это обеспечивает доступ к актуальным научным разработкам и возможность интеграции современных алгоритмов в систему.

### Библиотеки для обработки аудио и извлечения признаков

Для извлечения мел-спектрограмм из аудиосигнала выбрана библиотека librosa. Выбор обоснован следующими преимуществами:

* **Специализация на анализе аудио и тонкая настройка параметров**. Librosa предоставляет обширный набор функций для анализа и обработки аудиосигналов, включая извлечение различных спектральных признаков, ритмических характеристик и тональных особенностей. Библиотека позволяет тонко настраивать параметры извлечения мел-спектрограмм, включая число мел-уровней, размер окна анализа, величину перекрытия окон, что критически важно для качества распознавания и тонкой настройки обучаемой модели;
* **Совместимость с обучением модели.** Внутренняя реализация алгоритмов библиотеки оптимизирована для быстрой работы с использованием NumPy для векторизации вычислений и подходит для дальнейшей передачи результатов в нейронную модель;
* **Документация и поддержка**. Библиотека имеет исчерпывающую документацию с примерами использования и активное сообщество пользователей, что упрощает интеграцию и отладку.

Для записи звука в реальном времени выбрана библиотека sounddevice, что обосновано следующими факторами:

* **Низкоуровневый доступ к аудиоустройствам и функции обработки**. Библиотека предоставляет прямой доступ к системным аудиоустройствам через интерфейсы PortAudio, написанном на c++. Sounddevice поддерживает потоковую обработку аудиоданных через механизм обратных вызовов, что необходимо для работы системы распознавания речи в реальном времени. Также библиотека позволяет точно настраивать параметры записи: частоту дискретизации, число каналов, размер буфера, что делает возможной тонкую настройку для производительности системы;
* **Кроссплатформенность**. Благодаря использованию PortAudio, библиотека обеспечивает единый интерфейс доступа к аудиоустройствам на различных операционных системах (Windows, Linux);
* **Интеграция с NumPy**. Sounddevice предоставляет аудиоданные в виде массивов NumPy, что упрощает их последующую обработку и анализ далее в программной системе.

### Библиотека машинного обучения

В качестве библиотеки для разработки и обучения нейронных моделей выбран PyTorch вместо TensorFlow из-за того, что лучше подходит для исследований и прототипирования благодаря лёгкости отладки и гибкости. Также выбор обоснован следующими возможностями:

* **Оптимизированные операции для обработки тензоров**. Библиотека предоставляет эффективные реализации операций с тензорами, оптимизированные для выполнения на GPU, что существенно ускоряет обучение и вывод результата нейронных моделей;
* **Богатый набор предопределенных слоев**. PyTorch включает в себя обширную библиотеку готовых компонентов нейронных сетей (свёрточные слои, рекуррентные блоки, механизмы внимания), что ускоряет процесс создания архитектуры модели;
* **Поддержка распределенного обучения**. Библиотека обеспечивает интеграцию с CUDA для выполнения вычислений на GPU, что критически важно для обучения сложных нейронных моделей. Также предоставляет инструменты для распределенного обучения моделей на нескольких GPU или вычислительных узлах, что позволяет эффективно использовать доступные вычислительные ресурсы.

### Средства разработки пользовательского интерфейса

Для разработки графического пользовательского интерфейса выбрана библиотека tkinter, что обосновано следующими факторами:

* **Стандартная библиотека Python**. Tkinter входит в стандартную поставку Python, что исключает необходимость установки дополнительных зависимостей и упрощает развертывание приложения;
* **Кроссплатформенность**. Интерфейсы, разработанные с использованием tkinter, работают на различных операционных системах без изменений, что обеспечивает широкий охват платформ;
* **Низкие требования к ресурсам**. Tkinter отличается низким потреблением ресурсов, что особенно важно для системы распознавания речи, где основная вычислительная нагрузка приходится на обработку аудио и вывод результата нейронной моделью.

### Вспомогательные библиотеки

Для обработки и подготовки данных для обучения модели выбрана библиотека pandas, что обосновано следующими факторами:

* **Эффективная работа с табличными данными**. Pandas предоставляет удобные структуры данных (DataFrame) для эффективной обработки табличных данных;
* **Интеграция с различными форматами данных**. Библиотека поддерживает чтение и запись данных в различных форматах (CSV, Excel, SQL, JSON), что упрощает подготовку и анализ обучающих данных;
* **Мощные возможности предобработки данных**. Pandas предоставляет широкий спектр функций для очистки, трансформации и нормализации данных, необходимых для подготовки качественных обучающих наборов;
* **Оптимизация производительности**. Внутренняя реализация библиотеки оптимизирована для эффективной обработки больших объемов данных. При этом тесно интегрирован с NumPy, что обеспечивает совместимость с другими компонентами системы распознавания речи.

Для оценки качества распознавания речи выбрана библиотека jiwer, что обосновано следующими факторами:

* **Специализация на метриках качества распознавания речи**. JiWER предоставляет реализации стандартных метрик оценки качества распознавания речи, включая Word Error Rate (WER), Character Error Rate (CER), Match Error Rate (MER);
* **Высокая производительность**. Библиотека внутри реализована с использованием C++, что обеспечивает высокую скорость вычисления метрик на больших наборах данных;
* **Простота использования**. JiWER предоставляет простой и понятный API, что упрощает интеграцию метрик в процесс обучения и оценки модели.

Выбранные средства разработки образуют целостную экосистему, обеспечивающую решение всех аспектов создания системы автоматического распознавания речи: от подготовки обучающего набора данных до обучения нейронной модели и создания пользовательского интерфейса. Сочетание этих инструментов позволяет достичь баланса между производительностью, гибкостью и скоростью разработки.

# Разработка системы

## Архитектура разрабатываемой системы

Общая архитектура системы была представлена на рисунке 1. Разрабатываемая система состоит из трёх основных модулей, каждый из которых выполняет свою функцию в процессе обработки аудиосигнала и получения распознанного текста:

* Модуль предварительной обработки звукового сигнала считывает аудиосигнал с устройства записи (микрофона) и буферизует его во внутренней очереди. Далее производится извлечение признаков — формируется мел-спектрограмма, которая является информативным представлением звуковых данных для последующего анализа.
* Модуль распознавания речи принимает на входе мел-спектрограмму, которая далее обрабатывается обученной нейронной моделью, распознающей речевые шаблоны. Модель преобразует акустические признаки в текстовую форму, интерпретируя последовательность звуков в последовательность токенов. Токены представляют собой русский алфавит и два специальных символа (пробел и “\_”) для работы нейронной модели. Последовательность токенов преобразуется в текст.
* Модуль графического интерфейса получает текстовые результаты распознавания и обеспечивает отображение текста для пользователя в реальном времени. Также по окончании записи текст сохраняется в выбранную пользователем директорию.

Модули взаимодействуют последовательно: выходные данные одного модуля служат входом для следующего. Это позволяет обеспечить модульность системы, упрощая её развитие и масштабирование.

## Разработка модуля предобработки звукового сигнала

Весь обучающий набор данных представляет собой аудиозаписи в формате opus с частотой дискретизации 16 КГц и их транскрибированным текстом в формате txt. При формировании обучающего набора в программе обучения из записей в формате opus извлекается мел-спектрограмма. Преобразование в мел-спектрограмму позволяет получить более компактное и информативное представление речи по сравнению с обычной спектрограммой, уменьшая корреляцию между соседними частотными бинами. То есть несколько бинов обычной спектрограммы могут соответствовать одному мел-уровню, что уменьшает число хранимых данных, но также и несет более ценную информацию за счет того, что на низких частотах мел-шкала почти линейна, а на высоких частотах она становится все более логарифмической, то есть более широкие диапазоны реальных частот отображаются в более узкие диапазоны на мел-шкале.

Параметры мел-спектрограммы следующие:

1. Число мел-уровней 80 – среднее значение. Большое значение не увеличивает качество распознавания речи (Смотреть рисунок 2), но увеличивает размерность входных данных в нейронную модель. Слишком малое количество мел-уровней может привести к потери информации;
2. Окно равно 400 отсчетам, что равно 25 мс – среднее значение. Диапазон окна выбирается от 20 мс до 30 мс;
3. Сдвиг окна равен 160 отсчетам, что равно 10 мс;
4. Максимальная частота 8 КГц (Частота дискретизации 16 КГц).

При нехватке отсчетов до длины окна, отсчеты дополняются нулевыми значениями.

Параметры для считывания звукового сигнала следующие:

* Частота дискретизации 16000 Гц (частота дискретизации в обучающем наборе данных).
* Одноканальная запись.
* Число отсчетов в блоке данных 1024 при обратном вызове (Базовое значение для занесения в буфер).

В функции обратного вызова при получении данных они добавляются во внутренний буфер, после чего считается число отсчетов в этом буфере. Если число отсчетов в буфере больше или равно числу отсчетов для извлечения мел-спектрограммы, то это число отсчетов собирается вместе и отправляется в очередь audio\_queue на обработку для получения мел-спектрограммы. Алгоритм представлен на рисунке 5.

Из считанных данных мел-спектрограмма извлекается каждые n секунд (по умолчанию 5). Время по умолчанию выбрано для улучшения распознающих функций модели и удобства работы с отсчетами.

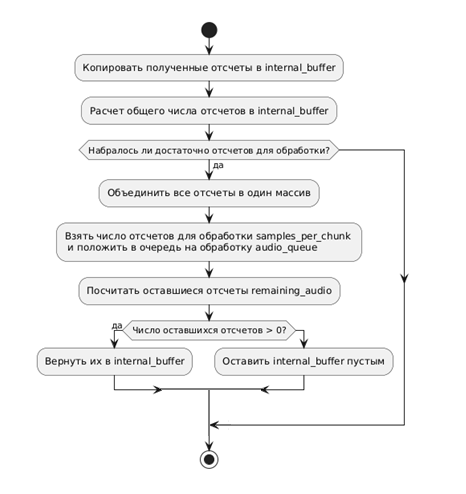


Рисунок 5 — Диаграмма обработки отсчетов.

При появлении данных в очереди они извлекаются из нее и идут на обработку для получения мел-спектрограммы (Алгоритм на рисунке 6).

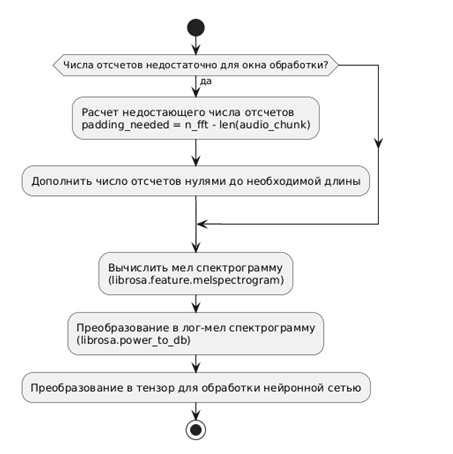


Рисунок 6 — Диаграмма получения мел-спектрограммы.

Далее мел-спектрограмма передается в модуль распознавания речи для дальнейшей обработки.

## Разработка модуля распознавания речи

### Архитектура нейронной модели

На вход модели подаются мел-спектрограммы(-а), которые после преобразуются в форму (B, 1, 80, T), где B - число поданных мел-спектрограмм, 1 - канал признаков, 80 - мел-уровни, T - число временных отсчетов. Число поданных мел-спектрограмм зависит от режима работы. При уже обученной модели это значение всегда будет равно 1, так как у нас будет только одно представление речи при распознавании в реальном времени. Но во время обучения применяется метод пакетной обработки для ускорения обучения и увеличения точности модели, тогда значение B является числом элементов в пакете обработки.

По форме (B, 1, 80, T) идет свертка два раза в такой последовательности:

1. Свертка с ядром 3x3, сдвигом 2x2, дополнением 1x1;
2. Функция активации ReLU;
3. Нормализация пакета (набора) данных.

Для выполнения свертки необходимо привести тензор к форме вида (B, C, H, W), где B-число экземпляров мел-спектрограмм в наборе данных, С-число каналов в наборе данных, H и W – высота (мел-уровни) и ширина (временные отсчеты) входных данных соответственно. Так как исходная реализация требует именно такой входной формы тензора, а текущая форма данных представляет (B, H, W), то необходимо добавить 1 фиктивный канал, который будет представлять те же данные, но в требуемом формате. Для этого используется метод unsqueeze(1) на тензоре (B, H, W) для получения тензора (B, 1, H, W).

Ядро свертки (kernel size) выбрано размерностью 3x3 на основании того, что это наименьший размер, который может захватывать пространственные признаки, а также на основании большого числа успешных архитектур сетей [6], использующих данные ядра. Ядра с большей размерностью увеличивают число параметров и следовательно вычислительные операции. Также при использовании нескольких последовательных слоев сверток после каждого из них используется функция активации линейного выпрямления (ReLU), что позволяет сети выучивать более сложные нелинейные признаки, если сравнивать с одним слоем свертки большей размерности и одной функцией активации.

Сдвиг ядра 2x2 (stride) является известной практикой для уменьшения размерности. Для мел-спектрограмм это является уменьшением временной и частотной размерностей в 2 раза. При этом это уменьшение положительно влияет при обучении на последующих слоях, так как большая временная последовательность может негативно сказаться в слоях LSTM и вызывать такие негативные эффекты как исчезающие градиенты (градиенты функции потерь относительно весов сети становятся чрезвычайно малыми по мере того, как они распространяются назад от выходного слоя к предыдущим слоям). Также слои LSTM работают быстрее на более коротких последовательностях и требуют меньше памяти.

Дополнение 1x1 (padding) исходных данных по краям необходимо для того, чтобы ядро свертки размером 3x3 могло полностью обработать все данные, включая крайние элементы, без их пропуска или неполной обработки. Это позволяет захватить признаки по всей области входных данных, включая границы.

Количество выходных каналов выбрано равным 32 на основании того, что это наименьшее из значений успешных архитектур сетей распознавания изображений VGGNet, ResNet, Inception. И при увеличении этого числа увеличивается вычислительная нагрузка при обучении, а так как вычислительные ресурсы ограничены, было принято решение начать с этого числа как с начальной отправной точки.

В глубоких сетях при обратном распространении ошибки малые градиенты перемножаются на каждом слое. В результате градиенты для ранних слоев сети могут стать исчезающе малыми, что практически останавливает их обучение. ReLU решает эту проблему для положительных активаций: Когда вход x > 0, производная ReLU равна 1. Это означает, что градиент проходит через активные ReLU нейроны без ослабления, позволяя ему эффективно распространяться вглубь сети и обучать даже самые первые слои. Также использование функции активации ReLU показало свою эффективность в следующих статьях [7,8].

Операция нормализации набора данных (BatchNorm2d) используется для ускорения обучения модели и уменьшения переобучения. В оригинальной статье [9] она используется после слоя свертки с целью привести среднее по набору данных к 0 и дисперсию к 1. Это помогает обобщить данные за счет того, что обучающий пример рассматривается вместе с другими примерами в наборе данных, и обучающаяся модель больше не производит детерминированных данных. Появляется избегание явления взрыва градиентов при обучении, то есть отсутствие переобучения в тех моментах, где аналогичная архитектура нейронной модели, но без нормализации, обучалась. Также эта нормализация позволяет либо уменьшить значение слоя прореживания весов (Dropout), либо полностью избавиться от него. Приведенные в статье данные показывают, что при нормализации на наборе данных модель достигает той же точности, но при в 14 раз меньшем количестве шагов обучения и превосходит исходную модель по точности распознавания при дальнейшем обучении.

Для данной работы производилась проверка нормализации до/после функции активации и экспериментальным путем при обучении было выяснено, что точность модели практически не изменяется (Разница на первой эпохе 0,31% на последующих еще меньше) при выборе расположения нормализации. Представленные ниже графики (Рисунок 7) отображают точность модели при разном порядке расположения нормализации набора данных. Для итоговой модели принято расположение нормализации данных после ReLU.

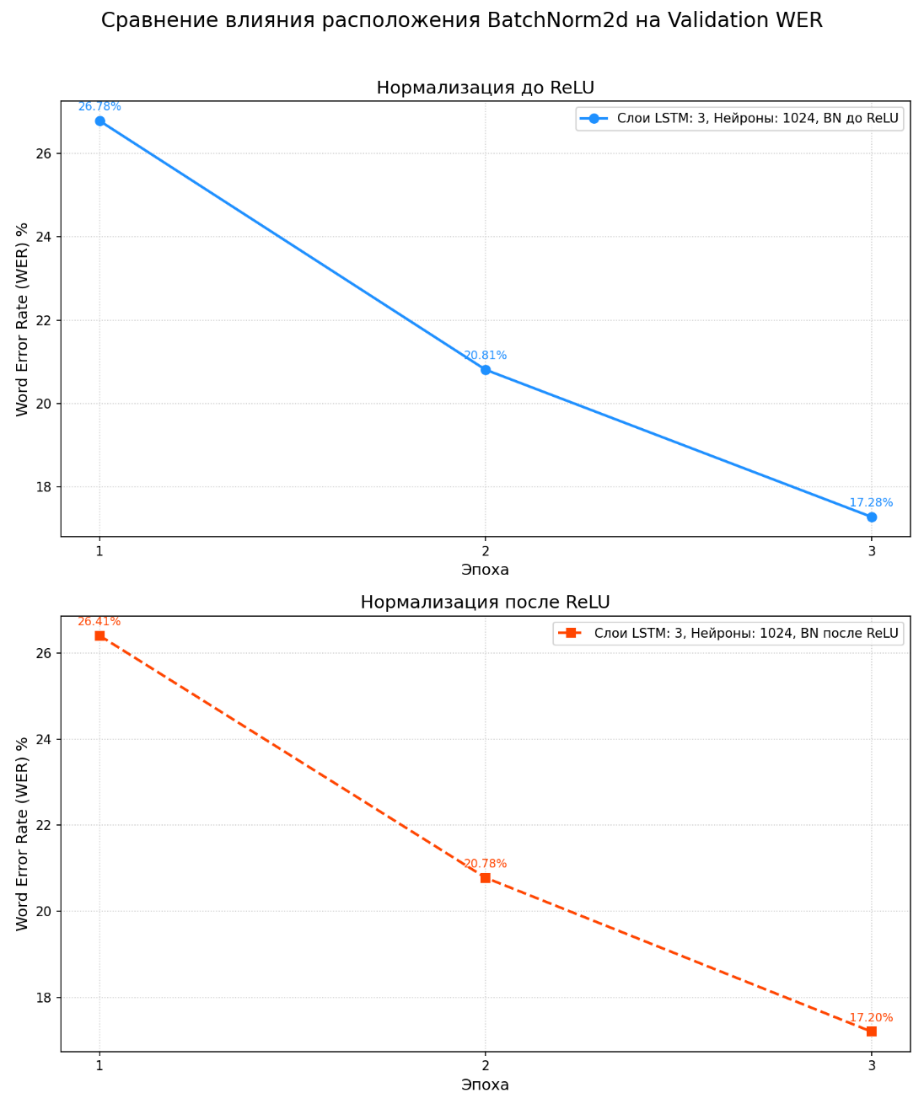


Рисунок 7 — Сравнение моделей с разным расположением операции нормализации данных.

Выполнив операции свертки, ReLU, нормализации 2 раза на выходе получается тензор признаков формы (B, 32, 20, T/4). Перед подачей в слой LSTM мы меняем форму тензора в (B, T/4, 32, 20). И совмещаем два последние измерения, чтобы получить полную размерность признаков (B, T/4, 640), так как наши признаки — это каналы свертки и частоты. Это совмещение измерений признаков в одно является необходимым требованием перед подачей во вход слоя LSTM.

Применяем на этом этапе метод обрубания весов для избегания переобучения DropOut с значением 0.3 (широко распространенное значение).

Речь — это временная последовательность, представленная амплитудными отсчетами. Значение текущего звука или слова часто зависит от предыдущих элементов. LSTM как раз предназначены для последовательностей: они являются типом рекуррентных нейронных сетей (RNN), специально разработанных для обучения на последовательных данных. Они обладают "памятью", позволяющей им учитывать предыдущую информацию при обработке текущего элемента последовательности. Благодаря своей способности удерживать информацию на протяжении многих временных шагов, они лучше справляются с улавливанием таких долгосрочных зависимостей по сравнению с другими моделями. Также для лучшего распознавания в данной архитектуре модели используется двунаправленное распознавание, то есть входная последовательность обрабатывается LSTM в двух направлениях:

* Прямое направление обрабатывает последовательность от начала до конца, используя прошлый контекст;
* Обратное направление обрабатывает последовательность от конца к началу, используя будущий контекст.

Текущий тензор поступает на вход LSTM слоя с числом скрытых признаков равным 1024. Число LSTM слоев выбрано равным 3, т.к. при увеличении числа слоев оценка по метрике WER критически высока (56-97%). На графике ниже (Рисунок 8) приведена статистика обучения моделей по метрике WER.

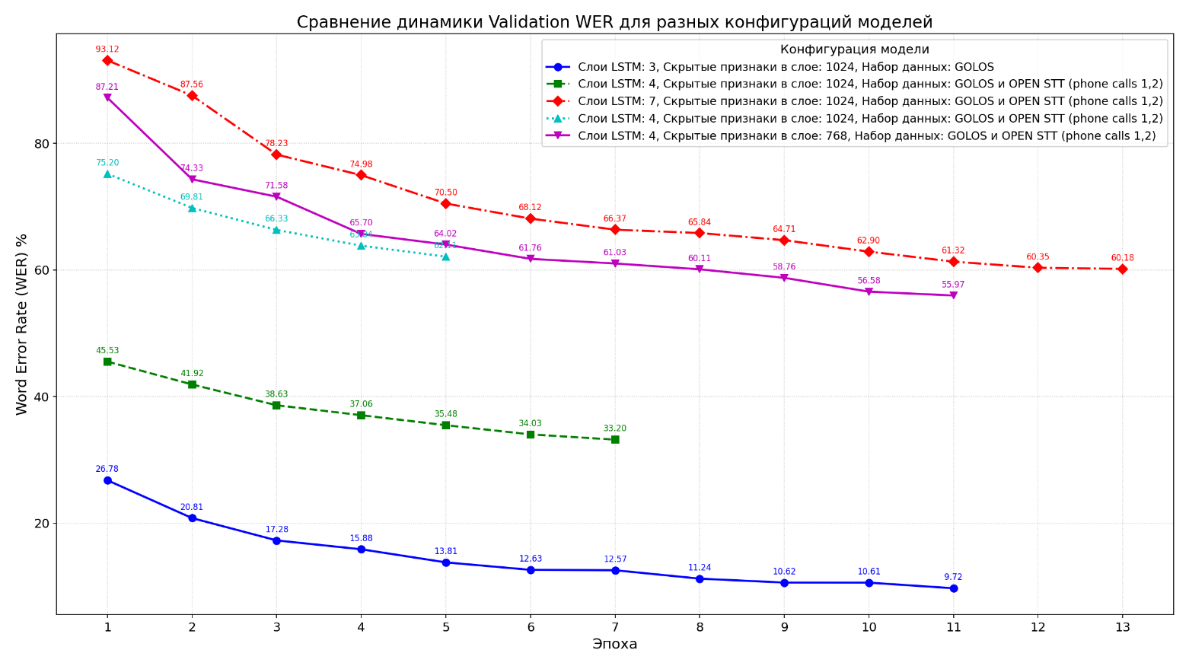


Рисунок 8 — Сравнение моделей с разными гиперпараметрами.

Выход представляет полно-связный слой из 2048 нейронов, который отображается на 35 классов (Русский алфавит, пробел и специальный символ для CTC “\_”). В конце применятся функция для вычисления вероятностей 35 классов. В итоге общая архитектура модели представляет собой рекуррентно-сверточную модель LSTM для распознавания последовательностей букв с помощью алгоритма CTC. Вся архитектура нейронной модели представлена на рисунке 9.

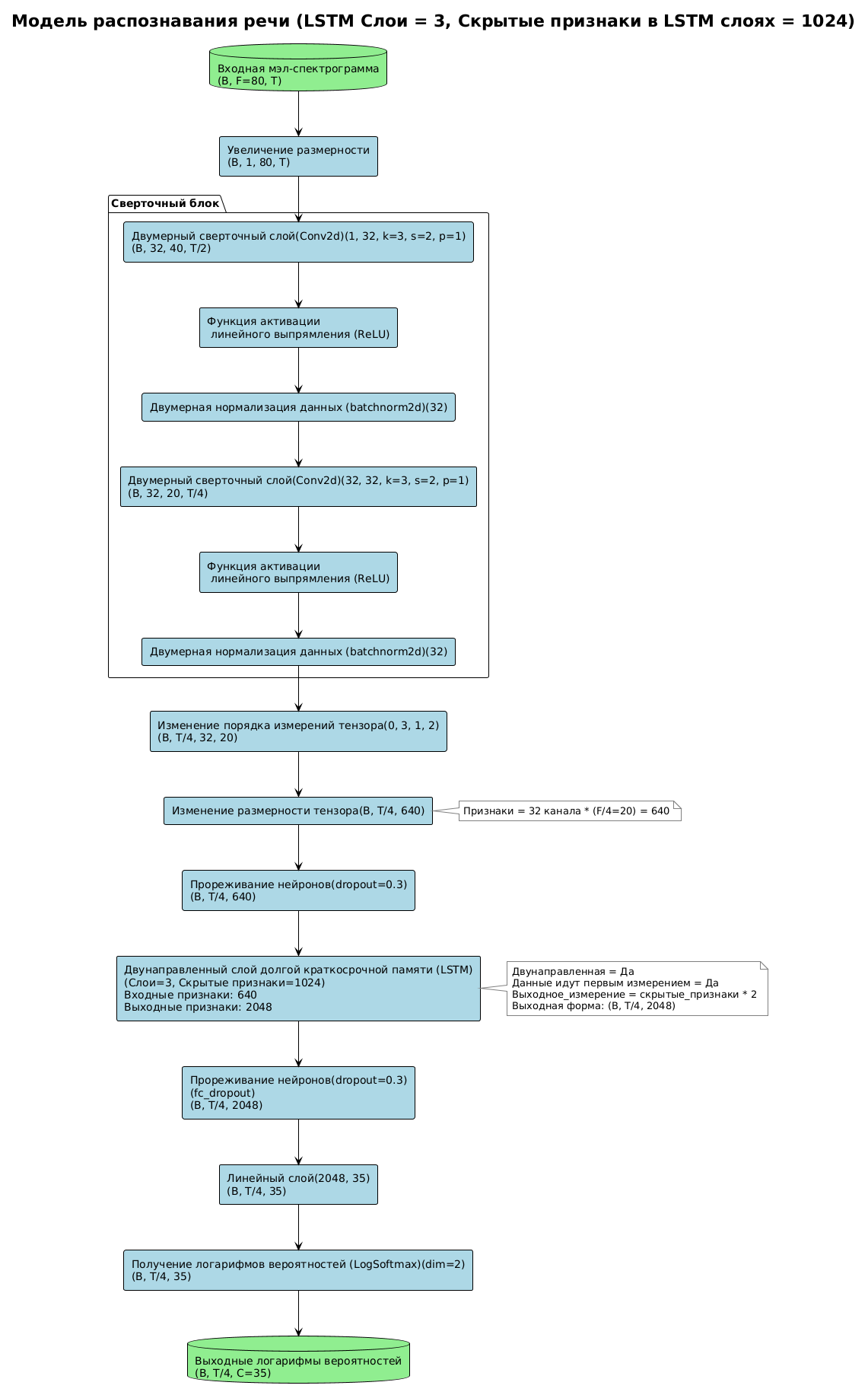


Рисунок 9 — Общая архитектура нейронной модели.

### Обучение нейронной модели

Для обучения модели создается класс SingleSourceAudioDataset, наследуемый от torch.Dataset и представляющий обучающий набор данных. Для его инициализации передаются файл(-ы) csv, содержащий названия аудиофайлов (столбец filename) и соответствующий им транскрибированный текст (столбец text), и путь к директории аудио файлов. Во время инициализации класса файл csv считывается с помощью функции read\_csv из библиотеки pandas. После считывания файла csv происходит фильтрация данных, соответствующая следующим критериям:

1. Должно быть значение в поле text файла данных csv;
2. Это значение text должно быть строкой без лишних символов (Знаки пунктуации, цифры и т.п), потому что алфавит распознавания ограничен русским алфавитом, пробелом и специальным символом “\_”;
3. Данные должны быть заново переупорядочены при удалении части из них, т.е. не должно быть такого порядка элементов 1,3,4,7, так как это вызовет ошибки при обучении, когда модель будет итерироваться по наборам данным.

Эти условия выполняются с помощью методов:

1. pandas.dropna удаляются все строки, где отсутствуют значения транскрибированного текста для аудиофайла. Это позволяет отсеять пустые данные из обучающей выборки;
2. audio\_target[audio\_target['text'].apply(lambda x: isinstance(x, str))] проверка, что значение является строкой из букв. Отсеивает данные, содержащие лишние символы;
3. pandas.reset\_index переупорядочивает данные, чтобы при последовательном взятии данных из класса не было ошибок.

Также в классе для итерации по данным реализован метод getitem. При вызове метода из текущего элемента извлекаются название файла и текст, соответствующий этому файлу. Далее идет предобработка файла по пути, получаемому с помощью директории хранения файлов и названия файла.

Предобработка заключается в получение амплитудных отсчетов из аудио файла с помощью библиотеки librosa, извлечения мел-спектрограммы из них и преобразование ее в ДБ шкалу. Весь алгоритм был описан ранее в главе 2.2 (Смотреть рисунки 5 и 6).

После получения мел-спектрограммы еще необходимо получить цифровое представление транскрибированной строки (далее будем называть целевой строки), то есть перевести символы в целочисленный вид для обучения модели. Для этого используется функция text\_to\_int, которая проверяет текущий символ и ищет его номер в алфавите распознавания (Таблица 1). При совпадении записывает это значение в список. Как весь список сформирован, он преобразуется в тензор для обработки нейронной моделью.

Таблица 1. Алфавит распознавания

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № в алфавите | Значение | № в алфавите | Значение | № в алфавите | Значение |
| 1 | “\_” | 15 | М | 29 | Ъ |
| 2 | А | 16 | Н | 30 | Ы |
| 3 | Б | 17 | О | 31 | Ь |
| 4 | В | 18 | П | 32 | Э |
| 5 | Г | 19 | Р | 33 | Ю |
| 6 | Д | 20 | С | 34 | Я |
| 7 | Е | 21 | Т | 35 | Пробел |
| 8 | Ё | 22 | У |  |  |
| 9 | Ж | 23 | Ф |  |  |
| 10 | З | 24 | Ч |  |  |
| 11 | И | 25 | Ц |  |  |
| 12 | Й | 26 | Ч |  |  |
| 13 | К | 27 | Ш |  |  |
| 14 | Л | 28 | Щ |  |  |

В конце функции getitem возвращаются мел-спектрограмма, тензор из чисел целевой строки и целевая строка (для оценки WER).

Сформированный обучающий набор разделяется на 90% обучающей и 10% валидационной выборки и передается в класс torch.DataLoader для загрузки пакетами данных в модель при обучении. Пакет данных представляет собой часть обучающего набора данных, при этом размер пакета ограничивается объемом доступной видеопамяти при использовании GPU. При обучении в данной работе размер обучающего пакета равнялся 32, то есть 32 аудио за один проход могло обрабатываться при обучении.

Для работы пакетами необходимо данные преобразовать в пакетную форму. Для этого используется функция collate\_fn\_asr\_wer, в которой входным параметром является ранее указанное число данных из обучающего набора (в данной работе 32). Входные данные фильтруются по следующему принципу, если что-либо отсутствует из мел-спектрограммы, тензора численных значений или целевой строки, то эти данные не проходят дальше.

При формировании пакета необходимо все данные привести к одинаковой длине, но перед этим для вычисления CTC нужны реальные длины этих данных, поэтому необходимо запомнить реальную длину временных отсчетов и реальную длину тензоров целочисленных значений целевой строки. Также требованием CTC является получение конкатенированного тензора целевого результата, поэтому необходимо из всех входных данных совместить целевые данные. Выходом функции collate\_fn\_asr\_wer являются дополненные до одной длины мел-спектрограммы, конкатенированный целочисленный целевой результат, исходные длины спектрограмм, исходные длины для конкатенированного целочисленного целевого результата, целевые результаты в виде текста. Алгоритм представлен на рисунке 10.

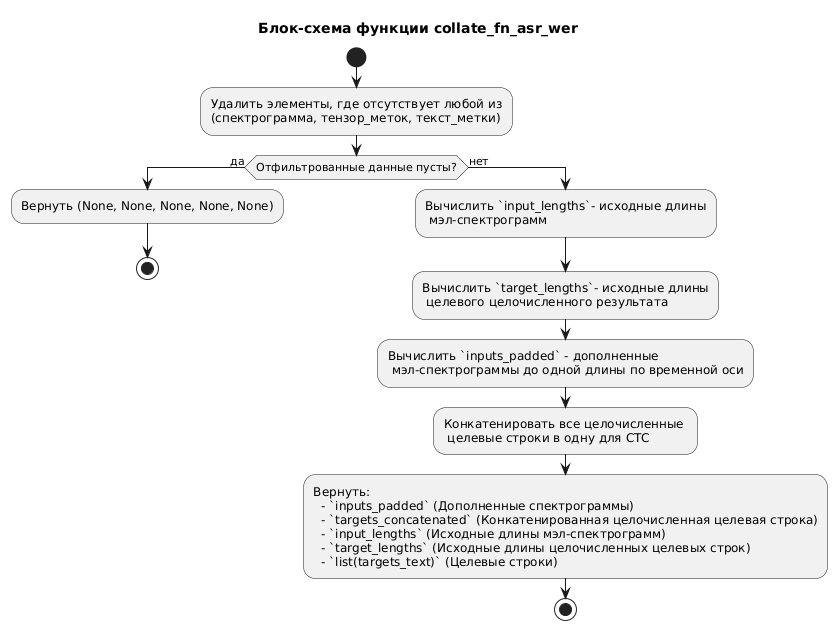


Рисунок 10 — Диаграмма формирования пакета данных.

Цикл обучения проходит по эпохам в каждой эпохе идет обучение пока все пакеты данных не будут обработаны. В начале каждой эпохи для статистики обнуляем значения общей ошибки CTC и числа обработанных данных. Далее распаковывается пакет и данные внутри него (Мел-спектрограммы одинаковой длины, целевая конкатенированная строка, длины исходных мел-спектрограмм, длины исходных целевых строк) передаются на устройство для обработки, которым является либо CPU, либо GPU (В зависимости от аппаратного обеспечения).

В модель для прямого прохода передаются только дополненные до одинаковой длины мел-спектрограммы. Прямой проход представляет собой применение последовательно всех слоев из 2.3.1. Выход прямого прохода модели представляет логарифмы вероятностей для каждого символа алфавита распознавания. Следующим шагом идет проверка, что выход модели по числу временных отсчетов больше либо равен целевой строке, при несоблюдении этого расчет CTC не будет корректно работать, так как это обязательное условие.

Производится расчет длины каждого элемента из пакета данных после сверток по формулам из документации pytorch [1010] из раздела shape:

,

где – входная последовательность отсчетов;

– выходная последовательность отсчетов после сверток;

Padding [1] – дополнение по краям мел-спектрограмм (В работе 1x1);

Dilation [1] – расширение ядра свертки путем вставки пустот между элементами ядра (по умолчанию 1);

Kernel\_size [1] – ядро (В работе 3x3);

Stride [1] – смещение ядра (В работе 2x2).

Тогда итоговая формула расчета длины каждого элемента пакета после сверток по текущим параметрам модели следующая:

После получения длин сравниваем, что все они больше либо равны целевым строкам. Если все не совпадают, то обрабатываем следующий пакет данных. Если не совпадает часть, то отфильтровываем эти элементы и оставляем только те, что удовлетворяют условию. При этом при фильтрации необходимо создать заново конкатенированный тензор целевых строк и оставить только те значения в логарифмах вероятностей, выходных длинах строк модели и целевых длинах строк, которые прошли фильтр. Алгоритм представлен на рисунке 11.

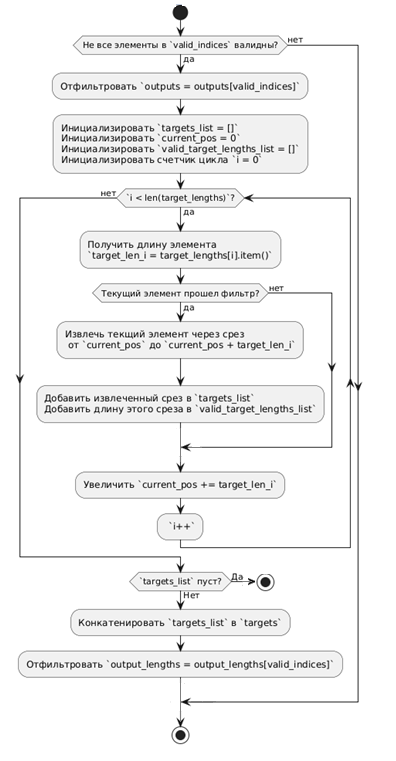


Рисунок 11 — Диаграмма фильтрации данных.

В конце применяем функцию CTC, в которую передаются выход модели (вероятности), целевая конкатенированная строка, истинные выходные длины модели, исходные входные длины целевых строк. После расчета CTC веса обновляются обратным проходом и выводится статистика по эпохе. После каждой эпохи результат обучения сохраняется в файл весов модели, который после можно либо дообучить, либо использовать при распознавании речи в реальном времени. Алгоритм обучения представлен на рисунке 12.

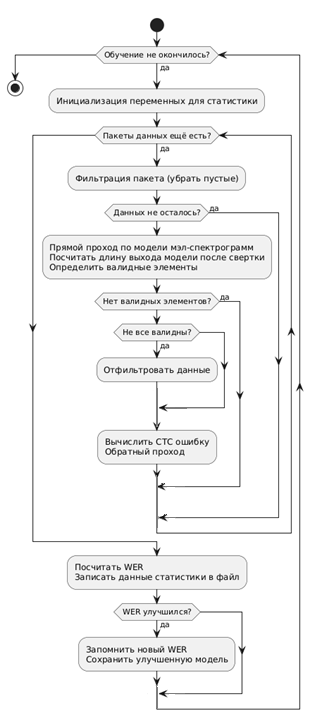


Рисунок 12 — Диаграмма обучения нейронной модели.

Вычисление WER происходит по полученным данным и целевым, при этом полученные данные должны быть переведены в форму текста с помощью функции int\_to\_text, алгоритм жадного декодирования которой представлен на рисунке 3.

Визуальный пример вывода из последовательности токенов текста рисунок 13. В упрощенном виде выполняется два действия по порядку:

1. Удалить все повторы токенов между “\_”;
2. Удалить все специальные символы “\_”.

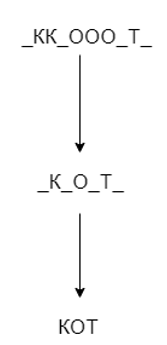


Рисунок 13 — Пример вывода текста из токенов при удалении повторов и специального токена.

## Разработка пользовательского интерфейса

Для взаимодействия пользователя с системой автоматического распознавания речи реализован графический интерфейс с использованием стандартной библиотеки Python – tkinter. Tkinter — кроссплатформенная событийно-ориентированная графическая библиотека на основе средств Tk (широко распространённая в мире GNU/Linux и других UNIX‐подобных систем, портирована также и на Microsoft Windows). При инициализации создается главное окно приложения (Рисунок 14) с заголовком "Автоматическая транскрибация речи". В конструктор передаются предварительно загруженная модель (из раздела 2.3).

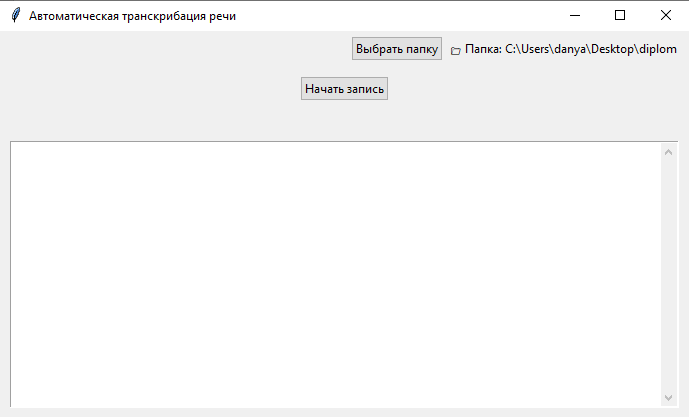


Рисунок 14 — Графический интерфейс приложения.

Визуальные элементы включают верхнюю панель с кнопкой выбора папки сохранения ("Выбрать папку") и текстом, отображающим выбранный путь. Ниже расположены основная кнопка управления "Начать запись", которая при нажатии меняет свою надпись на “Остановить запись” и отображает текстовый индикатор состояния записи “Идет запись…” (Рисунок 15).

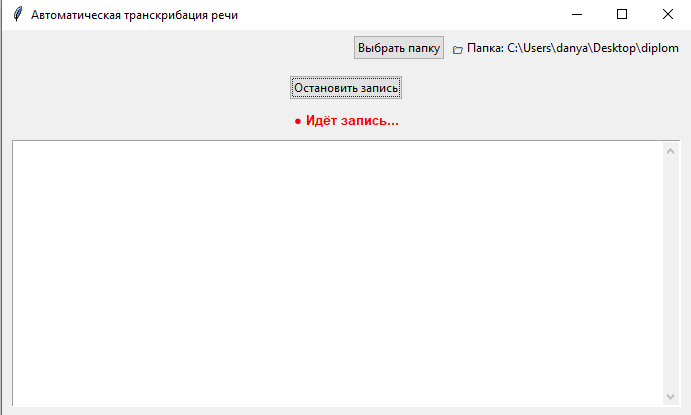


Рисунок 15 — Отображение изменения кнопки и индикатора записи.

Основную часть окна занимает многострочное текстовое поле с вертикальной полосой прокрутки для вывода результатов транскрибации, по умолчанию оно защищено от редактирования (см. Рисунок 16).

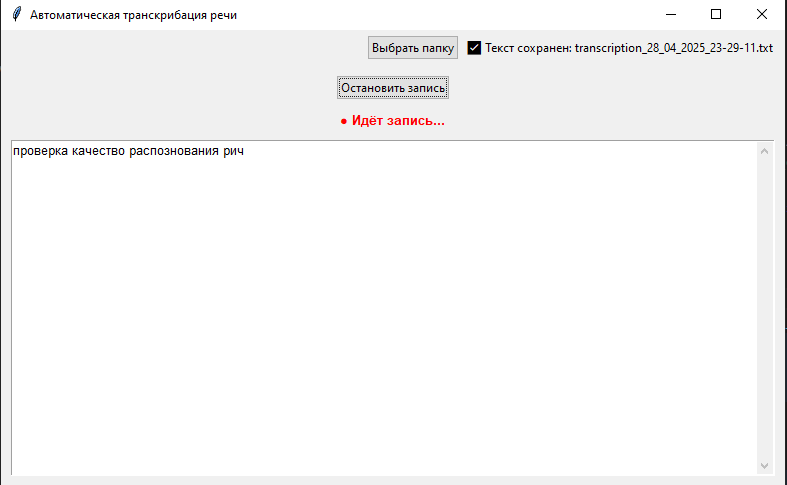


Рисунок 16 — Отображение результатов распознавания речи в текстовом поле.

Сохранение транскрибированного аудио выполняется методом save\_transcribation, который получает весь текст из текстового поля виджета, генерирует имя файла на основе текущей даты/времени и сохраняет текст в выбранную пользователем директорию. Пользователь выбирает эту директорию с помощью метода choose\_directory. Корректное завершение работы приложения при закрытии окна обеспечивается методом on\_closing, который останавливает все активные процессы и потоки перед уничтожением окна.

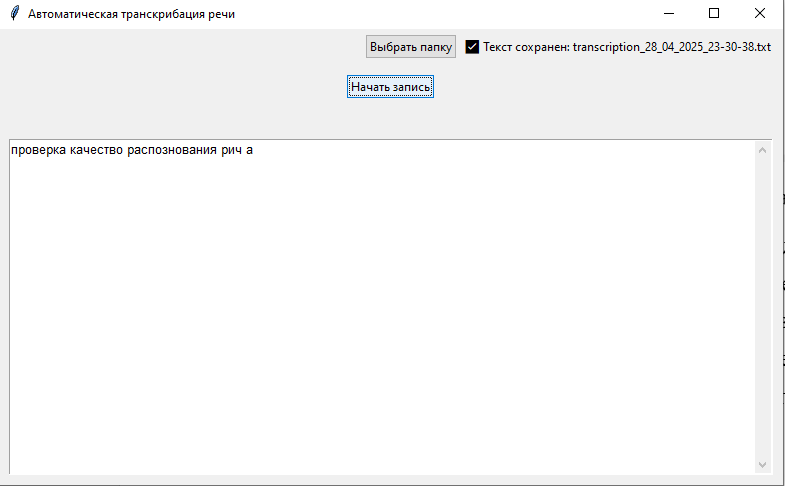


Рисунок 17 —В правом верхнем углу отображен путь сохранения результата.

Для обмена данными между потоками используются потокобезопасные очереди: audio\_queue для передачи аудиоданных из потока записи в поток обработки и transcribation\_queue для передачи распознанного текста из потока обработки в главный поток GUI. Состояние записи контролируется флагом is\_recording, а ссылки на потоки записи и обработки хранятся в recording\_thread и processing\_thread соответственно.

Управление записью осуществляется методами toggle\_recording, start\_recording и stop\_recording. Метод start\_recording инициализирует процесс записи: устанавливает флаг is\_recording, очищает буферы и текстовое поле, обновляет виджеты, запускает поток обработки очереди (processing\_thread) и поток записи (recording\_thread). Поток записи входит в цикл ожидания, в котором sounddevice непрерывно вызывает audio\_callback, возвращающий амплитудные отсчеты звукового сигнала. Метод stop\_recording сбрасывает флаг is\_recording, дожидается завершения потоков записи и обработки (поместив маркер конца в audio\_queue и дождавшись его обработки), обрабатывает последние данные из буфера, останавливает обновление GUI и вызывает сохранение транскрибированного аудио в файл.

Обновление графического интерфейса происходит в главном потоке. Метод check\_transcribation\_queue вызывается каждые 100 мс с помощью root.after для проверки и обновления графического интерфейса. Он извлекает все доступные текстовые фрагменты из transcribation\_queue и передает их методу update\_transcribation\_widget, который обновляет текстовое поле, добавляя текст и при полном заполнении экрана прокручивая окно вывода текста вниз.

# Тестирование программного обеспечения

## Проведение тестирования программы

Тестирование разработанной системы автоматического распознавания речи проводилось по нескольким направлениям, включающим проверку функциональности отдельных модулей и системы в целом.

### Тестирование модуля предобработки звукового сигнала

Предобработка звукового сигнала является важным этапом в системах распознавания речи, анализа аудио и других приложениях обработки звука. От качества этой предобработки напрямую зависит эффективность работы всей системы, поскольку именно на этом этапе формируются признаки, используемые для последующего анализа.

Для модуля предобработки звукового сигнала было проведено тестирование следующих функциональных возможностей:

1. Проверка считывания аудиосигнала различными устройствами записи звука;
2. Корректность сохранения амплитудных отсчетов во внутреннем буфере.

Тестирование формирования самой мел-спектрограммы не производится, так как эта функциональность была протестирована самими разработчиками библиотеки librosa [11]. Поэтому главный аспект — это корректная подача амплитудных отсчетов в функцию формирования мел-спектрограммы.

Для тестирования первого пункта проведем следующие действия: при первом запуске приложения, если отсутствует записывающее устройство, то всплывает информационное сообщение, уведомляющее пользователя, что запись невозможна (см. Рисунок 18). При попытке начать запись появится новое уведомление с ошибкой о том, что запись невозможна при отсутствии записывающего устройства (см. Рисунок 19).

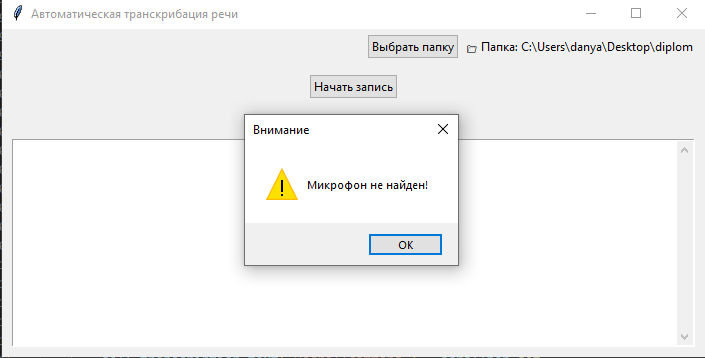


Рисунок 18 — Отображение информационного сообщения об отсутствии микрофона при запуске приложения.

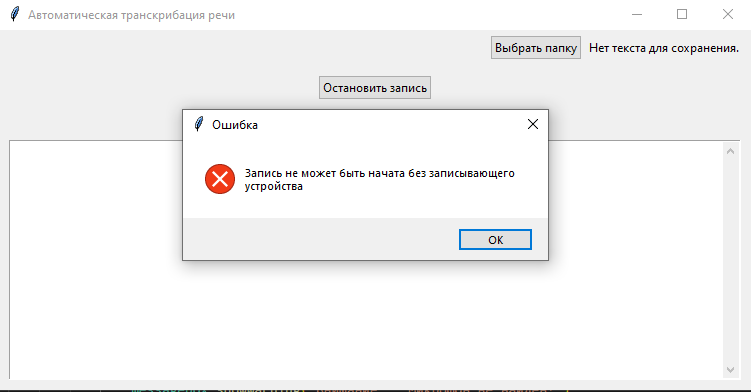


Рисунок 19 — Отображение информационного сообщения об отсутствии микрофона при попытке начать запись.

При наличии записывающий устройств при первом запуске будет информация, что микрофон есть (см. Рисунок 20), а при начале записи никаких уведомлений не будет, так как все нормально.

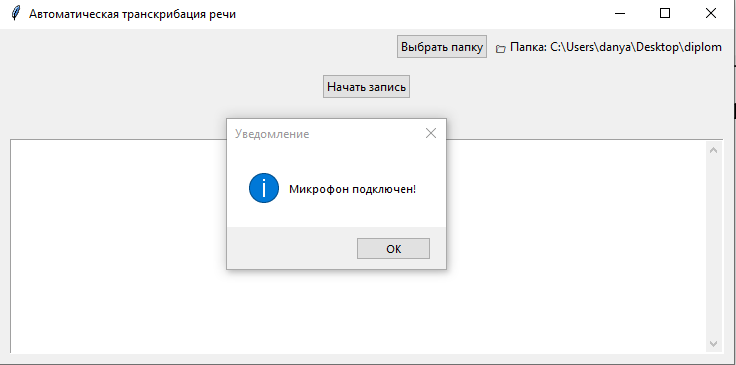


Рисунок 20 — Отображение информационного сообщения о наличии микрофона при первом запуске приложения.

Для проведения второго теста производится запись в реальном времени, которая с помощью механизма обратного вызова возвращает по 1024 амплитудных отсчета. При накоплении достаточного числа отчетов (количество секунд \* частоту дискретизации) из буфера берется это число, а остаток остается в буфере. Ниже приведена эхо печать формата отсчетов во внутреннем буфере (см. Рисунок 21) и эхо печать изменения размера буфера (см. Рисунок 22).

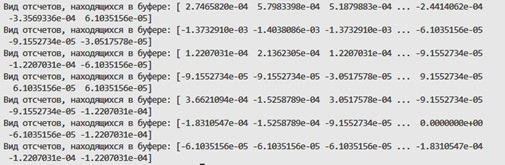


Рисунок 21 — Отображение формата отсчетов во внутреннем буфере.

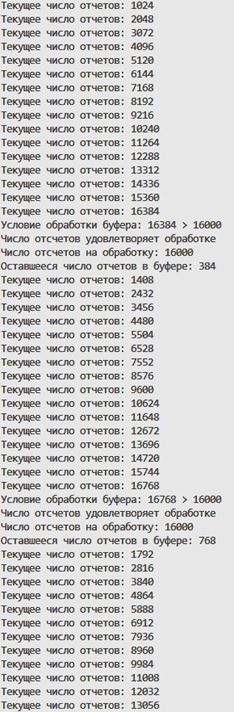


Рисунок 22 — Отображение изменения размера внутреннего буфера с отсчетами.

### Тестирования модуля графического интерфейса

Тестирование графического интерфейса системы было направлено на проверку корректности работы всех элементов управления и отображения на платформах Windows и Linux. Для этого было проведено тестирование следующих функциональных возможностей:

1. Проверка корректности работы кнопок управления записью ("Начать запись" / "Остановить запись") и изменения состояния в соответствии с нажатыми кнопками;
2. Тестирование функциональности выбора директории для сохранения результатов;
3. Тестирование функциональности прокрутки текстового поля при выводе результатов распознавания;

Тестирование пункта 1. При нажатии кнопки управления “Начать запись” она меняет свое состояние и текст изменяется на “Остановить запись”. Также появляется индикатор записи “Идет запись…” (см. Рисунок 23 и 24).

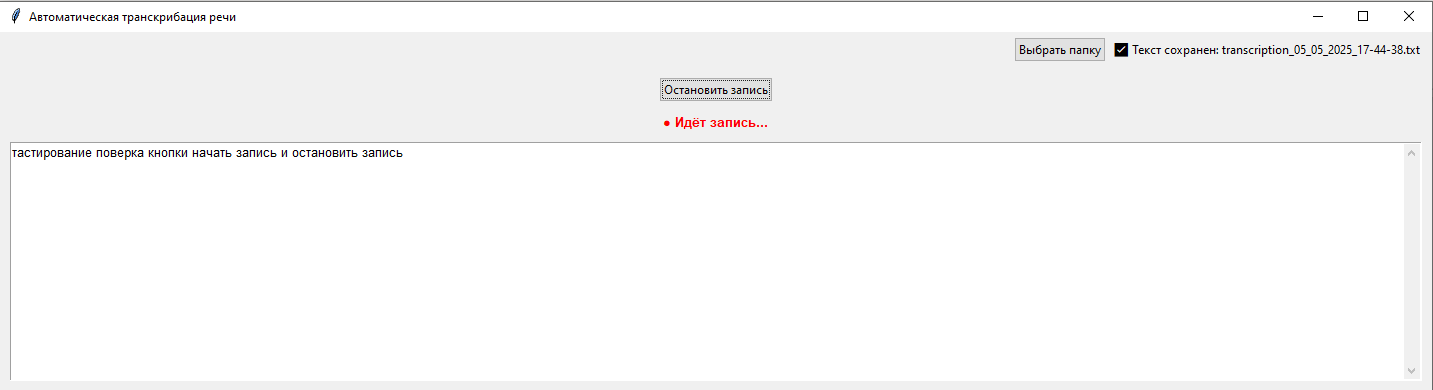


Рисунок 23 — Изменение состояния кнопки при начале записи Windows.

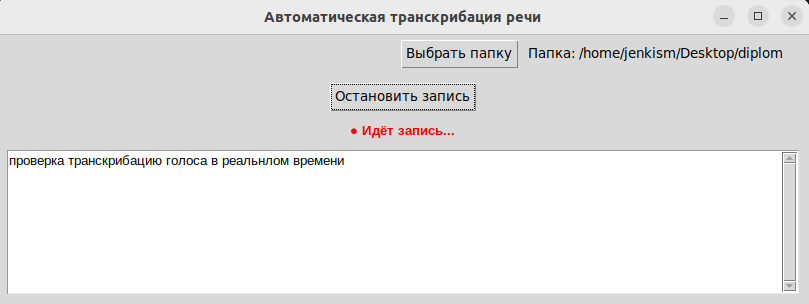


Рисунок 24 — Изменение состояния кнопки при начале записи Linux.

При нажатии “Остановить запись” она меняет свое состояние и текст изменяется на “Начать запись”. Также исчезает индикатор записи (см. Рисунок 25 и 26).

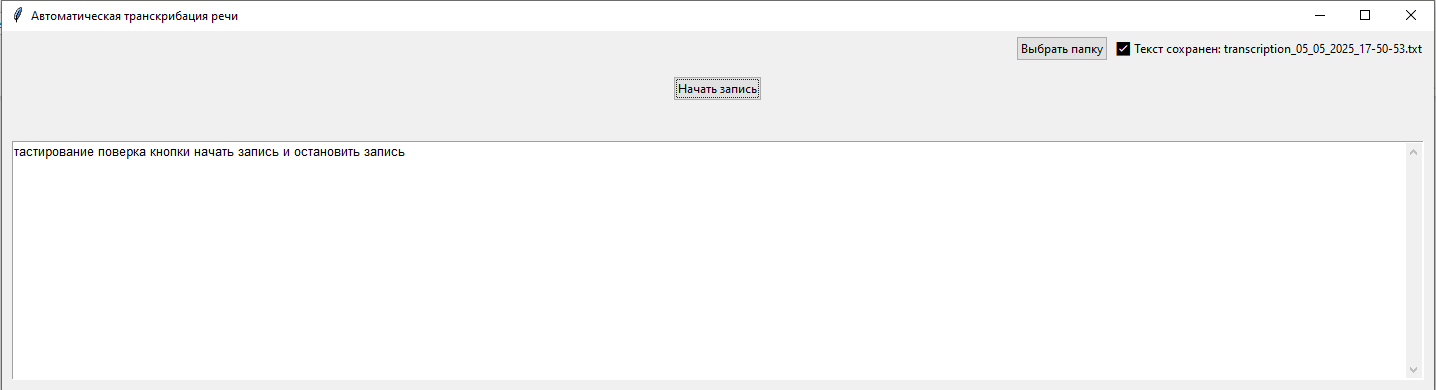


Рисунок 25 — Изменение состояния кнопки при конце записи Windows.

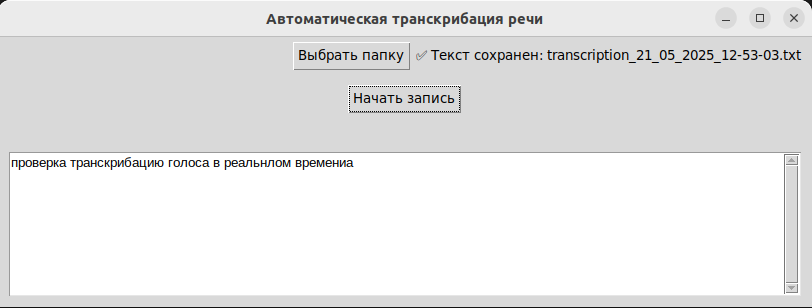


Рисунок 26 — Изменение состояния кнопки при конце записи Linux.

При повторном нажатии кнопки “Начать запись” поле отображения текста очищается, чтобы нельзя было спутать новые данные со старыми (см. Рисунок 27 и 28):

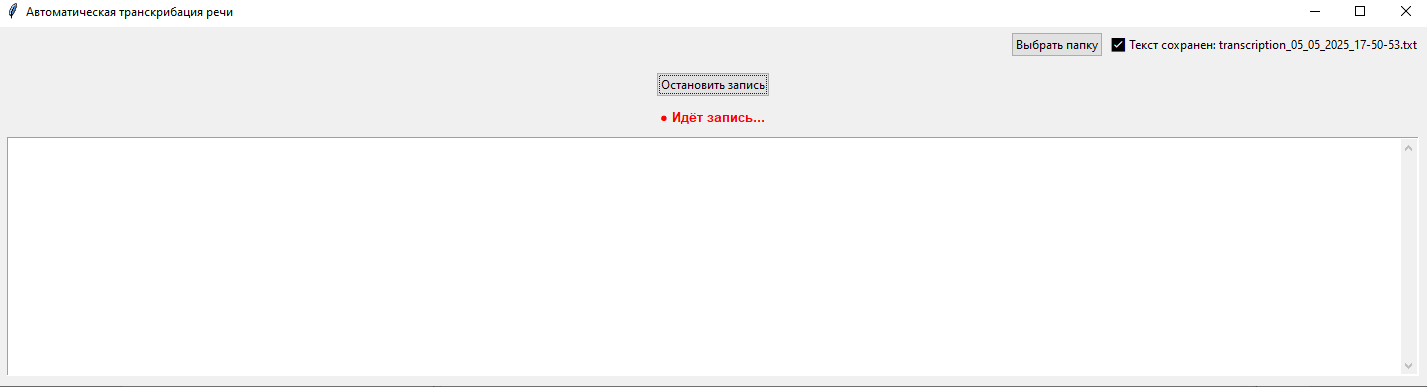


Рисунок 27 — Очищение поля отображения текста Windows.

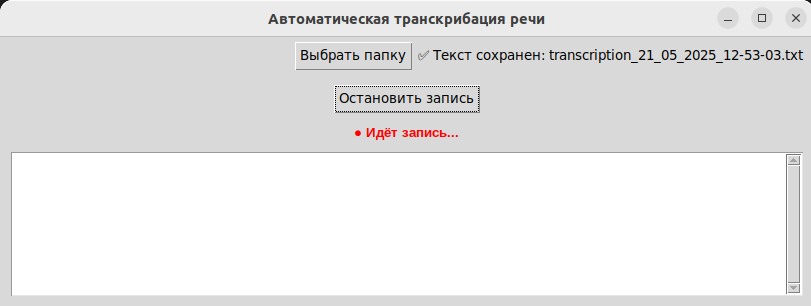


Рисунок 28 — Очищение поля отображения текста Linux.

Тестирование пункта 2. При нажатии кнопки “Выбрать папку”, появляется диалоговое окно выбора (см. Рисунок 29 и 30).

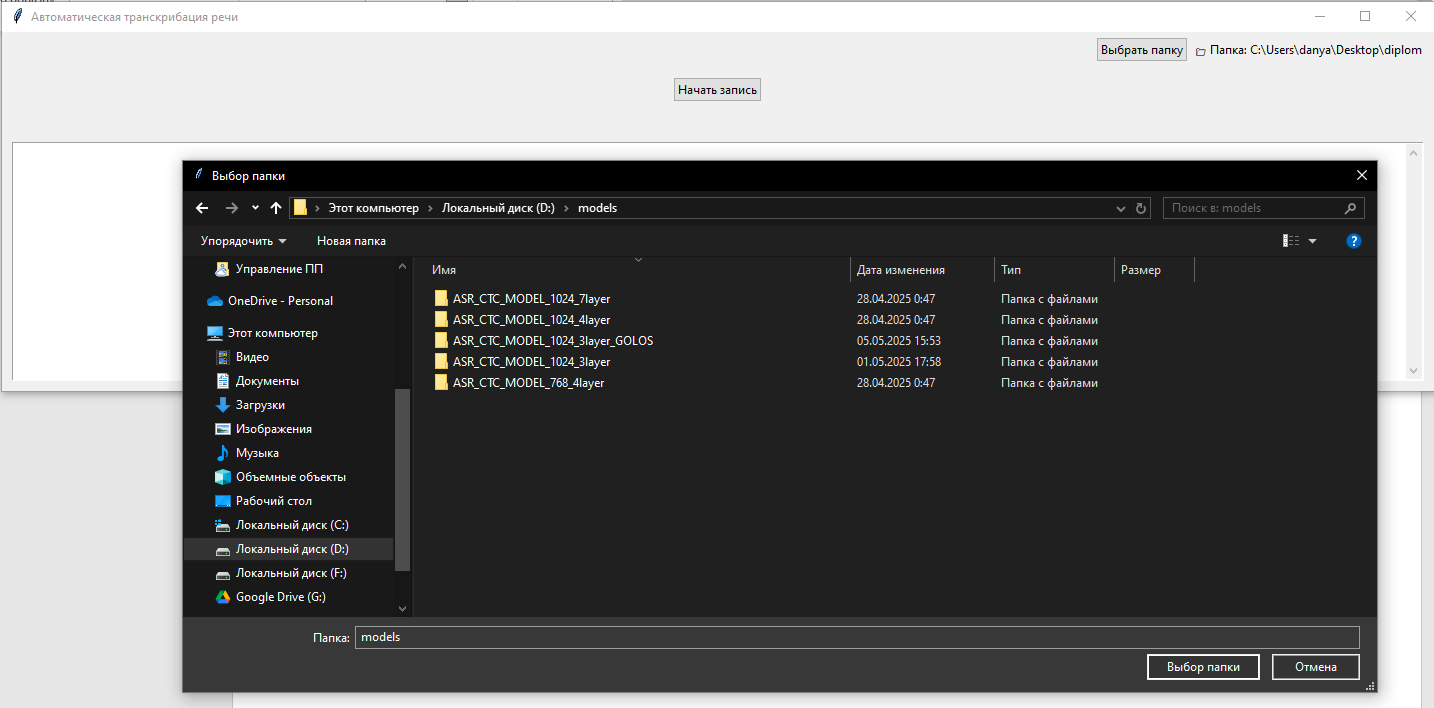


Рисунок 29 — Окно выбора директории для сохранения результата распознавания Windows.

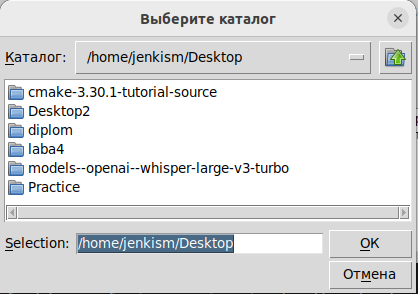


Рисунок 30 — Окно выбора директории для сохранения результата распознавания Linux.

При выборе директории окно закрывается и в правом верхнем углу отображается директория, в которую сохраняется результат распознавания (см. Рисунок 31 и 32).

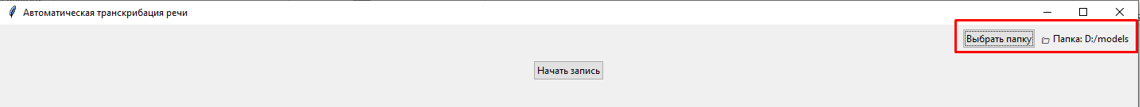


Рисунок 31 — Отображение измененной директории Windows.

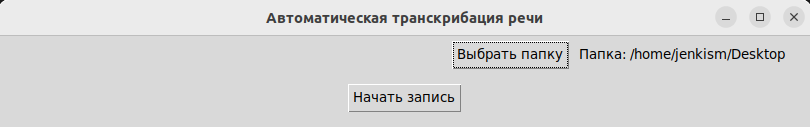


Рисунок 32 — Отображение измененной директории Linux.

Тестирование пункта 3. Огромный текст полностью не помещается в поле вывода результата распознавания. В таких случаях становится активна клавиша прокрутки текста (см. Рисунки 33, 34, 35, 36).

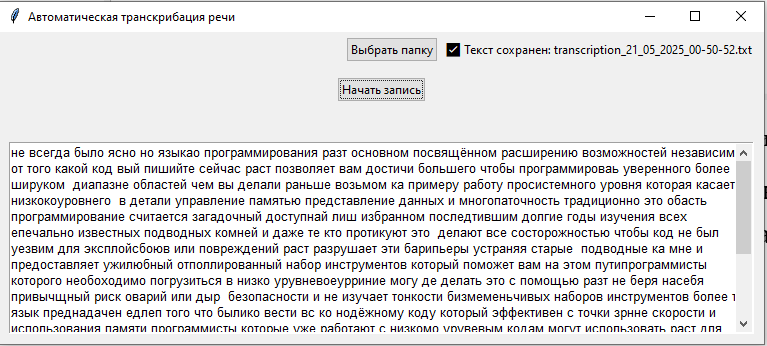


Рисунок 33 — Отображение начального текста в окне результата на Windows.

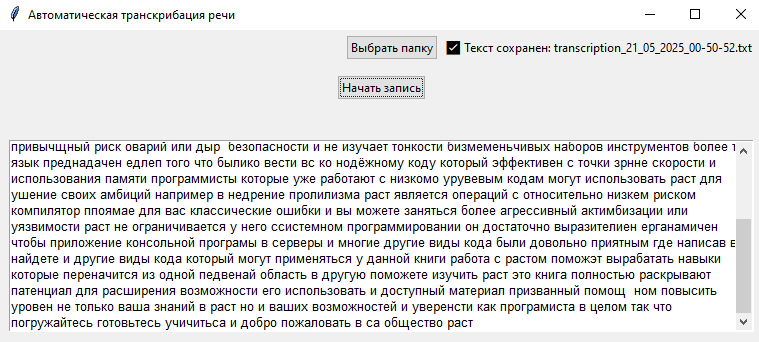


Рисунок 34 — Отображение после прокрутки текста в окне результата на Windows.

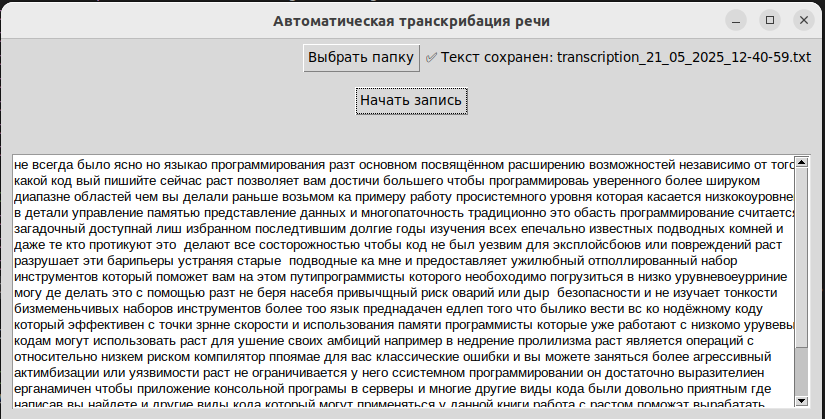


Рисунок 35 — Отображение начального текста в окне результата на Linux.

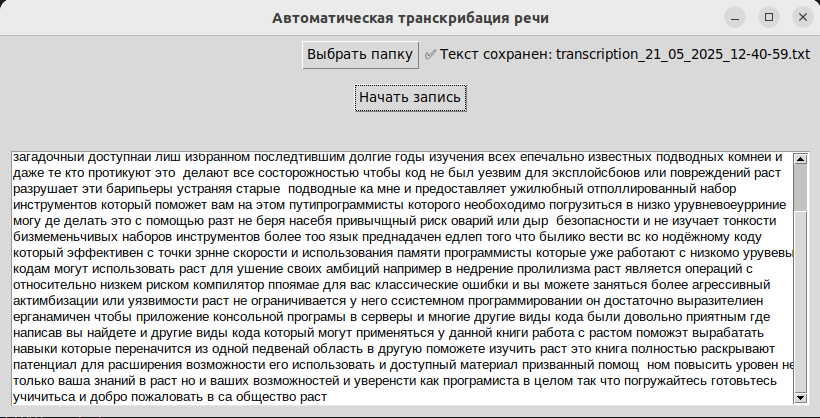


Рисунок 36 — Отображение после прокрутки текста в окне результата на Linux.

## Анализ результатов тестирования

### Результаты тестирования модуля предобработки звукового сигнала

Тестирование модуля предобработки звукового сигнала показало корректность работы следующих функциональных возможностей:

1. Система правильно определяет наличие или отсутствие записывающих устройств и выдает соответствующие информационные сообщения пользователю;
2. При наличии записывающих устройств система корректно осуществляет запись, и внутренний буфер системы правильно накапливает и обрабатывает амплитудные отсчеты звукового сигнала.

### Результаты тестирования модуля графического интерфейса

Тестирование графического интерфейса системы было направлено на проверку корректности работы всех элементов управления и отображения. Были получены следующие результаты:

1. Все проверенные аспекты функциональности кнопок управления записью работают корректно, что обеспечивает понятное и предсказуемое взаимодействие пользователя с системой;
2. Функциональность выбора директории работает без сбоев, обеспечивая пользователю полный контроль над местом сохранения результатов работы системы;
3. Тестирование подтвердило, что текстовое поле для вывода результатов распознавания обеспечивает корректное отображение и навигацию по тексту любого объема, что является важным аспектом удобства использования системы.

### Общие результаты тестирования

Тестирование разработанной системы автоматического распознавания речи не выявило сбоев в исполнении программы. Цель функционального тестирования достигнута. Работоспособность подтверждена, поэтому можно сделать вывод, что весь заявленный функционал системы автоматического распознавания речи реализован и готов к использованию.

##### ЗАКЛЮЧЕНИЕ

В настоящей выпускной квалификационной работе были проведены и описаны этапы проектирования, разработки и тестирования системы автоматического распознавания русской речи. Работа охватывала ключевые аспекты создания системы, начиная от выбора архитектурных решений и заканчивая реализацией пользовательского интерфейса.

Были выбраны средства разработки, соответствующие современным подходам к созданию систем на основе машинного обучения и обеспечивающие кроссплатформенность (Windows, Linux). В качестве языка программирования использовался Python с привлечением специализированных библиотек для обработки аудио и построения нейронных сетей. По результатам анализа предметной области и существующих подходов к распознаванию речи была спроектирована модульная архитектура разрабатываемого программного обеспечения, включающая модуль предобработки звука, модуль распознавания речи на основе нейронной сети и модуль пользовательского интерфейса.

Центральной частью работы стала разработка модуля распознавания речи: спроектирована и реализована рекуррентно-сверточная нейронная модель на базе слоев Conv2D и LSTM с использованием механизма CTC для обучения и декодирования последовательностей. Описан процесс обучения модели на наборе русскоязычных аудиоданных и выбор гиперпараметров. Также был разработан пользовательский интерфейс на базе библиотеки tkinter, обеспечивающий взаимодействие пользователя с системой.

Проведены эксперименты по обучению нейронной модели, и проанализированы результаты оценки качества распознавания по метрике WER. Хотя достигнутые значения WER указывают на необходимость дальнейшего совершенствования модели и процесса обучения, продемонстрирована работоспособность предложенной архитектуры. Были рассмотрены различные аспекты, влияющие на качество распознавания, такие как архитектура модели, параметры сверточных и LSTM слоев, функции активации и методы нормализации.

Разработанная система автоматического распознавания речи готова к дальнейшему развитию. Для усовершенствования программного обеспечения могут быть реализованы следующие идеи:

* Дальнейшая оптимизация гиперпараметров нейронной сети и процесса обучения для снижения WER, а также расширение обучающего набора данных для формирования модели способной распознавать более широкий спектр особенностей человеческой речи;
* При наличии более продвинутого аппаратного обеспечения появляется возможность организации более современных архитектур нейронных сетей на основе Transformer или Conformer;
* Интеграция языковой модели для улучшения качества распознавания посредством удаления орфографических ошибок и добавления пунктуации;
* Усовершенствование пользовательского интерфейса для большего удобства пользования и добавления новых функций (редактирование текста, работа с аудиофайлами).

##### Список использованных источников

1. Brendan Shillingford**,** Yannis Assael, Matthew W. Hoffman, Thomas Paine, Cían Hughes, Utsav Prabhu, Hank Liao, Hasim Sak, Kanishka Rao Large-scale visual speech recognition. URL: <https://arxiv.org/pdf/1807.05162> (Дата обращения: 13.02.2025);
2. Nikolay Karpov, Alexander Denisenko, Fedor Minkin Golos: Russian Dataset for Speech Research. Лицензия: Public license with attribution and conditions reserved. URL: GitHub, <https://github.com/sberdevices/golos> (Дата обращения: 03.03.2025);
3. Anna Slizhikova, Alexander Veysov, Diliara Nurtdinova, Dmitry Voronin Russian Open Speech To Text (STT/ASR) Dataset. Лицензия: CC-BY-NC. URL: GitHub, <https://github.com/snakers4/open_stt> (Дата обращения: 03.03.2025);
4. Muhammad A. Shah, Bhiksha Raj Revisiting Acoustic Features for Robust ASR. URL: <https://arxiv.org/pdf/2409.16399> (Дата обращения: 16.03.2025);
5. Boyang Zhang, Jared Leitner, Sam Thornton Audio Recognition using Mel Spectrograms and Convolution Neural Networks. URL: <https://noiselab.ucsd.edu/ECE228_2019/Reports/Report38.pdf?utm_source=chatgpt.com> (Дата обращения: 29.03.2025);
6. Aqeel Anwar Difference between AlexNet, VGGNet, ResNet, and Inception. URL: <https://medium.com/data-science/the-w3h-of-alexnet-vggnet-resnet-and-inception-7baaaecccc96> (Дата обращения: 30.03.2025);
7. Phone recognition with deep sparse rectifier neural networks. URL: <https://www.inf.u-szeged.hu/~tothl/pubs/ICASSP2013.pdf> (Дата обращения: 02.04.2025);
8. Andrew L. Maas, Awni Y. Hannun, Andrew Y. Ng Rectifier Nonlinearities Improve Neural Network Acoustic Models. URL: <https://ai.stanford.edu/~amaas/papers/relu_hybrid_icml2013_final.pdf> (Дата обращения: 06.04.2025);
9. Sergey Ioffe, Christian Szegedy Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. URL: <https://arxiv.org/abs/1502.03167> (Дата обращения: 10.04.2025);
10. Документация PyTorch об слое Conv2d. URL: <https://docs.pytorch.org/docs/stable/generated/torch.nn.Conv2d.html> (Дата обращения: 10.04.2025);
11. A python package for music and audio analysis, compatibility tests for librosa's core features. URL: GitHub, <https://github.com/librosa/librosa/tree/main/tests> (Дата обращения: 20.04.2025).

##### Приложение А. Код системы

## Приложение А.1. CTC\_train.py

import os

import pandas as pd

import torch

import torch.nn as nn

import torch.optim as optim

from torch.utils.data import Dataset, DataLoader, Subset, ConcatDataset, random\_split

from torch.nn.utils.rnn import pad\_sequence

import librosa

import numpy as np

import time

from tqdm import tqdm

import jiwer

# Добавлены импорты для TensorBoard и Аугментации

from torch.utils.tensorboard import SummaryWriter

import torchaudio.transforms as T

# Русский алфавит + пробел + пустой символ CTC

RUSSIAN\_ALPHABET = "\_абвгдеёжзийклмнопрстуфхцчшщъыьэюя "

BLANK\_CHAR = '\_'

char\_map = {char: idx for idx, char in enumerate(RUSSIAN\_ALPHABET)}

index\_map = {idx: char for char, idx in char\_map.items()}

# Функции кодирования/декодирования текста (без изменений)

def text\_to\_int(text, char\_map):

"""Преобразует текст в индексы по `char\_map`."""

text = text.lower()

# Пропускаем символы, которых нет в карте, чтобы избежать ошибок

return [char\_map[char] for char in text if char in char\_map]

def int\_to\_text(indices, index\_map, blank\_char=BLANK\_CHAR):

"""Преобразует индексы обратно в текст, убирая CTC-пустые символы и повторы."""

text = ""

blank\_idx = char\_map.get(blank\_char, -1)

last\_idx = -1

for idx in indices:

if idx == last\_idx: continue

if idx == blank\_idx:

last\_idx = idx

continue

# Проверяем наличие индекса в карте перед добавлением

if idx in index\_map:

text += index\_map[idx]

last\_idx = idx

text = ' '.join(text.split())

return text

# Улучшенная обработка аудио (возвращает F, T)

def preprocess\_audio(audio\_path, sample\_rate=16000, n\_mels=80, n\_fft=400, hop\_length=160):

"""Загружает аудио, преобразует в лог-мел-спектрограмму и нормализует."""

try:

audio, orig\_sr = librosa.load(audio\_path, sr=sample\_rate, mono=True)

except Exception as e:

print(f"Ошибка загрузки файла {audio\_path}: {e}")

return None

# Дополнение слишком коротких аудио, которые могут вызвать ошибки в Conv слоях

if len(audio) < n\_fft: # Если аудио короче одного окна FFT

padding\_needed = n\_fft - len(audio)

# Дополняем нулями (тишиной) в конец

audio = np.pad(audio, (0, padding\_needed), mode='constant', constant\_values=0)

# Вычисляем мел-спектрограмму

try:

mel\_spectrogram = librosa.feature.melspectrogram(y=audio, sr=sample\_rate, n\_fft=n\_fft, hop\_length=hop\_length, n\_mels=n\_mels, fmax=8000)

except Exception as e:

print(f"Ошибка при вычислении melspectrogram для {audio\_path} (даже после паддинга): {e}")

return None # Проверка на нулевую энергию (тишину), которая может дать -inf после power\_to\_db

if np.max(mel\_spectrogram) == 0:

# print(f"Предупреждение: Нулевая энергия в файле {audio\_path}, пропускаем.")

return None

log\_mel\_spectrogram = librosa.power\_to\_db(mel\_spectrogram, ref=np.max)

# Нормализация

mean = np.mean(log\_mel\_spectrogram)

std = np.std(log\_mel\_spectrogram)

# Проверка на нулевое стандартное отклонение (монотонный сигнал)

if std < 1e-6:

# print(f"Предупреждение: Нулевое стд. отклонение в файле {audio\_path}, пропускаем.")

return None

log\_mel\_spectrogram = (log\_mel\_spectrogram - mean) / (std + 1e-6)

log\_mel\_spectrogram\_tensor = torch.tensor(log\_mel\_spectrogram, dtype=torch.float32) # (F, T)

return log\_mel\_spectrogram\_tensor

# ASR\_CTC\_Model

class ASR\_CTC\_Model(nn.Module):

def \_\_init\_\_(self, input\_dim, hidden\_dim, output\_dim, num\_layers=3, dropout=0.15):

super(ASR\_CTC\_Model, self).\_\_init\_\_()

# Слои Conv2d ожидают вход (B, C, H, W) или (B, C, F, T)

self.conv\_layers = nn.Sequential(

# Вход: (B, 1, F=input\_dim, T)

nn.Conv2d(1, 32, kernel\_size=(3, 3), stride=(2, 2), padding=(1, 1)), # -> (B, 32, F/2, T/2)

nn.ReLU(),

nn.BatchNorm2d(32),

nn.Conv2d(32, 32, kernel\_size=(3, 3), stride=(2, 2), padding=(1, 1)), # -> (B, 32, F/4, T/4)

nn.ReLU(),

nn.BatchNorm2d(32)

)

# Расчет входной размерности для LSTM

lstm\_input\_dim = 32 \* (input\_dim // 4) # Каналы \* (F уменьшается в 2 раза дважды (из-за stride[0]=2))

# Dropout слой

self.conv\_dropout = nn.Dropout(dropout)

self.lstm = nn.LSTM(

lstm\_input\_dim,

hidden\_dim,

num\_layers=num\_layers,

batch\_first=True,

bidirectional=True,

dropout=dropout if num\_layers > 1 else 0 # Dropout между слоями LSTM

)

self.fc\_dropout = nn.Dropout(dropout)

self.fc = nn.Linear(hidden\_dim \* 2, output\_dim) # Bidirectional -> \*2

def forward(self, x):

# x: (batch, F, T) - входной формат от DataLoader

x = x.unsqueeze(1) # -> (B, 1, F, T) - добавляем канал для Conv2d

x = self.conv\_layers(x) # -> (B, 32, F/4, T/4)

# Изменяем размерность для LSTM: (batch, T', features)

batch\_size, channels, F\_prime, T\_prime = x.shape

x = x.permute(0, 3, 1, 2) # -> (B, T', C, F') -> T'=T/4, F'=F/4

x = x.reshape(batch\_size, T\_prime, channels \* F\_prime) # -> (B, T', 32\*F/4)

# Dropout после сверток

x = self.conv\_dropout(x)

x, \_ = self.lstm(x) # -> (B, T', H\*2)

# Dropout перед FC

x = self.fc\_dropout(x)

x = self.fc(x) # -> (B, T', Output)

# Применяем LogSoftmax для CTC Loss

# CTC ожидает (T, N, C), где N=batch\_size, T=длина посл., C=классы

x = nn.functional.log\_softmax(x, dim=2) # -> (B, T', Output)

return x

def get\_output\_lengths(self, input\_lengths\_T): # Принимает длины по оси T

""" Рассчитывает длину ВЫХОДА по оси T после сверток """

lengths = input\_lengths\_T

# Уменьшение по оси T определяется stride[1] для Conv2d при входе (B, C, F, T)

# Первый Conv: stride=(2, 2) -> stride[1]=2

if isinstance(self.conv\_layers[0], nn.Conv2d) and self.conv\_layers[0].stride[1] > 1:

lengths = torch.div(lengths - 1, self.conv\_layers[0].stride[1], rounding\_mode='floor') + 1

# Второй Conv: stride=(2, 2) -> stride[1]=2

if isinstance(self.conv\_layers[3], nn.Conv2d) and self.conv\_layers[3].stride[1] > 1:

lengths = torch.div(lengths - 1, self.conv\_layers[3].stride[1], rounding\_mode='floor') + 1

# Итого T\_out = T\_in / 4

return lengths

class SingleSourceAudioDataset(Dataset):

def \_\_init\_\_(self, annotation\_file, audio\_dir, char\_map, apply\_augmentation=False, freq\_mask\_param=27, time\_mask\_param=70):

""" Работает с ОДНИМ файлом аннотации и ОДНОЙ аудио директорией. """

self.audio\_dir = audio\_dir

self.char\_map = char\_map

self.apply\_augmentation = apply\_augmentation

self.spec\_augment = None

try:

self.audio\_target = pd.read\_csv(annotation\_file, on\_bad\_lines="skip")

self.audio\_target.dropna(subset=['text'], inplace=True)

self.audio\_target = self.audio\_target[self.audio\_target['text'].apply(lambda x: isinstance(x, str))]

self.audio\_target = self.audio\_target[self.audio\_target['text'].str.strip() != '']

self.audio\_target.reset\_index(drop=True, inplace=True) # Важно для Subset/ConcatDataset

except FileNotFoundError:

print(f"Ошибка: Файл аннотации не найден: {annotation\_file}")

self.audio\_target = pd.DataFrame(columns=['filename', 'text']) # Пустой

except Exception as e:

print(f"Ошибка при чтении {annotation\_file}: {e}")

self.audio\_target = pd.DataFrame(columns=['filename', 'text'])

if self.apply\_augmentation and len(self.audio\_target) > 0:

self.spec\_augment = nn.Sequential(T.FrequencyMasking(freq\_mask\_param=freq\_mask\_param),T.TimeMasking(time\_mask\_param=time\_mask\_param)).eval()

def \_\_len\_\_(self):

return len(self.audio\_target)

def \_\_getitem\_\_(self, idx):

if idx >= len(self.audio\_target):

raise IndexError(f"Индекс {idx} за пределами датасета (размер {len(self.audio\_target)})")

try:

row = self.audio\_target.iloc[idx]

audio\_filename\_no\_ext = row['filename']

label\_text = row['text']

if audio\_filename\_no\_ext.endswith(".mp3") or audio\_filename\_no\_ext.endswith(".wav"):

dir\_path = os.path.join(self.audio\_dir, str(audio\_filename\_no\_ext))

else:

dir\_path = os.path.join(self.audio\_dir, str(audio\_filename\_no\_ext) + ".opus")

if not os.path.exists(dir\_path):

return None, None, None

log\_mel\_spectrogram = preprocess\_audio(dir\_path)

if log\_mel\_spectrogram is None:

return None, None, None

if self.apply\_augmentation and self.spec\_augment is not None:

log\_mel\_spectrogram = self.spec\_augment(log\_mel\_spectrogram)

if not isinstance(label\_text, str):

return None, None, None

label\_int = text\_to\_int(label\_text, self.char\_map)

label\_tensor = torch.tensor(label\_int, dtype=torch.long)

return log\_mel\_spectrogram, label\_tensor, label\_text.lower()

except Exception as e:

print(f"Ошибка в \_\_getitem\_\_ (idx={idx}): {e}"); return None, None, None

# collate\_fn для обработки формата (F, T) и 5 элементов

def collate\_fn\_asr\_wer(batch):

# Фильтруем None (кортежи из 3-х элементов)

batch = [(spec, target\_tensor, target\_text)

for spec, target\_tensor, target\_text in batch

if spec is not None and target\_tensor is not None and target\_text is not None]

if not batch:

return None, None, None, None, None

inputs\_feat\_time, targets\_tensor, targets\_text = zip(\*batch) # inputs в формате (F, T)

# Длины по ВРЕМЕННОЙ оси (T) - вторая размерность

input\_lengths = torch.tensor([x.shape[1] for x in inputs\_feat\_time], dtype=torch.long)

target\_lengths = torch.tensor([len(t) for t in targets\_tensor], dtype=torch.long)

# Паддинг входов (спектрограмм) по временной оси T

try:

inputs\_permuted = [x.permute(1, 0) for x in inputs\_feat\_time] # список (T, F)

inputs\_padded\_permuted = pad\_sequence(inputs\_permuted, batch\_first=True, padding\_value=0.0) # (B, T\_padded, F)

inputs\_padded = inputs\_padded\_permuted.permute(0, 2, 1) # (B, F, T\_padded)

except Exception as e:

print(f"Ошибка паддинга в collate\_fn: {e}")

# Попытка вывести размеры для отладки

for i, spec in enumerate(inputs\_feat\_time):

print(f" Элемент {i} форма: {spec.shape}")

return None, None, None, None, None # Пропускаем батч при ошибке

targets\_concatenated = torch.cat(targets\_tensor)

return inputs\_padded, targets\_concatenated, input\_lengths, target\_lengths, list(targets\_text)

# Расчет WER

@torch.no\_grad()

def evaluate\_wer(model: nn.Module,

dataloader: DataLoader,

device: torch.device,

index\_map: dict) -> tuple[float, list, list]:

model.eval()

all\_predictions = []

all\_references = []

print(f"\nНачало оценки WER на устройстве {device}...")

for batch\_data in tqdm(dataloader, desc="WER Evaluation"):

if batch\_data is None or batch\_data[0] is None: continue

try:

inputs, \_, input\_lengths\_T, \_, ref\_texts = batch\_data # Длины по оси T

except ValueError as e:

print(f"Ошибка распаковки батча в evaluate\_wer: {e}")

print(f"Получено элементов: {len(batch\_data) if isinstance(batch\_data, (list, tuple)) else 'Не список/кортеж'}")

continue

inputs = inputs.to(device)

input\_lengths\_T\_dev = input\_lengths\_T.to(device)

try:

outputs = model(inputs) # (B, T\_out, C)

output\_lengths = model.get\_output\_lengths(input\_lengths\_T\_dev) # (B)

except Exception as e:

print(f"Ошибка model forward/get\_output\_lengths в evaluate\_wer: {e}")

# Попытка вывести размеры для отладки

print(f" Форма inputs: {inputs.shape}, Длины входа T: {input\_lengths\_T\_dev}")

continue

preds\_indices = torch.argmax(outputs, dim=2) # (B, T\_out)

preds\_indices\_cpu = preds\_indices.cpu().tolist()

output\_lengths\_cpu = output\_lengths.cpu().tolist()

for i in range(len(inputs)):

actual\_len = output\_lengths\_cpu[i]

if actual\_len <= 0 :

pred\_text = ""

else:

pred\_text = int\_to\_text(preds\_indices\_cpu[i][:actual\_len], index\_map)

ref\_text = ref\_texts[i]

all\_predictions.append(pred\_text)

all\_references.append(ref\_text)

calculated\_wer = float('inf')

if all\_references and all\_predictions:

if len(all\_references) != len(all\_predictions):

print(f"Предупреждение: Разная длина референсов ({len(all\_references)}) и предсказаний ({len(all\_predictions)})!")

else:

print("Расчет WER...")

try:

calculated\_wer = jiwer.wer(all\_references, all\_predictions) \* 100

print(f"Ошибка WER = {calculated\_wer:.2f}%")

except Exception as e:

print(f"Ошибка при расчете WER с помощью jiwer: {e}")

else:

print("Предупреждение: Не удалось собрать данные для расчета WER.")

# model.train() # Убрали, пусть вызывающий код решает, когда переключать режим

return calculated\_wer, all\_references, all\_predictions

if \_\_name\_\_ == "\_\_main\_\_":

# Гиперпараметры

INPUT\_DIM = 80 # n\_mels число уровней мэл

HIDDEN\_DIM = 1024 # Число скрытых признаков LSTM

OUTPUT\_DIM = len(RUSSIAN\_ALPHABET)

NUM\_LAYERS = 3 # Число слоев LSTM

DROPOUT = 0.3 # Обрубание весов

BATCH\_SIZE = 32 # Число обрабатываемых аудио за проход

NUM\_EPOCHS = 1 # Число эпох обучения

LEARNING\_RATE = 4e-4 # Adam с большими моделями лучше сходится с меньшим LR

WEIGHT\_DECAY = 1e-4 # Небольшая L2 регуляризация

CLIP\_GRAD\_NORM = 5.0 # Для предотвращения взрыва градиентов

MODEL\_PATH = "/kaggle/input/ep-10-all-datasets/pytorch/default/1/all\_data\_1024\_3layer\_epoch10\_WER28.70.pth" # Путь для дообучения модели

TRAIN\_SPLIT\_RATIO = 0.9 # 90% на обучение, 10% на валидацию

PATIENCE\_SCHEDULER = 3 # для ReduceLROnPlateau

PATIENCE\_EARLY\_STOPPING = 10 # Число эпох без улучшения для выхода

FREQ\_MASK\_PARAM = 27 # Аугментация по частоте

TIME\_MASK\_PARAM = 70 # Аугментация по времени

data\_sources = [

("/kaggle/input/russian-asr-open-stt-public-youtube-700-hours/dataset\_target.csv", "/kaggle/input/russian-asr-open-stt-public-youtube-700-hours/0"),

("/kaggle/input/russian-asr-librispeech/manifest.csv", "/kaggle/input/russian-asr-librispeech/0"),

("/kaggle/input/russian-asr-common-voice-corpus-21-0/manifest.csv", "/kaggle/input/russian-asr-common-voice-corpus-21-0/0"),

("/kaggle/input/russian-asr-golos/golos/golos/dataset\_target.csv", "/kaggle/input/russian-asr-golos/golos/golos/0"),

("/kaggle/input/russian-asr-sova-ai-rudevices/manifest.csv", "/kaggle/input/russian-asr-sova-ai-rudevices/0"),

("/kaggle/input/audiosets/dataset\_target.csv","/kaggle/input/audiosets/asr\_public\_phone\_calls\_1")

]

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

print(f"Используется устройство: {device}")

# Инициализация TensorBoard

run\_name = f"asr\_lr{LEARNING\_RATE}\_wd{WEIGHT\_DECAY}\_dr{DROPOUT}\_aug"

log\_dir = os.path.join("runs", run\_name)

writer = SummaryWriter(log\_dir)

print(f"Логи TensorBoard: {log\_dir}")

# Загрузка и разделение данных

print("Создание датасетов...")

all\_datasets\_train\_version = [

SingleSourceAudioDataset(csv, adir, char\_map, apply\_augmentation=False)

for csv, adir in data\_sources

]

all\_datasets\_val\_version = [

SingleSourceAudioDataset(csv, adir, char\_map, apply\_augmentation=False)

for csv, adir in data\_sources

]

# Фильтруем пустые Datasets

all\_datasets\_train\_version = [ds for ds in all\_datasets\_train\_version if len(ds) > 0]

all\_datasets\_val\_version = [ds for ds in all\_datasets\_val\_version if len(ds) > 0]

if not all\_datasets\_train\_version:

print("Ошибка: Нет валидных данных!")

exit()

combined\_dataset\_train\_ver = ConcatDataset(all\_datasets\_train\_version)

combined\_dataset\_val\_ver = ConcatDataset(all\_datasets\_val\_version)

dataset\_size = len(combined\_dataset\_train\_ver)

print(f"Общий размер объединенного датасета: {dataset\_size}")

train\_size = int(TRAIN\_SPLIT\_RATIO \* dataset\_size)

val\_size = dataset\_size - train\_size

train\_indices, val\_indices = random\_split(range(dataset\_size), [train\_size, val\_size])

train\_subset = Subset(combined\_dataset\_train\_ver, train\_indices) # Train subset с аугментацией

val\_subset = Subset(combined\_dataset\_val\_ver, val\_indices) # Val subset БЕЗ аугментации

print(f"Размер обучающей выборки: {len(train\_subset)}")

print(f"Размер валидационной выборки: {len(val\_subset)}")

# DataLoaders

train\_dataloader = DataLoader(train\_subset, batch\_size=BATCH\_SIZE, shuffle=True, collate\_fn=collate\_fn\_asr\_wer, num\_workers=4, pin\_memory=True if device.type == 'cuda' else False, persistent\_workers=True if torch.cuda.is\_available() and 4 > 0 else False)

val\_dataloader = DataLoader(val\_subset, batch\_size=BATCH\_SIZE, shuffle=False, collate\_fn=collate\_fn\_asr\_wer, num\_workers=4, pin\_memory=True if device.type == 'cuda' else False, persistent\_workers=True if torch.cuda.is\_available() and 4 > 0 else False)

# Инициализация модели

model = ASR\_CTC\_Model(INPUT\_DIM, HIDDEN\_DIM, OUTPUT\_DIM, NUM\_LAYERS, DROPOUT)

model.to(device)

# Загрузка весов, если нужно продолжить обучение

if os.path.exists(MODEL\_PATH):

print(f"Загрузка весов из {MODEL\_PATH}")

model.load\_state\_dict(torch.load(MODEL\_PATH, map\_location=device))

print(model)

total\_params = sum(p.numel() for p in model.parameters() if p.requires\_grad)

print(f"Общее количество обучаемых параметров: {total\_params:,}")

# Loss, Optimizer, Scheduler

ctc\_loss = nn.CTCLoss(blank=char\_map[BLANK\_CHAR], reduction='mean', zero\_infinity=True).to(device)

optimizer = optim.AdamW(model.parameters(), lr=LEARNING\_RATE, weight\_decay=WEIGHT\_DECAY)

scheduler = optim.lr\_scheduler.ReduceLROnPlateau(optimizer, mode='min', factor=0.5, patience=PATIENCE\_SCHEDULER, verbose=True) # mode='min' для WER

# Цикл обучения

best\_val\_wer = float('inf')

epochs\_no\_improve = 0

start\_epoch = 0 # Для продолжения обучения

print(f"\n Начало обучения ({NUM\_EPOCHS} эпох) ")

for epoch in range(start\_epoch, NUM\_EPOCHS):

start\_time\_epoch = time.time()

# Обучение

model.train()

train\_loss\_accum = 0.0

processed\_samples\_train = 0

for batch\_idx, batch\_data in enumerate(tqdm(train\_dataloader, desc=f"Epoch {epoch+1}/{NUM\_EPOCHS} Training")):

if batch\_data is None or batch\_data[0] is None: continue

try:

inputs, targets\_concat, input\_lengths\_T, target\_lengths, \_ = batch\_data

except ValueError as e:

print(f"\nОшибка распаковки трен. батча {batch\_idx+1}: {e}")

continue

# Проверка на пустые тензоры после фильтрации в collate

if inputs.numel() == 0 or targets\_concat.numel() == 0:

# print(f"\nПропущен пустой батч {batch\_idx+1} после collate.")

continue

inputs = inputs.to(device, non\_blocking=True)

targets = targets\_concat.to(device, non\_blocking=True)

input\_lengths = input\_lengths\_T.to(device, non\_blocking=True)

target\_lengths = target\_lengths.to(device, non\_blocking=True)

optimizer.zero\_grad()

try:

outputs = model(inputs) # (B, T\_out, C)

output\_lengths = model.get\_output\_lengths(input\_lengths).to(device) # (B)

# Проверка валидности длин

valid\_indices = output\_lengths >= target\_lengths

valid\_indices &= (output\_lengths > 0) & (target\_lengths > 0)

if not valid\_indices.any(): # Если нет валидных в батче

# print(f"\nПропущен батч {batch\_idx+1}: нет валидных длин.")

continue

# Фильтруем батч для лосса, если не все валидны

if not valid\_indices.all():

# print(f"\nФильтрация батча {batch\_idx+1} для лосса...")

outputs = outputs[valid\_indices]

targets\_list = []

current\_pos = 0

valid\_target\_lengths\_list = []

target\_lengths\_cpu = target\_lengths.cpu() # Для итерации

targets\_cpu = targets.cpu() # Для срезов

for i in range(len(target\_lengths\_cpu)):

target\_len\_i = target\_lengths\_cpu[i].item()

if valid\_indices[i]:

targets\_list.append(targets\_cpu[current\_pos : current\_pos + target\_len\_i])

valid\_target\_lengths\_list.append(target\_len\_i)

current\_pos += target\_len\_i

targets = torch.cat(targets\_list).to(device, non\_blocking=True)

output\_lengths = output\_lengths[valid\_indices]

target\_lengths = torch.tensor(valid\_target\_lengths\_list, dtype=torch.long, device=device)

# Если после фильтрации ничего не осталось

if targets.numel() == 0:

continue

log\_probs = outputs.permute(1, 0, 2) # (T\_out, B\_filtered, C)

loss = ctc\_loss(log\_probs, targets, output\_lengths, target\_lengths)

if torch.isnan(loss) or torch.isinf(loss):

print(f"\nNaN/Inf loss на батче {batch\_idx+1}, пропуск.")

continue

except Exception as e:

print(f"\nОшибка forward/loss на трен. батче {batch\_idx+1}: {e}")

continue

# Backward и Step

try:

loss.backward()

torch.nn.utils.clip\_grad\_norm\_(model.parameters(), CLIP\_GRAD\_NORM)

optimizer.step()

except Exception as e:

print(f"\nОшибка backward/step на батче {batch\_idx+1}: {e}")

optimizer.zero\_grad() # Сбрасываем градиенты при ошибке

continue

# Успешный шаг

batch\_size\_processed = outputs.size(0) # Реальный размер батча после фильтрации

train\_loss\_accum += loss.item() \* batch\_size\_processed

processed\_samples\_train += batch\_size\_processed

# Логирование батча

if (batch\_idx + 1) % 100 == 0: # Логируем реже

global\_step = epoch \* len(train\_dataloader) + batch\_idx

writer.add\_scalar('Loss/train\_batch', loss.item(), global\_step)

# print(f"DEBUG: Logged train\_batch loss at step {global\_step}") # Отладочный вывод

avg\_train\_loss = train\_loss\_accum / processed\_samples\_train if processed\_samples\_train > 0 else float('inf')

# Валидация

# Переключаем модель в eval режим ВНУТРИ evaluate\_wer

epoch\_wer, val\_refs, val\_preds = evaluate\_wer(model, val\_dataloader, device, index\_map)

# evaluate\_wer возвращает модель в режим train в конце

epoch\_duration = time.time() - start\_time\_epoch

# Логирование эпохи

writer.add\_scalar('Loss/train\_epoch', avg\_train\_loss, epoch + 1) # Логируем по номеру эпохи (начиная с 1)

writer.add\_scalar('WER/validation', epoch\_wer, epoch + 1)

writer.add\_scalar('LearningRate', optimizer.param\_groups[0]['lr'], epoch + 1)

writer.add\_scalar('EpochDuration\_sec', epoch\_duration, epoch + 1)

# Вывод результатов эпохи

print(f"\nEpoch {epoch+1}/{NUM\_EPOCHS} Summary:")

print(f" Duration: {epoch\_duration:.2f}s")

print(f" Avg Train Loss: {avg\_train\_loss:.4f}")

print(f" Validation WER: {epoch\_wer:.2f}%")

print(f" Current LR: {optimizer.param\_groups[0]['lr']:.3e}")

print(" Example Predictions (Predicted | Target):")

for i in range(min(5, len(val\_preds))):

print(f" - '{val\_preds[i]}' | '{val\_refs[i]}'")

# Обновление планировщика, сохранение лучшей модели, ранняя остановка

scheduler.step(epoch\_wer)

if epoch\_wer < best\_val\_wer:

cur\_epoch = epoch + 11

best\_val\_wer = epoch\_wer

torch.save(model.state\_dict(), f"/kaggle/working/all\_data\_1024\_3layer\_epoch{cur\_epoch}\_WER{epoch\_wer:.2f}.pth")

print(f" Validation WER улучшился до {best\_val\_wer:.2f}%. Модель сохранена.")

epochs\_no\_improve = 0

else:

epochs\_no\_improve += 1

torch.save(model.state\_dict(), f"/kaggle/working/all\_data\_1024\_3layer\_epoch{cur\_epoch}\_WER{epoch\_wer:.2f}.pth")

print(f" Validation WER не улучшился. Эпох без улучшения: {epochs\_no\_improve}/{PATIENCE\_EARLY\_STOPPING}")

if epochs\_no\_improve >= PATIENCE\_EARLY\_STOPPING:

print(f"\nРанняя остановка на эпохе {epoch+1}! Validation WER не улучшался {PATIENCE\_EARLY\_STOPPING} эпох.")

break

print("-" \* 50)

writer.close()

print(f"Обучение завершено. Лучший Validation WER: {best\_val\_wer:.2f}% сохранен")

## Приложение A.2. app.py

import tkinter as tk

from tkinter import ttk, filedialog, messagebox

import sounddevice as sd

import numpy as np

import threading

import os

import datetime

import queue

import time

# Импорты для модели и обработки

import torch

import torch.nn as nn

import librosa

from transformers import WhisperFeatureExtractor, WhisperTokenizer, WhisperProcessor, WhisperForConditionalGeneration

# КОНСТАНТЫ И ФУНКЦИИ МОДЕЛИ

RUSSIAN\_ALPHABET = "\_абвгдеёжзийклмнопрстуфхцчшщъыьэюя "

BLANK\_CHAR = '\_'

char\_map = {char: idx for idx, char in enumerate(RUSSIAN\_ALPHABET)}

index\_map = {idx: char for char, idx in char\_map.items()}

def text\_to\_int(text, char\_map):

text = text.lower()

return [char\_map[char] for char in text if char in char\_map]

def int\_to\_text(indices, index\_map, blank\_char=BLANK\_CHAR):

"""Преобразует индексы обратно в текст, убирая CTC-пустые символы и повторы."""

text = ""

blank\_idx = char\_map.get(blank\_char, -1)

last\_idx = -1

for idx in indices:

if idx == last\_idx:

continue

if idx == blank\_idx:

last\_idx = idx

continue

# Проверяем наличие индекса в карте перед добавлением

if idx in index\_map:

text += index\_map[idx]

last\_idx = idx

text = ' '.join(text.split())

return text

# Функция предобработки для получения лог-мел-спетрограммы

def preprocess\_buffer(audio\_chunk: np.ndarray, sample\_rate=16000, n\_mels=80, n\_fft=400, hop\_length=160):

"""Преобразует NumPy массив аудио в лог-мел-спектрограмму (F, T)."""

# Убедимся, что данные в формате float32, как ожидает librosa

if audio\_chunk.dtype != np.float32:

audio\_chunk = audio\_chunk.astype(np.float32)

# Дополнение тишиной на всякий случай (хотя из буфера должна приходить нужная длина)

if len(audio\_chunk) < n\_fft:

padding\_needed = n\_fft - len(audio\_chunk)

audio\_chunk = np.pad(audio\_chunk, (0, padding\_needed), mode='constant', constant\_values=0)

try:

mel\_spectrogram = librosa.feature.melspectrogram(y=audio\_chunk, sr=sample\_rate, n\_fft=n\_fft, hop\_length=hop\_length, n\_mels=n\_mels, fmax=8000)

except Exception as e:

print(f"Ошибка при вычислении melspectrogram из буфера: {e}")

return None

if np.max(mel\_spectrogram) < 1e-10: return None # Пропускаем тишину

log\_mel\_spectrogram = librosa.power\_to\_db(mel\_spectrogram, ref=np.max)

mean = np.mean(log\_mel\_spectrogram)

std = np.std(log\_mel\_spectrogram)

if std < 1e-6:

return None

log\_mel\_spectrogram = (log\_mel\_spectrogram - mean) / (std + 1e-6)

return torch.tensor(log\_mel\_spectrogram, dtype=torch.float32) # (F, T)

# ASR\_CTC\_Model

class ASR\_CTC\_Model(nn.Module):

def \_\_init\_\_(self, input\_dim, hidden\_dim, output\_dim, num\_layers=3, dropout=0.15):

super(ASR\_CTC\_Model, self).\_\_init\_\_()

# Слои Conv2d ожидают вход (B, C, H, W) или (B, C, F, T)

self.conv\_layers = nn.Sequential(

# Вход: (B, 1, F=input\_dim, T)

nn.Conv2d(1, 32, kernel\_size=(3, 3), stride=(2, 2), padding=(1, 1)), # -> (B, 32, F/2, T/2)

nn.ReLU(),

nn.BatchNorm2d(32),

nn.Conv2d(32, 32, kernel\_size=(3, 3), stride=(2, 2), padding=(1, 1)), # -> (B, 32, F/4, T/4)

nn.ReLU(),

nn.BatchNorm2d(32)

)

# Расчет входной размерности для LSTM

lstm\_input\_dim = 32 \* (input\_dim // 4) # Каналы \* (F уменьшается в 2 раза дважды (из-за stride[0]=2))

# Dropout слой

self.conv\_dropout = nn.Dropout(dropout)

self.lstm = nn.LSTM(

lstm\_input\_dim,

hidden\_dim,

num\_layers=num\_layers,

batch\_first=True,

bidirectional=True,

dropout=dropout if num\_layers > 1 else 0 # Dropout между слоями LSTM

)

self.fc\_dropout = nn.Dropout(dropout)

self.fc = nn.Linear(hidden\_dim \* 2, output\_dim) # Bidirectional -> \*2

def forward(self, x):

# x: (batch, F, T) - входной формат от DataLoader

x = x.unsqueeze(1) # -> (B, 1, F, T) - добавляем канал для Conv2d

x = self.conv\_layers(x) # -> (B, 32, F/4, T/4)

# Изменяем размерность для LSTM: (batch, T', features)

batch\_size, channels, F\_prime, T\_prime = x.shape

x = x.permute(0, 3, 1, 2) # -> (B, T', C, F') -> T'=T/4, F'=F/4

x = x.reshape(batch\_size, T\_prime, channels \* F\_prime) # -> (B, T', 32\*F/4)

# Dropout после сверток

x = self.conv\_dropout(x)

x, \_ = self.lstm(x) # -> (B, T', H\*2)

# Dropout перед FC

x = self.fc\_dropout(x)

x = self.fc(x) # -> (B, T', Output)

# Применяем LogSoftmax для CTC Loss

# CTC ожидает (T, N, C), где N=batch\_size, T=длина посл., C=классы

x = nn.functional.log\_softmax(x, dim=2) # -> (B, T', Output)

return x

def get\_output\_lengths(self, input\_lengths\_T): # Принимает длины по оси T

""" Рассчитывает длину ВЫХОДА по оси T после сверток """

lengths = input\_lengths\_T

# Уменьшение по оси T определяется stride[1] для Conv2d при входе (B, C, F, T)

# Первый Conv: stride=(2, 2) -> stride[1]=2

if isinstance(self.conv\_layers[0], nn.Conv2d) and self.conv\_layers[0].stride[1] > 1:

lengths = torch.div(lengths - 1, self.conv\_layers[0].stride[1], rounding\_mode='floor') + 1

# Второй Conv: stride=(2, 2) -> stride[1]=2

if isinstance(self.conv\_layers[3], nn.Conv2d) and self.conv\_layers[3].stride[1] > 1:

lengths = torch.div(lengths - 1, self.conv\_layers[3].stride[1], rounding\_mode='floor') + 1

# Итого T\_out = T\_in / 4

return lengths

# ПАРАМЕТРЫ МОДЕЛИ И ПУТЬ

INPUT\_DIM = 80

HIDDEN\_DIM = 1024 # Число нейронов в скрытом слое LSTM

OUTPUT\_DIM = len(RUSSIAN\_ALPHABET)

NUM\_LAYERS = 3 # Число слоев LSTM

DROPOUT = 0.3

#MODEL\_LOAD\_PATH = "D:/models/1024\_3layer\_all\_datasets/all\_data\_1024\_3layer\_epoch9\_WER29.26.pth"

MODEL\_LOAD\_PATH = "D:/models/ASR\_CTC\_MODEL\_1024\_3layer\_GOLOS, common voice, sova ai, librispeech/gslc\_asr\_ctc\_model\_1024\_3layer\_epoch20\_WER11.60.pth" # Пример

# ОПРЕДЕЛЕНИЕ УСТРОЙСТВА

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

print(f"Используется устройство для инференса: {device}")

# ЗАГРУЗКА МОДЕЛИ

print("Загрузка модели...")

model = ASR\_CTC\_Model(INPUT\_DIM, HIDDEN\_DIM, OUTPUT\_DIM, NUM\_LAYERS, DROPOUT).to(device)

try:

# map\_location=device гарантирует загрузку на правильное устройство

model.load\_state\_dict(torch.load(MODEL\_LOAD\_PATH, map\_location=device))

model.eval() # <--- ОБЯЗАТЕЛЬНО переводим модель в режим оценки

print("Модель успешно загружена и переведена в режим оценки.")

except FileNotFoundError:

print(f"Ошибка: Файл модели не найден по пути: {MODEL\_LOAD\_PATH}")

print("Транскрибация будет невозможна.")

model = None # Ставим модель в None, чтобы не было ошибок дальше

except Exception as e:

print(f"Ошибка при загрузке модели: {e}")

model = None

# Функция транскрибации чанка

@torch.no\_grad() # Отключаем градиенты для инференса

def transcribe\_chunk(audio\_chunk, model, device, index\_map):

"""Транскрибирует один чанк аудио (NumPy array)."""

if model is None: # Если модель не загрузилась

return "[Модель не загружена]"

# Предобработка буфера -> Тензор (F, T) на CPU

input\_features = preprocess\_buffer(audio\_chunk) # sample\_rate и др. параметры берутся по умолчанию

if input\_features is None:

# print("Предобработка не удалась.")

return "[Ошибка обработки]"

# Перенос на device и добавление batch-измерения -> (1, F, T)

input\_tensor = input\_features.unsqueeze(0).to(device)

# Длина по времени (T) для get\_output\_lengths

input\_length\_T = torch.tensor([input\_features.shape[1]], dtype=torch.long).to(device)

try:

# Получение выхода модели

outputs = model(input\_tensor) # (1, T\_out, C)

# Получение длины выхода

output\_lengths = model.get\_output\_lengths(input\_length\_T) # (1)

if output\_lengths[0] <= 0:

# print("Нулевая длина выхода модели.")

return "" # Пустая строка, если выход нулевой

# Обрезаем выход до реальной длины

actual\_output\_len = output\_lengths[0].item()

logits\_for\_decoder = outputs.squeeze(0)[:actual\_output\_len] # (T\_out\_actual, C)

# Greedy декодирование

pred\_indices = torch.argmax(outputs, dim=2).squeeze(0) # (T\_out)

pred\_indices\_cpu = pred\_indices[:output\_lengths[0]].cpu().tolist() # Обрезаем и переносим на CPU

decoded\_text = int\_to\_text(pred\_indices\_cpu, index\_map)

return decoded\_text

except Exception as e:

print(f"Ошибка во время инференса модели: {e}")

return "[Ошибка инференса]"

QUEUE\_END\_MARKER = object()

# Класс GUI приложения

class AudioApp:

def \_\_init\_\_(self, root, loaded\_model, feauture\_extractor=None, whisper\_model=None, tokenizer=None):

self.root = root

self.root.title("Автоматическая транскрибация речи")

self.model = loaded\_model # Модель для моей

self.whisper\_feature\_extractor = feauture\_extractor

self.whisper\_model = whisper\_model

self.whisper\_tokenizer = tokenizer

self.device = device # CPU или GPU

# Параметры записи

self.samplerate = 16000

self.channels = 1

self.blocksize = 1024 # Размер блока от sounddevice

self.seconds\_per\_chunk = 5 # Сколько секунд накапливать для транскрибации

self.samples\_per\_chunk = self.samplerate \* self.seconds\_per\_chunk

# GUI элементы

self.save\_directory = os.getcwd()

top\_frame = ttk.Frame(root) # Создание группирующего виджета

top\_frame.pack(fill=tk.X, padx=10, pady=5) # Соединение top\_frame с root отступы 10 и 5 и растягивание по горизонтали

# Создание надписи по правому краю top\_frame

self.dir\_label = ttk.Label(top\_frame, text="📁 Папка: " + self.save\_directory, anchor="e")

self.dir\_label.pack(side=tk.RIGHT)

# Создание кнопки в правом краю

self.dir\_button = ttk.Button(top\_frame, text="Выбрать папку", command=self.choose\_directory)

self.dir\_button.pack(side=tk.RIGHT, padx=5)

# Создание кнопки с отступом вниз на 10

self.record\_button = ttk.Button(root, text="Начать запись", command=self.toggle\_recording)

self.record\_button.pack(pady=10)

# Создание индикатора под кнопкой записи

self.recording\_indicator = ttk.Label(root, text="", foreground="red", font=("Arial", 10, "bold"))

self.recording\_indicator.pack()

# Область для вывода текста

self.transcription\_text = tk.Text(root, height=10, width=60, wrap=tk.WORD, state=tk.DISABLED, font=("Arial", 10))

self.transcription\_text.pack(pady=10, padx=10, fill=tk.BOTH, expand=True)

# Добавим прокрутку текста

scrollbar = ttk.Scrollbar(self.transcription\_text, command=self.transcription\_text.yview)

scrollbar.pack(side=tk.RIGHT, fill=tk.Y)

self.transcription\_text['yscrollcommand'] = scrollbar.set

try:

default\_input = sd.default.device[0] # индекс устройства ввода

if default\_input is None or default\_input < 0:

messagebox.showwarning("Внимание", "Микрофон не найден!")

else:

messagebox.showinfo("Уведомление", "Микрофон подключен!")

except:

messagebox.showwarning("Внимание", "Микрофон не найден!")

# Очередь и буфер

self.audio\_queue = queue.Queue()

self.transcription\_queue = queue.Queue()

self.internal\_buffer = [] # Накапливаем здесь до нужного размера

# Состояние

self.recording\_thread = None

self.processing\_thread = None

self.is\_recording = False # Флаг для основного потока записи

self.gui\_update\_timer = None # Для таймера обновления GUI

self.gui\_update\_ms = 100 # Как часто проверять очередь текста (мс)

def choose\_directory(self):

folder = filedialog.askdirectory()

if folder:

self.save\_directory = folder

self.dir\_label.config(text="📁 Папка: " + self.save\_directory)

def audio\_callback(self, indata: np.ndarray, frames: int, time, status: sd.CallbackFlags):

"""Callback функция от sounddevice."""

if status: print(status, flush=True) # Выводим статус, если есть

if not self.is\_recording: return

# Копируем данные и добавляем в буфер и берем только первый канал

self.internal\_buffer.append(indata[:, 0].copy())

# Проверяем, накопилось ли достаточно сэмплов (shape[0] размерность)

total\_samples\_in\_buffer = sum(chunk.shape[0] for chunk in self.internal\_buffer)

if total\_samples\_in\_buffer >= self.samples\_per\_chunk:

# Объединяем части буфера в один длинный сегмент

concatenated\_audio = np.concatenate(self.internal\_buffer, axis=0)

# Берем ровно samples\_per\_chunk

chunk\_to\_process = concatenated\_audio[:self.samples\_per\_chunk]

# Оставшиеся данные возвращаем в буфер

remaining\_audio = concatenated\_audio[self.samples\_per\_chunk:]

# Хвост аудиоданных для обработки с "памятью"

#tail\_audio = chunk\_to\_process[-1600:]

# Кладем готовый чанк в очередь

self.audio\_queue.put(chunk\_to\_process)

# Обновляем внутренний буфер

if remaining\_audio.size > 0:

self.internal\_buffer = [remaining\_audio] #[tail\_audio, remaining\_audio]

else:

self.internal\_buffer = [] #[tail\_audio]

def toggle\_recording(self):

if not self.is\_recording:

self.start\_recording()

else:

self.stop\_recording()

def start\_recording(self):

if self.is\_recording:

return # Уже запущено

print("Запуск записи...")

self.is\_recording = True

self.internal\_buffer = [] # Очищаем буфер

self.record\_button.config(text="Остановить запись")

self.recording\_indicator.config(text="● Идёт запись...")

self.clear\_transcription() # Очищаем текстовое поле

# Запускаем поток для обработки аудио из очереди

self.start\_processing\_thread()

self.start\_gui\_updater() # Запускаем обновление GUI

# Функция для выполнения в потоке записи

def \_record\_thread():

try:

# контекстный менеджер для потока

with sd.InputStream(callback=self.audio\_callback,

samplerate=self.samplerate,

channels=self.channels,

blocksize=self.blocksize, # Размер блока для callback

dtype='float32'): # Используем float32

while self.is\_recording:

sd.sleep(100) # Небольшая пауза, чтобы не грузить CPU

print("Поток записи завершен.")

except Exception as e:

messagebox.showerror("Ошибка", "Запись не может быть начата без записывающего устройства")

print(f"Ошибка в потоке записи: {e}")

self.root.after(0, self.stop\_recording) # Вызываем stop\_recording в главном потоке

# Запускаем поток записи

self.recording\_thread = threading.Thread(target=\_record\_thread, daemon=True)

self.recording\_thread.start()

def stop\_recording(self):

if not self.is\_recording: return # Уже остановлено

print("Остановка записи...")

self.is\_recording = False # Сигнализируем потоку записи остановиться

# Ждем завершения потока записи (с таймаутом)

if self.recording\_thread is not None:

print("Ожидание завершения потока записи...")

self.recording\_thread.join(timeout=1.0) # Ждем не более 1 секунд

if self.recording\_thread.is\_alive():

print("Предупреждение: Поток записи не завершился вовремя.")

self.recording\_thread = None

# Проверка, что буфер пуст

if self.internal\_buffer:

# Объединяем все, что осталось

final\_chunk = np.concatenate(self.internal\_buffer, axis=0)

if final\_chunk.size > 0: # Только если что-то есть

self.audio\_queue.put(final\_chunk) # Кладем последний кусок в очередь

self.internal\_buffer = [] # Очищаем буфер

self.audio\_queue.put(QUEUE\_END\_MARKER)

# Ждем завершения потока обработки (он должен доопработать очередь)

if self.processing\_thread is not None:

print("Ожидание завершения потока обработки...")

self.audio\_queue.join() # Даем БОЛЬШЕ времени на дообработку

self.processing\_thread.join()

self.processing\_thread = None

print("Поток обработки остановлен.")

self.stop\_gui\_updater()

self.check\_transcription\_queue(run\_once=True)

self.root.update\_idletasks()

self.save\_transcription()

self.record\_button.config(text="Начать запись")

self.recording\_indicator.config(text="")

print("Запись остановлена.")

def save\_transcription(self):

"""Сохраняет весь текст из текстового поля в файл."""

full\_text = self.transcription\_text.get("1.0", tk.END).strip() # Получаем весь текст

if not full\_text:

print("Нет текста для сохранения.")

self.dir\_label.config(text="Нет текста для сохранения.")

return

filename = datetime.datetime.now().strftime("transcription\_%d\_%m\_%Y\_%H-%M-%S.txt")

path = os.path.join(self.save\_directory, filename)

try:

with open(path, 'w', encoding='utf-8') as f:

f.write(full\_text)

print(f"Транскрипция сохранена: {path}")

# basename для короткого сообщения в UI

self.dir\_label.config(text=f"✅ Текст сохранен: {os.path.basename(path)}")

except Exception as e:

print(f"Ошибка сохранения транскрипции: {e}")

self.dir\_label.config(text="⚠️ Ошибка сохранения текста!")

def start\_processing\_thread(self):

"""Запускает поток обработки очереди."""

self.processing\_thread = threading.Thread(target=self.process\_audio\_from\_queue, daemon=True)

self.processing\_thread.start()

def process\_audio\_from\_queue(self):

"""Поток, который берет чанки из очереди и транскрибирует их."""

print("Поток обработки очереди запущен.")

if self.whisper\_model:

forced\_decoder\_ids = self.whisper\_tokenizer.get\_decoder\_prompt\_ids(language="ru", task="transcribe")

transcribed\_text = ""

while True:

try:

# Ждем данные из очереди (с таймаутом, чтобы можно было подождать данные)

audio\_chunk = self.audio\_queue.get(timeout=0.5) # Таймаут 0.5 сек

if audio\_chunk is QUEUE\_END\_MARKER:

print("Поток обработки: получен маркер конца.")

self.audio\_queue.task\_done()

break

rms = np.sqrt(np.mean(audio\_chunk\*\*2)) # Корень из среднего квадрата — RMS

db = 20 \* np.log10(rms + 1e-9) # Преобразуем в dB, добавляем ε чтобы не делить на 0

if db < -50:

self.audio\_queue.task\_done()

#print("Тишина, пропускаем...")

#print(f"db {db}")

continue

#print(f"Не шум db {db}")

if self.whisper\_model:

feature = self.whisper\_feature\_extractor(audio\_chunk, sampling\_rate=16000, language="ru", return\_tensors="pt").input\_features

feature = feature.to(self.device)

generated\_tokens =self.whisper\_model.generate(input\_features=feature, forced\_decoder\_ids=forced\_decoder\_ids)

transcribed\_text = self.whisper\_tokenizer.batch\_decode(generated\_tokens.cpu(), skip\_special\_tokens=True)[0]

else:

# Транскрибация моей моделью

transcribed\_text = transcribe\_chunk(audio\_chunk, self.model, self.device, index\_map)

# Обновление GUI

if transcribed\_text and transcribed\_text not in ["[Ошибка обработки]", "[Ошибка инференса]", "[Модель не загружена]"]:

self.transcription\_queue.put(transcribed\_text)

self.audio\_queue.task\_done()

except Exception as e:

#print(f" Очередь пустая: {e}")

time.sleep(0.1) # Небольшая пауза при ошибке

try:

self.audio\_queue.task\_done() # Пытаемся пометить задачу как выполненную

except ValueError:

pass

print("Поток обработки очереди остановлен.")

# обновления GUI из очереди

def check\_transcription\_queue(self, run\_once=False):

"""Проверяет очередь текста и обновляет виджет."""

try:

while True: # Обрабатываем все сообщения в очереди

text\_to\_add = self.transcription\_queue.get\_nowait()

self.update\_transcription\_widget(text\_to\_add + " ")

except queue.Empty:

pass # Очередь пуста

except Exception as e:

print(f"Ошибка обновления GUI из очереди: {e}")

finally:

# Планируем следующую проверку, только если не запрошен однократный запуск

# и если окно еще существует

if not run\_once and self.root.winfo\_exists():

self.gui\_update\_timer = self.root.after(self.gui\_update\_ms, self.check\_transcription\_queue)

def update\_transcription\_widget(self, text\_to\_add):

"""Безопасно обновляет текстовое поле из главного потока."""

self.transcription\_text.config(state=tk.NORMAL) # Разрешаем редактирование

self.transcription\_text.insert(tk.END, text\_to\_add)

self.transcription\_text.see(tk.END) # Автопрокрутка вниз

self.transcription\_text.config(state=tk.DISABLED) # Снова запрещаем

def clear\_transcription(self):

"""Очищает текстовое поле."""

self.transcription\_text.config(state=tk.NORMAL)

self.transcription\_text.delete('1.0', tk.END)

self.transcription\_text.config(state=tk.DISABLED)

# управление таймером обновления GUI

def start\_gui\_updater(self):

"""Запускает периодическую проверку очереди текста."""

self.stop\_gui\_updater() # Останавливаем предыдущий таймер, если был

self.check\_transcription\_queue() # Запускаем первый раз

def stop\_gui\_updater(self):

"""Останавливает периодическую проверку очереди текста."""

if self.gui\_update\_timer is not None:

self.root.after\_cancel(self.gui\_update\_timer)

self.gui\_update\_timer = None

def on\_closing(self):

"""Обработка закрытия окна."""

print("Закрытие приложения...")

if self.is\_recording:

self.stop\_recording() # Останавливаем запись, если идет

else:

if self.processing\_thread is not None and self.processing\_thread.is\_alive():

print("Отправка маркера конца при закрытии...")

self.audio\_queue.put(QUEUE\_END\_MARKER)

# Не ждем здесь долго, просто даем шанс завершиться

self.processing\_thread.join(timeout=1.0)

self.stop\_gui\_updater() # Останавливаем таймер GUI

self.root.destroy() # Закрытие окна

if \_\_name\_\_ == "\_\_main\_\_":

whisper\_model = None

feauture\_extractor = None

tokenizer = None

# whisper-tiny

# feauture\_extractor = WhisperFeatureExtractor.from\_pretrained("models--openai--whisper-tiny/snapshots/169d4a4341b33bc18d8881c4b69c2e104e1cc0af")

# whisper\_model = WhisperForConditionalGeneration.from\_pretrained("models--openai--whisper-tiny/snapshots/169d4a4341b33bc18d8881c4b69c2e104e1cc0af")

# tokenizer = WhisperTokenizer.from\_pretrained("models--openai--whisper-tiny/snapshots/169d4a4341b33bc18d8881c4b69c2e104e1cc0af", language = "russian", task="transcribe")

# whisper-large-v3-turbo

# feauture\_extractor = WhisperFeatureExtractor.from\_pretrained("models--openai--whisper-large-v3-turbo/snapshots/41f01f3fe87f28c78e2fbf8b568835947dd65ed9")

# whisper\_model = WhisperForConditionalGeneration.from\_pretrained("models--openai--whisper-large-v3-turbo/snapshots/41f01f3fe87f28c78e2fbf8b568835947dd65ed9")

# tokenizer = WhisperTokenizer.from\_pretrained("models--openai--whisper-large-v3-turbo/snapshots/41f01f3fe87f28c78e2fbf8b568835947dd65ed9", language = "russian", task="transcribe")

# whisper\_model.to(device)

root = tk.Tk()

if whisper\_model:

app = AudioApp(root, model, feauture\_extractor, whisper\_model, tokenizer) # Загруженная модель Whisper

else:

app = AudioApp(root, model) # Загруженная модель моя

root.protocol("WM\_DELETE\_WINDOW", app.on\_closing) # Обработка закрытия окна

root.mainloop()

##### Приложение Б. Демонстрационный материал

