|  |
| --- |
|  |
| МИНОБРНАУКИ РОССИИ |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА − Российский технологический университет»**  **РТУ МИРЭА** |
| Институт кибербезопасности и цифровых технологий |
| Цифровая кафедра |

|  |
| --- |
| **РАБОТА ДОПУЩЕНА К ЗАЩИТЕ** |
| Заведующий программой  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Ш.Г. Магомедов |
| «31» мая 2024 г. |

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| по программе | |  |  |  |
| **«Программные средства решения прикладных задач искусственного интеллекта»** | | | | |
| на тему: | Модель распознавания вариант 88 | | | |
|  |  | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Обучающийся | \_\_\_\_\_\_\_\_\_\_\_ *подпись* | Вдовичев Никита Сергеевич  *Фамилия, имя, отчество* | |
| Руководитель  работы | \_\_\_\_\_\_\_\_\_\_\_ *подпись* | \_\_\_\_\_\_\_\_\_\_\_\_\_\_  *ученая степень, должность* | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ *Фамилия, имя, отчество* |

**Москва 2024 г.**

|  |
| --- |
| ÐÐ°ÑÑÐ¸Ð½ÐºÐ¸ Ð¿Ð¾ Ð·Ð°Ð¿ÑÐ¾ÑÑ Ð±ÐµÐ»ÑÐ¹ ÑÐ¾Ð½ |
| МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  РОССИЙСКОЙ ФЕДЕРАЦИИ |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** |
| **Институт кибербезопасности и цифровых технологий** |
| Цифровая кафедра |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | | СОГЛАСОВАНО | | | |
|  | | Заведующий программой  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  *(подпись)* | | | |
|  | **Ш.Г. Магомедов** | | | |
|  | «27» | | мая | 2024 г. |

**ЗАДАНИЕ**

**на выполнение выпускной квалификационной работы**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Обучающийся | **Вдовичев Никита Сергеевич** | | | |
|  |  | | | |
| Направление программы | **«Программные средства решения прикладных задач искусственного интеллекта»** | | |
|  |  |  |  | | |

**1. Тема выпускной квалификационной работы**

|  |
| --- |
| Модель распознавания вариант 88 |

**2. Цель и задачи выпускной квалификационной работы**

|  |  |  |
| --- | --- | --- |
| Цель работы: | | Получение практических навыков в решении прикладных задач ИИ |
|  |  | |
| Задачи работы: | | Обучение архитектуры QuartzNet15x5 для создания транскрипций аудиосообщений на русском языке |

**3. Этапы выпускной квалификационной работы**

|  |  |  |  |
| --- | --- | --- | --- |
| №  этапа | Содержание этапа выпускной квалификационной работы | Результат выполнения этапа ВКР | Срок выполнения |
| 1. | **Постановка прикладной задачи** |  | — |
| 2. | **Алгоритм решения поставленной задачи** |  | — |
| 3. | **Реализация поставленной задачи** |  | — |

**4. Перечень разрабатываемых документов и графических материалов**

Цель и задачи ВКР, характеристика предметной области, метод и алгоритм решения задачи, апробация (моделирование или программное обеспечение), выводы.

|  |  |
| --- | --- |
|  | Задание принял к исполнению |
|  | Обучающийся: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  | *подпись* |
|  | «27» мая 2024 г. |

# АННОТАЦИЯ

Данная тема посвящена программным средствам решения прикладных задач в области искусственного интеллекта. В работе рассматриваются основные этапы процесса обработки речи, которые решаются с помощью общих алгоритмов и прикладных программных средств. Описывается категория программных средств, включающая свёрточные нейронные сети и нейронные сети, основанные на трансформерах. В работе приводятся результаты исследований и практических применений нейронных сетей, а также оценивается их эффективность и перспективы для развития технологий искусственного интеллекта.

**СОДЕРЖАНИЕ**

[Термины и определения 7](#_Toc169117073)

[1 ПОСТАНОВКА ПРИКЛАДНОЙ ЗАДАЧИ 8](#_Toc169117074)

[2 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ 9](#_Toc169117075)

[2.1 Определение ASR 9](#_Toc169117076)

[2.2 Используемые метрики 10](#_Toc169117077)

[3 ПОДГОТОВКА ДАННЫХ ДЛЯ ДООБУЧЕНИЯ 11](#_Toc169117078)

[3.1 Выбор датасетов 11](#_Toc169117079)

[3.2 Подготовка шумов 11](#_Toc169117080)

[3.3 Создание файлов манифеста 12](#_Toc169117081)

[4 whisper 13](#_Toc169117082)

[4.1 Определение 13](#_Toc169117083)

[4.2 Вариации моделей 14](#_Toc169117084)

[5 QUARTZNET15X5 15](#_Toc169117085)

[5.1 Nvidia NeMo 15](#_Toc169117086)

[5.2 Архитектура QuartzNet15x5 15](#_Toc169117087)

[6 ТЕСТИРОВАНИЕ МОДЕЛЕЙ 17](#_Toc169117088)

[6.1 Тестирование Whisper 17](#_Toc169117089)

[6.2 Тестирование QuartzNet15x5 19](#_Toc169117090)

[6.3 Выводы 20](#_Toc169117091)

[7 ОБУЧЕНИЕ МОДЕЛИ 21](#_Toc169117092)

[7.1 Настройка конфигурации 21](#_Toc169117093)

[7.2 Затрачиваемые ресурсы 22](#_Toc169117094)

[7.3 Результат обучения 23](#_Toc169117095)

[ЗАКЛЮЧЕНИЕ 24](#_Toc169117096)

[СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ 25](#_Toc169117097)

[ПРИЛОЖЕНИЕ 26](#_Toc169117098)

[Приложение А 27](#_Toc169117099)

[Приложение Б 29](#_Toc169117100)

[Приложение В 32](#_Toc169117101)

**Термины и определения**

В настоящем отчете применяют следующие термины с соответствующими определениями.

|  |  |
| --- | --- |
| Нейронная сеть | * математическая модель, построенная по принципу функционирования биологических нейронов. Способна выявлять сложные зависимости между входными данными и выходными, а также выполнять обобщение |
| Энкодер  (шифратор) | * часть архитектуры нейронной сети, которая преобразует входные данные в сжатое представление. Используется для извлечения свойств данных, уменьшения размерности и др. |
| Декодер  (дешифратор) | * часть архитектуры нейронной сети, которая преобразует внутренние представления данных в человеко-читаемый формат |
| Инференс | * использование обученной модели для получения предсказаний на новых данных. |
| Датасет | * набор данных |
| Квантование | * это процесс уменьшения точности представления чисел (весов и активаций) в нейронной сети. Квантизация помогает снизить размер модели, ускоряет вычисления, поскольку операции с целыми числами выполняются более эффективно. |
| "State of the Art" решение | * передовая технология или метод на данный момент. Представляет собой лучший практический результат на текущий момент, эталон для сравнения с новыми исследованиями и разработками. |

**1 ПОСТАНОВКА ПРИКЛАДНОЙ ЗАДАЧИ**

Для процесса создания транскрипций аудиосообщений в мессенджерах на русском языке необходимо провести анализ предметной области и рассмотреть существующие современные решения в области обработки речи.

Требуется осуществить поиск наборов данных, соответствующих поставленной задаче, выполнить тестирование моделей на выбранных датасетах и определить самую перспективную модель для данного направления.

После этого произвести дообучение выбранной модели для получения минимальной ошибки в процессе транскрибации.

**2 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ**

## 2.1 Определение ASR

**ASR** (Automatic Speech Recognition) - это технология, направленная на автоматическое преобразование речи из аудиоформата в текст. Она использует глубокие нейронные сети для обработки и понимания аудиосигналов и последующего вывода соответствующего текста.

Процесс ASR включает в себя следующие основные шаги:

1. **Захват аудиосигнала:**

• Запись (захват) аудиосигнала с использованием микрофона или другого устройства записи.

2. **Предварительная обработка аудиосигнала:**

• Преобразование аудиосигнала в представление, удобное для обработки нейронной сетью. Это может включать в себя дискретизацию, выравнивание и другие техники обработки сигналов.

3. **Использование акустической модели:**

• Чаще всего используются нейронные сети, такие как рекуррентные нейронные сети (RNN), сверточные нейронные сети (CNN), или их комбинации, используются для извлечения важных признаков из аудиосигнала. Глубокие нейронные сети, такие как рекуррентные нейронные сети с долгой краткосрочной памятью (LSTM) или трансформеры, часто применяются в современных системах ASR.

4. **Преобразование признаков в текст:**

• Преобразование извлеченных признаков в текстовый вывод. Это может быть выполнено с использованием различных методов, включая CTC (Connectionist Temporal Classification) или seq2seq (sequence-to-sequence) подходы.

## 2.2 Используемые метрики

WER, или Word Error Rate (Коэффициент ошибок слов), является метрикой, используемой для измерения качества распознавания речи или автоматического распознавания речи (ASR). WER происходит от расстояния Левенштейна, работающего на уровне слов, а не на уровне фонем.

Эта метрика определяет процент ошибок в распознавании, измеряя отношение минимального количества операций (вставок, удалений и замен) к общему числу слов в эталонной и распознанной фразах.

Формула WER приведена на рисунке 1.1

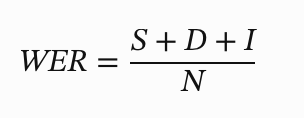


Рисунок 1.1 – Формула WER

где:

• S - количество замен слов. Замена происходит, когда слово в распознанном или переведенном тексте отличается от соответствующего слова в исходном тексте.

• D - количество удалений слов. Удаление происходит, когда слово из исходного текста отсутствует в распознанном или переведенном тексте.

• I - количество вставок слов. Вставка происходит, когда в распознанном или переведенном тексте присутствует дополнительное слово, которого нет в исходном тексте.

• N - общее количество слов в исходном тексте

Итак, WER измеряет, насколько точно система распознавания речи воспроизводит эталонный текст. Чем меньше WER, тем лучше. WER выражается в процентах либо в десятичных дробях. 0% и 0.0 означает идеальное совпадение между распознанным и эталонным текстом.

# 3 ПОДГОТОВКА ДАННЫХ ДЛЯ ДООБУЧЕНИЯ

## 3.1 Выбор датасетов

Для обучения модели необходимы данные, которые будут соответствовать поставленной задаче. Датасеты должны содержать реплики людей на различные бытовые темы, а также содержать фоновый шум для приближения реальной ситуации. Обучающий датасет состоит из 113 часов аудио и собран из 4 датасетов:

* RuDevices - русская живая речь, запись на мобильные устройства и другую непрофессиональную аппаратуру, ручная разметка [1];
* asr\_calls\_2\_val – записи телефонных разговоров [2];
* buriy\_audiobooks\_2\_val – чтение аудио книг [2];
* public\_youtube700\_val – транскрипции видео с хостингов [2].

После загрузки датасетов они были разархивированы, включена

дополнительная настройка «извлечение файлов без путей». Данные распределены на обучающие и валидационные.

Аудиозаписи имеют частоту дискретизации 16000 Гц и длину от 2 до 10 секунд.

## 3.2 Подготовка шумов

Для приближения обучающих данных к аудиозаписям, которые люди отправляют друг другу в жизни к обучающим данным необходимо добавить шумы и искажения [3].

RIR (Импульсная характеристика помещения) — это характеристика аудиосигнала, который отражается от стен и других поверхностей помещения (или среды) перед тем, как достигнет микрофона или слушателя. Эта характеристика описывает, как помещение воздействует на звуковой сигнал, изменяя его в зависимости от времени.

Background Noise – это шум, который присутствует на заднем плане и не является основным объектом акустической сцены. Обычно это шум, вызванный окружающей средой или другими источниками, которые находятся вне основного интереса. Фоновый шум может включать в себя шумы дорожного движения, фоновую речь, шум вентиляции, шум кондиционера и т.д.

Загруженный датасет [4] необходимо разархивировать, включив дополнительную настройку «извлечение файлов без путей».

## 3.3 Создание файлов манифеста

Для дальнейшего использования данных созданы манифесты для обучающих данных и шумов (Листинг кода находится в приложении А).

**4 whisper**

## 4.1 Определение

Whisper - это система автоматического распознавания речи (ASR) [5], разработанная OpenAI. Является многоязычной моделью. Она обучена на большом наборе данных, состоящем из 680 000 часов многоязычных и многозадачных данных. Благодаря использованию такого большого и разнообразного набора данных, Whisper демонстрирует улучшенную устойчивость к акцентам, фоновому шуму и техническому языку.

На текущий момент, "State of the Art" решениями считаются модели, основанные на транформерах. Они позволяют достичь максимальной точности, однако требуют больших вычислительных мощностей при обучении и инференсе. Такой моделью является Whisper.

Архитектура Whisper представляет собой end-to-end подход, реализованный как кодировщик-декодер трансформер:

1. **Входное аудио**: Входное аудио разбивается на блоки по 30 секунд, преобразуется в log-Mel спектрограмму и затем передается в кодировщик.
2. **Кодировщик**: Кодировщик преобразует аудио в векторные представления, которые затем используются декодером.
3. **Декодер**: Декодер обучается предсказывать соответствующую текстовую подпись, перемешанную с специальными токенами. Эти специальные токены направляют одну модель на выполнение задач, таких как определение языка, метки времени на уровне фразы, многоязычная транскрипция речи и перевод речи на английский.

Эта архитектура позволяет Whisper быть гибким и многофункциональным, способным обрабатывать различные задачи распознавания речи(Рисунок 2.1)

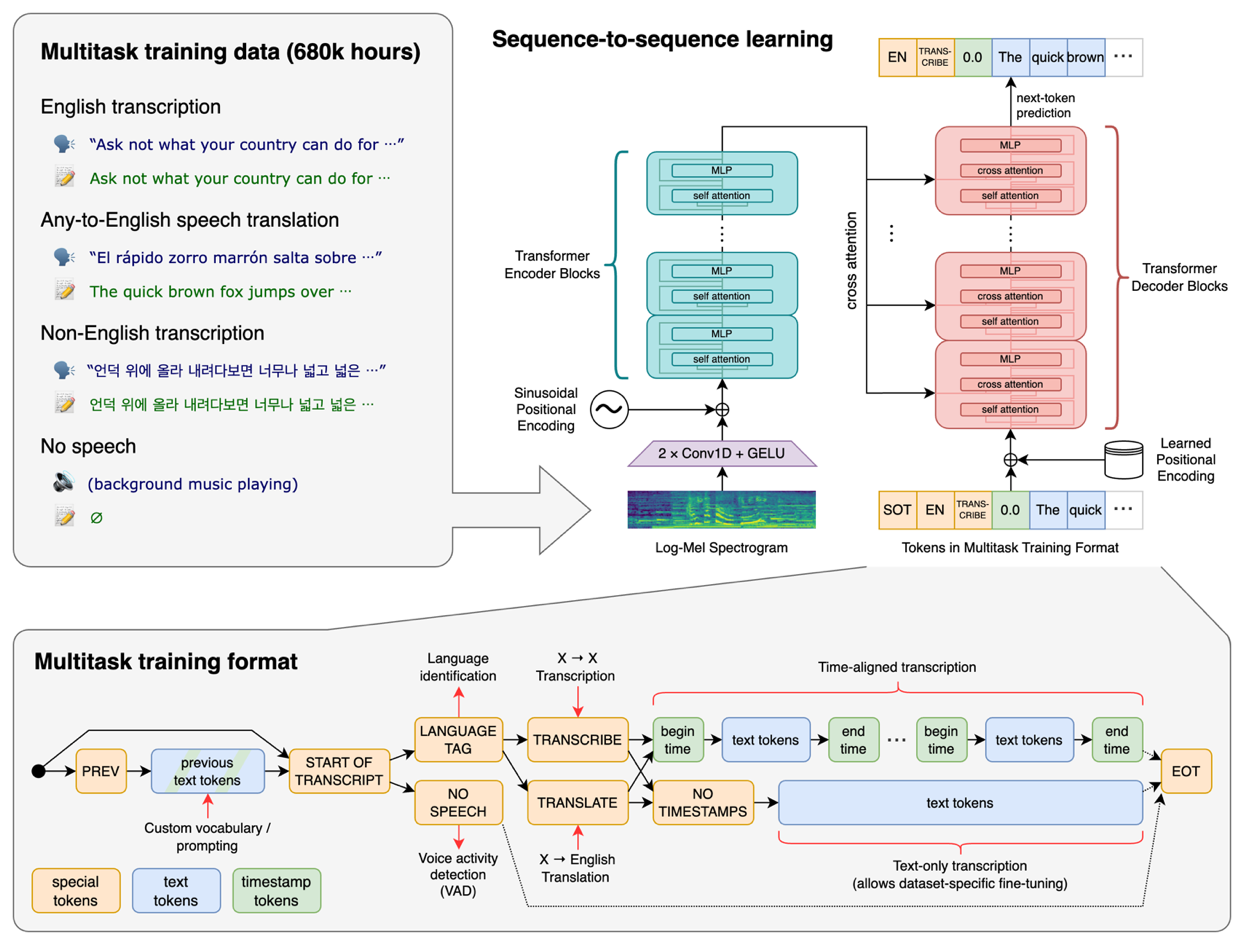


Рисунок 4.1 – Архитектура Whisper

## 4.2 Вариации моделей

Модель Whisper представлена несколькими вариациями:

1. Tiny – 39 миллионов параметров
2. Base – 74 миллиона параметров
3. Small – 244 миллиона параметров
4. Medium – 769 миллионов параметров
5. Large – 1550 миллионов параметров

Все модели имеют одинаковую архитектуру и обучены на одном датасете

# 5 QUARTZNET15X5

## 5.1 Nvidia NeMo

NVIDIA NeMo - фреймворк для создания, настройки и развертывания моделей ИИ. [6] Он включает в себя фреймворки для обучения и вывода, инструменты для установки ограждений, инструменты для обработки данных и предварительно обученные модели, предлагая предприятиям простой, экономически эффективный и быстрый способ принятия генеративного ИИ.

## 5.2 Архитектура QuartzNet15x5

В фреймоворке Nvidia NeMo представлено множнство моделей, основнанных на сверточных нейронных сетях. Такие модели обладают рядом преимуществ, таких как использование общих весов, сверточных фильтров и пулинга, благодаря чему CNN показывают впечатляющие результаты в ASR.

Рассмотрим модель QuartzNet15x5:

QuartzNet - это сеть, которая использует разделимые свертки и большие размеры фильтров. Эта модель имеет 15 блоков, каждый из которых повторяется 5 раз

QuartzNet состоит из 79 слоев и имеет общее количество 18,9 миллиона параметров, с пятью блоками, которые повторяются пятнадцать раз, плюс четыре дополнительных сверточных слоя. Модель состоит из нескольких блоков с остаточными связями между ними, обученных с использованием потерь CTC. Каждый блок состоит из одного или нескольких модулей с 1D временно-канальными разделимыми сверточными слоями, нормализацией по пакетам и слоями ReLU(рисунок 5.1).

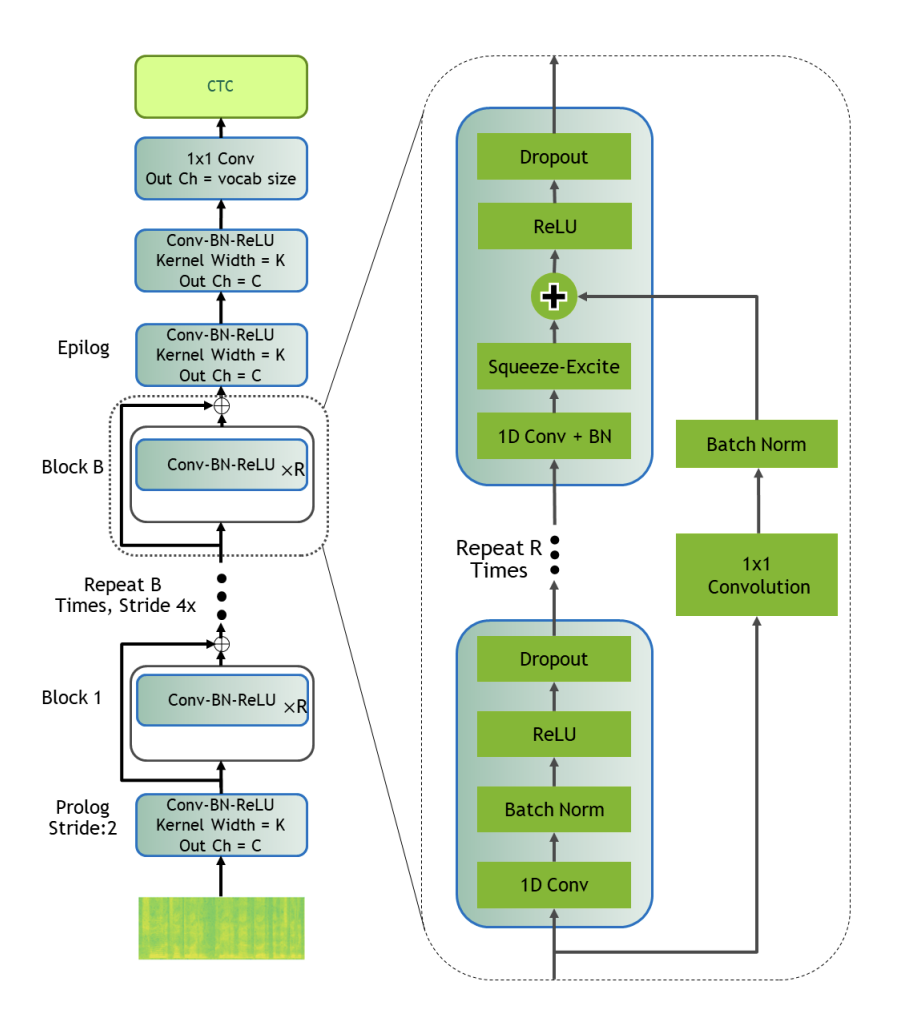


Рисунок 5.1 – Архитектура QuartzNet15x5

# 6 ТЕСТИРОВАНИЕ МОДЕЛЕЙ

**6.1 Тестирование Whisper**

Модели Whisper представлены несколькими вариациями, необходимо определить наиболее релевантную модель. Необходимо учитывать, что модели были обучены преимущественно на английском языке, поэтому тестирование нужно выполнить на англоязычном датасете. Был выбран датасет librispeech\_asr [7]. Тестирование проводилось на 300 аудиозаписях. Тестирование проводилось на CPU в Google Colab.

Абсолютные результаты тестирования представлены в Таблице 6.1, относительные в Таблице 6.2.

Таблица 6.1 – абсолютные результаты тестирования моделей Whisper

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название | Размер (ГБ) | время(сек) | Время на 1 аудио(сек) | wer |
| large - whisper | 6.17 | 16617.71 | 55.4 | 2.11% |
| medium - whisper | 3.06 | 6600.15 | 22 | 2.46% |
| small - whisper | 0.97 | 2176.65 | 7.25 | 3.03% |
| base - whisper | 0.29 | 1412.48 | 4.7 | 4.70% |
| tiny - whisper | 0.15 | 501.17 | 1.67 | 7.25% |

Таблица 6.2 – относительные результаты тестирования моделей Whisper

|  |  |  |  |
| --- | --- | --- | --- |
| Название | Размер | Скорость | Точность |
| large - whisper | x | x | x |
| medium - whisper | **0.50x** | 13.17x | 0.996x |
| small - whisper | **0.16x** | 21.73x | 0.991x |
| base - whisper | **0.05x** | 18.74x | 0.973x |
| tiny - whisper | **0.02x** | 33.16x | 0.947x |

Для сравнения с моделями из библиотеки Nvidia NeMo была выбрана вариация Base так как она является оптимальной по размеру, скорости работы и точности работы среди моделей этого семейства.

Произведено тестирование модели Whisper-Base для тестовой выборки датасета, созданного для выполнения задачи, которая состоит из 2050 аудиозаписей. Тестирование выполнено на GPU RTX 4070 Laptop 8GB, результаты представлены в Таблице 6.3

Таблица 6.3 – Результаты тестирования модели Whisper-Base для созданного датасета

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название | Размер (МБ) | время(сек) | Время на 1 аудио(сек) | wer |
| base - whisper | 290 | 856.91 | 0.40 | 49.35% |

## 6.2 Тестирование QuartzNet15x5

Сравним производительность QuartzNet15x5 от NVidia NeMo

и Quartznet15x5\_golos (получена с помощью трансферного обучения QuartzNet15x5 от NVidia NeMo) [8].

Тестирование проводилось для тестовой выборки датасета, созданного для выполнения задачи, которая состоит из 2050 аудиозаписей. Тестирование выполнено на CPU i7-12700H и GPU RTX 4070 Laptop 8GB, результаты представлены в Таблице 6.4.

Таблица 6.4 – результаты тестирования моделей QuartzNet15x5

|  |  |  |
| --- | --- | --- |
|  | QuartzNet15x5 | QuartzNet15x5\_golos |
| Время CPU (сек) | 637.31 | 632.96 |
| Среднее время CPU (сек) | 0.31 | 0.31 |
| Время CUDA (сек) | 62.42 | 61.51 |
| Среднее время CUDA (сек) | 0.03 | 0.03 |
| WER CPU | 72.15% | 50.51% |
| WER CUDA | 72.12% | 50.47% |

## 6.3 Выводы

Рассмотрим таблицу с результатами тестирования (Листинг кода находится в приложении Б) моделей Whisper и QuartzNet15x5 при обработке данных на GPU (Таблица 6.5).

Таблица 6.5 – Результаты тестирования моделей Whisper и QuartzNet15x5

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название | Размер (МБ) | время(сек) | Время на 1 аудио(сек) | wer |
| base - whisper | 290 | 856.91 | 0.40 | 49.35% |
| QuartzNet15x5 | 70 | 62.42 | 0.03 | 72.12% |
| QuartzNet15x5\_golos | 70 | 61.51 | 0.03 | 50.47% |

Для обучения стоит использовать модель Quartznet15x5\_golos от Сбера. Так как она демонстрирует лучшие показатели для созданного датасета, работа с ней будет наиболее эффективной и менее ресурсоемкой.

# 7 ОБУЧЕНИЕ МОДЕЛИ

## 7.1 Настройка конфигурации

При обучении Quartznet15x5 были использованы несколько комбинаций параметров оптимизатора и планировщика обучения. Лучший результат обучения получен при использовании оптимизатора NovoGrad и планировщика обучения с косинусным аннулированием (Листинг 7.1)

*Листинг 7.1 – Конфигурация оптимизатора и планировщика*

optim:

name: novograd

lr: 0.005

betas:

- 0.9

- 0.98

weight\_decay: 0.001

sched:

name: CosineAnnealing

warmup\_steps: 10000

warmup\_ratio: null

min\_lr: 0.0005

last\_epoch: -1

Обучение показало лучший результат с использованием Room Impulse Response и Background Noise, а также применением RIR к шуму (Листинг 7.2).

*Листинг 7.2 - Применение шумов и искажений*

augmentor:

rir\_noise\_aug:

prob: 1.0

rir\_manifest\_path:

- C:\Users\vdovichev\Documents\DatasetForFineTune\SimulatedRIRSnoises\manifestRirs.jsonl

rir\_prob: 0.6

bg\_noise\_manifest\_paths:

- C:\Users\vdovichev\Documents\DatasetForFineTune\PointSourceNoises\manifestNoises.jsonl

bg\_noise\_tar\_filepaths:

- null

bg\_noise\_prob: 0.6

bg\_min\_snr\_db:

- -3

bg\_max\_snr\_db:

- 5

apply\_noise\_rir: True

Также можно использовать более простые варианты шумов, например белый шум (Листинг 7.3) или шум без импульсной характеристики помещения (Листинг 7.4).

*Листинг 7.3 - Применение шума без импульсной характеристики помещения*

augmentor:

noise:

prob: 0.5

min\_snr\_db: -5

max\_snr\_db: 5

max\_gain\_db: 300

manifest\_path:

- C:\Users\vdovichev\Documents\DatasetForFineTune\PointSourceNoises\manifestNoises.jsonl

*Листинг 3.3 - Применение белого шума*

augmentor:

white\_noise:

prob: 0.3

min\_level: -50

max\_level: -15

## 7.2 Затрачиваемые ресурсы

Обучающий датасет состоит из 113 часов аудио. Лучший результат показывает обучение, состоящее из 2 итераций 20 эпохах, скорость обучения зависит от выбора GPU (Таблица 7.1)

Таблица 7.1 – Сравнительная таблица для различных GPU

|  |  |
| --- | --- |
| GPU | Время на 1 эпоху |
| RTX 4070 Laptop 8GB | 104 минуты |
| Tesla T4 15GB | 72 минуты |
| Tesla V100 32GB | 35 минут |

## 7.3 Результат обучения

После обучения модель Quartznet15x5 достигает метрики Word Error Rate = 27.51%, при незначительном увеличении размера модели и скорости её работы. Назовём эту модель bestQuartz1206. Составим сравнительные таблицы абсолютных (Таблица 7.2) и относительных (Таблица 7.3) результатов тестирования на тестовой выборке датасета, созданного для выполнения задачи, которая состоит из 2050 аудиозаписей. Тестирование выполнено на GPU RTX 4070 Laptop 8GB.

Таблица 7.2 – Абсолютные результаты тестирования моделей

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название | Размер (МБ) | время(сек) | Время на 1 аудио(сек) | wer |
| base - whisper | 290 | 856.91 | 0.40 | 49.35% |
| QuartzNet15x5 | 70 | 62.42 | 0.03 | 72.12% |
| QuartzNet15x5\_golos | 70 | 61.51 | 0.03 | 50.47% |
| bestQuartz1206 | 74 | 62.7 | 0.03 | 27.51% |

Таблица 6.2 – относительные результаты тестирования моделей Whisper

|  |  |  |  |
| --- | --- | --- | --- |
| Название | Размер | Скорость | Точность |
| base - whisper | 4.14x | 0.07x | 1.02x |
| QuartzNet15x5 | x | 0.99x | 0.56x |
| QuartzNet15x5\_golos | **x** | x | x |
| bestQuartz1206 | **1.06x** | 0.98x | 1.46x |

**ЗАКЛЮЧЕНИЕ**

В ходе выполнения выпускной квалификационный работы проанализированы ключевые модели и технологии позволяющие достичь результата в обучении модели для задачи создания транскрипций аудиосообщений на русском языке, собран датасет для обучения.

Полученная во время исследования модель на 46% превосходит в точности ближайшего конкурента (модель QuartzNet15x5, обученную Сбером) в рамках задачи обработки аудиосообщений. При увеличении размера модели на 4 МБ и потери скорости в 2%.

Поставленная задача была выполнена. В дальнейшем можно увеличить точность модели путём дальнейшего обучения с помощью коммерческого использования более мощных GPU.

**СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ**

1. Набор данных RuDevices <https://github.com/sovaai/sova-dataset>

2. Набор данных Open STT <https://github.com/snakers4/open_stt/>

3. Исходный код для применения искажений <https://docs.nvidia.com/deeplearning/nemo/user-guide/docs/en/main/_modules/nemo/collections/asr/parts/preprocessing/perturb.html>

4. Набор шумов <https://www.kaggle.com/datasets/nhattruongdev/rirs-noises>

5. Репозиторий Whisper <https://github.com/openai/whisper>

6. Репозиторий Nvidia NeMo <https://github.com/NVIDIA/NeMo>

7. Набор данных librispeech\_asr <https://huggingface.co/datasets/openslr/librispeech_asr>

8. Модель Golos <https://github.com/salute-developers/golos/blob/master/golos/README_ru.md>

# ПРИЛОЖЕНИЕ

Приложение А — Создание манифестов для датасетов

Приложение Б — Тестирование моделей Whisper и QuartzNet15x5

Приложение В — Обучение модели QuartzNet15x5

## Приложение А

*Листинг А.1 — Создание манифестов для обучающих датасетов*

import os

import json

import wave

# Путь к папке с файлами .wav и .txt

folder\_path = r"C:\Users\vdovichev\Documents\DatasetForFineTune\RuDevicesDataset\files"

# Путь к выходному файлу manifest.jsonl

output\_file = r"C:\Users\vdovichev\Documents\DatasetForFineTune\RuDevicesDataset\manifestRuDevices.jsonl"

# Список для хранения данных о каждом элементе

manifest\_data = []

# Функция для получения длительности аудио файла

def get\_audio\_duration(audio\_path):

with wave.open(audio\_path, 'rb') as audio\_file:

frames = audio\_file.getnframes()

rate = audio\_file.getframerate()

duration = frames / float(rate)

return round(duration, 1)

# Перебираем все файлы .txt в папке

for file in os.listdir(folder\_path):

if file.endswith('.txt'):

text\_file\_path = os.path.join(folder\_path, file)

audio\_file\_path = os.path.join(folder\_path, file[:-4] + '.wav')

if os.path.exists(audio\_file\_path):

# Создаем уникальный ID из имени файла

file\_id = os.path.splitext(file)[0]

# Получаем текст из .txt файла

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

text = text\_file.read().strip()

# Получаем длительность аудио файла

duration = get\_audio\_duration(audio\_file\_path)

# Создаем объект данных для добавления в manifest.jsonl

data = {

"id": file\_id,

"audio\_filepath": f"files/{file\_id}.wav",

"text": text,

"duration": duration

}

manifest\_data.append(data)

# Записываем данные в manifest.jsonl

with open(output\_file, 'w', encoding='utf-8') as manifest\_file:

for data in manifest\_data:

json.dump(data, manifest\_file, ensure\_ascii=False)

manifest\_file.write('\n')

*Листинг А.2 — Создание манифестов для шумов*

import os

import json

import wave

# Путь к папке с файлами .wav

folder\_path = r"C:\Users\vdovichev\Documents\DatasetForFineTune\PointSourceNoises\files"

# Путь к выходному файлу manifest.jsonl

output\_file = r"C:\Users\vdovichev\Documents\DatasetForFineTune\PointSourceNoises\manifestNoises.jsonl"

# Список для хранения данных о каждом элементе

manifest\_data = []

# Функция для получения длительности аудио файла (не более 15 секунд)

def get\_audio\_duration(audio\_path):

with wave.open(audio\_path, 'rb') as audio\_file:

frames = audio\_file.getnframes()

rate = audio\_file.getframerate()

duration = frames / float(rate)

return round(min(duration, 15.0), 1)

# Перебираем все файлы .wav в папке

for file in os.listdir(folder\_path):

if file.endswith('.wav'):

audio\_file\_path = os.path.join(folder\_path, file)

if os.path.exists(audio\_file\_path):

# Создаем уникальный ID из имени файла

file\_id = os.path.splitext(file)[0]

# Получаем длительность аудио файла

duration = get\_audio\_duration(audio\_file\_path)

# Создаем объект данных для добавления в manifest.jsonl

data = {

"audio\_filepath": f"files/{file\_id}.wav",

"duration": duration,

"label": "noise",

"text": "\_",

"offset": 0.0,

}

manifest\_data.append(data)

# Записываем данные в manifest.jsonl

with open(output\_file, 'w', encoding='utf-8') as manifest\_file:

for data in manifest\_data:

json.dump(data, manifest\_file, ensure\_ascii=False)

manifest\_file.write('\n')

## Приложение Б

*Листинг Б.1 — Тестирование моделей Whisper*

from transformers import WhisperProcessor, WhisperForConditionalGeneration

import time

import torch

from datasets import load\_dataset

dset = load\_dataset("librispeech\_asr", 'clean', split="test", streaming=True)

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

# Загрузите модель и процессор

processor = WhisperProcessor.from\_pretrained("openai/whisper-large")

model = WhisperForConditionalGeneration.from\_pretrained("openai/whisper-large")

model.config.forced\_decoder\_ids = None

# Перенесите модель на GPU, если CUDA доступен

model.to(device)

print("Device:", device)

# Дополнительная информация о GPU, если доступно

if device.type == 'cuda':

print("GPU Name:", torch.cuda.get\_device\_name(0))

print("CUDA Version:", torch.version.cuda)

dset\_iter = iter(dset) # создаем итератор

#для пропуска первых n элементов датасета

n = 250

for \_ in range(n):

next(dset\_iter)

predictions = []

references = []

total\_time = 0

num\_of\_audio = 25

for i in range(num\_of\_audio):

current\_element = next(dset\_iter)

audio = current\_element["audio"]

input\_features = processor(audio["array"], sampling\_rate=audio["sampling\_rate"], return\_tensors="pt").input\_features

reference = processor.tokenizer.\_normalize(current\_element['text'])

references.append(reference)

print(reference)

start\_time = time.time()

predicted\_ids = model.generate(input\_features)

transcription = processor.batch\_decode(predicted\_ids, skip\_special\_tokens=True)

end\_time = time.time()

prediction = processor.tokenizer.\_normalize(transcription[0])

predictions.append(prediction)

print(prediction)

file\_time = end\_time - start\_time

total\_time += file\_time

*Продолжение Листинг Б.1*

print(len(references))

print(len(predictions))

from jiwer import wer

wer\_result = wer(references, predictions)

print(f"Result wer: {wer\_result \* 100}")

accuracy = 1 - wer\_result

print("Accuracy: %.5f" % accuracy)

print(f"Общее время: {total\_time:.2f} секунд")

print(f"Среднее время на одно аудио: {total\_time/num\_of\_audio:.2f} секунд")

*Листинг Б.2 — Тестирование моделей QuartzNet15x5*

import json

import os

import numpy as np

import torch

from ruamel.yaml import YAML

import torchaudio

import time

from omegaconf import DictConfig, OmegaConf, open\_dict

import nemo

import nemo.collections.asr as nemo\_asr

asr\_model = nemo\_asr.models.EncDecCTCModel.from\_pretrained(model\_name="stt\_ru\_quartznet15x5", map\_location='cuda')

# Получение конфигурации модели и сохранение

config\_model = asr\_model.cfg

config\_yaml = OmegaConf.to\_yaml(config\_model)

with open(r'C:\Users\vdovichev\Documents\VKR\models\_and\_configs\QuartzNet15x5\_Nvidia\_config.yaml', 'w') as yaml\_file:

yaml\_file.write(config\_yaml

yaml = YAML(typ='safe')

with open(r'C:\Users\vdovichev\Documents\VKR\models\_and\_configs\QuartzNet15x5\_Nvidia\_config.yaml') as f:

config\_model = yaml.load(f)

preprocessor = nemo\_asr.models.EncDecCTCModel.from\_config\_dict(config\_model['preprocessor'])

decoder = nemo\_asr.metrics.wer.CTCDecoding(config\_model['decoding'], vocabulary=config\_model['decoder']['vocabulary'])

print(asr\_model.device) # проверим что cuda подключилась

manifest\_paths = [

r"C:\Users\vdovichev\Documents\proj1-STT\DatasetForFineTune\RuDevicesDataset\manifestValidationRuDevices.jsonl",

r"C:\Users\vdovichev\Documents\proj1-STT\DatasetForFineTune\OpenSTTruDatasets\asr\_calls\_2\_val\manifestValidationSTTcalls.jsonl",

r"C:\Users\vdovichev\Documents\proj1-STT\DatasetForFineTune\OpenSTTruDatasets\buriy\_audiobooks\_2\_val\manifestValidationSTTaudiobooks.jsonl",

r"C:\Users\vdovichev\Documents\proj1-STT\DatasetForFineTune\OpenSTTruDatasets\public\_youtube700\_val\manifestValidationSTTyoutube.jsonl"

]

*Продолжение Листинг Б.2*

base\_dirs = [

r"C:\Users\vdovichev\Documents\proj1-STT\DatasetForFineTune\RuDevicesDataset",

r"C:\Users\vdovichev\Documents\proj1-STT\DatasetForFineTune\OpenSTTruDatasets\asr\_calls\_2\_val",

r"C:\Users\vdovichev\Documents\proj1-STT\DatasetForFineTune\OpenSTTruDatasets\buriy\_audiobooks\_2\_val",

r"C:\Users\vdovichev\Documents\proj1-STT\DatasetForFineTune\OpenSTTruDatasets\public\_youtube700\_val",

]

asr\_model.eval()

asr\_model.encoder.freeze()

asr\_model.decoder.freeze()

num\_of\_audio\_all = 0

hypotheses = []

y\_target = []

total\_time = 0

start\_time = time.time()

for manifest\_path, base\_dir in zip(manifest\_paths, base\_dirs):

print(manifest\_path)

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

manifest\_data = [json.loads(line) for line in f]

num\_of\_audio = len(manifest\_data)

num\_of\_audio\_all += num\_of\_audio

for audio\_index in range(num\_of\_audio):

current\_audio\_entry = manifest\_data[audio\_index]

y\_target.append(current\_audio\_entry.get('text', ''))

current\_audio\_filename = current\_audio\_entry.get('audio\_filepath', '')

file\_path = os.path.join(base\_dir, current\_audio\_filename)

#print(file\_path)

waveform, sample\_rate = torchaudio.load(file\_path)

processed\_signal, processed\_signal\_len = preprocessor(input\_signal=waveform, length=torch.tensor([len(waveform[0])]))

processed\_signal = processed\_signal.to('cuda')

processed\_signal\_len = processed\_signal\_len.to('cuda')

encoder\_output = asr\_model.encoder(audio\_signal=processed\_signal, length=processed\_signal\_len)

logits = asr\_model.decoder(encoder\_output=encoder\_output[0])

current\_hypotheses, \_ = decoder.ctc\_decoder\_predictions\_tensor(logits, decoder\_lengths=processed\_signal\_len)

hypotheses.extend(current\_hypotheses)

total\_time = time.time() - start\_time

print(f"Число аудио: {num\_of\_audio\_all}")

print(f"Общее время: {total\_time:.2f} секунд")

print(f"Среднее время на одно аудио: {total\_time/num\_of\_audio\_all:.2f} секунд")

print(f"Word Error Rate (WER): {nemo\_asr.metrics.wer.word\_error\_rate(hypotheses, y\_target) \* 100:.2f}%")

## Приложение В

*Листинг В.1 — Обучение модели QuartzNet15x5*

import librosa

import IPython.display as ipd

import nemo

import torch

import nemo.collections.asr as nemo\_asr

import pytorch\_lightning as pl

from ruamel.yaml import YAML

from omegaconf import DictConfig

from nemo.core.classes import ModelPT

from nemo.utils import exp\_manager

from omegaconf import OmegaConf

import os

yaml = YAML(typ='safe')

params\_path = r"C:\Users\vdovichev\Documents\VKR\models\_and\_configs\augmentationConf.yaml"

with open(params\_path, encoding="utf-8") as f:

params = yaml.load(f)

asr\_model = ModelPT.restore\_from(r"C:\Users\vdovichev\Documents\VKR\models\_and\_configs\bestQuartz1206.nemo")

asr\_model.setup\_optimization(optim\_config= DictConfig(params['optim']))

asr\_model.setup\_validation\_data(val\_data\_config=params['validation\_ds'])

asr\_model.setup\_training\_data(train\_data\_config=params['train\_ds'])

print(asr\_model.\_cfg)

if torch.cuda.is\_available():

accelerator = 'gpu'

else:

accelerator = 'gpu'

EPOCHS = 20 # 100 epochs would provide better results

trainer = pl.Trainer(devices=1,

accelerator=accelerator,

max\_epochs=EPOCHS,

enable\_checkpointing=False,

logger=False,

limit\_val\_batches = 1.0,

check\_val\_every\_n\_epoch = 1,

val\_check\_interval = 1.0)

# Setup model with the trainer

asr\_model.set\_trainer(trainer)

os.environ.pop('NEMO\_EXPM\_VERSION', None)

*Продолжение Листинг В.1*

config = exp\_manager.ExpManagerConfig(

exp\_dir="C:\\Users\\vdovichev\\Documents\\proj1-STT\\FineTune\\nemo\_experiments\\",

name=f"test2",

create\_tensorboard\_logger = True,

create\_checkpoint\_callback = True,

checkpoint\_callback\_params=exp\_manager.CallbackParams(

monitor="val\_wer",

mode="min",

always\_save\_nemo=True,

save\_best\_model=True,

),

create\_wandb\_logger = False,

wandb\_logger\_kwargs={

"name": None,

"project": None,

},

)

config = OmegaConf.structured(config)

logdir = exp\_manager.exp\_manager(trainer, config)

%load\_ext tensorboard

%tensorboard --logdir "C:\\Users\\vdovichev\\Documents\\proj1-STT\\FineTune\\nemo\_experiments\\test2\\" --port 8888

trainer.validate(asr\_model)

trainer.fit(asr\_model)