

**ОТЧЕТ ПО ПРОЕКТУ**

**«Автоматическое распознавание речи (ASR)**

**на базе BiLSTM и CTC Loss»**

2026

## 1. Введение и постановка задачи

Цель проекта - разработка и обучение end-to-end системы автоматического распознавания речи (ASR), преобразующей аудиосигнал в текстовую последовательность.

Таблица 1 - Цель, ограничения и задачи проекта

Пункт	Описание
Цель работы	Разработать и обучить ASR-систему end-to-end на основе СТС.
Ключевое ограничение	Не использовать готовые ASR-блоки (torchaudio.models, NeMo, SpeechBrain и т.п.).
Задача 1	Реализовать архитектуру модели (препроцессинг + энкодер BiLSTM + CTC-head).
Задача 2	Реализовать декодирование: Greedy Search и Beam Search (на чистом Python).
Задача 3	Обучить модель на LibriSpeech (train-clean-100) и добиться CER < 20%.
Задача 4	Реализовать инференс на валидации и пользовательских данных, а также подсчет WER/CER.
Задача 5	Подготовить демо-ноутбук для интерактивного тестирования (Google Colab).

## 2. Обзор реализованного решения

### 2.1. Архитектура модели (BiLSTM + СТС)

В качестве входных признаков используется логарифмическая Mel-спектrogramма. Для сокращения временной размерности перед рекуррентным энкодером применяется сверточный подсемплинг (convolutional subsampling). Далее признаки подаются в многослойный двунаправленный LSTM-энкодер,

после чего линейная проекция формирует логиты по алфавиту с учетом CTC-blank токена.

Таблица 2 - Ключевые параметры архитектуры

Компонент	Значение
Входные признаки	Log Mel Spectrogram
Частота дискретизации	16 000 Гц (по конфигу)
n_mels	80 (по конфигу)
Подсемплинг	Conv2D subsampling (сокращение времени)
Энкодер	BiLSTM
Количество слоев	3 (базово) / 4 (в финальном конфиге)
Hidden size	256
Выходной слой	Linear -> LogSoftmax
Функция потерь	CTC Loss

## 2.2. Подготовка данных и аугментации

Для снижения риска переобучения на подвыборке LibriSpeech train-clean-100 применен пайплайн аугментаций. Аугментации накладываются на сигнал (waveform) и на спектrogramму, повышая устойчивость модели к вариативности темпа, частотных характеристик и пропускам фрагментов речи.

Таблица 3 - Использованные аугментации и назначение

Метод	Описание
Frequency Masking	Случайно зануляются полосы частот на спектрограмме, формируя устойчивость к спектральным искажениям.
Time Masking	Случайно зануляются временные окна, модель учится восстанавливать пропуски по контексту.
Time Stretch	Изменение скорости воспроизведения аудио (ускорение/замедление), устойчивость к темпу речи.
Цель аугментаций	Снижение переобучения и рост обобщающей способности модели.

```

%%writefile ./content/asr_project/src/utils/dataloader.py
from __future__ import annotations
from typing import Callable, Dict, List, Optional
import json

import torch
from torch.utils.data import DataLoader
from src.utils.collate import make_batch

class ASRCollator:
    def __init__(self,
                 tokenizer,
                 sr: int = 16000,
                 n_fft: int = 400,
                 hop: int = 160,
                 win: int = 400,
                 n_mels: int = 89,
                 wav_augs: Optional[List[Callable]] = None,
                 spec_augs: Optional[List[Callable]] = None,
                 return_aug_debug: bool = False,
                 aug_debug_max_items: int = 2):
        self.tokenizer = tokenizer
        self.sr = int(sr)
        self.n_fft = int(n_fft)
        self.hop = int(hop)
        self.win = int(win)
        self.n_mels = int(n_mels)
        self.wav_augs = wav_augs or []
        self.spec_augs = spec_augs or []
        self.return_aug_debug = bool(return_aug_debug)
        self.aug_debug_max_items = int(aug_debug_max_items)

    def __call__(self, batch: List[Dict]) -> dict:
        aug_items: List[Dict] = []
        # 1) Waveform aug + сохраняем wav_before / wav_after

```

Рисунок 1 - Фрагмент реализации ASRCollator и параметров препроцессинга/аугментаций

```

  del sys.modules['datasets']
  import datasets as hf_datasets

  class LibriSpeechHF(Dataset):
    def __init__(_
      self,
      split: str = "train-clean-100",
      target_sr: int = 16000,
      max_items: Optional[int] = None,
    ) -> None:
      self.target_sr = int(target_sr)

      ds = hf_datasets.load_dataset("librispeech_asr", split=split, trust_remote_code=True)

      self.ds = ds.decode(False)

      if max_items is not None:
        limit = min(int(max_items), len(ds))
        self.ds = self.ds.select(range(limit))

    def __len__(self):
      return len(self.ds)

    def __getitem__(self, idx: int) -> Dict:
      item = self.ds[idx]
      audio_dict = item["audio"]
      import io

      try:
        if isinstance(audio_dict, dict) and audio_dict.get("bytes"):
          wav, _ = librosa.load(io.BytesIO(audio_dict["bytes"])), sr=self.target_sr
        else:
          wav, _ = librosa.load(audio_dict["path"], sr=self.target_sr)
      except Exception:
        wav = np.zeros(self.target_sr, dtype=np.float32)

      wav = torch.from_numpy(wav).float()
      if wav.ndim > 1:
        wav = wav.mean(dim=0)

      text = (item.get("text", "") or "").strip().lower()
      uid = str(item.get("id", idx))
      return {"uid": uid, "wav": wav, "sr": self.target_sr, "text": text}

  Writing /content/eror_project/src/asr_datasets/librispeech_hf.py

```

Рисунок 2 - Класс датасета LibriSpeech с загрузкой данных и подготовкой аудио

### 3. Процесс обучения

Обучение выполнялось в среде Google Colab с использованием GPU. Для стабильной сходимости применены оптимизатор AdamW и планировщик OneCycleLR. В ходе экспериментов контролировались метрики CTC Loss, CER и WER на валидационной выборке, а также диагностические показатели (например, доля blank-токена).

#### 3.1. Гиперпараметры обучения

Таблица 4 - Основные параметры обучения

Параметр	Значение
Оптимизатор	AdamW
Scheduler	OneCycleLR
Learning rate	1e-3 (по конфигу)
Batch size	16 (по конфигу); подбор под память GPU
Количество эпох	50
Gradient clipping	5.0 (по конфигу)
Устройство	CUDA
Логирование	Weights & Biases (W&B)

#### 3.2. Мониторинг обучения и логирование (W&B)

Для контроля динамики обучения и анализа ошибок использовался сервис W&B. В логах фиксируются значения train/valid loss, CER/WER для Greedy и Beam, а также вспомогательные метрики.

```
wandb: [!] setting up run 0666ouhv (0.1s)
wandb: [!] setting up run 0666ouhv (0.1s)
wandb: [!] setting up run 0666ouhv (0.1s)
wandb: Tracking run with wandb version 0.23.1
wandb: Run data is saved locally in /content/asr_project/wandb/run-20260120_215244-0666ouhv
wandb: Syncing run BigModel_OneCycle
wandb: ★ View project at https://wandb.ai/olga\_tsybakova-hse/asr\_project
wandb: ★ View run at https://wandb.ai/olga\_tsybakova-hse/asr\_project/runs/0666ouhv
train e1: 0% 0/1190 [00:02<, ?it/s, blank=0, loss=20.6][2026-01-20 21:52:51,110][numexpr.utils][INFO] - NumExpr defaulting to 12 threads.
Epoch 1/50 | train_loss=5.1276 | valid_loss=3.1985 | WER_G=1.0000 | WER_B=1.0000 | CER_G=1.0000 | CER_B=1.0000
Epoch 2/50 | train_loss=2.8379 | valid_loss=2.7988 | WER_G=0.9985 | WER_B=0.9985 | CER_G=0.9988 | CER_B=0.9985
train e3: 30% 354/1190 [01:11<02:32, 5.47it/s, blank=0.813, loss=2.76]
```

---

Рисунок 3 - Пример вывода обучения и логирования метрик (W&B)

```
wandb: Run history:
*** wandb: epoch
wandb: train/blank_ratio
wandb: train/grad_norm
wandb: train/loss
wandb: train/lr
wandb: valid/avg_hyp_chars_b
wandb: valid/avg_hyp_chars_g
wandb: valid/blank_ratio
wandb: valid/cer_beam
wandb: valid/cer_greedy
wandb: +3 ...
wandb:
wandb: Run summary:
wandb: epoch 50
wandb: train/blank_ratio 0.7952
wandb: train/grad_norm 1.57427
wandb: train/loss 0.50983
wandb: train/lr 0.0
wandb: valid/avg_hyp_chars_b 144.81
wandb: valid/avg_hyp_chars_g 144.78
wandb: valid/blank_ratio 0.79089
wandb: valid/cer_beam 0.11981
wandb: valid/cer_greedy 0.11982
wandb: +3 ...
wandb:
wandb: ★ View run BigModel_OneCycle at: https://wandb.ai/olga\_tsybakova-hse/asr\_project/runs/0666ouhv
wandb: ★ View project at: https://wandb.ai/olga\_tsybakova-hse/asr\_project
wandb: Synced 5 W&B file(s), 7250 media file(s), 105 artifact file(s) and 0 other file(s)
wandb: Find logs at: ./wandb/run-20260120_215244-0666ouhv/logs
```

---

Рисунок 4 - Итоговая сводка прогона в W&B (epoch=50 и ключевые метрики)

#### 4. Алгоритмы декодирования

Поскольку выход модели представляет собой распределение вероятностей по символам на каждом временном шаге, для преобразования в текст реализованы два алгоритма декодирования: Greedy Search и Beam Search. Оба алгоритма учитывают правила СТС (удаление blank-токенов и схлопывание повторов).

Таблица 5 - Сравнение Greedy Search и Beam Search

Алгоритм	Описание
Greedy Search	Берется argmax на каждом шаге; быстро, но может давать локально-оптимальные ошибки.
Beam Search	Поддерживается к гипотез; выбирается наиболее вероятная последовательность; устойчивее на сложных фразах.
Реализация	Собственная реализация на Python (без готовых ctcdecode-библиотек).



```

Writing /content/asr_project/src/model/ctc_model.py
In [1]:
```py
#>>> %%writefile /content/asr_project/src/trainer/ctc_trainer.py
from __future__ import annotations
from typing import Dict, List, Optional

import numpy as np
import torch
import torch.nn.functional as F
import matplotlib.pyplot as plt
from tqdm import tqdm
import math
import math

from src.utils.metrics import get_cer, get_wer

def ctc_greedy_decode(log_probs: torch.Tensor, blank_id: int) -> List[int]:
    ids = torch.argmax(log_probs, dim=-1).tolist()
    out: List[int] = []
    prev = -1
    for i in ids:
        if i == blank_id and i != prev:
            out.append(i)
        prev = i
    return out

def _mask(lengths: torch.Tensor, t: int) -> torch.Tensor:
    return torch.arange_(device=lengths.device)[None, :] < lengths[:, None]

class CTCTrainer:
    def __init__(_
        self,
        net: nn.Module,
        optimizer: torch.optim.Optimizer,
        tokenizer,
        device: str = "cuda",
        blank_id: int = 0,
        use_wandb: bool = True,
        grad_clip: float = 1.0,
        log_avg_examples: bool = True,
        # Аргументы для жадного декодирования
        entropy_weight: float = 0.0,
        blank_weight: float = 0.0,
        blank_prior_target: float = 0.0,
        space_prior_weight: float = 0.0,
        space_id: int = None):
```

```

Рисунок 5 - Фрагмент реализации СТС-тренера и жадного декодирования

## 5. Результаты и оценка качества

Финальная оценка выполнялась по метрикам CER (Character Error Rate) и WER (Word Error Rate). Дополнительно анализировались примеры предсказаний по отдельным аудиофрагментам (Target/Prediction) и различия между Greedy и Beam.

Таблица 6 - Итоговые метрики на test-clean (Librispeech)

| Метрика | Greedy Search | Beam Search |
|---------|---------------|-------------|
| CER     | ~11.98%       | ~11.9%      |
| WER     | ~38.5%        | ~38.0%      |

| Ref                            | Greedy                     | Beam Search                | WER (G) | WER (B) | CER (G) | CER (B) | Audio           |
|--------------------------------|----------------------------|----------------------------|---------|---------|---------|---------|-----------------|
| 241 our virtues it is probable | i horvertse it is          | i hor vertuse it is        | 0.4688  | 0.5     | 0.1813  | 0.1703  | ▶ 00:00:00:15 ↴ |
| 242 saw a wild waste of raging | saw a wild waste of raging | saw a wild waste of raging | 0.5     | 0.5     | 0.161   | 0.1695  | ▶ 00:00:00:16 ↴ |
| 243 the brighten boys and      | the brightn boys in their  | the brighten boys in their | 0.5     | 0.5     | 0.2052  | 0.1965  | ▶ 00:00:00:12 ↴ |
| 244 and nothing so truly       | and nothing so tru the     | and nothing so tru the     | 0.3889  | 0.3889  | 0.1978  | 0.1978  | ▶ 00:00:00:12 ↴ |
| 245 and his own was unknown    | and his own was akno and   | and his own was a knoe and | 0.4     | 0.4286  | 0.1399  | 0.1451  | ▶ 00:00:00:14 ↴ |
| 246 our virtues it is probable | i horvertse it is          | i hor vertuse it is        | 0.4688  | 0.5     | 0.1813  | 0.1703  | ▶ 00:00:00:15 ↴ |
| 247 saw a wild waste of raging | saw a wild waste of raging | saw a wild waste of raging | 0.5     | 0.5     | 0.161   | 0.1695  | ▶ 00:00:00:16 ↴ |
| 248 the brighten boys and      | the brightn boys in their  | the brighten boys in their | 0.5     | 0.5     | 0.2052  | 0.1965  | ▶ 00:00:00:12 ↴ |
| 249 and nothing so truly       | and nothing so tru the     | and nothing so tru the     | 0.3889  | 0.3889  | 0.1978  | 0.1978  | ▶ 00:00:00:12 ↴ |
| 250 and his own was unknown    | and his own was akno and   | and his own was a knoe and | 0.4     | 0.4286  | 0.1399  | 0.1451  | ▶ 00:00:00:14 ↴ |

Рисунок 6 - Таблица валидации W&B: примеры распознавания и метрики (WER/CER) по аудио

## 6. Техническая реализация и воспроизводимость

Проект организован как репозиторий со скриптами обучения, инференса и оценки качества. Конфигурации вынесены в YAML, структура модулей разделяет ответственность (датасеты, модель, трейнер, утилиты).

Таблица 7 - Основные модули и назначение

| Компонент       | Назначение   |
|-----------------|--|
| run_train.py    | Запуск обучения по конфигам Hydra, сохранение чекпойнтов.                        |
| inference.py    | Инференс на выбранном датасете (libriSpeech/custom), выбор режима декодирования. |
| calc_metrics.py | Подсчет CER/WER между папками гипотез и эталонных транскрипций.                  |
| demo.ipynb      | Интерактивное демо (Google Colab), авторазвертывание, интеграция с Google Drive. |
| scripts/*       | Служебные скрипты (например, загрузка демо-весов).                               |

```

1 | import os
2 |
3 | dirs = [
4 |     "asr_project/src/config/dataset",
5 |     "asr_project/src/config/model",
6 |     "asr_project/src/datasets",
7 |     "asr_project/src/trainer",
8 |     "asr_project/src/utils",
9 |     "asr_project/src/augmentations",
10 |     "asr_project/weights",
11 |     "asr_project/predictions"
12 | ]
13 |
14 | for d in dirs:
15 |     os.makedirs(f"/content/{d}", exist_ok=True)
16 |
17 | for d in ["datasets", "model", "trainer", "utils", "text", "augmentations"]:
18 |     with open(f"/content/asr_project/src/{d}/__init__.py", "w") as f:
19 |         pass
20 |
21 | print("Структура проекта создана успешно!")
22 |
23 | --- Структура проекта создана успешно!
24 |
25 | %cd /content/asr_project
26 | !ls -l
27 |
28 | /content/asr_project
29 |
30 | import os
31 | from pathlib import Path
32 |
33 | # Known projects & collab
34 | ROOT = Path("/content/asr_project")
35 |
36 | # 1) Базовая структура (такая), но через ROOT, чтобы не было расхождений
37 | dirs = [
38 |     ROOT / "src" / "config",
39 |     ROOT / "src" / "model",
40 |     ROOT / "src" / "datasets",
41 | ]

```

Рисунок 7 - Создание базовой структуры директорий проекта в Google Colab

```

1 | !tar -czf contents.tar weights/
2 |
3 | !cd /content/asr_project
4 | !python3 scripts/download_demo_weights.py
5 |
6 | /content/asr_project
7 | [OK] Project created: /content/asr_project
8 | [OK] Downloading demo_asr_weights...
9 | [OK] URL : https://huggingface.co/facebook/mawvvec2-base-960h/resolve/main/pytorch_model.bin
10 | OUT : /content/asr_project/weights/demo_mawvvec2.bin
11 | [OK] Saved to: /content/asr_project/weights/demo_mawvvec2.bin
12 |
13 | !rm -rf /content/asr_project/src/datasets
14 |
15 | !hydrafile /content/asr_project/src/config/config.yaml
16 | defaults:
17 |   dataset: libriSpeech
18 |   model: ctc_bilstm
19 |   _self_
20 |
21 | training:
22 |   lr: 1e-3
23 |   batch_size: 16
24 |   epochs: 50
25 |   prav_citec: 0.0
26 |   device: "cuda"
27 |   use_wandb: true
28 |   run_name: "final_run"
29 |   save_dir: "weights"
30 |   log_every: 50
31 |   log_steps: 5
32 |   log_avg_examples: true
33 |   num_workers: 8
34 |
35 | Writing /content/asr_project/src/config/config.yaml
36 |
37 | !hydrafile /content/asr_project/src/config/dataset/libriSpeech.yaml
38 | .target.: src.utils.dataloader.LibriSpeechDataset
39 | .path.: "/content/datasets/libriSpeech"
40 | .target._src: 10000
41 | .max_items: 40000
42 |
43 | Writing /content/asr_project/src/config/dataset/libriSpeech.yaml
44 |
45 | !hydrafile /content/asr_project/src/config/model/ctc_bilstm.yaml
46 | n_hidden: 100
47 | n_layers: 256
48 | num_layers: 4
49 | .target.: a

```

Рисунок 8 - Автозагрузка демо-весов и формирование конфигурации (Hydra YAML)

The screenshot shows a GitHub repository page for 'speech-recognition-ctc'. At the top, it displays 'main' branch, 1 branch, 0 tags, a search bar ('Go to file'), an 'Add file' button, and a 'Code' dropdown. Below this is a list of commits by 'Olga57' (a60efc9 · 1 hour ago) with 4 commits. The commits include updates to README.md, LibriSpeech, scripts, src, weights, .gitignore, and various Python files like calc\_metrics.py, demo.ipynb, inference.py, overfit\_debug.py, requirements.txt, and run\_train.py. The last commit was an update to README.md. The README file itself is shown below, featuring a title 'ASR Project (CTC-BiLSTM)' and a description in Russian about the project being an ASR implementation using CTC and BiLSTM, trained on the LibriSpeech dataset.

| File             | Description                                  | Time Ago     |
|------------------|--|--------------|
| README.md        | Update README.md                             | 1 hour ago   |
| LibriSpeech      | Initial commit of ASR project (CTC baseline) | 19 hours ago |
| scripts          | Initial commit of ASR project (CTC baseline) | 19 hours ago |
| src              | Initial commit of ASR project (CTC baseline) | 19 hours ago |
| weights          | Add trained model weights                    | 19 hours ago |
| .gitignore       | Initial commit of ASR project (CTC baseline) | 19 hours ago |
| README.md        | Update README.md                             | 1 hour ago   |
| calc_metrics.py  | Initial commit of ASR project (CTC baseline) | 19 hours ago |
| demo.ipynb       | Add interactive demo notebook                | 19 hours ago |
| inference.py     | Initial commit of ASR project (CTC baseline) | 19 hours ago |
| overfit_debug.py | Initial commit of ASR project (CTC baseline) | 19 hours ago |
| requirements.txt | Initial commit of ASR project (CTC baseline) | 19 hours ago |
| run_train.py     | Initial commit of ASR project (CTC baseline) | 19 hours ago |

**ASR Project (CTC-BiLSTM)**

Этот репозиторий содержит реализацию системы автоматического распознавания речи (ASR) на базе СТС-модели.

Проект выполнен в рамках учебного курса. Модель обучена на датасете LibriSpeech.

Рисунок 9 - Структура репозитория проекта и основные файлы пайплайна

## 7. Заключение

В рамках проекта реализована и обучена система автоматического распознавания речи на основе СТС-подхода и BiLSTM-энкодера. Код построен модульно и соответствует требованию реализации «с нуля» на примитивах PyTorch без использования готовых ASR-решений. Достигнуто целевое качество ( $CER < 20\%$ ), реализованы два режима декодирования (Greedy и

Beam), подготовлены скрипты инференса и подсчета метрик, а также демонстрационный ноутбук для интерактивной проверки.

Полученные результаты показывают, что даже без внешней языковой модели возможно получить стабильное качество на чистой речи, однако уровень WER остается ограниченным из-за отсутствия LM и сложности редких/омофонных слов. Дальнейшее улучшение возможно за счет интеграции языковой модели в Beam Search, расширения обучающей выборки и/или перехода к более современным архитектурам (например, Conformer).

## **Список использованных источников**

1. Graves A., Fernandez S., Gomez F., Schmidhuber J. Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks. 2006.
2. Panayotov V., Chen G., Povey D., Khudanpur S. LibriSpeech: An ASR corpus based on public domain audio books. 2015.
3. Park D. S., Chan W., Zhang Y., et al. SpecAugment: A Simple Data Augmentation Method for Automatic Speech Recognition. 2019.
4. Loshchilov I., Hutter F. Decoupled Weight Decay Regularization (AdamW). 2019.
5. Smith L. N. A disciplined approach to neural network hyper-parameters: Part 1 - learning rate, batch size, momentum, and weight decay (OneCycle policy). 2018.