Final Project Individual Report

Marcus Mellor

May 06, 2025

1: Introduction

This document records the work I did individually on the BERFA filler detection system that Andrew Ash and I prepared for our final project. While much of the work we did was quite collaborative, some implementation work was done separately. We assigned tasks according to our strengths. For example, as an experienced Python programmer, I handled most of the code to interface with hardware that Andrew would have been less familiar with. And since his primary Python experience is in training neural networks with Python frameworks (gained primarily through this course), he handled tuning the training code after I set up the initial loop.

Our project was to design and implement a system capable of real-time identification of filler words in speech. This is heavily inspired by the "Ah-Counter" role from Toastmasters, an organization in which we have both participated previously. We call our design a Binary Estimator for Robust Filler Analysis, or BERFA for short. The final product far exceeded our expectations, with a consistent accuracy of nearly 90% on the test dataset and impressive real-time performance. All code is freely available at https://github.com/infinitymdm/Final-Project-1.

This document is divided into several sections. Section 2 details my contributions to the project, broken down by task. Section 3 describes experiments I performed as part of the project development process. Finally, Section 4 summarizes our results and discusses lessons learned.

2: Project Contributions

2.a: Dataset Exploration

The very first work I did on the project was to download the two datasets we were considering and take a look at the data and labels. During this process we determined that the TED-LIUM dataset, which we had initially planned to use as our primary training data, was not actually suitable for our use case as it had many mislabeled or unlabeled filler words.

```python
#! /usr/bin/env python3

import librosa
import time
import numpy as np
import sounddevice as sd
from datasets import load_dataset

tedlium = load_dataset("LIUM/tedlium", "release3")


for i in range(3000):
    sample = tedlium['train'][i]
    if 'fill' in sample['text'].lower():
        print(i)
        break

print(sample['text'])
audio = sample['audio']['array']
sr = sample['audio']['sampling_rate']
dur = librosa.get_duration(y=audio, sr=sr)
sd.play(audio, sr)
time.sleep(dur)
```

Listing 1: Initial script for loading the TED-LIUM dataset and finding a sample that contained a filler word. The transcript is printed to the display and the sample audio is played on the device's speakers.

Much of this code later evolved into the dataset and dataloader code we wrote to easily access the PodcastFillers dataset. The final version of this script, as displayed in Listing 2, contains a functional DataSet subclass and tests it with a DataLoader. This is very similar to the code actually used in the project, though it is missing some augmentation that Andrew added.

```python
class PodcastFillersDataset(torch.utils.data.Dataset):
    '''
    Custom dataset for PodcastFillers labeled wav clips

    Based on https://pytorch.org/tutorials/beginner/basics/data_tutorial.html#creating-a-custom-dataset-for-your-files
    '''

    def __init__(self, annotations_csv: str, wav_dir: str, split: str, transform=None, target_transform=relabel_fillers_bool):
        '''
        Construct the dataset

        Args:
            annotations_csv (str or path-like): path to the annotations csv file
            wav_dir (str or path-like): path to the 1-second clipped wav files
            split (str): which split to use (extra, test, train, validation)
            transform (callable): a transformation to apply to audio data (defaults to none)
            target_transform (callable): a transformation to apply to target labels (defaults to relabeling "filler/nonfiller")
        '''
        unsplit_annotations = pandas.read_csv(annotations_csv)
        self.annotations = unsplit_annotations[unsplit_annotations['clip_split_subset'] == split]
        self.wav_dir = Path(wav_dir) / split
        self.transform = transform
        self.target_transform = target_transform

    def __len__(self):
        '''
        Return the number of annotated clips in the dataset

        This is a magic method for the "len(dataset)" syntax
        '''
        return len(self.annotations)

        ...
```

Listing 2: An excerpt from the final version of the load_dataset.py script. See https://github.com/infinitymdm/Final-Project-1/blob/main/Code/scripts/load_dataset.py for the complete script.

See https://github.com/infinitymdm/Final-Project-1/issues/5 and https://github.com/infinitymdm/Final-Project-1/issues/6 for project management related to this task.

2.b: Dataset Download Script

The first work I did that directly contributed to the final product was to prepare a script for downloading and unzipping the PodcastFillers dataset. The dataset is hosted as a series of .zip files that must be combined before decompression. The code is included in Listing 3.

```
#! /usr/bin/env sh

set -e

# Fetch and assemble the dataset if we don't have the full zip
if [ ! -f "PodcastFillers-full.zip" ]; then
    # Download the files for PodcastFillers from the hosting service.
    wget https://zenodo.org/records/7121457/files/PodcastFillers.csv
    wget https://zenodo.org/records/7121457/files/PodcastFillers.z01
    wget https://zenodo.org/records/7121457/files/PodcastFillers.z02
    wget https://zenodo.org/records/7121457/files/PodcastFillers.z03
    wget https://zenodo.org/records/7121457/files/PodcastFillers.zip

    # Assemble the files into a single record
    zip -FF PodcastFillers.zip --out PodcastFillers-full.zip

    # Clean up downloaded zips
    rm PodcastFillers.*
fi

# Unpack the dataset
unzip PodcastFillers-full.zip

# Move the dataset into Code/data
mkdir ../data
mv PodcastFillers ../data/.
```

Listing 3: A bash script I wrote to make downloading the dataset easier.

While the script worked perfectly on my own computer, we ran into several small snags when we first tried to run it on an AWS instance. For example, the `zip` package was not installed, so the script would fail to load the dataset. This led to several iterative improvments, such as adding the `set -e` flag so that the script would exit on any error.

2.c: Initial Training Loop

After Andrew prepared an initial filler detector model architecture, I wrote a simple training loop (heavily based on exam code) to train it. See https://github.com/infinitymdm/Final-Project-1/pull/11 for project management and discussion related to this task. This also included setting

up the "real" version of the `FillerDetector` model architecture that Andrew later made major improvements to.

The file containing this initial training loop is far too large to include in this document. See https://github.com/infinitymdm/Final-Project-1/blob/0e78779ea206b86bd42aacec0f61bbc3ea1c2d22/Code/train.py for the code as it existed upon completion of this task.

Upon completion of this task, I was left with a trained (though not great) model that I later used to inform the design of the real-time processing code.

2.d: Real-Time Processing Infrastructure

After I set up an initial training loop, Andrew took over refining the model. I instead turned my attention to the code that would let us use the model for inference. The challenge here was that this code had to be real-time. After some reading, I decided to try using the `sounddevice` library and the `asyncio` Python module. This code ended up being much simpler than I anticipated.

```
#! /usr/bin/env python3

import asyncio
import sounddevice as sd
# See https://python-sounddevice.readthedocs.io/en/0.5.1/usage.html#callback-streams for info
# on real-time recording & playback
import torch

async def audiostream(channels=1, **kwargs):
    '''Generator yielding blocks of input audio data'''
    q_in = asyncio.Queue()
    loop = asyncio.get_event_loop()

    def callback(data, frame_count, time_info, status):
        loop.call_soon_threadsafe(q_in.put_nowait, (data.copy(), status))

    stream = sd.InputStream(callback=callback, channels=channels, **kwargs)
    with stream:
        while True:
            data, status = await q_in.get()
            yield data, status
```

Listing 4: Code for sampling an input device in real time. The function operates as a generator, yielding audio data into a queue as soon as it becomes available.

See https://github.com/infinitymdm/Final-Project-1/pull/17 for project management related to this task.

2.e: Real-Time Classifier CLI Tool

My final (and perhaps most proud) contribution to the project is the berfa.py script that we used during our presentation to demonstrate filler detection live. This script provides a fairly robust command line interface, making use of Python's `argparse` library to provide helpful feedback to the user. This script loads a specified model's weights, runs until a timeout, and allows custom thresholds to adjust how strict the classifier is. It makes use of the generator discussed in the previous section to load and classify data asynchronously. Last but not least, it provides a count of the total number of fillers detected during its runtime.

```
# Parse arguments from command line
parser = argparse.ArgumentParser(
    prog='berfa',
    description='Binary Estimator for Robust Filler Analysis'
)
parser.add_argument('model', help='path to model weights')
parser.add_argument('-t', '--threshold', default=0.5, help='classifier decision threshold (1=filler)', type=float)
parser.add_argument('-r', '--runtime', default=1e6, help='number of seconds the program should run', type=int)
args = parser.parse_args()

# Load the classifier model
torch_device = torch.device('cpu')
classifier = FillerDetector(out_dim=1)
classifier.load_state_dict(torch.load(args.model, weights_only=True, map_location=torch_device))
classifier.to(torch_device)

async def classify_loop(timeout):
    try:
        await asyncio.wait_for(classify_audiostream(classifier, torch_device, args.threshold, blocksize=16000, samplerate=16000), timeout=timeout)
    except asyncio.TimeoutError:
        print('Timeout reached. Thank you for using BERFA!')

try:
    asyncio.run(classify_loop(args.runtime))
except KeyboardInterrupt:
    print('Keyboard interrupt detected. Thank you for using BERFA!')
```

Listing 5: An excerpt from berfa.py, which was used for live demonstrations of our results. See https://github.com/infinitymdm/Final-Project-1/blob/main/Code/berfa.py for the complete file.

The script instantiates a classifier from Andrew's tuned model architecture, loads its weights from a file, and

See https://github.com/infinitymdm/Final-Project-1/issues/15 for project management related to this task.

3: Experiments and Results

Much of my experimentation was related to the practical challenges of this project. Some of the questions my experiments helped us answer included:

- Which dataset is suitable to train our classifier model?
- How do we sample and classify audio in real time?
- How should the user interact with the model for inference?

The experimentation process included a large amount of trial and error, with designs iterated until we achieved our project requirements. The results were a polished CLI and tools that make this model usable for our desired application. Figure 1 displays usage information given by the program, and Figure 2 displays a demonstration of the program using inference with a trained model to detect filler words.

```
…/final/Code  main !?  v2.7.18  23:34
❯ ./berfa.py -h
usage: berfa [-h] [-t THRESHOLD] [-r RUNTIME] model

Binary Estimator for Robust Filler Analysis

positional arguments:
  model                 path to model weights

options:
  -h, --help            show this help message and exit
  -t THRESHOLD, --threshold THRESHOLD
                        classifier decision threshold (1=filler)
  -r RUNTIME, --runtime RUNTIME
                        number of seconds the program should run
```

```
…/final/Code  main !?  v2.7.18  23:37
❯ ./berfa.py models/weight_0.5_try-thresh_0.5-0.75/model.pt -r 20
Waiting for filler words...
/home/marcus/Documents/osu/25spring/ecen5060/exams/final/Code/./berfa.py:17: Deprecation
Warning: Conversion of an array with ndim > 0 to a scalar is deprecated, and will error
in future. Ensure you extract a single element from your array before performing this op
eration. (Deprecated NumPy 1.25.)
  is_filler = float(probs) > threshold
Filler detected! (total: 1)
Filler detected! (total: 2)
Filler detected! (total: 3)
Filler detected! (total: 4)
Filler detected! (total: 5)
Filler detected! (total: 6)
Timeout reached. Thank you for using BERFA!
```

4: Summary and Conclusions

This project was a fascinating exercise in applying a deep neural network to a real-world problem. It was incredibly satisfying to see our model converge to a high accuracy, then use that model to actually do something useful that helps us improve our speech ettiquette.

There are number of improvements I would like to make if we were to continue working on this project. First, I would like to see whether fine-tuning the transfer model would result in better overall performance. Second, I would want to try augmenting our dataset to help prevent overfitting; perhaps some unsupervised clustering methods on the TED-LIUM dataset could help with this, though that may be a huge project of its own. I'd also like to continue to polish the tools used for inference, perhaps adding a graphical interface so that this is easy for speakers to use in a real-world setting.

Bibliography

[1] A. Baevski, H. Zhou, A. Mohamed, and M. Auli, “wav2vec 2.0: A Framework for Self-Supervised Learning of Speech Representations.” Accessed: Apr. 29, 2025. [Online]. Available: http://arxiv.org/abs/2006.11477

[2] F. Hernandez, V. Nguyen, S. Ghannay, N. Tomashenko, and Y. Estève, “TED-LIUM 3: Twice as Much Data and Corpus Repartition for Experiments on Speaker Adaptation,” in *Speech and Computer*, Springer International Publishing, 2018, pp. 198–208. doi: 10.1007/978-3-319-99579-3_21.

[3] G. Zhu, J.-P. Caceres, and J. Salamon, “Filler Word Detection and Classification: A Dataset and Benchmark,” in *23rd Annual Cong.~of the Int.~Speech Communication Association (INTERSPEECH)*, Incheon, Korea, Sep. 2022. [Online]. Available: https://arxiv.org/abs/2203.15135

[4] “Toastmasters Club Meeting Roles.” [Online]. Available: https://www.toastmasters.org/membership/club-meeting-roles

[5] “CrossEntropyLoss — PyTorch 2.7 documentation.” Accessed: May 05, 2025. [Online]. Available: https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html

[6] clueless, “Using weights in CrossEntropyLoss and BCELoss (PyTorch).” Accessed: Apr. 24, 2025. [Online]. Available: https://stackoverflow.com/q/67730325

[7] J. Vuurens, “Answer to "Using weights in CrossEntropyLoss and BCELoss (PyTorch)".” Accessed: Apr. 24, 2025. [Online]. Available: https://stackoverflow.com/a/67778392

[8] “BCELoss — PyTorch 2.7 documentation.” Accessed: May 05, 2025. [Online]. Available: https://pytorch.org/docs/stable/generated/torch.nn.BCELoss.html

[9] “BCELoss with class weights - vision.” Accessed: Apr. 23, 2025. [Online]. Available: https://discuss.pytorch.org/t/bceloss-with-class-weights/196991

[10] “BCEWithLogitsLoss — PyTorch 2.7 documentation.” Accessed: May 05, 2025. [Online]. Available: https://pytorch.org/docs/stable/generated/torch.nn.BCEWithLogitsLoss.html

[11] “precision_recall_curve.” Accessed: May 06, 2025. [Online]. Available: https://scikit-learn/stable/modules/generated/sklearn.metrics.precision_recall_curve.html