

Working with AI models at the US Air
Force Academy : from image
generation, through speech-to-text
models, and building an interface

Internship report

NED

SLT Thibaut Hazebroucq, EA22

1. Table of contents

1.	Table of contents	2
2.	Acknowledgements	4
3.	Introduction	6
4.	Discovery Phase	7
a.	Introduction to Theory	7
b.	Tools Installation, Visual Code Setup, SSH Connection, and Linux Commands	10
c.	Using Initial Models for Image or Short Video Generation	11
d.	Using the Integrated API on HuggingFace - StableDiffusion.....	13
e.	Using Gradio for Creating Simple Interfaces.	13
5.	Using Audio Transcription Models and Searching for Result Improvements.....	15
f.	Retrieving MP3 Samples	15
g.	Creating an Interface for Model Comparison	16
h.	Using LLM to Correct or Summarize Transcriptions.....	17
i.	Transcription of Noise-Reduced Versions	18
j.	Filtering Sample Attempts	19
6.	Implementing an Interface for live transcriptions of aeronautical communications 21	
k.	Reasons for Implementing the Interface Instead of Continuing Research	21
l.	Preliminary Study	22
m.	Live Audio Stream Transcription Script	23
n.	Main Script: Starting the Web Interface and Managing Sub-processes	25
o.	Description of the web interface and its functionalities	26
p.	Proof of concept : development choices	28
q.	Improvements and Additions	29
r.	Discussions and outlook	31
7.	Conclusion and Outlook	32

2. Acknowledgements

I would like to express my deepest gratitude to all the people who contributed to the completion of this thesis and enriched my professional experience during this internship.

First of all, I would like to thank both the École de l'Air et de l'Espace and the US Air Force Academy for offering me this exceptional internship opportunity. It was a formative and stimulating experience that allowed me to continue the guided learning of this year independently, to discover and experiment with the latest advancements in AI using the technical resources provided by the US Air Force Academy.

I extend my sincere thanks to COL. Mayer, Dr. Mello, and the entire staff of the department. They welcomed me into their department, where I was able to work surrounded by a healthy intellectual environment amidst the students of the US Air Force Academy and their interesting projects. Dr. Mello supported me from the very beginning while granting me great autonomy, always available to discuss, review my experiments, or advise me through our exchanges.

I would also like to thank Professors Barache and Bartheye, who have been a source of inspiration and motivation through their pedagogy throughout the school year, instilling dynamism and motivation in exciting subjects. Although I did not interact with them during the internship, I wish to acknowledge them here.

Mr. Buisson provided valuable advice through emails and our video call, and I am grateful to him.

I also want to acknowledge Thomas Gallard for his technical assistance, particularly in configuring server access and for his insightful advice on programming and using software tools.

Thank you all for your contribution and support. This thesis is the result of your combined efforts and dedication.

3. Introduction

The subject of this internship was to explore the use of Large Language Models (LLM) and AI models in the field of defense. Initially, I had the freedom to explore, test, and familiarize myself with various available AI models. The detailed steps of my research and experiments will be discussed later in this report. After my initial experiments and observing the impressive results achieved by my colleague Thomas Gallard with LLMs, I decided to focus on another type of model, specifically those that enable the transcription of audio files, particularly in the aviation field.

My goal was to develop an interface for an application that could be used for real-time strategic and informative monitoring, as well as for the transfer of classified information within operational systems. These innovative systems greatly inspire me. Combining this type of tool with LLM systems presents a significant interest within the defense forces, allowing for intelligent real-time monitoring of our territory or other areas of interest.

This thesis presents the work carried out during my internship, divided into three main parts: the discovery phase, the use of audio transcription models, and the search for improvements in the results, and finally, the creation of an interface. Each of these sections details the objectives, methods, results obtained, and challenges encountered.

The objective of this internship was not only to familiarize myself with AI technologies but also to produce a concrete tool that could be used in practical applications, particularly in the field of aviation communications. This thesis reflects this approach, highlighting the learnings and achievements obtained during this period.

4. Discovery Phase

a. Introduction to Theory

As an introduction, I had the opportunity to discuss with Dr. Mello, my mentor at the US Air Force Academy. This gave me an initial glimpse into the "gap" between the very basic knowledge I learned in AI at the École de l'Air and what is currently being done.

To recap, we used several models such as k-nearest neighbors, decision trees, and regression models to try to match data with predicted results. The simplest example being Fisher's Iris dataset, which I will not revisit here but will provide my previous work if needed.

[illegible]

Between predicting an output value based on multiple input values and generating images from simple text, there are significant intermediate steps.

Here are some explanations of these steps, simplified and translated from my initial conversation with Dr. Mello:

Convolutional Models:

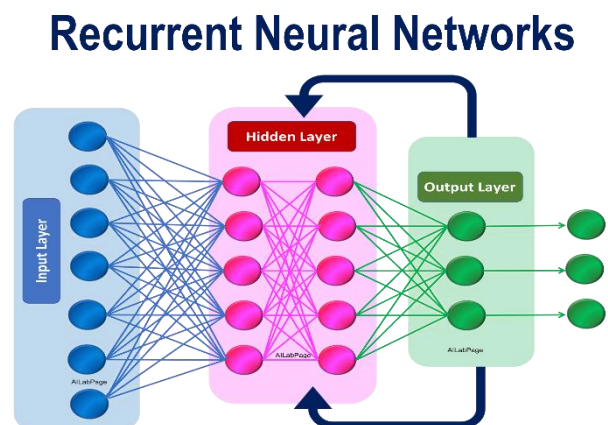
These are similar to convolution matrices in image processing, where the neural network itself calculates the convolution matrix. As seen in AI courses, this can lead to limitations such as overfitting depending on the training data provided, like an orange being perceived as a helicopter.

(Reference: <https://www.science-et-vie.com/article-magazine/i-a-la-faille-inattendue>)



RNN - Recursives Neural Networks :

The output of these networks is looped back into their input. They can, for example, predict several moves ahead in chess by reintegrating the final/predicted situation as the input situation. The principle of recurrent neural networks, as their name suggests, is to loop the output back as the input state, allowing it to make a new projection/prediction for the next step. Another example is a vehicle following a path on the ground represented by successive white rectangles: A simple CNN would identify the pattern each time and determine a direction to move the vehicle. But an RNN could predict behavior in the coming seconds and make a smoother turn, for instance, by predicting a curve if it has learned from a sequence of images. These images must be ordered sequentially for the model to understand the sequence/order as an input variable.



Transformers :

While RNNs allow following a certain logic over multiple predictions, they can still lose the thread, whereas a transformer aims to "contain" the predictions within a certain "volume." The attention mechanism and transformers are key concepts in artificial intelligence, especially in natural language processing (NLP).

Attention Mechanism:

The attention mechanism allows a model to focus on different parts of the input when producing an output. It assigns weights to different parts of the input based on their relevance to the task. For example, when a translation model processes a sentence, the attention mechanism allows it to focus on the most important words to produce a correct translation.

Transformers :

Transformers are neural network architectures based on the attention mechanism. They consist of multiple layers that process inputs and produce outputs sequentially. The main components of a transformer are:

- **Multi-head attention:** Multiple attention mechanisms work in parallel to capture different relationships in the data.
- **Feed-forward networks:** Simple neural networks that process data after attention.
- **Residual connections:** Residual connections help retain information between layers to facilitate learning.

More clearly, if you compare it to a discussion, here's the effect of a transformer:

Case 1 :

« What's the weather like ? »

Answer = local weather

Case 2 :

« I want to go on vacation to Dubai. How much does a ticket cost? »

Answer = ticket price

« And what's the weather like ? »

non-transformer Answer = local weather

transformer Answer = Dubai's weather

b. Tools Installation, Visual Code Setup, SSH Connection, and Linux Commands

After a first week of adaptation, where I settled in and practiced some exercises from Aurelien Geron's book, "Hands-On Machine Learning," the second week of my internship was dedicated to installing and configuring the necessary tools for my work. I familiarized myself with Visual Code and learned to connect to remote servers via SSH with the help of Thomas Gallard. Setting up the development environments on the Linux servers involved several steps:

Some plugins were necessary to connect and develop remotely from Visual Studio. (details in the appendix)

Once again, Thomas, who had participated beforehand in configuring the server, helped me with problems he had already encountered or not. For example, we were denied access to the server due to a full disk. We had to investigate to determine the origin of the problem. The necessary steps included:

- Running a Linux command to study disk usage and directory distribution: This helped us identify large files and directories that were taking up too much space.
- Changing the configuration file during SSH setup: We redefined the library installation directory to a different disk than the home folder to have more space.
- Creating and activating Python virtual environments: We set up development bubbles specific to each project with only the necessary libraries for those tests instead of a single heavy environment containing libraries for all projects. This approach is advantageous as it isolates dependencies, although it sometimes presents risks if a project requires a specific and older version of a library.
- Changing directory permissions: This ensured that the virtual environment could write to the directories.

Later, I also developed locally from my Windows machine without using the LINUX server. I had to use the commands to create and activate the virtual environment from the command line on Windows, while having to circumvent certain security rules, not having found another functional solution.

c. Using Initial Models for Image or Short Video Generation

The initial tests involved using models for generating images and short videos. Thanks to the fast generation capabilities of the USAFA servers, I experimented with several models available on HuggingFace. The initial results allowed me to familiarize myself with the capabilities and limitations of these models.

I started by getting acquainted with HuggingFace models, particularly by using pipelines after installing the necessary libraries, including diffusers and torch.



AnimateDiff could generate less realistic animations than LCM (closer to animated films or drawings), likely due to the type of training and the images used for training. AnimateLCM produced more realistic results.

I also tested other models like epiCRealism and Hotshot-XL. The results varied depending on the prompts and parameters used.

To push my experiments further, I tried to improve the resolution or duration of the animations, but without positive results. Specifying a resolution in the prompt did not change the output. I experimented with parameters such as `guidance_scale` and `num_inference_steps`, which influenced the generation time, but I couldn't create actual video clips showing different scenes. The models can depict and derive a subject but cannot create a logical sequence that would, for instance, illustrate a news broadcast clip from a simple text/prompt describing a current event.

It might be necessary to use LLMs to create and generate different text excerpts describing successive scenes to then feed a video-generating AI multiple times.

Here is an example of the code used to test these models:

```
# AnimateDiff.py script excerpt

import huggingface_hub
from transformers import AnimateDiffPipeline

pipeline = AnimateDiffPipeline.from_pretrained("huggingface/animate-diff")

prompt = "A dog running in a park"
animation = pipeline(prompt)

animation.save("dog_running.mp4")

# AnimateLCM.py script excerpt

import huggingface_hub
from transformers import AnimateLCPipeline

pipeline = AnimateLCPipeline.from_pretrained("huggingface/animate-lcm")

prompt = "A cat playing with a ball"
animation = pipeline(prompt)

animation.save("cat_playing.mp4")
```

These experiments helped me understand the steps needed to prepare, execute, and evaluate animation generation models. I also learned how to adjust prompts and parameters to optimize the results obtained.

What impressed me the most was the simplicity of use. In a single line, you load the pre-trained model and with a simple prompt, you get a text-to-image conversion.

d. Using the Integrated API on HuggingFace - StableDiffusion

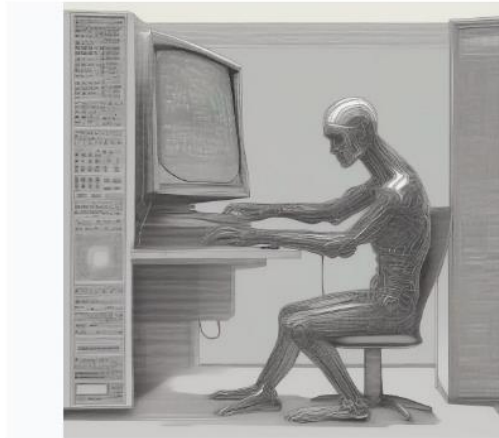
🔥 Inference API ⓘ

🖼️ Text-to-Image

a computer programming a human

Compute

Computation time on gpu: 6.724 s



✕ I also discovered Stable Diffusion for pure image generation, rather than animation. I continued my tests with Stable Diffusion, obtaining results that were consistent with the prompts, much more so than with video generation models when it comes to requesting unusual things (for example: a cat chasing a mouse while breathing through an inhaler). I was also able to adjust the image resolution using the HuggingFace interface, unlike with video generations.

e. Using Gradio for Creating Simple Interfaces.

The downside of these scripts is having to reload the model every time you want to change the prompt. One solution is to use a Jupyter Notebook (format .ipynb), which allows the code to be divided into blocks and executed separately without reloading everything.

```
Exploite_liste.ipynb ✕
Donnees serveur > saveAIMODELES > AUDIO > Exploite_liste.ipynb > ...
+ Code + Markdown | ▶ Run All | 🗑️ Clear All Outputs | 📖 Outline ...
Select Kernel

SHOW_WEBSITE = False
Reset_all_improved_files = True

numbers = {}

numbers["INITIAL_displayed_lines"] = 2
numbers["INITIAL_starting_line"] = 0
numbers["INITIAL_stoping_line"] = numbers["INITIAL_starting_line"] + numbers["INITIAL_displayed_lines"]

Python

print("""
#-0-##### Loading : Imports #####
""")

#Génériques
import os, glob, random, csv, time, pygame, warnings, copy
from tabulate import tabulate
import requests
import io
from io import BytesIO

from pydub.playback import play

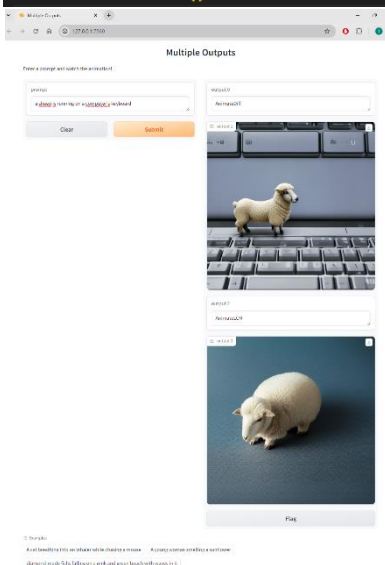
from scipy.signal import find_peaks

import gradio as gr
```

Another solution, similar to current practices, is to have a graphical interface via a web page to rerun the script with a prompt. An easy-to-use tool for this is Gradio.

```
# Création de l'interface utilisateur avec Gradio
interface = gr.Interface(
    fn=generate_animations,
    inputs="text",
    outputs=["text", "image", "text", "image"], # Le modèle renvoie un fichier GIF, donc nous définissons le t
    title='Multiple Outputs',
    description="Enter a prompt and watch the animation!",
    examples=
    [
        "A cat breathing into an inhaler while chasing a mouse",
        "A young woman smelling a sunflower",
        "diamond-made fishs falling on a pink and green beach with waves in it"],
)

# Lancement de l'interface utilisateur
interface.launch()
```



Here is a simple example that allows you to create a web interface with a prompt and the output of two different models. Another model, i2vgen-xl, can generate a short video from an image and a prompt.

You could imagine a successive "pipeline" of a model that generates an image, which is then transformed into a video, or even described by an AI to generate an action prompt with a language model.

5. Using Audio Transcription Models and Searching for Result Improvements

After conducting other tests trying to perform live speech recognition, I focused on one of the initial topics of the internship: the transcription of aeronautical communications. Searching on HuggingFace, only these four models seem specialized in recognizing aeronautical communications:

- Jzuluaga/wav2vec2-large-960h-lv60-self-en-atc-atcosim
- facebook/wav2vec2-base-960h
- scy0208/whisper-aviation-base
- billodal/whisper-small-atc

f. Retrieving MP3 Samples

A site, liveATC.net, provides downloadable samples of aeronautical communications in MP3 format, which can serve as a basis for testing specialized transcription models. After attempting to automate the script using datamining, I realized it would take too long to develop and would be off-topic for the internship. By displaying the page's source code, I was able to transform the HTML table into a CSV file by changing tags to semicolons to list all the available audio files (file liste_liveATC_use.csv).

LiveATC.net
Live Air Traffic - From Their Headsets to You.

Report/ARTCC Code

Frequency Search

Frequency Search

Frequency Search

Frequency Search

Frequency Search

Frequency Search

Frequency Search

Frequency Search

Frequency Search

Frequency Search

Frequency Search

Frequency Search

Frequency Search

Frequency Search

Frequency Search

Frequency Search

Frequency Search

Frequency Search

Frequency Search

Frequency Search

Frequency Search

Frequency Search

Frequency Search

Frequency Search

Frequency Search

Frequency Search

Frequency Search

Frequency Search

Frequency Search

Frequency Search

Archived LiveATC Recordings

These recordings have been captured from a variety of sources, primarily LiveATC audio streams. You can post your own in the ATC/Airline Audio Clip Forum (requires forum registration).

NOTE TO PRESE: Please use LiveATC.net as audio source.

Time (GMT)	Subject	Member Name
2024-05-30 02:09:26	Diamond DA40 lands on Hwy 501 near Hyattsville Beach	glimmers
2024-05-29 19:01:22	Southwest Airlines smoke in the cabin Colorado Springs	ts4462
2024-05-29 19:46:00	Small plane makes hard landing at RDU due to nose gear malfunction	ts4462
2024-05-29 19:22:06	F35B Crashes Near Albuquerque Airport New Mexico	ts4462
2024-05-28 19:49:22	United 8th main engine fire on takeoff Chicago	ts4462
2024-05-28 17:00:41	Wing and tail strike, Livermore CA	ts4462
2024-05-28 15:34:40	De Havilland Twin Otter power lost lands outside Fairbanks AK	ts4462
2024-05-28 15:01:57	Canopy Door flies off Charleston	ts4462
2024-05-28 14:14:30	Nose Gear collapse Savannah GA	ts4462
2024-05-24 19:11:47	Skywest Flight diverts to KGRB due to engine flame out	ts4462
2024-05-24 12:06:11	Plane lands on road near Colorado Springs	ts4462
2024-05-23 12:22:55	Boeing 737 MAX 8 lands on runway	ts4462
2024-05-22 10:21:43	Small plane overshoot runway before hitting travel trailer in north Peoria	ts4462
2024-05-21 16:17:52	Small plane crash at Hondo Airport no brakes	ts4462
2024-05-20 13:05:36	Wheel separation on landing SAN LUIS OBISPO	ts4462
2024-05-17 13:52:54	Cessna lands on road near Yuma AZ	ts4462
2024-05-16 18:18:42	Wing stall after nose gear crash in Williamson County	ts4462
2024-05-15 14:33:00	UPS 747-400 Flight Control Issues Returns to KDMX	ts4462
2024-05-10 12:34:24	Sheared off nose gear Sacramento Exec	ts4462
2024-05-10 10:08:28	Plane makes emergency landing near Puerto Tequillo	ts4462
2024-05-10 01:50:15	DALLAS: Nose Gear Problem Descending into KATL	ts4462
2024-05-10 01:31:40	Re: Orlando, rejected takeoff, conflicting ATC clearing takeoff and revy crossing	ts4462
2024-05-09 16:41:26	Orlando, rejected takeoff, conflicting ATC clearing takeoff and revy crossing	ts4462
2024-05-09 02:31:47	Re: SAC Executive - Porpoise landing gear failure	ts4462
2024-05-08 10:58:06	Plane crashes along side of Apopka road	ts4462
2024-05-08 14:03:06	Gear up landing Punta Gorda	ts4462



Time (GMT)	Subject	Member Name
2024-05-30 02:09:26	Diamond DA40 lands on Hwy 501 near Hyattsville Beach	glimmers
2024-05-29 19:01:22	Southwest Airlines smoke in the cabin Colorado Springs	ts4462
2024-05-29 19:46:00	Small plane makes hard landing at RDU due to nose gear malfunction	ts4462
2024-05-29 19:22:06	F35B Crashes Near Albuquerque Airport New Mexico	ts4462
2024-05-28 19:49:22	United 8th main engine fire on takeoff Chicago	ts4462
2024-05-28 17:00:41	Wing and tail strike, Livermore CA	ts4462
2024-05-28 15:34:40	De Havilland Twin Otter power lost lands outside Fairbanks AK	ts4462
2024-05-28 15:01:57	Canopy Door flies off Charleston	ts4462
2024-05-28 14:14:30	Nose Gear collapse Savannah GA	ts4462
2024-05-24 19:11:47	Skywest Flight diverts to KGRB due to engine flame out	ts4462
2024-05-24 12:06:11	Plane lands on road near Colorado Springs	ts4462
2024-05-23 12:22:55	Boeing 737 MAX 8 lands on runway	ts4462
2024-05-22 10:21:43	Small plane overshoot runway before hitting travel trailer in north Peoria	ts4462
2024-05-21 16:17:52	Small plane crash at Hondo Airport no brakes	ts4462
2024-05-20 13:05:36	Wheel separation on landing SAN LUIS OBISPO	ts4462
2024-05-17 13:52:54	Cessna lands on road near Yuma AZ	ts4462
2024-05-16 18:18:42	Wing stall after nose gear crash in Williamson County	ts4462
2024-05-15 14:33:00	UPS 747-400 Flight Control Issues Returns to KDMX	ts4462
2024-05-10 12:34:24	Sheared off nose gear Sacramento Exec	ts4462
2024-05-10 10:08:28	Plane makes emergency landing near Puerto Tequillo	ts4462
2024-05-10 01:50:15	DALLAS: Nose Gear Problem Descending into KATL	ts4462
2024-05-10 01:31:40	Re: Orlando, rejected takeoff, conflicting ATC clearing takeoff and revy crossing	ts4462
2024-05-09 16:41:26	Orlando, rejected takeoff, conflicting ATC clearing takeoff and revy crossing	ts4462
2024-05-09 02:31:47	Re: SAC Executive - Porpoise landing gear failure	ts4462
2024-05-08 10:58:06	Plane crashes along side of Apopka road	ts4462
2024-05-08 14:03:06	Gear up landing Punta Gorda	ts4462

The goal was then to be able to download samples “on the go”, when any script requires it to test the effectiveness of the audio transcription models.

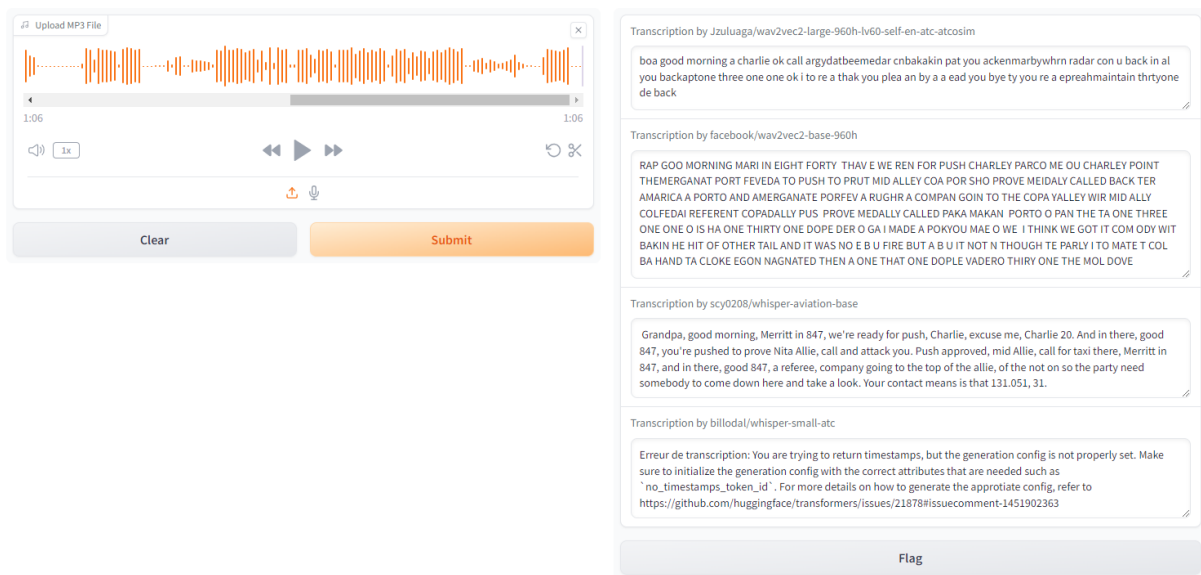
g. Creating an Interface for Model Comparison

A Gradio interface allowed me to compare the results of four different transcription models.

The following work uses the script "Audio/Exploit_Data_list.ipynb" and "Audio/Improved_speech_recognition.ipynb".

Here is a simplified version of the script "Audio/Exploit_Data_list.ipynb":

- Loading the CSV file
- Downloading the first samples, the number being set by the user (the script checks for the presence of the file to avoid re-downloading it)
- Each sample is then transcribed through the four models
- Starting the Gradio interface, which allows listening to each audio sample in the different versions and viewing the transcriptions of the different versions



It quickly becomes apparent that two of the models produce inconsistent results on a large number of samples, while the fourth encounters a recurring error. My attempts to resolve these two problems were unsuccessful.

h. Using LLM to Correct or Summarize Transcriptions


The third model, "whisper-aviation-base", gives the best results. However, inconsistencies persist. Two approaches will be attempted: correcting transcription inconsistencies with an LLM or improving transcription through sound quality enhancement.

The samples are downloaded with a short description, so successfully retrieving this summary would be a significant achievement to potentially automatically categorize audio samples by summarizing their content. The idea is to use an LLM, BART in this case, to attempt to summarize the transcription.

Audio Transcription Service, Specialized in Aeronautical Speech

Upload an MP3 file to get its transcription using various fine-tuned models.

Upload MP3 File



0:001:06

<|> 3x

⏮ ⏪ ⏩ ⏭

🔊 🔇

ClearSubmit

Transcription by Jzuluaga/wav2vec2-large-960h-lv60-self-en-atc-atcosim

boa good morning a charlie ok call argydatbeemedar cnbakakin pat you ackenmarbywhrn radar con u back in al you backptone three one one ok i to re a thank you plea an by a a ead you bye ty you re a epreahmaintain thrityone de back

Summarized (BERT) of Jzuluaga/wav2vec2-large-960h-lv60-self-en-atc-atcosim

Boa: Good morning a charlie ok call argydatbeemedar cnbakakin pat you ackenmarbywhrn radar con u back in al you backptone three one one ok i to re a

Transcription by facebook/wav2vec2-base-960h

RAP GOO MORNING MARI IN EIGHT FORTY THAV E WE REN FOR PUSH CHARLEY PARCO ME OU CHARLEY POINT THEMERGANAT PORT FEVEDA TO PUSH TO PRUT MID ALLEY COA POR SHO PROVE MEIDALY CALLED BACK TER AMARICA A PORTO AND AMERGANATE PORFEV A RUGHR A COMPAN GOIN TO THE COPA YALLEY WIR MID ALLY COLFEDAI REFERENT COPADALLY PUS PROVE MEDALLY CALLED PAKA MAKAN PORTO O PAN THE TA ONE THREE ONE ONE O IS HA ONE THIRTY ONE DOPE DER O GA I MADE A POKYOU MAE O WE I THINK WE GOT IT COM ODY WIT BAKIN HE HIT OF OTHER TAIL AND IT WAS NO E B U FIRE BUT A B U IT NOT N THOUGH TE PARLY I TO MATE T COL BA HAND TA CLOKE EGON NAGNATED THEN A ONE THAT ONE DOPE VADERO THIRY ONE THE MOL DOVE

Summarized (BERT) of facebook/wav2vec2-base-960h

Rapper Mari hits the streets of New York City in search of a hit song. The song is called "RAP GOO MORNING MARI IN EIGHT FORTY THAV"

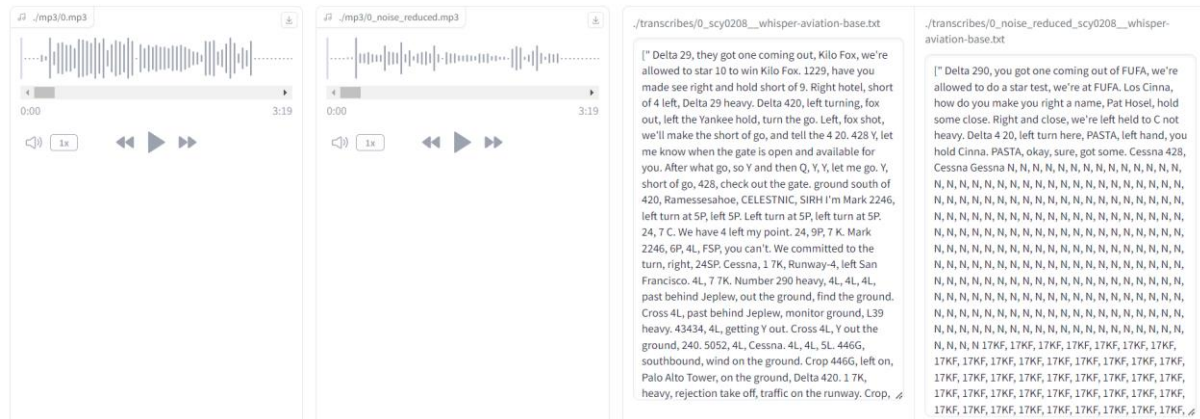
Transcription by scy0208/whisper-aviation-base

Grandpa, good morning, Merritt in 847, we're ready for push, Charlie, excuse me, Charlie 20. And in there, good 847, you're pushed to prove Nita Allie, call and attack you. Push approved, mid Allie, call for taxi there, Merritt in 847. and in there. good 847. a referee. commany going to the top of the allie. of the not on so the narty need

The results are not promising, even when modifying the prompt to explain that it is aeronautical communications.

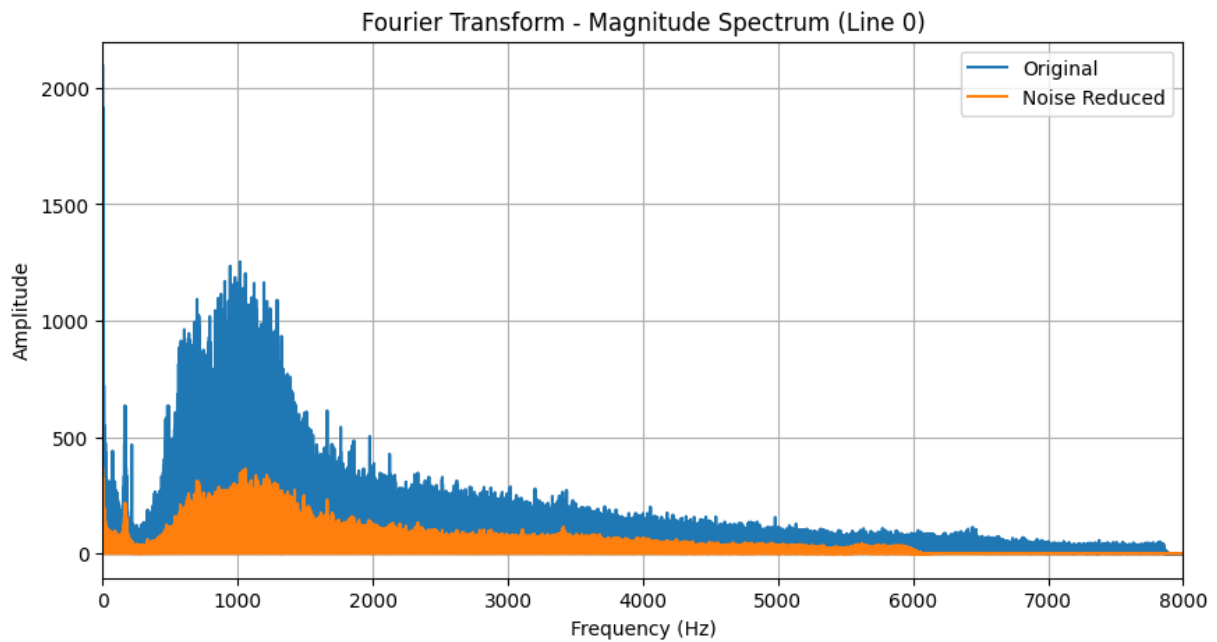
i. Transcription of Noise-Reduced Versions

One of the first attempts to improve transcription focused on the third model, whisper-aviation-base. Its results being relatively good, an improvement in sound quality could translate to an improvement in transcription (although this is not guaranteed: if the model was trained with degraded quality samples, it will not perform well on crystal clear samples). A "noisereducer" library is available on Python, so I modified the script to create a second version of the MP3 file following this treatment.

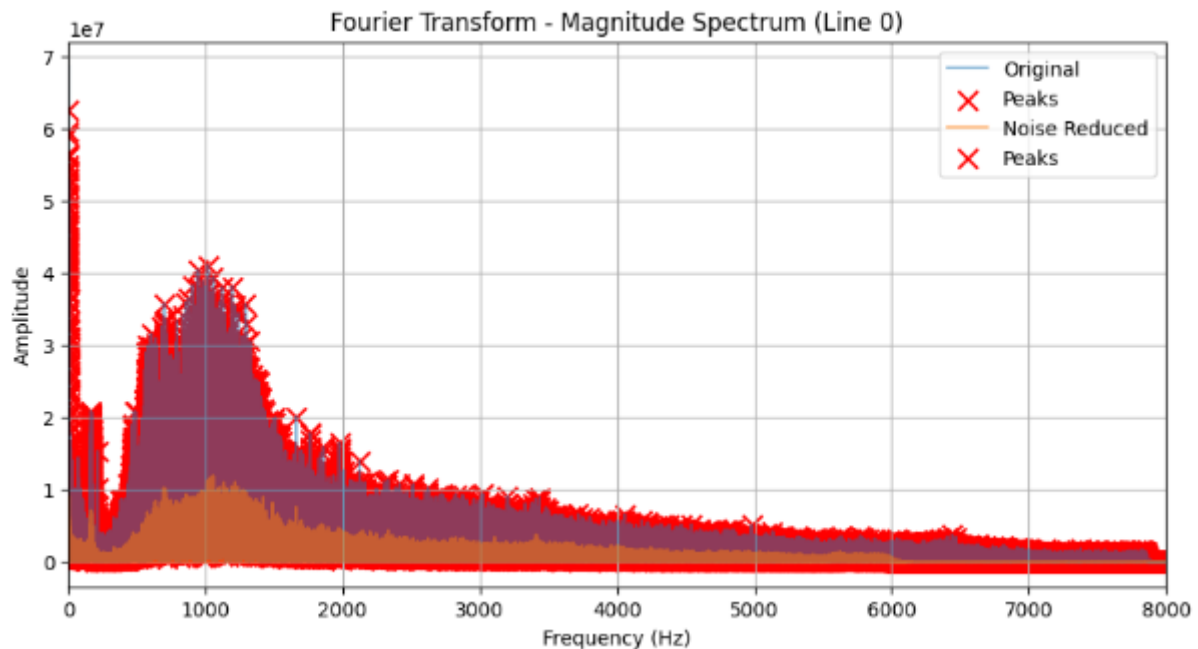


The transcriptions are not better, sometimes worse. Another approach would be to filter the audio samples by applying a band-pass filter adapted to the human timbre, in addition or as a replacement for noise reduction.

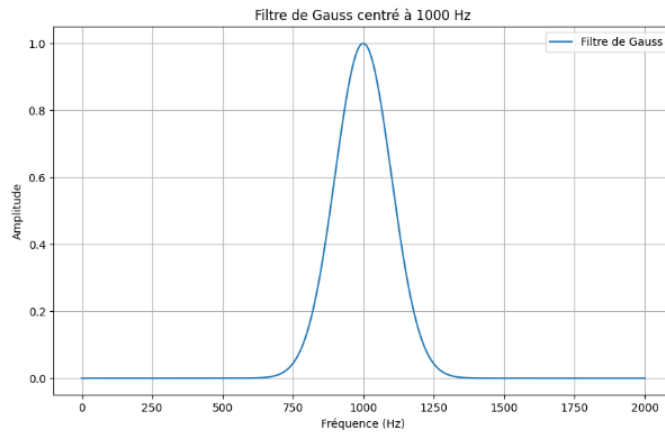
j. Filtering Sample Attempts



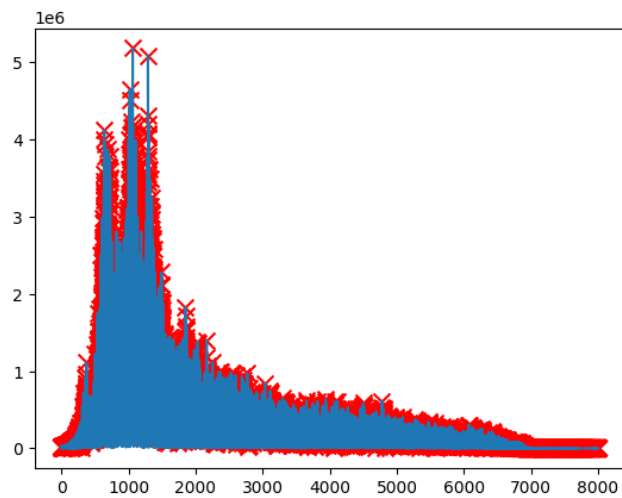
Here, we see the frequency spectrum of the first downloaded sample. A Fourier transform is applied to visualize the amplitude of the frequencies. My attempts using a Butterworth filter were not conclusive (strong voice modulation). I wanted to create a filter by detecting the speaker's voice by detecting the frequency peaks. The scipy library offers a "find_peaks" function.



However, I couldn't adjust the parameters to get correct results: between detecting all frequencies or none, I couldn't create manual filtering.



Out of curiosity, I applied a manually created Gaussian filter, intended for when I could find the frequencies and bands to filter, but once again, the result was too modulated, and the transcriptions worsened.



6. Implementing an Interface for live transcriptions of aeronautical communications

k. Reasons for Implementing the Interface Instead of Continuing Research

The decision to implement an interface rather than continuing to improve the audio transcription models was based on several reasons:

- Lack of Functional Models and Databases for Improvement:

During my initial research, I tested several audio transcription models available on HuggingFace. Despite some successes, many models did not provide satisfactory results. For example, two of the four models tested produced incorrect transcriptions, and one model did not work at all.

Additionally, the search for a quality database containing audio samples and their associated transcriptions was unsuccessful. Such a database would have allowed fine-tuning the existing models to improve their accuracy.

Creating an interface to allow the transcriptions and storage of those linked to the audio samples might allow to build such database, with users correcting the transcriptions, allowing a future improvement of such model.

- Technical and Scientific Limitations:

My technical and scientific knowledge in signal processing and manipulation of language models (LLM) was limited, as shown in the previous chapter. Despite several attempts to improve transcription results (noise reduction, filtering, using Fourier transforms), the results remained unsatisfactory.

The treatments applied to the audio signals did not significantly improve the quality of the transcriptions. For example, noise reduction and various filtering techniques did not yield the expected improvements.

- Desire to Create a Concrete Interface:

Rather than continuing to seek ways to improve transcriptions, I wanted to create a concrete and practical tool. In order to create something concrete during this short internship, an interface would not only demonstrate the capabilities of real-time

transcription models but also might be a tool to create a new database with correct transcriptions to improve models in the future, as said before.

The development of the interface also presents practical and operational interest. In the context of my future assignment, a real-time transcription interface could be used for strategic monitoring and intelligent surveillance of aeronautical communications.

I. Preliminary Study

First, I had to study and develop a module capable of performing this task for a single audio stream. The site I experimented on (liveAtc.com – note: it does not allow commercial or professional use; I requested permission via email to use it for this project) provides .pls files corresponding to live wave streams for each airport and even for different channels of each available airport if desired.

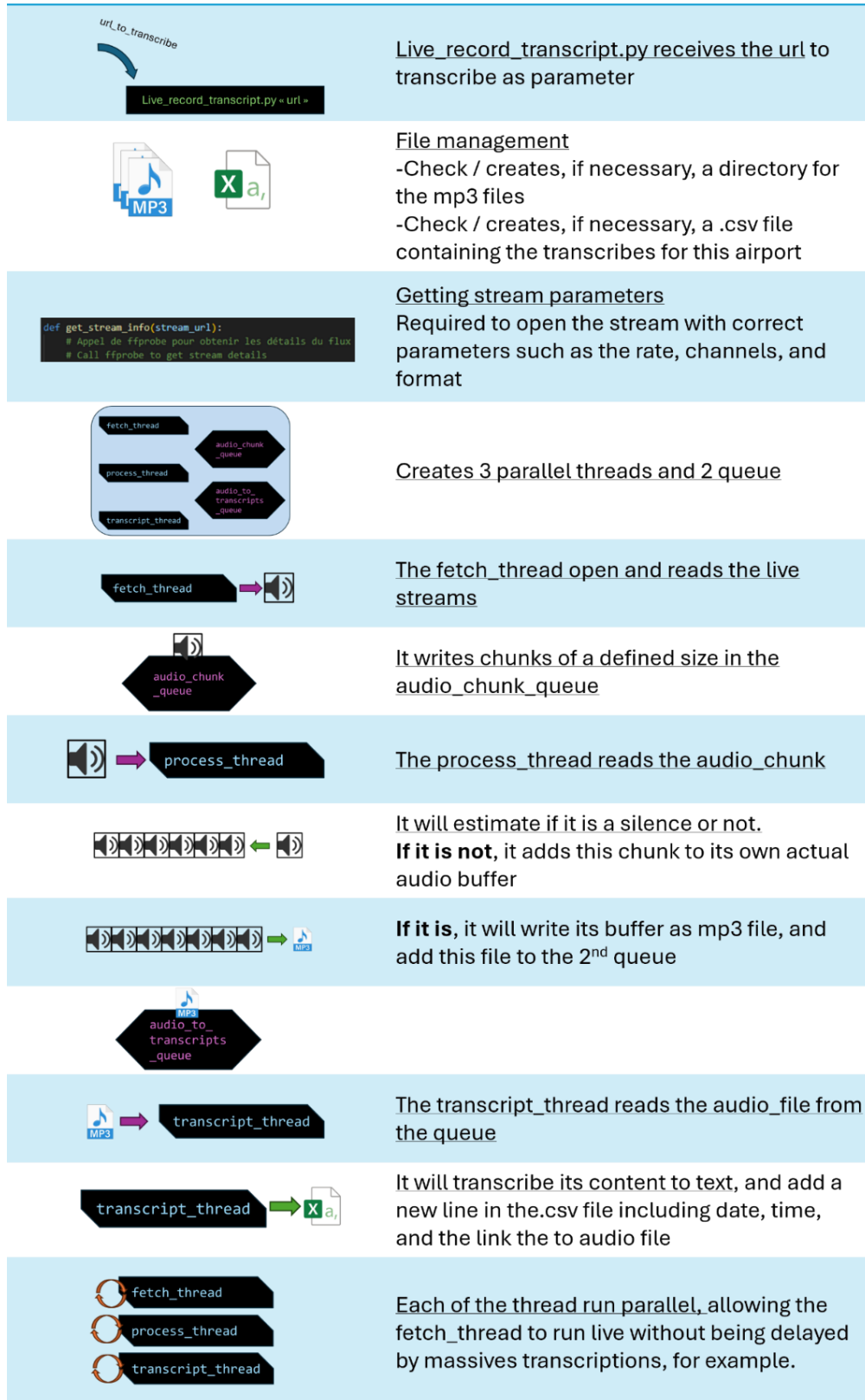
The work involved several aspects:

- Opening a "live stream" type audio file instead of standard MP3 files.
- Working on the arriving audio blocks, including speech/silence detection.
- Storing these blocks as files, each describing a speech segment.
- Coordinating the architecture to allow simultaneous operation and transcription without slowing down the capture of the incoming audio stream.
- Choosing a web interface was relatively quick due to my experience and the tools available, even in Python, to develop one easily. However, Gradio did not seem to offer the same freedom in creating the desired interface, so I used another library suggested by ChatGPT: Flask.
- Starting from the liveATC.com website, where you can search for airports and download .pls files to listen to their live audio streams, it was necessary to integrate this into the website to display airports and allow selection.

The complete interface script is available in the Aero_voice_rec folder, including a demonstration video.

m.Live Audio Stream Transcription Script

The code will be provided, and the encountered difficulties and possible improvements will be mentioned later. Here is a schematic of the transcription script's operation:



So as to resume, it is divided into three threads running in parallel:

- ✓ The first thread accesses the audio stream and stores audio data "chunks" (chunk_size = 256) at fixed intervals. These chunks are placed in a queue to be processed by a second thread.
- ✓ The second thread starts by evaluating whether it is silence or not. If it is, it "cuts" the audio chunk, considering it the end of a sentence. It converts the resulting string to MP3 format and sends it to the third thread via a queue.
- ✓ The third thread handles transcription: it processes the received MP3 files one by one, converting them into text strings.

n. Main Script: Starting the Web Interface and Managing Sub-processes

Interface.py

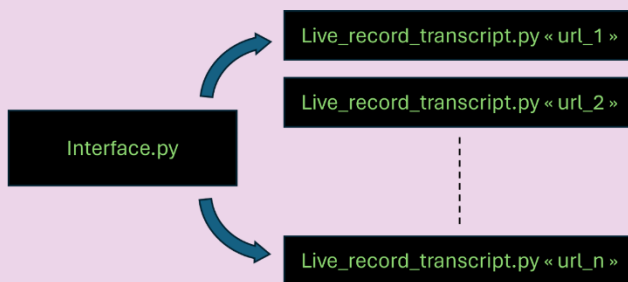
Interface.py is the main of this interface.
It is the script which shall be launched



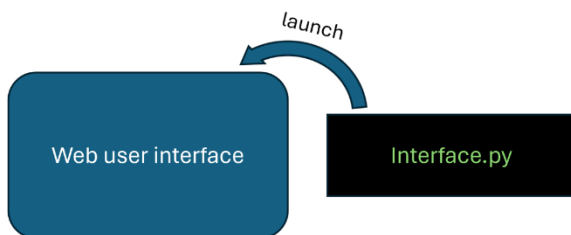
id	Name	Latitude	Longitude	Code	Web Type	URL
1	Budapest Ferenc Liszt International Airport	47.43692020	19.06099999	BUD	UNRP Approach	http://td.liveatc.net/tdp_app2
2	John F. Kennedy International Airport	40.63985333	-73.77893333	JFK	UNRP Approach	http://td.liveatc.net/tdp_app2
3	London Heathrow Airport	51.47002222	-0.45409722	LHR	UNRP Approach	http://td.liveatc.net/tdp_app2
4	San Francisco International Airport	37.62135278	-122.37901667	SFO	UNRP Approach	http://td.liveatc.net/tdp_app2
5	San Jose International Airport (Costa Rica)	9.90033333	-84.08055556	SJO	UNRP Approach	http://td.liveatc.net/tdp_app2
6	San Jose International Airport (Panama)	9.12905556	-79.41888889	SJO	UNRP Approach	http://td.liveatc.net/tdp_app2
7	San Jose International Airport (Guatemala)	16.73666667	-90.58333333	SJO	UNRP Approach	http://td.liveatc.net/tdp_app2
8	San Jose International Airport (Honduras)	15.38333333	-86.08333333	SJO	UNRP Approach	http://td.liveatc.net/tdp_app2
9	San Jose International Airport (Nicaragua)	12.13333333	-86.08333333	SJO	UNRP Approach	http://td.liveatc.net/tdp_app2
10	San Jose International Airport (El Salvador)	13.63333333	-88.91666667	SJO	UNRP Approach	http://td.liveatc.net/tdp_app2

Read Data/airports_updated.csv

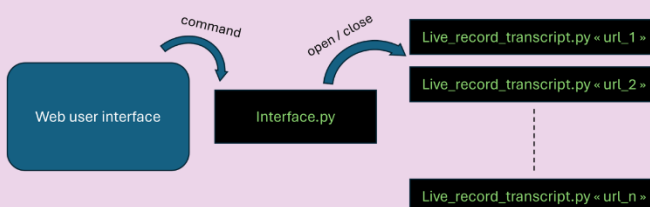
Contains a list of airports with
-**name** and **IATA code**, **latitude** and **longitude** of the airport
-**url** to live audio, **bool** « **being transcribed** »



Open a subprocess for each airport already being transcribed, regarding airport_updated.csv['being_transcribed'] var
Also keep a link to the subprocess to kill it afterward



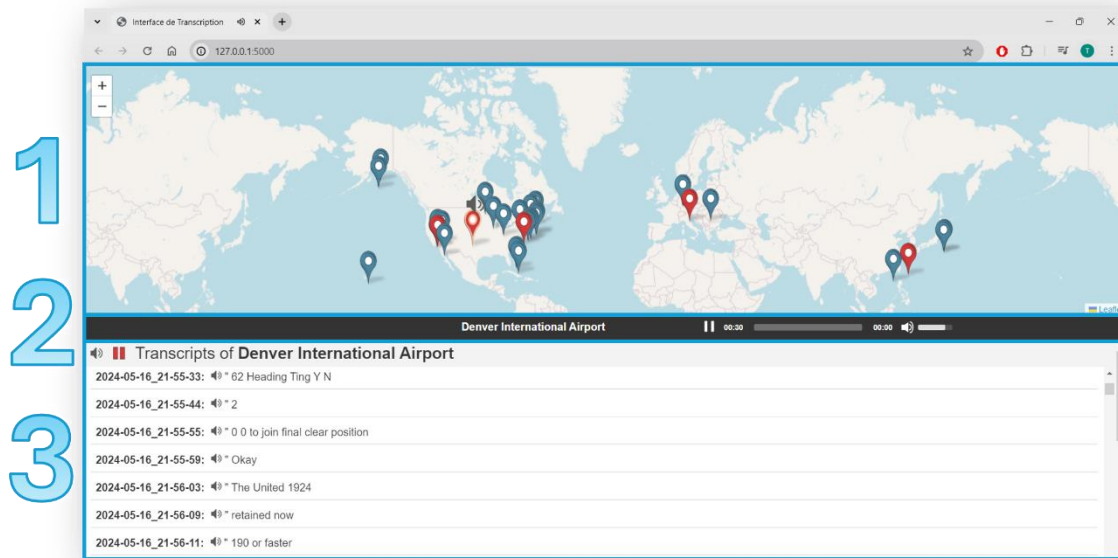
Launch the Web User Interface using Flask library.
The Interface is splitted in 3 frames : main, media player, and transcriptions.
The main receives the content of airports while being launched



The Web User Interface calls Interface.py through buttons calling:
manage_current_transcribes_url.html?to_do=[start/stop]_transcribe_[url_pls_airport]
This launches or kill a subprocess transcribing url_pls_airport

o. Description of the web interface and its functionalities

The interface.py script launches a server, allowing local access to the application. Typically hosted on 127.0.0.1:5000, accessing the page triggers a GET request visible in the console. The script is designed to respond with index.html, a page mainly divided into three divs: [mapid], [media_player], and [transcripts].]



- 1 [mapid]
Generated in JavaScript directly within the index.html code, which has received the content of airports.csv to be accessible in JavaScript.
Using JavaScript, it:
 - ✓ Loads an OpenStreetMap map.
 - ✓ Creates and manages pins (selection color, popup display onmouseover).
 - ✓ Triggers the loading of [transcripts] content based on the selected pin, transmitting information in the request (GET method).
- 3 [transcripts]
Managed by the pin selected in [mapid].
JavaScript:
 - ✓ Loads the content of the CSV file corresponding to the chosen airport to display messages with timestamps and a button to listen to the audio file. If transcription is ongoing, it automatically refreshes the file's content to display the latest messages.
 - ✓ Triggers the loading of [media_player] content upon clicking the speaker icon, transmitting information in the request (GET method).

Simultaneously adds a speaker icon on the [mapid] map above the chosen airport's pin.

- ✓ Triggers the start or stop of transcriptions via an HTML request on [/manage_current_transcribes_url] upon clicking the record/pause icon, transmitting information in the request (GET method). This also changes the pin's color and activates/deactivates automatic message refresh.

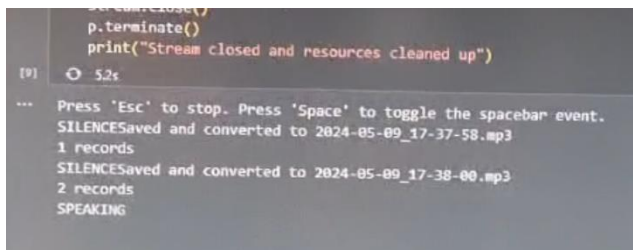
- 2 [media_player]

A banner displaying a media player playing the current stream.

JavaScript: Automatically loads the audio stream (approx. 10s) received in the URL (GET method).

p. Proof of concept : development choices

- Multi-threading was necessary. Transcription could slow down the rest of the process, so losing audio excerpts during transcription was inconceivable if it took time. Additionally, this approach allows handling periods of high activity by continuing to record the live incoming audio stream and catching up on transcriptions when communications calm down.
- To lighten the server load and optimize its use for potential massive transcription, JavaScript was preferred over PHP, despite my limited experience with it, for the advantage of being executed on the client side.
- Silence evaluation is a simple mathematical sum of the sent audio block values. Initially, ChatGPT recommended using a speech recognition API to detect silence: this proved very heavy and inefficient on such short audio excerpts. Wanting to materialize the sound signal's amplitude to manually detect silences,



```
p.terminate()
print("Stream closed and resources cleaned up")
[1] 52s
... Press 'Esc' to stop. Press 'Space' to toggle the spacebar event.
SILENCESaved and converted to 2024-05-09_17-37-58.mp3
1 records
SILENCESaved and converted to 2024-05-09_17-38-00.mp3
2 records
SPEAKING
```

I created a script displaying the sum of values when pressing the space bar or not, allowing me to set a threshold on spoken or unspoken passages. Despite the varying quality of streams from

different airports, this threshold works well, even on streams where noise has not been reduced.

- Initially, for testing, I added a fourth thread monitoring keyboard movements, particularly for the space bar (cf. above) and the escape key: I had to create a function closing all threads upon the "escape" signal; otherwise, transcription continued in the background in the console despite a "CTRL + C" to close the script.
- Creating a handle (symbolic link) to /dev/null was necessary to redirect the audio stream; otherwise, a server error message indicated the absence of a sound card.

q. Improvements and Additions

Interface Side:

- Displaying current CPU Usage : Possible improvements include **displaying the transcription queues' status** to visualize potential overloads.
- Getting rid of too short excerpt: **Implementing a minimum duration for recording an audio excerpt**. Some pilot manipulations consist of quickly opening and closing the channel, triggering the transcription mechanism with random results.
- CSV vs SQL Storage: Additionally, the server storing the list of currently transcribed airports in a CSV file is not problematic. However, if there were remote access, the script (transcripts.html) **currently loads transcriptions from a CSV file stored in the server directories**. This would not be possible in client-side JavaScript or at least not recommended. Other techniques (PHP or others) should be studied to allow displaying a server-side stored file while only sending the newly transcribed lines to the client to avoid reloading the entire CSV file every second.
- Independent Thread for transcription: It might also be interesting to study the benefit of **creating an independent script for transcription**, evaluating the memory consumption of loading a single or multiple times the model. In any case, loading the transcription pipeline at the script's start takes the most time (50 – 66%) and would allow transcription to start more quickly and have a more responsive program to user requests. However, having only one transcription process when multiple streams are being listened to risks not using the servers' capabilities and accumulating transcription delays.

Transcription Side:

- **Transform this interface into a “AI model feeding tool” : By adding the possibility to manually correct each transcription**, this could automatically generate a database associating an audio excerpt with the correct transcription, allowing fine-tuning a generic model like Whisper or improving the learning of the current model.
- **Excerpt preprocessing: Continuing to reduce noise or filter input signals to improve transcription** (even if it means using a non-specialized speech-to-text model for aeronautics once the signal is clear enough).
- **Usage of LLM : Continuing the possibility of using an LLM** light enough to handle each transcription request without significantly slowing the process. It could be tasked with detecting minor transcription errors, searching for inconsistencies, or merging transcriptions from different models or original and denoised audio files.

r. Discussions and outlook

- Combination with LLM Systems:

A practical interface could also be combined with LLM systems for advanced tasks such as detecting intentions or inconsistencies in communications. This could help raise alerts in case of abnormal behaviors or pre-established harmful patterns.

The tool could also be used to generate strategic advice inspired by the art of war or to monitor enemy behaviors in real-time through in-depth analysis of transcribed communications.

- Learning and Demonstrating Acquired Skills:

The implementation of this interface was a project of a complexity that I consider decent for an individual project at my level, and with the assistance of ChatGPT, I was able to use features never seen before, while maintaining a critical eye when using this tool.

This allowed me to achieve a concrete realization with an AI model, moving beyond the phase of pure experimentation.

The interface also serves as a proof of concept to show what can be achieved with current technologies and to identify future areas for improvement.

7. Conclusion and Outlook

Key Points :

1) Skills Acquired :

During this internship, I acquired numerous skills in artificial intelligence, including manipulating image and video generation models, using audio transcription models, and developing user interfaces with tools like Gradio and Flask.

2) Relevance for Future Position :

This internship allowed me to better understand the concrete applications of AI technologies in the defense sector, particularly for strategic monitoring and processing of aeronautical communications. These skills and knowledge will be particularly useful in my future position.

3) Limitations and Future Prospects :

Despite the progress made, several limitations were identified, notably in terms of transcription accuracy and data management. Future research could focus on improving transcription models, optimizing signal processing, and integrating advanced machine learning techniques to detect inconsistencies and anomalies in transcriptions.