

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет

вычислительной техники

Кафедра

«Вычислительная техника»

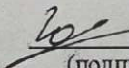
Направление подготовки: 09.04.01 «Информатика и вычислительная техника»
Магистерская программа: «Прикладной искусственный интеллект»

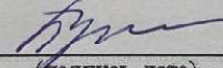
МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

на тему

РАЗРАБОТКА СЕРВИСА

РАСПОЗНАВАНИЯ ЦЕЛЕВОГО ГОЛОСА В
УСЛОВИЯХ ШУМОВЫХ ПОМЕХ

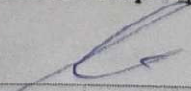
Студент  25.05.26 Горбунов Никита Алексеевич
(подпись, дата) (ФИО полностью)

Руководитель  25.05.26 Бурукина И.П.
(подпись, дата) (фамилия, инициалы)

Нормоконтролёр  25.05.26 Бычков А.С.
(подпись, дата) (фамилия, инициалы)

Рецензент главный специалист 22.05.26
АО «НПП «Рубин» Васильев Васильев АВ
(должность, место работы) (подпись, дата) (фамилия, инициалы)

Работа допущена к защите (протокол зас. кафедры от 22.05.26 № 14)

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

Работа защищена с отметкой отлично (протокол заседания ГЭК
от 04 июня 2026 № 10)

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

Пенза 2026

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

Факультет

вычислительной техники

Кафедра

«Вычислительная техника»

«Утверждаю»

Заведующий кафедрой

М.А. Митрохин

« 20 » 09 2024 г.

**ЗАДАНИЕ
НА МАГИСТЕРСКУЮ ДИССЕРТАЦИЮ**

1. Студент Горбунов Никита Алексеевич 24ВВИМ2 факультета вычислительной техники, направления 09.04.01

2. Тема работы «Разработка сервиса распознавания целевого голоса в условиях шумовых помех»

Тема утверждена приказом ПГУ № 920/0 от "20" сентября 2024 г.

3. Руководитель магистерской диссертации к.т.н., доцент Бурукина И.П.

4. Задание на работу (назначение разработки, исходные данные и т.п.)

Назначение разработки: автоматическое распознавание и транскрибация речи целевого говорящего в многоголосых аудиозаписях на основе методов биометрической идентификации голоса и нейросетевой обработки акустических сигналов.

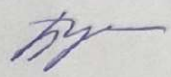
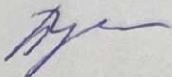
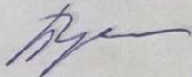
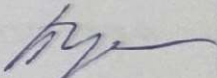
Разработать программный сервис, принимающий на вход аудиозапись с несколькими участниками и короткий голосовой образец целевого диктора, выполняющий идентификацию и транскрибацию речи только этого участника. Сервис должен функционировать в офлайн режиме без передачи данных во внешние сервисы.

5. Перечень подлежащих разработке вопросов

1. Анализ существующих методов диаризации речи, шумоподавления и автоматического распознавания речи, оценка возможности их применения для идентификации целевого говорящего,
2. Разработка алгоритма идентификации целевого диктора по голосовому образцу на основе сравнения голосовых отпечатков с применением косинусного сходства,

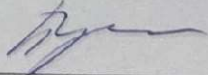
3. Проектирование и реализация пайплайна, объединяющего диаризацию, идентификацию целевого говорящего и транскрибацию его речи, включая механизмы коррекции ошибок диаризации на основе акустического сходства и языковой модели.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
	Анализ предметной области и существующих систем	10%	20.09.2024- 21.11.2024	
	Исследование и сравнительный анализ методов диаризации, извлечения голосовых отпечатков и транскрибации речи	30%	22.11.2024- 22.01.2025	
	Разработка алгоритма идентификации целевого диктора по голосовому образцу на основе косинусного сходства	20%	23.01.2025- 23.04.2025	
	Проектирование и реализация пайплайна (диаризация, идентификация, транскрибация, коррекция ошибок)	20%	24.04.2025- 29.01.2026	
	Тестирование сервиса и оценка качества по метрике WER	10%	30.01.2026- 30.03.2026	
	Сравнение с аналогами и оформление работы	10%	31.03.2026- 31.05.2026	

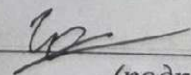
Дата выдачи задания "20" сентября 2024 г.

Руководитель магистерской диссертации


(подпись, дата)

Бурукина И.П.

Задание к исполнению принял студент


(подпись, дата)

Горбунов Н.А.

Работу к защите допустить
Декан факультета


(подпись, дата)

Фионова Л.Р.

02.06.2026

РЕФЕРАТ

Магистерская работа содержит 116 листов, 21 рисунок, 8 таблиц, 40 источников, 2 приложения.

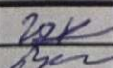
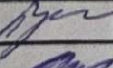
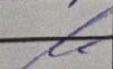
ДИАРИЗАЦИЯ РЕЧИ, РАСПОЗНАВАНИЕ ЦЕЛЕВОГО ГОВОРЯЩЕГО, ГОЛОСОВОЙ ОТПЕЧАТОК, КОСИНУСНОЕ СХОДСТВО, ТРАНСКРИБАЦИЯ РЕЧИ, НЕЙРОСЕТЕВАЯ ОБРАБОТКА АКУСТИЧЕСКИХ СИГНАЛОВ, ШУМОВЫЕ ПОМЕХИ, МНОГОГОЛОСЫЕ АУДИОЗАПИСИ, БИОМЕТРИЧЕСКАЯ ИДЕНТИФИКАЦИЯ, ЯЗЫКОВАЯ МОДЕЛЬ

Цель работы – разработка программного сервиса, принимающего многоголосую аудиозапись и голосовой образец целевого участника и формирующего расшифровку только его речи в офлайн-режиме.

Объектом разработки является программный сервис идентификации и транскрибации речи целевого говорящего в условиях акустических помех. Сервис принимает на вход многоголосую аудиозапись и короткий голосовой образец нужного участника, выполняет диаризацию, сравнивает голосовые отпечатки всех говорящих с референсом и формирует текстовую расшифровку речи только целевого диктора.

В результате работы разработан и протестирован сервис, объединяющий диаризацию, идентификацию диктора по голосовому образцу и транскрибацию в единый пайплайн. Целевой диктор верно определён во всех тестовых записях. Ни один из трёх проверенных аналогов не располагает механизмом идентификации конкретного диктора по голосовому образцу.

Результаты могут применяться для протоколирования совещаний, в юридической и медицинской практике, а также в дальнейших разработках в области персонализированного распознавания речи.

					ПГУ1.090401.3.001 ПЗ			
Изм.	Лист	№ докум.	Подпись	Дата				
Разраб.		Горбунов Н.А.		25.05	Разработка сервиса распознавания целевого голоса в условиях шумовых помех Пояснительная записка	Лит.	Лист	Листов
Провер.		Бурукина И.П.		25.05			3	116
Н. Контр		Бычков А.С.		25.05		ФВТ, каф. ВТ гр. 24ВВИм2		
Утверд.		Митрохин М.А.		25.05				

СОДЕРЖАНИЕ

РЕФЕРАТ	4
ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ	10
ВВЕДЕНИЕ.....	11
1 Анализ предметной области	14
1.1 Актуальность задачи выделения целевого голоса.....	14
1.2 История развития методов распознавания речи	17
1.3 Обзор существующих методов и систем.....	19
1.3.1 Спектральное вычитание	20
1.3.2 Вейвлет-преобразование	20
1.3.3 Фильтр Винера	20
1.3.4 Сравнительный анализ классических методов.....	22
1.4 Диаризация речи	23
1.5 Постановка задачи	25
1.6 Выводы по первому разделу	25
2 Теоретические основы и выбор методов.....	26
2.1 Цифровое представление аудиосигнала	26
2.2 Методы акустической обработки сигналов.....	30
2.3 Методы шумоподавления и разделения источников звука.....	32
2.4 Подходы к распознаванию и верификации диктора.....	34
2.5 Метрики качества и методология оценки.....	35
2.6 Выводы по второму разделу	36
3 Проектирование и разработка сервиса	38
3.1 Архитектура сервиса	38
3.2 Предобработка аудиосигнала.....	40

3.3	Диаризация аудиозаписи	42
3.4	Извлечение голосовых отпечатков	43
3.5	Идентификация целевого диктора	44
3.6	Слияние ошибочно разделенных говорящих	46
3.7	Слияние говорящих на основе анализа текста.....	47
3.8	Транскрибация речи целевого диктора	48
3.9	Постобработка результатов транскрибации	49
3.10	Сохранение промежуточных результатов.....	50
3.11	Выводы по третьему разделу	51
4	Экспериментальное исследование и оценка результатов	52
4.1	Описание экспериментальной базы данных	52
4.2	Условия проведения экспериментов	53
4.3	Результаты диаризации и идентификации диктора.....	54
4.4	Результаты распознавания целевого голоса	57
4.5	Сравнение с существующими системами	58
4.5.1	Методология сравнения	59
4.5.2	Результаты сравнения.....	59
4.5.3	Анализ результатов.....	60
4.5.4	Сводная оценка	68
4.6	Выводы по четвёртому разделу	69
	ЗАКЛЮЧЕНИЕ	71
	СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	73
	ПРИЛОЖЕНИЕ А – ЛИСТИНГ	77
	ПРИЛОЖЕНИЕ В – ПРЕЗЕНТАЦИЯ	109

ПРИЛОЖЕНИЕ А
(обязательное)
Листинг программы

Файл «main.py»

```
"""
```

```
Target Speaker Extraction — точка входа
```

```
Магистерская работа: Распознавание целевого голоса в условиях  
шумовых помех
```

```
Запуск:
```

```
python main.py
```

```
"""
```

```
import os
```

```
import shutil
```

```
from datetime import datetime
```

```
import torch
```

```
import whisper
```

```
from pyannote.audio import Pipeline
```

```
from speechbrain.inference.speaker import EncoderClassifier
```

```
import config
```

```
from audio_utils import convert_to_wav
```

```
from speaker import extract_reference_embedding,  
compute_speaker_embeddings, filter_target_segments
```

```
from merge_speakers import merge_similar_speakers, merge_via_llm
```

```
from diarization import diarize_audio
```

```
from transcription import transcribe_segments,  
merge_close_segments, save_result
```

```
from debug_save import save_diarization, save_per_speaker_audio,  
save_target_audio, save_similarity_scores
```

```
from postprocess import postprocess_segments
```

```
def archive_temp(temp_path: str) -> None:
```

```
    """Перемещает содержимое TEMP в подпапку с датой-временем."""
```

```
    if not os.path.exists(temp_path):
```

```

        return
    files = [f for f in os.listdir(temp_path)
              if os.path.isfile(os.path.join(temp_path, f))]
    if not files:
        return
    stamp = datetime.now().strftime("%Y-%m-%d_%H-%M-%S")
    archive_dir = os.path.join(temp_path, f"temp_{stamp}")
    os.makedirs(archive_dir, exist_ok=True)
    for f in files:
        shutil.move(os.path.join(temp_path,
                                   f),
                    os.path.join(archive_dir, f))
    print(f"    Предыдущие файлы TEMP → {archive_dir}")

```

```

def main():
    device = "cuda" if torch.cuda.is_available() else "cpu"
    print(f"Device: {device}")

```

```

if config.DEBUG:
    archive_temp(config.TEMP_PATH)
    os.makedirs(config.TEMP_PATH, exist_ok=True)

```

#	—	Загрузка	моделей
---	---	----------	---------

```

print("\n[1/3] Загрузка ECAPA-TDNN (speaker encoder)...")
encoder = EncoderClassifier.from_hparams(
    source=config.ECAPA_MODEL,
    savedir=config.ECAPA_SAVEDIR,
    run_opts={"device": device},
)

```

```

print("[2/3] Загрузка pyannote diarization pipeline...")
diarization_pipeline = Pipeline.from_pretrained(
    config.DIARIZATION_MODEL,
    use_auth_token=config.HF_TOKEN,
)

```

```

)
diarization_pipeline.to(torch.device(device))

print("[3/3] Загрузка Whisper...")
whisper_model = whisper.load_model(config.WHISPER_MODEL,
device=device)

```

#	—	Конвертация	в	WAV	если	нужно
---	---	-------------	---	-----	------	-------

```

tmp_wav = None
audio_wav = config.AUDIO_PATH
if not config.AUDIO_PATH.lower().endswith(".wav"):
    print("\n[Pre] Конвертация аудио в WAV через ffmpeg...")
    audio_wav = convert_to_wav(config.AUDIO_PATH)
    tmp_wav = audio_wav
    print(f"    Временный файл: {audio_wav}")

```

```

try:

```

#	—	Шаг	1:	референс
---	---	-----	----	----------

```

    print("\n[Step 1] Извлечение эмбединга референсного
голоса...")
    ref_emb =
extract_reference_embedding(config.REFERENCE_PATH, encoder,
device)
    print(f"    Shape: {ref_emb.shape}")

```

#	—	Шаг	2:	диаризация
---	---	-----	----	------------

```

print("\n[Step 2] Диаризация аудио...")
segments = diarize_audio(audio_wav, diarization_pipeline)
speakers = len(set(s["speaker"] for s in segments))
print(f"    {len(segments)} сегментов, {speakers} спикеров")
if config.DEBUG:
    save_diarization(segments, config.TEMP_PATH)

```

#	—	Шаг	3:	эмбединги	+	scores
---	---	-----	----	-----------	---	--------

```

print("\n[Step 3] Вычисление эмбедингов спикеров...")
speaker_emb = compute_speaker_embeddings(
    audio_wav, segments, encoder, device,
config.MIN_SEGMENT_DUR,
)

# Считаем scores до merge — нужны для проверки минимального
порога
_, scores = filter_target_segments(
    segments, speaker_emb, ref_emb,
    config.THRESHOLD, threshold_min=config.THRESHOLD_MIN,
)

if config.MERGE_SPEAKERS:
    print("\n[Step 3b] Слияние похожих спикеров
(эмбединги) ...")
    segments, _ = merge_similar_speakers(
        segments, speaker_emb,
        merge_threshold=config.MERGE_THRESHOLD,
        ref_scores=scores,
        threshold_min=config.THRESHOLD_MIN,
    )
    speaker_emb = compute_speaker_embeddings(
        audio_wav, segments, encoder, device,
config.MIN_SEGMENT_DUR,
)

```

#	—	Шаг	4:	поиск	целевого	спикера
---	---	-----	----	-------	----------	---------

```

print("\n[Step 4] Поиск целевого спикера...")
target_segs, scores = filter_target_segments(
    segments, speaker_emb, ref_emb,

```

```

        config.THRESHOLD, threshold_min=config.THRESHOLD_MIN,
    )
    print(f" Совпало: {len(target_segs)} сегментов")

    if config.DEBUG:
        save_similarity_scores(scores, config.TEMP_PATH)
        save_per_speaker_audio(audio_wav, segments, scores,
config.TEMP_PATH)
        save_target_audio(audio_wav, target_segs,
config.TEMP_PATH)

    if not target_segs:
        print(f"\nСовпадений нет. Попробуй снизить THRESHOLD
"
            f"(сейчас {config.THRESHOLD}, например до
0.65)")
        return

# — Шаг 5: транскрипция кандидатов


---


    print("\n[Step 5] Транскрипция через Whisper...")

    if config.MERGE_CONFIRM_LLM:
        # Транскрибируем ВСЕХ кандидатов выше порога
        spk_texts: dict[str, str] = {}
        for label, sim in sorted(scores.items(), key=lambda x:
-x[1]):
            if sim < config.THRESHOLD_MIN:
                continue
            spk_segs = [s for s in segments if s["speaker"] ==
label]
            print(f" {label} (sim={sim:.4f}):
{len(spk_segs)} сегментов...")
            spk_transcribed = transcribe_segments(

```

```

        audio_wav,        spk_segs,        whisper_model,
config.LANGUAGE,
    )
    spk_texts[label] = " ".join(s["text"] for s in
spk_transcribed)
    spk_path = os.path.join(
        config.TEMP_PATH,
f"transcription_{label}_sim{sim:.3f}.txt"
    )
    save_result(spk_transcribed, spk_path)

    print(f"\n[Step 6] LLM объединяет спикеров по смыслу
({config.LLM_MODEL})...")
    merged_text = merge_via_llm(
        segments, scores, spk_texts,
        threshold_min=config.THRESHOLD_MIN,
        temp_path=config.TEMP_PATH,
        model=config.LLM_MODEL,
    )

    if config.LLM_CORRECT:
        print(f"\n[Step 7] Постобработка текста через LLM
({config.LLM_MODEL})...")
        from postprocess import correct_text,
ensure_model

        ensure_model(config.LLM_MODEL)
        merged_text = correct_text(merged_text,
model=config.LLM_MODEL)

        os.makedirs(os.path.dirname(config.OUTPUT_PATH) or
".", exist_ok=True)
        with open(config.OUTPUT_PATH, "w", encoding="utf-8")
as f:

            f.write("TARGET SPEAKER TRANSCRIPT\n")
            f.write("=" * 50 + "\n\n")

```

```

        f.write(merged_text + "\n")
    print(f"\n✓ Сохранено: {config.OUTPUT_PATH}")

    else:
        # Обычный режим — только целевой спикер
        transcribed = transcribe_segments(
            audio_wav, target_segs, whisper_model,
            config.LANGUAGE,
        )
        transcribed = merge_close_segments(transcribed,
            gap=config.MERGE_SEGMENTS_GAP)

        if config.DEBUG:
            raw_path = os.path.join(config.TEMP_PATH,
                "transcription_raw.txt")
            save_result(transcribed, raw_path)

        if config.DEBUG_ALL_SPEAKERS:
            print("\n[DEBUG] Транскрипция всех спикеров для
сравнения...")
            for label, sim in sorted(scores.items(),
                key=lambda x: -x[1]):
                if sim < config.THRESHOLD_MIN:
                    continue
                spk_segs = [s for s in segments if
                    s["speaker"] == label]
                print(f"        {label} (sim={sim:.4f}):
{len(spk_segs)} сегментов...")
                spk_transcribed = transcribe_segments(
                    audio_wav, spk_segs, whisper_model,
                    config.LANGUAGE,
                )
                spk_path = os.path.join(
                    config.TEMP_PATH,
                    f"transcription_{label}_sim{sim:.3f}.txt"

```

```

        )
        save_result(spkr_transcribed, spkr_path)

    if config.LLM_CORRECT:
        print(f"\n[Step 6] Постобработка текста через LLM
({config.LLM_MODEL})...")
        transcribed = postprocess_segments(transcribed,
model=config.LLM_MODEL)

        save_result(transcribed, config.OUTPUT_PATH)

    finally:
        if tmp_wav and os.path.exists(tmp_wav):
            os.remove(tmp_wav)

if __name__ == "__main__":
    main()

```

Файл «config.py»

```
"""
Настройки пайплайна — меняй здесь
"""

REFERENCE_PATH =
r"D:/University/Dissertation/TestWork/TestSounds/reference_mine.
wav"
AUDIO_PATH =
r"D:/University/Dissertation/TestWork/TestSounds/Я+Юля_Сложный.m
4a"
OUTPUT_PATH = r"D:/University/Dissertation/TestWork/result.txt"

HF_TOKEN = "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"

THRESHOLD = 0.7 # основной порог cosine similarity (0.0-
1.0)
THRESHOLD_MIN = 0.35 # минимальный порог — ниже него спикер
игнорируется полностью
WHISPER_MODEL = "medium" # tiny / base / small / medium / large
LANGUAGE = "ru"
MIN_SEGMENT_DUR = 1.0 # минимальная длина сегмента (сек) для
вычисления эмбединга

ECAPA_MODEL = "speechbrain/spkrec-ecapa-voxceleb"
ECAPA_SAVEDIR =
r"D:/University/Dissertation/TestWork/models/ecapa"

DIARIZATION_MODEL = "pyannote/speaker-diarization-3.1"

SUPPORTED_FORMATS = {".wav", ".mp3", ".m4a", ".aac", ".flac",
".ogg", ".wma"}

FFMPEG_BIN = r"C:\ffmpeg\bin\ffmpeg.exe"
```

```

# ————— Отладка


---


DEBUG      = True  # сохранять промежуточные файлы в TEMP (wav-ки,
scores, diarization.json)
TEMP_PATH = r"D:/University/Dissertation/TestWork/TEMP"

# ————— LLM


---


LLM_MODEL      = "qwen2.5:7b"  # модель Ollama
MERGE_CONFIRM_LLM = True      # транскрибировать всех кандидатов
и объединить через LLM
                                # (когда True —
DEBUG_ALL_SPEAKERS и MERGE_SEGMENTS_GAP не нужны)
LLM_CORRECT     = True      # постобработка итогового текста
через LLM (орфография, пунктуация)

# — Слияние спикеров (эмбединг-based, быстрое)


---


MERGE_SPEAKERS   = True  # объединять очевидно похожих спикеров
по эмбедингам
MERGE_THRESHOLD  = 0.70  # минимальный порог speaker↔speaker для
слияния
                                # (работает совместно с критерием
взаимного ближайшего соседа)

# — Только когда MERGE_CONFIRM_LLM = False


---


MERGE_SEGMENTS_GAP = 1.5  # склеивать сегменты с паузой < N
сек в один блок
DEBUG_ALL_SPEAKERS = False # транскрибировать всех кандидатов
в TEMP для сравнения

```

Файл «audio_utils.py»

"""

Загрузка аудио с поддержкой множества форматов.

WAV / FLAC — грузит torchaudio напрямую.

MP3 / M4A / AAC / OGG / WMA — конвертирует через ffmpeg CLI (нужен в PATH).

```
"""
```

```
import pathlib
import subprocess
import tempfile
import os
import torch
import torchaudio
import numpy as np
```

```
from config import SUPPORTED_FORMATS, FFMPEG_BIN
```

```
# Форматы, которые torchaudio читает без ffmpeg
_NATIVE_FORMATS = {".wav", ".flac"}
```

```
def _load_via_ffmpeg(path: str, target_sr: int) ->
tuple[torch.Tensor, int]:
    """Декодирует файл через ffmpeg CLI, возвращает (waveform [1,
N], sr)."""
    cmd = [
        FFMPEG_BIN, "-hide_banner", "-loglevel", "error",
        "-i", path,
        "-ar", str(target_sr),
        "-ac", "1",
        "-f", "f32le", # raw 32-bit float, little-endian
        "pipe:1",
    ]
    try:
```

```

        proc = subprocess.run(cmd, capture_output=True,
                                check=True)
    except FileNotFoundError:
        raise RuntimeError(
            "ffmpeg не найден. Установи его и добавь в PATH:\n"
            "  https://ffmpeg.org/download.html"
        )
    except subprocess.CalledProcessError as e:
        raise RuntimeError(
            f"ffmpeg не смог декодировать"
            f'{{path}}':\n{e.stderr.decode()}'
        )

    samples = np.frombuffer(proc.stdout, dtype=np.float32).copy()
    waveform = torch.from_numpy(samples).unsqueeze(0) # [1, N]
    return waveform, target_sr

def convert_to_wav(path: str) -> str:
    """
    Конвертирует аудиофайл в временный WAV (16kHz, mono) через
    ffmpeg.

    Возвращает путь к временному файлу — удали его после
    использования.

    Если файл уже WAV — возвращает исходный путь без конвертации.
    """
    ext = pathlib.Path(path).suffix.lower()
    if ext == ".wav":
        return path # конвертация не нужна

    tmp = tempfile.NamedTemporaryFile(suffix=".wav",
                                        delete=False)
    tmp.close()
    cmd = [
        FFMPEG_BIN, "-hide_banner", "-loglevel", "error",

```

```

        "-i", path,
        "-ar", "16000", "-ac", "1",
        "-y", tmp.name,
    ]
    try:
        subprocess.run(cmd, check=True, capture_output=True)
    except FileNotFoundError:
        raise RuntimeError("ffmpeg не найден. Убедись, что он
установлен в conda-среде.")
    except subprocess.CalledProcessError as e:
        raise RuntimeError(f"ffmpeg не смог конвертировать
'{path}':\n{e.stderr.decode()}")

    return tmp.name


def load_audio(path: str, target_sr: int = 16000) ->
tuple[torch.Tensor, int]:
    """
    Загружает аудиофайл любого поддерживаемого формата.
    Возвращает (waveform [1, N], sample_rate=target_sr).
    """
    ext = pathlib.Path(path).suffix.lower()
    if ext not in SUPPORTED_FORMATS:
        raise ValueError(
            f"Неподдерживаемый формат: '{ext}'. "
            f"Допустимые: {'', '}.join(SUPPORTED_FORMATS)}"
        )

    if ext in _NATIVE_FORMATS:
        waveform, sr = torchaudio.load(path)
        if waveform.shape[0] > 1:
            waveform = waveform.mean(dim=0, keepdim=True)
        if sr != target_sr:

```

```
        waveform = torchaudio.transforms.Resample(sr,
target_sr)(waveform)
    return waveform, target_sr
else:
    return _load_via_ffmpeg(path, target_sr)
```

Файл «diarization.py»

```
"""
Диаризация аудио через pyannote.audio
"""

from pyannote.audio.pipelines.utils.hook import ProgressHook

def diarize_audio(path: str, pipeline) -> list[dict]:
    """
    Запускает диаризацию и возвращает список сегментов:
    [{"start": float, "end": float, "speaker": str}, ...]
    """
    with ProgressHook() as hook:
        diarization = pipeline(path, hook=hook)

    segments = []
    for turn, _, speaker in diarization.itertracks(yield_label=True):
        segments.append({
            "start": turn.start,
            "end": turn.end,
            "speaker": speaker,
        })
    return segments
```

Файл «merge_speakers.py»

"""

Объединение спикеров которые оказались одним человеком.

Два метода:

1. Эмбединг-based: попарное cosine similarity между спикерами.
2. LLM-based: скармливаем LLM таблицу (время + сходство + текст), она решает кого объединить. Хорошо ловит случаи когда один человек

разбился на 2 спикера из-за паузы или смены тональности.

"""

```
import torch
```

```
from transcription import format_time
```

```
from speaker import cosine_similarity
```

```
def merge_similar_speakers(
```

```
    segments: list[dict],
```

```
    speaker_emb: dict[str, torch.Tensor | None],
```

```
    merge_threshold: float = 0.70,
```

```
    ref_scores: dict[str, float] | None = None,
```

```
    threshold_min: float = 0.0,
```

```
) -> tuple[list[dict], dict[str, str]]:
```

```
    """
```

Объединяет спикеров используя два критерия одновременно:

1. Попарная cosine similarity $\geq$ merge_threshold

2. Взаимный ближайший сосед — каждый из пары наиболее похож именно на другого

(не зависит от качества референса, хорошо ловит разбитых спикеров)

```
    """
```

```
    labels = [lbl for lbl, emb in speaker_emb.items() if emb is not None]
```

```

if len(labels) < 2:
    return segments, {lbl: lbl for lbl in labels}

# Считаем все попарные сходства
sims: dict[tuple[str, str], float] = {}
print("\n— Speaker-to-speaker similarity —")
for i in range(len(labels)):
    for j in range(i + 1, len(labels)):
        a, b = labels[i], labels[j]
        sim = cosine_similarity(speaker_emb[a],
speaker_emb[b])
        sims[(a, b)] = sim
        sims[(b, a)] = sim
        print(f" {a} ↔ {b}: {sim:.4f}")

# Ближайший сосед для каждого спикера
nearest: dict[str, str] = {}
for lbl in labels:
    others = {other: sims[(lbl, other)] for other in labels if
other != lbl}
    if others:
        nearest[lbl] = max(others, key=others.__getitem__)

# Объединяем если: взаимные ближайшие соседи И similarity >=
порога
# И оба спикера преодолели минимальный порог сходства с
референсом
groups: list[set] = []
merged_pairs: set[frozenset] = set()
for a in labels:
    b = nearest.get(a)
    if b and nearest.get(b) == a: # взаимные соседи
        sim = sims.get((a, b), 0)
        pair = frozenset({a, b})
        # Проверяем минимальный порог с референсом для обоих

```

```

        if ref_scores is not None:
            score_a = ref_scores.get(a, 0)
            score_b = ref_scores.get(b, 0)
            if score_a < threshold_min or score_b <
threshold_min:

                print(f" X {a} ↔ {b}: пропускаем "
                    f"(sim с референсом: {score_a:.3f} /
{score_b:.3f} < {threshold_min})")
                continue
            if sim >= merge_threshold and pair not in
merged_pairs:

                merged_pairs.add(pair)
                print(f" → MERGE {a} ↔ {b} (sim={sim:.4f},
взаимные ближайшие соседи)")

                group_a = next((g for g in groups if a in g), None)
                group_b = next((g for g in groups if b in g), None)
                if group_a and group_b:
                    if group_a is not group_b:
                        group_a |= group_b
                        groups.remove(group_b)
                    elif group_a:
                        group_a.add(b)
                    elif group_b:
                        group_b.add(a)
                else:
                    groups.append({a, b})

grouped = {lbl for g in groups for lbl in g}
for lbl in labels:
    if lbl not in grouped:
        groups.append({lbl})

merge_map: dict[str, str] = {}
for group in groups:
    canonical = min(group)

```

```

        for lbl in group:
            merge_map[lbl] = canonical

    for seg in segments:
        if seg["speaker"] not in merge_map:
            merge_map[seg["speaker"]] = seg["speaker"]

    merged_segments = [
        **seg, "speaker": merge_map[seg["speaker"]]
        for seg in segments
    ]

    changed = {old: new for old, new in merge_map.items() if old
!= new}
    if changed:
        print("\n Объединено:")
        for old, new in changed.items():
            print(f"    {old} → {new}")
    else:
        print("\n Спикеры достаточно различны, объединение не
требуется.")

    return merged_segments, merge_map

def merge_via_llm(
    segments: list[dict],
    scores: dict[str, float],
    speaker_texts: dict[str, str],
    threshold_min: float,
    temp_path: str,
    model: str = "qwen2.5:7b",
) -> str:
    """
    Отдаёт LLM транскрипции всех кандидатов (выше threshold_min).

```

LLM сама объединяет тексты одного человека по смыслу и времени, возвращает один связный текст.

Сохраняет входной промпт в TEMP/llm_merge_input.txt для отладки.

```
"""
import os
import ollama

# Временные диапазоны спикеров
by_speaker: dict[str, list[dict]] = {}
for seg in segments:
    by_speaker.setdefault(seg["speaker"], []).append(seg)

# Строим блоки для всех переданных спикеров
speaker_blocks = []
for label in sorted(speaker_texts.keys()):
    segs = sorted(by_speaker.get(label, []), key=lambda x:
x["start"])
    if not segs:
        continue
    t_start = format_time(segs[0]["start"])
    t_end = format_time(segs[-1]["end"])
    sim = scores.get(label, 0)
    speaker_blocks.append(
        f"[{label}] время {t_start}-{t_end}, сходство с
референсом: {sim:.3f}\n"
        f"{speaker_texts[label]}"
    )

prompt = (
    "Тебе даны транскрипции нескольких спикеров из одной
аудиозаписи.\n"
    "Для каждого указано: временной диапазон, сходство голоса
с референсным, текст.\n\n"
```

```

+ "\n\n".join(speaker_blocks)
+ "\n\n"
"Задача:\n"

"Если некоторые спикеры являются одним человеком (текст
логически продолжается,\n"

"временные диапазоны идут друг за другом) — объедини их
речь в один связный текст.\n"

"Если спикеры явно разные люди — верни только речь того,
чья запись полнее и связнее.\n\n"

"Верни ТОЛЬКО итоговый текст, без пояснений, меток
спикеров и комментариев."
)

# Сохраняем входные данные в TEMP всегда — до любых проверок
os.makedirs(temp_path, exist_ok=True)
input_path = os.path.join(temp_path, "llm_merge_input.txt")
with open(input_path, "w", encoding="utf-8") as f:
    f.write(prompt)
print(f"  Входные данные для LLM → {input_path}")

if not speaker_blocks:
    print("Нет текста для отправки в LLM.")
    return ""

print(f"  Отправляю {len(speaker_blocks)} спикер(ов) на анализ
LLM...")
response = ollama.chat(
    model=model,
    messages=[{"role": "user", "content": prompt}],
)
result = response["message"]["content"].strip()

output_path = os.path.join(temp_path, "llm_merge_output.txt")
with open(output_path, "w", encoding="utf-8") as f:
    f.write(result)

```

```
print(f" Ответ LLM → {output_path}")
```

```
return result
```

Файл «target_speaker_extraction.py»

```
"""
Target Speaker Extraction Pipeline
Магистерская работа: Распознавание целевого голоса в условиях
шумовых помех
"""

import torch
import torchaudio
import whisper
import numpy as np
from pyannote.audio import Pipeline
from speechbrain.inference.speaker import EncoderClassifier
from pyannote.audio.pipelines.utils.hook import ProgressHook

REFERENCE_PATH = r"D:/"
AUDIO_PATH      = r"D:/"
OUTPUT_PATH     = r"D:/"
HF_TOKEN        = "TOKEN"
THRESHOLD       = 0.7 # порог similarity (0.0-1.0)
WHISPER_MODEL   = "medium" # tiny/base/small/medium
LANGUAGE        = "ru"

# -----
# 1. Загрузка аудио → 16kHz mono tensor
# -----

def load_audio(path: str, target_sr: int = 16000):
    waveform, sr = torchaudio.load(path)
    if waveform.shape[0] > 1:
        waveform = waveform.mean(dim=0, keepdim=True)
    if sr != target_sr:
        waveform = torchaudio.transforms.Resample(sr,
target_sr)(waveform)
```

```

    return waveform, target_sr

# -----
# 2. Speaker embedding из референсного файла
# -----
def extract_reference_embedding(path, encoder, device):
    waveform, _ = load_audio(path)
    waveform = waveform.to(device)
    with torch.no_grad():
        emb = encoder.encode_batch(waveform)
    return emb.squeeze()

# -----
# 3. Диаризация — кто говорил когда
# -----
def diarize_audio(path, pipeline):
    with ProgressHook() as hook:
        diarization = pipeline(path, hook=hook)
    segments = []
    for turn, _ , speaker in
diarization.itertracks(yield_label=True):
        segments.append({"start": turn.start, "end": turn.end,
"speaker": speaker})
    return segments

# -----
# 4. Cosine similarity
# -----
def cosine_similarity(a, b):
    a = a / (a.norm() + 1e-8)
    b = b / (b.norm() + 1e-8)
    return float(torch.dot(a, b).item())

```

```

# -----
# 5. Фильтрация сегментов целевого спикера
# -----

def filter_target_segments(audio_path, segments, ref_emb,
                           encoder, threshold, device, min_dur=1.0):
    waveform, sr = load_audio(audio_path)
    waveform = waveform.to(device)

    # Группируем сегменты по спикеру
    by_speaker = {}
    for seg in segments:
        by_speaker.setdefault(seg["speaker"], []).append(seg)

    # Считаем один эмбеддинг на спикера
    speaker_emb = {}
    for label, segs in by_speaker.items():
        chunks = []
        for seg in segs:
            s, e = int(seg["start"] * sr), int(seg["end"] * sr)
            chunk = waveform[:, s:e]
            if chunk.shape[1] > int(min_dur * sr):
                chunks.append(chunk)
        if not chunks:
            speaker_emb[label] = None
            continue
        combined = torch.cat(chunks, dim=1)
        with torch.no_grad():
            speaker_emb[label] = encoder.encode_batch(combined).squeeze()

    # Сравниваем с референсом
    matched = []
    print("\n— Similarity scores —")

```

```

for label, emb in speaker_emb.items():
    if emb is None:
        print(f" {label}: skipped (too short)")
        continue
    sim = cosine_similarity(ref_emb, emb)
    print(f" {label}: {sim:.4f} {'✓ MATCH' if sim >=
threshold else 'X'}")
    if sim >= threshold:
        matched.extend(by_speaker[label])

matched.sort(key=lambda x: x["start"])
return matched

```

```

# -----
# 6. Транскрипция через Whisper
# -----

def transcribe_segments(audio_path, segments, whisper_model,
language):
    waveform, sr = load_audio(audio_path)
    results = []
    for seg in segments:
        s, e = int(seg["start"] * sr), int(seg["end"] * sr)
        chunk = waveform[:,
s:e].squeeze().numpy().astype(np.float32)
        result = whisper_model.transcribe(chunk,
language=language, fp16=False)
        text = result["text"].strip()
        if text:
            results.append({"start": seg["start"], "end":
seg["end"], "text": text})
    return results

```

```

# -----

```

```

# 7. Сохранение результата
# -----

def format_time(s):
    m, s = divmod(int(s), 60)
    h, m = divmod(m, 60)
    return f"{h:02d}:{m:02d}:{s:02d}"

def save_result(segments, path):
    with open(path, "w", encoding="utf-8") as f:
        f.write("TARGET SPEAKER TRANSCRIPT\n")
        f.write("=" * 50 + "\n\n")
        for seg in segments:
            f.write(f"[{format_time(seg['start'])}
{format_time(seg['end'])}]\n{seg['text']}\n\n")
        print(f"\n✓ Saved: {path} ({len(segments)} segments)")

# =====
# MAIN
# =====

if __name__ == "__main__":
    device = "cuda" if torch.cuda.is_available() else "cpu"
    print(f"Device: {device}")

    print("\n[1/4] Loading ECAPA-TDNN speaker encoder...")
    encoder = EncoderClassifier.from_hparams(
        source="speechbrain/spkrec-ecapa-voxceleb",

savedir=r"D:/University/Dissertation/TestWork/models/ecapa",
    run_opts={"device": device}
)

    print("[2/4] Loading pyannote diarization pipeline...")
    diarization_pipeline = Pipeline.from_pretrained(
        "pyannote/speaker-diarization-3.1",

```

```

        use_auth_token=HF_TOKEN
    )
    diarization_pipeline.to(torch.device(device))

    print("[3/4] Loading Whisper...")
    whisper_model = whisper.load_model(WHISPER_MODEL,
device=device)

    print("\n[Step 1] Extracting reference embedding...")
    ref_emb = extract_reference_embedding(REFERENCE_PATH,
encoder, device)
    print(f" Shape: {ref_emb.shape}")

    print("\n[Step 2] Diarizing audio...")
    segments = diarize_audio(AUDIO_PATH, diarization_pipeline)
    print(f" {len(segments)} segments, {len(set(s['speaker'] for
s in segments))} speakers")

    print("\n[Step 3] Matching target speaker...")
    target_segs = filter_target_segments(AUDIO_PATH, segments,
ref_emb, encoder, THRESHOLD, device)
    print(f" Matched: {len(target_segs)} segments")

    if not target_segs:
        print("\nNo match. Try lowering THRESHOLD (e.g. 0.65)")
    else:
        print("\n[Step 4] Transcribing with Whisper...")
        transcribed = transcribe_segments(AUDIO_PATH,
target_segs, whisper_model, LANGUAGE)
        save_result(transcribed, OUTPUT_PATH)

```

Файл «transcription.py»

```
"""
Транскрипция сегментов через Whisper + сохранение результата
"""

import numpy as np
from audio_utils import load_audio

def transcribe_segments(
    audio_path: str,
    segments: list[dict],
    whisper_model,
    language: str,
) -> list[dict]:
    waveform, sr = load_audio(audio_path)
    results = []
    for seg in segments:
        s, e = int(seg["start"] * sr), int(seg["end"] * sr)
        chunk = waveform[:,
s:e].squeeze().numpy().astype(np.float32)
        result = whisper_model.transcribe(chunk,
language=language, fp16=False)
        text = result["text"].strip()
        if text:
            results.append({
                "start": seg["start"],
                "end": seg["end"],
                "text": text,
            })
    return results

def merge_close_segments(segments: list[dict], gap: float = 1.5)
-> list[dict]:
```

```

"""
Склеивает соседние сегменты с паузой меньше gap секунд в один
блок.

Уменьшает дробление текста на мелкие куски.
"""

if not segments:
    return []
merged = [dict(segments[0])]
for seg in segments[1:]:
    if seg["start"] - merged[-1]["end"] <= gap:
        merged[-1]["end"] = seg["end"]
        merged[-1]["text"] = merged[-1]["text"].rstrip() + "
" + seg["text"].lstrip()
    else:
        merged.append(dict(seg))
return merged

def format_time(seconds: float) -> str:
    m, s = divmod(int(seconds), 60)
    h, m = divmod(m, 60)
    return f"{h:02d}:{m:02d}:{s:02d}"

def save_result(segments: list[dict], path: str) -> None:
    # Сортируем по времени — гарантирует правильный порядок после
    слияния спикеров
    segments = sorted(segments, key=lambda x: x["start"])
    with open(path, "w", encoding="utf-8") as f:
        f.write("TARGET SPEAKER TRANSCRIPT\n")
        f.write("=" * 50 + "\n\n")
        for seg in segments:
            f.write(
                f"[{format_time(seg['start'])}
{format_time(seg['end'])}]\n"

```

```
        f"{seg['text']}\n\n"
    )
print(f"\n✓ Сохранено: {path}  ({len(segments)} сегментов)")
```

Файл «speaker.py»

```
"""
Speaker embedding: извлечение и сравнение с референсом (ECAPA-
TDNN)
"""

import torch
from audio_utils import load_audio

def extract_reference_embedding(path: str, encoder, device: str)
-> torch.Tensor:
    waveform, _ = load_audio(path)
    waveform = waveform.to(device)
    with torch.no_grad():
        emb = encoder.encode_batch(waveform)
    return emb.squeeze()

def cosine_similarity(a: torch.Tensor, b: torch.Tensor) -> float:
    a = a / (a.norm() + 1e-8)
    b = b / (b.norm() + 1e-8)
    return float(torch.dot(a, b).item())

def compute_speaker_embeddings(
    audio_path: str,
    segments: list[dict],
    encoder,
    device: str,
    min_dur: float = 1.0,
) -> dict[str, torch.Tensor | None]:
    """Считает один эмбеддинг на спикера по всем его сегментам."""
    waveform, sr = load_audio(audio_path)
    waveform = waveform.to(device)
```

```

by_speaker: dict[str, list[dict]] = {}
for seg in segments:
    by_speaker.setdefault(seg["speaker"], []).append(seg)

speaker_emb: dict[str, torch.Tensor | None] = {}
for label, segs in by_speaker.items():
    chunks = []
    for seg in segs:
        s, e = int(seg["start"] * sr), int(seg["end"] * sr)
        chunk = waveform[:, s:e]
        if chunk.shape[1] > int(min_dur * sr):
            chunks.append(chunk)
    if not chunks:
        speaker_emb[label] = None
        continue
    combined = torch.cat(chunks, dim=1)
    with torch.no_grad():
        speaker_emb[label] = encoder.encode_batch(combined).squeeze()

return speaker_emb

```

```

def filter_target_segments(
    segments: list[dict],
    speaker_emb: dict[str, torch.Tensor | None],
    ref_emb: torch.Tensor,
    threshold: float,
    threshold_min: float = 0.35,
) -> tuple[list[dict], dict[str, float]]:
    """
    Сравнивает эмбединги спикеров с референсом,
    возвращает сегменты совпавших спикеров и словарь scores.
    """

```

```

by_speaker: dict[str, list[dict]] = {}
for seg in segments:
    by_speaker.setdefault(seg["speaker"], []).append(seg)

scores: dict[str, float] = {}
print("\n— Similarity scores —")
for label, emb in speaker_emb.items():
    if emb is None:
        print(f" {label}: skipped (too short)")
        continue
    sim = cosine_similarity(ref_emb, emb)
    scores[label] = sim
    tag = "✓ MATCH" if sim >= threshold else "X"
    print(f" {label}: {sim:.4f} {tag}")

matched_labels = [lbl for lbl, sim in scores.items() if sim >=
threshold]

if not matched_labels:
    if scores:
        best_label = max(scores, key=scores.__getitem__)
        best_sim = scores[best_label]
        if best_sim >= threshold_min:
            print(f"\n Никто не прошёл THRESHOLD={threshold}.
"
                f"Берём          лучшего:          {best_label}
({best_sim:.4f})")
            matched_labels = [best_label]
        else:
            print(f"\n          Лучший    спикер    ({best_label}:
{best_sim:.4f}) "
                f"ниже минимального порога {threshold_min}.
Совпадений нет.")

matched: list[dict] = []

```

```
for lbl in matched_labels:
    matched.extend(by_speaker[lbl])

matched.sort(key=lambda x: x["start"])
return matched, scores
```

ПРИЛОЖЕНИЕ В

Презентация

Разработка сервиса распознавания целевого голоса в условиях шумовых помех

Автор работы: магистрант группы 24ВВИМ2 Горбунов Н.А.

Научный руководитель: к.т.н. доцент, Бурукина И.П.

Пенза, 2026

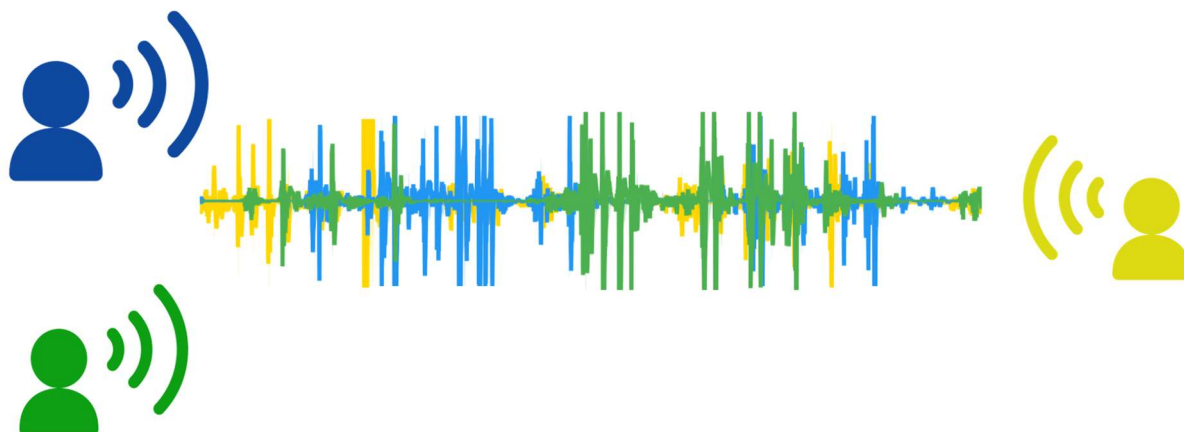
1

Опрос службы исследований HeadHunter среди работающих соискателей



2

Проблема коктейльной вечеринки



3

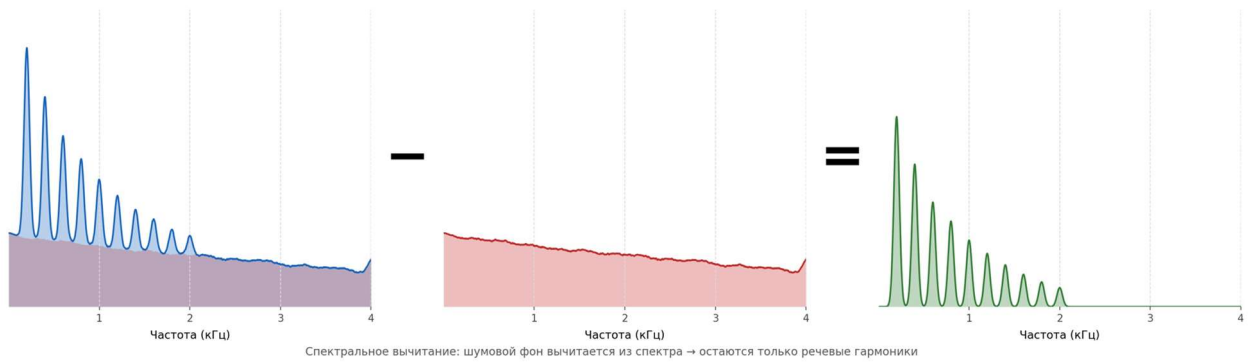
Цель работы: Разработать программный сервис, обеспечивающий распознавание и транскрибацию речи целевого говорящего в условиях шумовых помех на основе методов биометрической идентификации голоса и нейросетевой обработки акустических сигналов.

Для достижения цели решаются следующие задачи:

- Анализ существующих методов диаризации, идентификации говорящего и автоматического распознавания речи, а также оценка их применимости в условиях многоголосых записей с фоновыми шумами
- Разработка алгоритма идентификации целевого диктора по голосовому образцу на основе сравнения векторных представлений акустических характеристик говорящих
- Проектирование и реализация программного пайплайна, интегрирующего диаризацию, идентификацию целевого говорящего и транскрибацию его речи в единый сервис.
- Экспериментальная проверка разработанного сервиса на тестовых аудиозаписях с различным числом дикторов и характером речи, а также сравнение результатов с аналогичными доступными инструментами

4

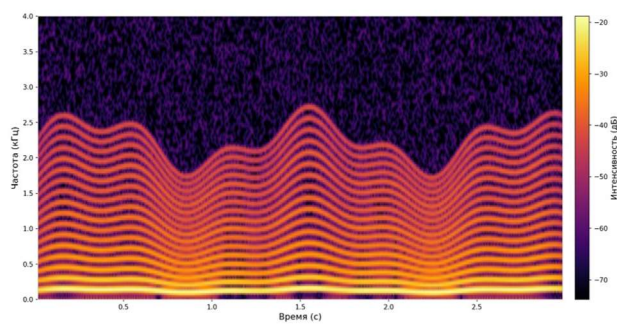
Классические методы могут устранять только постоянный шум



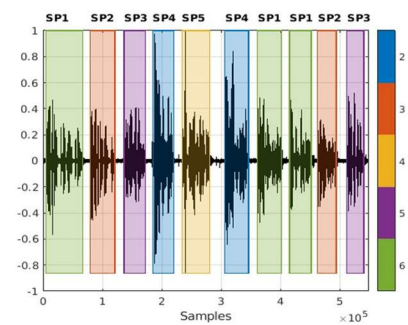
5

Чтобы найти нужного человека в записи нужны **три** ключевые вещи:

1) Голосовой отпечаток



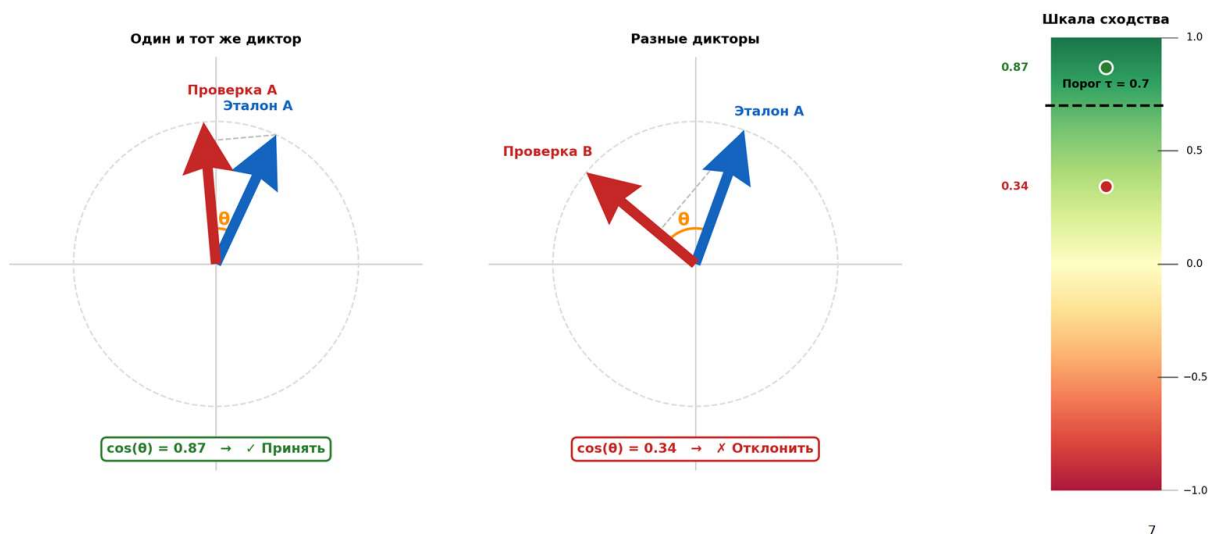
2) Диаризация



6

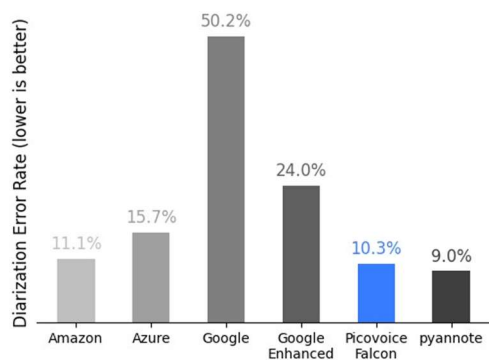
3) Косинусное сходство

$$\cos(a, b) = \frac{(a * b)}{(\|a\| * \|b\|)}$$

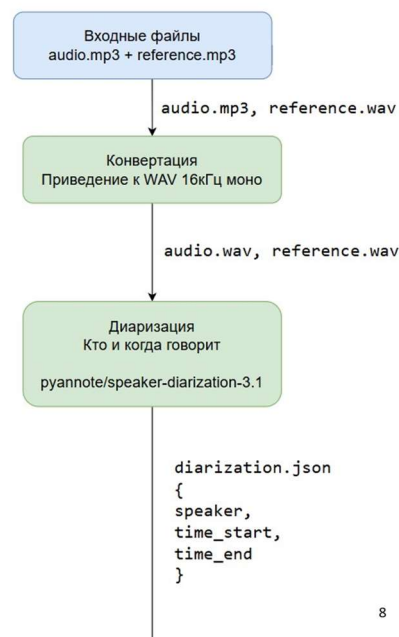


7

Сравнение моделей Диаризации

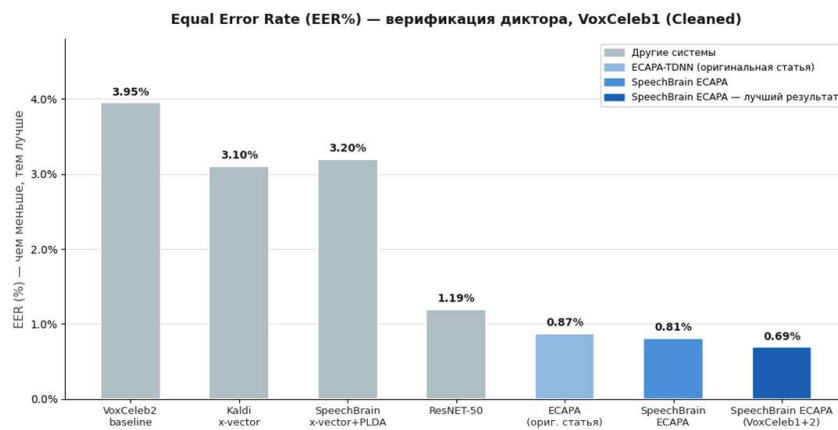


Источник:
<https://github.com/Picovoice/speaker-diarization-benchmark>

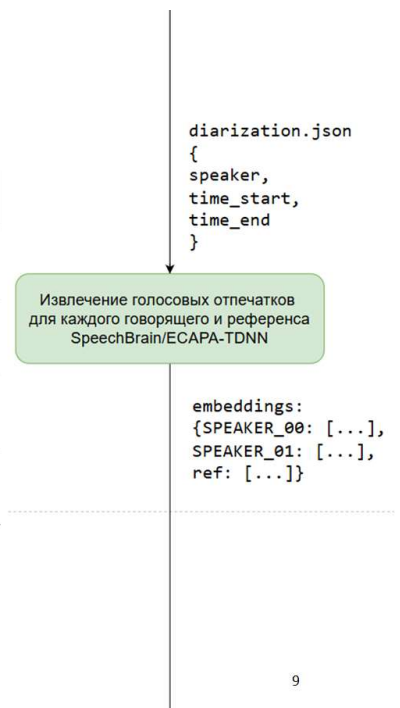


8

Сравнение моделей извлечений голосовых отпечатков



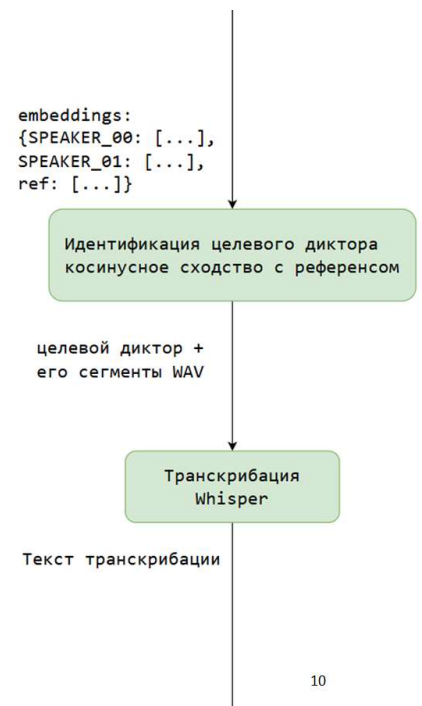
Источник:
Ravanelli et al., «SpeechBrain: A General-Purpose Speech Toolkit»,
<https://arxiv.org/pdf/2106.04624>



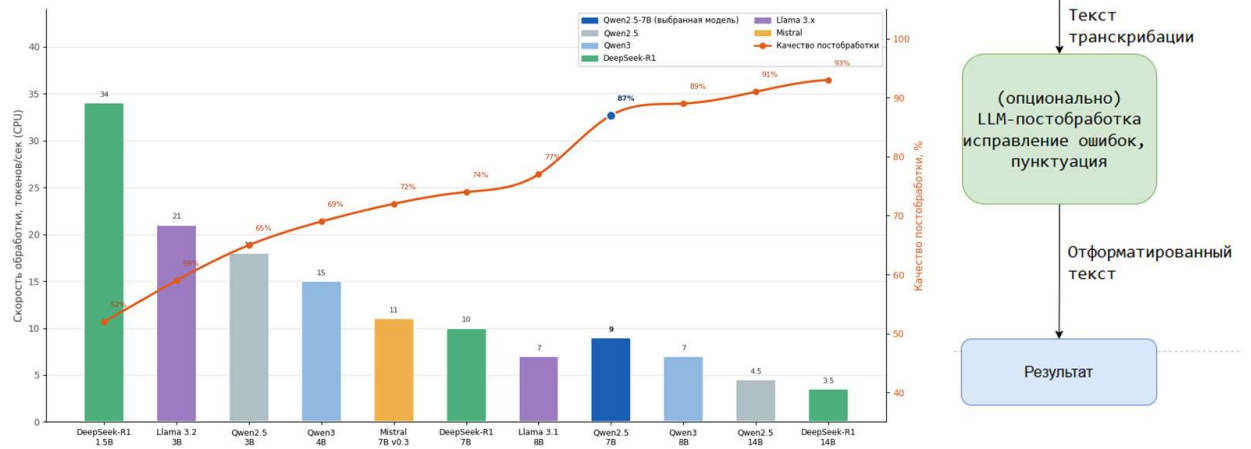
Сравнение моделей транскрибации

Модель	Точность	Языков	Требования к железу	Скорость
Whisper	Очень высокая	~99	Высокие	Медленнее
Vosk	Средняя	20+	Низкие	Быстрее
Kaldi	Высокая	Настраивается	Очень высокие	Зависит от конфига
Coqui STT	Средне-высокая	Мультиязычный	Средние	Умеренная

Источник:
Jamy.ai, «OpenAI whisper vs Other Open-Source Transcription Models»



Сравнение открытых LLM моделей



11

Сравнение результата с аналогами

Запись	Кол-во дикторов	Характер речи	Состав участников	WER Разработанного сервиса	WER Samsung	WER SpeechToText.ru	WER TurboScribe
1	2	Сильное перекрытие	M+M	31,2%	62.6%	68.8%	12.5%
2	2	Одновременная речь	M+Ж	4,5%	18.2%	40.9%	22.7%
3	2	Одновременная речь	M + Ж	55,2% 34,5%	27.6%	24.1%	10.3%
4	4	Поочерёдная речь с наложением	M + M + Ж + Ж	31,3%	31.3%	29.9%	100%

12

Пример результата транскрибации



Верный текст	Всем привет, меня зовут Никита, я работаю над задачей 1831, в ней я должен пофиксить звонок
Разработанный сервис	Всем привет! Меня зовут Никита. Я работаю над задачей №1831 и должен отсортировать список.
Samsung	Всем привет меня зовут Никита. Я.
SpeechToText	Всем привет. Меня зовут Никита.
TurboScribe	Всем привет, меня зовут Никита, я работаю над задачей 345, в ней я должен подписать звонок.

13

Что сделано:

- Проведён анализ существующих методов диаризации и распознавания речи
- Разработан алгоритм идентификации целевого диктора по голосовому образцу
- Реализован пайплайн: конвертация - диаризация - отпечатки - идентификация – транскрипция
- Проведено тестирование на реальных записях и сравнение с 3 аналогами

Результаты:

- Целевой диктор найден верно во всех записях
- Лучший WER: 4,5% - меньше чем у аналогов
- Более стабильная работа среди аналогов
- Возможность улучшения отдельных модулей
- Опубликовано статья на тему «Голосовые эмбендинги для идентификации дикторов»

14