Final Project Summary Report: Neural Lyric Transcriber

Project Title

Neural Lyric Transcriber: End-to-End Lyrics Transcription from Mixed Music Audio Using Deep Learning

Student

Gideon Szamet

Course

Deep Learning - Technion, 2025

Submission Date

July 24, 2025

---

1. Problem Statement

The objective of this project was to build a deep learning-based pipeline capable of transcribing sung lyrics directly from full mixed audio tracks. This task differs significantly from traditional automatic speech recognition (ASR) due to the musical context: background instrumentation, overlapping vocals, varying pitch, and vocal effects complicate the transcription process. The goal was to produce accurate word-level transcription with proper alignment from unprocessed music recordings, without requiring vocal isolation.

---

2. Dataset Selection and Preparation

Dataset Overview

I chose the MUSDB18 dataset, a public, high-quality music source separation benchmark, which includes 150 professionally recorded and mixed music tracks. Each track is provided in multitrack stem format with five stereo audio

channels: vocals, drums, bass, other, and the full mixture. The dataset is published under a Creative Commons license via Zenodo.

The dataset also comes with a predefined train-test split: 100 songs are designated for training and 50 for testing, maintaining a 2:1 ratio. This consistent split supports reproducible evaluations and benchmarking across different modeling strategies.

To provide aligned lyrics, I utilized a companion lyrics extension dataset published on Zenodo (MUSDB18 lyrics extension), which includes manually aligned time-stamped transcriptions for MUSDB18 songs. This enabled an automatic generation of audio-transcript pairs suitable for supervised training.

Data Cleaning and Parsing

All processing was performed exclusively on the isolated vocal stems from MUSDB18 to simplify the input space and enable focused learning on the lyric content alone.

- Parsed metadata from MUSDB18 README.
- filtered for English tracks only.
- Extracted start/end/duration timestamps for each lyric segment.
- Added a has_lyrics flag to separate true lyrics from placeholder segments.
- Resampled audio to 16 kHz mono for consistency with model input requirements.
- Tracks were split into audio chunks according to the timing of lyric segments, based on precise time-stamped annotations from the aligned dataset.

- Generated CSVs: train_segments_chunked.csv and test_segments_chunked.csv contain full metadata for each lyric-aligned audio chunk, including the source song, start and end timestamps, aligned lyric text, chunk file path, and whether the chunk was generated via augmentation. These CSVs serve as the primary input records for all training and evaluation stages, enabling seamless loading, filtering, and batching of preprocessed audio-text pairs.
-

The original dataset included 4,251 aligned lyric segments across both training and test splits. After filtering for meaningful lyrics, 2,975 usable segments remained: 1,940 allocated to the training set and 1,035 to the test set.

Data Augmentation

To enhance the diversity of the training data and mitigate overfitting, I applied systematic audio augmentation techniques to all lyric-containing vocal segments:

- **Pitch Shifting**: Applied transformations of +2 and -2 semitones to simulate vocal variations.
- **Time Stretching**: Modified the tempo by factors of 0.9x and 1.1x to simulate different rhythmic deliveries.

Each augmented clip retained its original label and was integrated into the training set. Normalized log-Mel spectrograms were extracted for all augmented samples. This process expanded the vocal-only training corpus from approximately 1,940 lyric-containing chunks to a total of **9,700** samples, significantly improving model robustness and generalization capacity.

Exploratory Data Analysis (EDA)

I conducted EDA to understand the structure and composition of the training data:

- Segment durations: most between 3–12 seconds (mean ≈ 7.2s)
- Words per segment: mean ≈ 6.9
- Character count distribution: majority between 10 and 80 characters
- Placeholder ratio: 30.44% of segments had no lyrics
- The most common words across the lyric segments were "the," "you," "I," "and," and "to." Their high frequency across segments justified using a character-level tokenizer, as the overall vocabulary was compact and repetitive, ideal for efficient character-based modeling.

These insights informed model input design and batching strategies.

---

3. Baseline Models and Feature Construction

Input Representation

Each audio chunk was preprocessed into log-Mel spectrograms, which represent the energy of the audio signal across different frequencies over time. The preprocessing included:

- Resampling each waveform to 16 kHz mono
- Computing 80 Mel frequency bands with a 20ms hop and 64ms window
- Converting the spectrogram values to decibels for log-scale representation

These spectrograms were saved as .pt tensors with shape [80, T], where T varies with segment duration.

Labeling Strategy

Lyric transcriptions were converted to target label sequences using a character-level vocabulary. The vocabulary included lowercase English letters, a space character, and a special <blank> token used by CTC. Each lyric line was cleaned (lowercased, punctuation removed) and encoded into a sequence of character indices. Segments without valid lyrics were excluded using the has_lyrics flag.

This labeling strategy ensured compatibility with CTC-based models, which require character-level input without relying on exact timing alignments between audio and text.

Dataset and Loader

A custom PyTorch Dataset class was implemented to load spectrogram-label pairs. Labels were encoded using a vocabulary built from the lyric corpus. A custom collate_fn enabled padding of variable-length sequences, preparing inputs for Connectionist Temporal Classification (CTC) loss.

Baseline Model: CNN + CTC

The first model was a basic convolutional neural network (CNN) composed of two 2D convolutional layers with ReLU activations and batch normalization, followed by a fully connected linear layer paired with a linear output projection and CTC (Connectionist Temporal Classification) loss. CNN layers captured local time-frequency patterns from spectrograms, and the final linear layer mapped these representations to character probabilities. CTC was used to align input audio features with target transcriptions without requiring explicit time-aligned labels. It enables the model to learn when and what to emit using a special blank token, which is essential for free-length alignment in speech and lyrics. Despite this, the model lacked temporal context and failed to learn

robust alignments from mixed-audio inputs. The decoded outputs were consistently blank or non-informative.

Improved Model: CNN + BiLSTM + CTC

To better capture temporal dependencies, I introduced bidirectional LSTMs (BiLSTMs) after an enhanced convolutional stack. The updated model architecture included two 2D convolutional layers (with kernel size 3x3, ReLU activations, batch normalization, dropout, and max pooling), followed by a reshaping and flattening step before input to the RNN.

The BiLSTM component consisted of two stacked bidirectional LSTM layers with a hidden size of 256, producing a 512-dimensional output per timestep. A final fully connected linear layer projected the LSTM outputs to character logits. Xavier initialization was applied to the RNN weights for stable training.

CTC loss was used to align predictions to labels without requiring exact timing supervision. This architecture showed improved training dynamics and convergence, though its decoding quality was still limited when trained directly on mixed audio.

Enhanced Model and Training Strategy

Using the CNN + BiLSTM + CTC architecture, I implemented the following improvements:

- Three convolutional layers with dropout and pooling
- Two-layer BiLSTM with 256 hidden units
- Final linear classifier projecting to vocabulary size
- Dropout rate increased (conv: 0.3, LSTM: 0.4)
- Added a projection layer and layer normalization

Training reached an average loss of ~0.0062 after 300 epochs, with smooth convergence observed after epoch 200. Predictions showed improved phonetic accuracy and alignment.

When the model was evaluated on the held-out vocal-only test set, the results revealed clear signs of overfitting:

- Word Error Rate (WER): 1.3749
- Character Error Rate (CER): 0.9368

Despite excellent convergence on training data, the model failed to generalize to unseen vocal segments. These metrics confirm that the model was heavily overfitted and unable to produce meaningful predictions beyond the training data.

4. Transformer CTC: Wav2Vec2-Based Model

Wav2Vec2 Model Architecture

Wav2Vec2 is a Transformer-based ASR model pretrained on large-scale audio data using a contrastive learning objective. During pretraining, it learns to distinguish correct audio segments from distractors using contrastive loss — this teaches the model useful speech representations without needing transcripts. For downstream tasks like this one, the model is then fine-tuned using CTC loss, which enables it to align audio features with target text. It learns latent speech representations from raw waveforms, making it well-suited for downstream tasks with limited data.

In this project, I fine-tuned the facebook/wav2vec2-base-960h model with a custom character-level tokenizer. The model consists of:

- A convolutional feature encoder for raw waveform processing
- A Transformer encoder stack with self-attention

- A linear layer projecting encoder outputs to character logits
- Connectionist Temporal Classification (CTC) as the loss function

Beyond simply applying a pretrained model, I designed a full pipeline to integrate Wav2Vec2 into my curriculum learning strategy. This included implementing a character-level tokenizer, building a CTC-compatible collator that padded input sequences and generated attention masks, and validating the training setup using manual forward passes and custom padding logic. This setup allowed precise control over input-label alignment and training stability, facilitating systematic debugging and extension into more complex data settings.

Stability and Debugging

To overcome persistent NaN losses during training with a CTC-based transcription model, I transitioned from initializing with pretrained weights to building the Wav2Vec2ForCTC model from scratch. This allowed full control over the vocabulary configuration, CTC loss function, and final linear head.

Key implementation details:

- **Vocabulary augmentation**: I added a <ctc_blank> token at the end of the vocabulary to explicitly define the CTC blank index. The tokenizer size was updated to reflect this addition.
- **Model configuration**: A new Wav2Vec2Config object was created with the correct vocab_size, blank_token_id, pad_token_id, and forced_decoder_ids=None to ensure clean decoding.
- **Weight initialization**: The lm_head was resized and reinitialized using Xavier uniform distribution, with biases zeroed.

- **CTC loss override**: The model's internal CTCLoss layer was replaced with a new one explicitly using the correct blank index and zero_infinity=True.
- **Memory optimizations**: Set gradient_checkpointing=True, fp16=True, and disabled output_hidden_states to reduce GPU memory use during training and evaluation.

Despite many rounds of reconfiguration and debugging, the model continued to report zero loss. Upon deeper investigation, I discovered that this 'zero' was masking a deeper issue—internally, the CTC loss was NaN, which was being silently handled and replaced by zero in the loss reporting. I further traced the instability down into the network: NaN values were propagating through the hidden layers themselves. This persistent breakdown led me to conclude that given the time and computational constraints of this project, I was unable to properly configure the pretrained model. At that point, I decided to abandon the pretrained Wav2Vec2 weights and instead trained the model architecture from scratch. However, due to limited data and computational constraints, the non-pretrained version failed to converge meaningfully. The resulting model was underfitted and unable to produce valid transcriptions.

---

5. Whisper Model: Pretrained Fine-Tuning

In the final phase of the curriculum learning strategy, I initiated full fine-tuning of the Whisper model using only clean vocal audio segments extracted from the MUSDB18 training set. This stage was designed to optimize the model's ability to transcribe lyrics from high-quality vocal stems before progressing to mixed audio environments.

Whisper Architecture

Whisper is a large-scale encoder-decoder Transformer ASR model trained by OpenAI. It performs end-to-end transcription using autoregressive decoding, integrating a multilingual language model prior. For this project, I used the "small" variant.

Key components:

- Audio encoder: converts raw audio into hidden states using convolutional and attention layers
- Text decoder: generates character sequences autoregressively with beam search
- No CTC: relies entirely on sequence-to-sequence alignment

**Pre-trained model Evaluation **

Before fine-tuning, I ran Whisper inference on the unmodified vocal-only segments to establish a baseline. The model performed reasonably well considering no tuning had been applied, but the predictions were inconsistent and frequently included hallucinated tokens, particularly in silent or low-SNR segments.

Initial zero-shot results:

- Clean vocals (2,789 segments): WER = 0.456, CER = 0.332
- Mixed audio (1,940 segments): WER = 0.665, CER = 0.512

These results established Whisper as a strong baseline model, despite its occasional tendency to hallucinate short words like "you" in non-lyrical segments. Despite lower WER, hallucinated endings and repetitive artifacts were observed. Whisper tended to overgenerate common English words (e.g., "you") even when no lyric was present.

Fine-Tune key modifications and configurations applied during this phase include:

- **Label Finalization**: Each lyric label was appended with an (end-of-sentence) token. This allowed the model to learn when to terminate its predictions naturally, avoiding forced stopping during inference.
- **Language and Task Control**: To ensure consistent transcription in English and prevent Whisper's default language detection from interfering, I explicitly set language="en" and task="translate" in the Whisper processor during preprocessing.
- **Attention Mask Handling**: Enabled return_attention_mask=True and passed the attention mask directly to the model. This addressed a known issue where Whisper's pad_token_id and eos_token_id are the same, which can confuse the decoder when padding is present.
- **Checkpoint Management**: Training was configured to save checkpoints both locally and to Google Drive, ensuring progress was retained even if runtime sessions were interrupted.
- **Preliminary Validation**: A 500-step training run on a small subset using this updated configuration showed promising results. The model began to produce correct early chunks of lyrics more frequently, although hallucinated endings still occurred. This validated the impact of using the <eos> token and language/task overrides.

This clean-vocal fine-tuning phase set the foundation for future experiments on mixed-audio transcription and continues the trajectory of curriculum learning.

Whisper Model – Full Vocal-Only Fine-Tuning (Phase 2.1)

- Training duration: 2000 steps
- Best checkpoint: Step 1500

- Validation WER: 0.088
- Validation CER: 0.084
- Final training loss: 0.0249

This model accurately transcribes lyrics from clean vocal stems with minimal errors. Performance stabilized by step 1500. Additional training to step 2000 showed overfitting (WER increased to 10.93%), confirming step 1500 as optimal.

Note: In this report, WER and CER values are presented as fractional values (e.g., 0.088 = 8.8%). In the training notebooks, these same metrics are displayed as percentages. This formatting difference explains why the values shown here appear ~100× smaller than those seen in the raw logs — they are in fact equivalent. Additionally, WER and CER computed during earlier Whisper evaluations (prior to fine-tuning) were originally calculated as fractional values in the notebook. This discrepancy in formatting only applies to the fine-tuned model logs, not the zero-shot baseline evaluation.

Whisper Model – Mixed Audio Fine-Tuning

This phase continued the curriculum learning trajectory by fine-tuning Whisper further on mixed-audio segments using the best checkpoint (step 1500) from the clean vocal-only model. This gradual progression allowed the model to first learn clean vocal transcription and then adapt to more challenging audio with background instrumentation, improving robustness while preserving alignment accuracy.

- Training duration: 1100 steps (starting from checkpoint-1500 of vocal model)
- Final WER: 0.11
- Final CER: 0.095

- Final training loss: 0.0029

Despite the added complexity of full musical backgrounds, the model retained much of its transcription accuracy. WER increased by only ~2 percentage points, demonstrating effective generalization from the vocal-only phase. Most hallucinations occurred in the trailing segments of predictions.

Fine-Tuning Procedure

- Used 2,789 vocal-only segments with aligned lyrics
- Added <eos> token to labels
- Set language = "en" and task = "translate"
- Passed attention_mask to address token ambiguity
- Ran a 500-step validation to confirm configuration

Preliminary results showed promising improvement, especially in early chunk predictions. Whisper now serves as the foundation for future noise-augmented training.

---

7. Tools and Frameworks

- Python 3.10
- Google Colab (A100 GPU runtime)
- PyTorch & HuggingFace Transformers
- OpenAI Whisper
- Wav2Vec2 (facebook)
- Librosa, pydub, ffmpeg
- Seaborn, matplotlib (EDA)

8. Conclusion and Future Directions

This project demonstrates the complexity of applying deep learning to lyric transcription in musical contexts. I progressed from simple CNN-CTC models to Transformer-based architectures and finally fine-tuned Whisper using a curriculum learning approach.

Key achievements include:

- Clean preprocessing of MUSDB18 with aligned lyrics
- Functional CTC training pipeline on clean vocals
- Integration of Wav2Vec2 with custom tokenizer and collator
- Whisper baseline and fine-tuning with attention control and <eos> token
- Augmentation pipeline with pitch/time variation

Future work includes:

- Training on augmented mixed-audio inputs at varying SNR levels
- Evaluating Whisper's robustness to instrumentation
- Incorporating forced alignment or intermediate CTC layers for timestamped output
- Addressing hallucinations with decoding constraints or token filtering

Through methodical experimentation and iterative refinement, I developed a robust pipeline capable of adapting large-scale ASR models to music transcription challenges.