Automated Transcription of Tapes in Air Traffic Control

https://github.com/luaParable/CS-6320-Project/

Paul Barela

CS 6320.001

5-5-25

Introduction

Air traffic control phraseology is characterized by rapid speech, clipped transmissions, heavy static, and a highly specialized vocabulary. Generic automatic-speech-recognition (ASR) systems, therefore, degrade considerably when applied to tower, local, and en-route recordings. My objective is to demonstrate that minimal adaptation of Whisper can produce more useful transcripts than the base model if tuned properly; a secondary objective of mine is to demonstrate named entity recognition capable of automating the transcription process for air traffic control entirely.

Contributions

The entirety of the report will consist of the work of Paul Barela, the only contributor.

Requirement and Data Gathering

My original idea for this project was to fine-tune a whisper model on a collection of air traffic control recordings sourced from liveATC.net, saved as mp3 files. The gathering of the data was actually incredibly easy, liveATC does not so much as require an account to access this data. The ease stopped almost immediately, as torchaudio is incapable of decoding mp3 files. From here I discovered the ffmpeg package and created a python script to convert all of my mp3 files to wav files, which were routed to my data folder.

Later in testing I discovered issues due to the bitrates, and so I had to go back and adjust the mp3 conversion script several times, for example to ensure 16-kHz wav files for processing by whisper. I also found that whisper requires significantly shorter clips than are available in the dataset I was creating originally; whisper needs clips less than 30 seconds long, while only a few of the original hundreds of clips were in that size range. This led to the creation of chunk_creation.py, in which I split the audio files appropriately and also used pydub's split on silence to create a hugging face dataset with train/dev splits. As for pre-processing, I created augment.py to apply a series of transformations on the audio files including speed perturbation, pink noise, and random gain/loss.

This is where the development slowed considerably and my hardware started to become a problem. Chunk caching exploded in disk space, and I didn't notice until I happened to see I had only a few gigabytes left in storage on my hard drive; I had filled up over 120 gb of disk usage from the caching and I had actually just started a second caching operation after making a few changes to chunk_creation.py. After days of incrementally adjusting this and developing training_text.py to add text data to the dataset, I ran into the nail in the coffin for my dataset creation aspirations: my computer would have to successfully run non-stop for over a week just to accomplish building the large in-house corpus I had planned.

Implementation, Integration, and Training

The repository implements an end-to-end air traffic control transcription system that starts with raw audio and ends with a web-served JSON/Markdown transcript enriched with rule-based named entities. The file convert_mp3_to_wav.py standardizes any audio dropped into /input/ to 16 kHz mono WAVs. Then, chunk_creation.py then segments each WAV by silence, exports the pieces, and builds a Hugging Face Dataset with a train/dev split that is cached on disk; training_text.py shows a CPU-only experiment that labels the chunk dataset with Faster-Whisper for semi-supervised text training. When following the full pipeline, your newly created dataset allows you to utilize train_asr_public_corpus.py, which performs a fine-tune of the public Whisper-tiny model on any given corpus, freezing all parameters except the LM head and a configurable number of final decoder blocks; if you are not following the full pipeline and are instead utilizing a preexisting dataset, this is also where you would begin the process.Once a fine-tuned checkpoint resides in /model/, transcribe_md.py loads it, calls the atc_ner.py spaCy component to label ATC-specific entities, and renders a Markdown report. The same helper is wrapped by api.py, which exposes a /transcribe endpoint, converts the Markdown to JSON, and serves a static single-page interface under /ui/. The front-end lets users upload an audio file, streams it to the FastAPI backend, and visualizes the returned transcript and entity table alongside an audio player.

Testing and Error Analysis

Testing focuses on each pipeline stage's correctness and the ASR model's generalization. The preprocessing scripts verify prerequisites and raise early errors if inputs are missing. Chunking is implicitly tested by re-loading and re-saving the resulting Hugging Face dataset; any I/O or schema issues surface there. Model training uses a deterministic seed, a held-out dev split, and periodic logging every 10 steps; the trainer's metrics (loss, WER when enabled) guide early stopping and hyper-parameter tweaks. After fine-tuning, qualitative spot checks are performed through the CLI and the web UI: transcripts are compared to a baseline Whisper-tiny run (--compare flag) and entity spans are validated by visual inspection. Runtime errors caught during API calls are returned with a plain-text stack trace, while temporary files are cleaned up in a finally block to avoid leakage. Together, these checks provide coverage across audio conversion, dataset integrity, model convergence, and end-user delivery, allowing detection and correction of integration regressions.

Lessons Learned

This project was incredibly invaluable in learning to apply machine learning and natural language processing concepts; I would go as far as to say it was the most informative and involved stretch of my academic career so far. One thing that stood out to me in the lessons I learned was the difficulty in implementing named entity recognition through pattern detection. Regex has a lot of shortcomings in creating easy to understand logic in these patterns, and the whole process was much more involved than I expected.

Adjustments of parameters was another challenge during this process; while not as challenging as brainstorming the named entity recognition, it was the most time consuming portion of the project due to the huge amount of trial and error involved in tuning a model to your liking.

Conclusion

In conclusion, this project comprised of iterative development, beginning with mp3-to-wav conversion of air traffic control dialogue, proceeding to silence chunking and dataset splitting, and attempting to fully realize the dataset with training text production. Here, the storage, hardware capability, and hour requirements led to the adoption of a Hugging Face dataset based on ATCO2 produced data. Iterative development led me to adjust my training prototype to accommodate the imported dataset, which was a huge breakthrough as it allowed me to perform measurable domain adaptation while staying extremely lightweight compared to my previous attempt. The constant-block configuration allowed me (and in turn users of the program) to easily adjust the fine tuning to prioritize processing power required vs affect of the changes to the model.

Self Scoring Table – Paul Barela

Significant Exploration Beyond Baseline: I spent a huge amount of time bug fixing, detecting and resolving errors, and developing a usable product	80/80
Innovation or Creativity: I think that my initial approach of creating a dataset was potentially novel, and was fully functional; the change to a public dataset only came as a result of hardware limitations	30/30
Highlighted Complexity: I think my potentially novel approach to data gathering and very involved preprocessing process was the highlight of complexity in this project. Additionally, I utilized concepts and programming skills learned in both this course and in my machine learning course to build the fine-tuning functionality, adjust learning rates appropriately, and make decisions regarding layers, batch size, training/devset splitting, etc.	10/10
Discussion of Lessons Learned and Potential Improvements: I believe I covered this well in my report.	10/10
Exceptional Visualization/Diagrams/Repo: My css and styling for the front-end of the program is certainly not impressive, but I believe my repository organization, changelog, and readme are worthy of some credit in regards to having an exceptional repo at the very least.	5/10
Discussion of Testing Outside of the Team: In brainstorming the functionality of this program, I called upon the air traffic experience not just of myself, but also of my wife (another air traffic controller) and several friends that remain in the career field. Their ideas were essential in the development of the named entity recognition logic, and they provided invaluable feedback.	10/10
Earned Money: I did not earn money with this project.	0/10
Total	145/160, 90.6%

Works Cited

[1] J. Tol, “atc-dataset,” *Huggingface.co*, Oct. 09, 2024.

https://huggingface.co/datasets/jacktol/atc-dataset (accessed Apr 17, 2025).

[2] LuigiSaetta, “atco2_atcosim,” *Huggingface.co*, May 12, 2023.

https://huggingface.co/datasets/luigisaetta/atco2_atcosim (accessed May 06, 2025).

[3] “Data available from the project,” *atco2.org*. https://www.atco2.org/data