INTERNSHIP REPORT

Project Title: Wake Word Detection – Evaluation Script for Alexa Pre-Trained Model

INTRODUCTION

During my internship, I worked on a project based on Wake Word Detection, specifically for the word **Alexa**. The main goal of this project was to understand how a pre-trained model can detect a wake word in audio and to evaluate the performance of this model using a custom script.

Wake word detection is a key part of any voice assistant system. It allows the device to **wake up** when a specific word is spoken. For example, Alexa-enabled devices respond only when they hear Alexa. This project helped me understand how such detection works in the background using machine learning models and audio processing.

I worked mostly with Python and used a public GitHub repository called openWakeWord (https://github.com/dscripka/openWakeWord), which provides tools and models for building wake word detection systems. Through this project, I learned how to work with ONNX and TFLITE models, extract audio features, write evaluation code, and analyze the results.

All work completed during this internship is organized and available in my GitHub repository: https://github.com/rmall2003/Evaluation

TOOLS and RESOURCES USED

- **Programming Language:**
  Python was used for writing the evaluation scripts, processing audio files, and running model inference.
- **Platforms:**
  I used **Google Colab** for generating audio samples, and maintained all development, testing, and script versions in my **GitHub repository** for version control and collaboration.
- **Libraries Used:**
  - onnxruntime: for running ONNX models
  - numpy, scipy: for numerical operations and audio handling
  - sklearn: to calculate metrics like recall, F1 score, and ROC AUC
  - matplotlib: to plot ROC curves
  - librosa: used initially for extracting MFCC features
  - yaml: to load configuration files

- **Repository Used:**
  The GitHub repository (https://github.com/dscripka/openWakeWord) provided the pre-trained Alexa model, sample generation notebook, and the utils.py file used for proper feature extraction.

- **Audio Files:**
  Synthetic audio samples were generated using the notebook in the repo. These included both positive (wake word present) and negative (wake word absent) samples. I also created augmented versions with noise and variations.

- **Model Type:**
  I used a pre-trained Alexa wake word detection model in ONNX format. ONNX was preferred due to TensorFlow installation issues on my system.

WORKING PROCESS

1. **Understanding the Repository:**
   I began by exploring the openWakeWord GitHub repository. This repo provides ready-to-use models and code for wake word detection using ONNX and TFLite. It also included tools to generate synthetic audio samples and perform model training.

   I studied the structure of the repo and tried to understand the training and evaluation workflows provided.

2. **Generating Synthetic Audio Samples:**
   The repository had a Jupyter notebook for automatically generating audio samples, both positive (with the wake word) and negative (without the wake word). I downloaded and tried running the notebook on my local system. However, the required module piper-sample-generator did not work on Windows, as it only supports Linux.

   Because of this limitation, I switched to Google Colab to run the notebook online. There, I was able to install the dependencies and start the sample generation process.

3. **Fixing Errors in Sample Generation**
   While using the sample generator notebook on Google Colab, I encountered a PyTorch model loading error while running train.py. The script failed while trying to load a model file using torch.load(model_path). The error message was: **"_pickle.UnpicklingError: Weights only load failed. This file can still be loaded..."**

   The error happened because in PyTorch 2.6, the default argument weights_only=True was introduced in torch.load(). But the model file in the

repository was a full checkpoint, not just weights. So, PyTorch was trying to load it as weights-only and failed.

To fix this, I updated the line in generate_samples.py to: "**torch_model = torch.load(model_path, weights_only=False)**"

By setting weights_only=False, I allowed PyTorch to load the full checkpoint instead of just the weights. This resolved the issue, and I was able to generate all the required synthetic audio samples successfully. I made sure to trust the source before applying this change.

4. **Writing the Evaluation Script (Version 1)**

   After generating the audio clips, I downloaded them to my system. Then I started writing a custom evaluation script in Python to test the model's accuracy. This script took in a set of audio files and evaluated how well the model could distinguish between wake word and non-wake word clips.

   In my first version (Evaluation_script.py), I used **MFCC** (Mel Frequency Cepstral Coefficients) for feature extraction. MFCC is a standard audio feature used in many voice and speech recognition systems.

   I used a tflite model at this point. However, I faced issues running TensorFlow on my machine. It failed to install or import properly. So, I moved on to using ONNX model later.

5. **Issue with Low Recall**

   After running the evaluation, I noticed a major issue: the Recall value was very low. This meant that the model was missing a lot of positive wake word detections. The model was not performing as expected.

   After discussion, we checked the original training configuration and realized the mistake: the pre-trained Alexa model was not trained using MFCC features. It was trained using **embeddings from melspectrograms**, a more advanced audio feature format.

   So, my script was giving poor results because the features I was feeding into the model were not compatible with what the model was trained on.

6. **Second Evaluation Script (The Final Version)**

   To solve the issue, I wrote a new evaluation script (Eval_Script.py). This time, I used the AudioFeatures class defined in utils.py from the repo, which extracts the correct features using ONNX model.

   In this script:

   - I used onnxruntime for inference
   - Loaded audio samples from directories using YAML config
   - Used the proper melspectrogram + embedding method for feature extraction

- Computed metrics like Recall, F1 Score, ROC Curve, etc.

Both the evaluation scripts made use of **argparse**, a Python module that helps take input parameters directly from the command line. This allowed the scripts to be reused easily with different model files or audio sample folders without changing the code every time.

Arguments like the model path, YAML config file, and optional feature model paths could be passed dynamically while running the script. This made the code cleaner, more flexible, and more professional.

This version gave much better results and was in line with the expected performance of the model.

7. **Testing with different audio types**

   To analyse the performance better, I tested the model on two sets of audios:

   - **Non-Augmented Audios:** Normal, clean speech samples
   - **Augmented Audios:** Noisy, stretched, or modified versions of the original clips

   I ran the evaluation script on both sets and compared the results.

EVALUATION RESULTS

Metric	Non-Augmented Audios	Augmented Audios
True Positive	108	96
True Negative	166	169
False Positive	34	31
False Negative	92	104
Recall	0.54	0.48
F1 Score	0.6316	0.5872
False Positive Rate	0.17	0.155
True Positive Rate	0.54	0.48
ROC AUC	0.7244	0.673

As expected, the model performed better on clean audio than on noisy/augmented data.

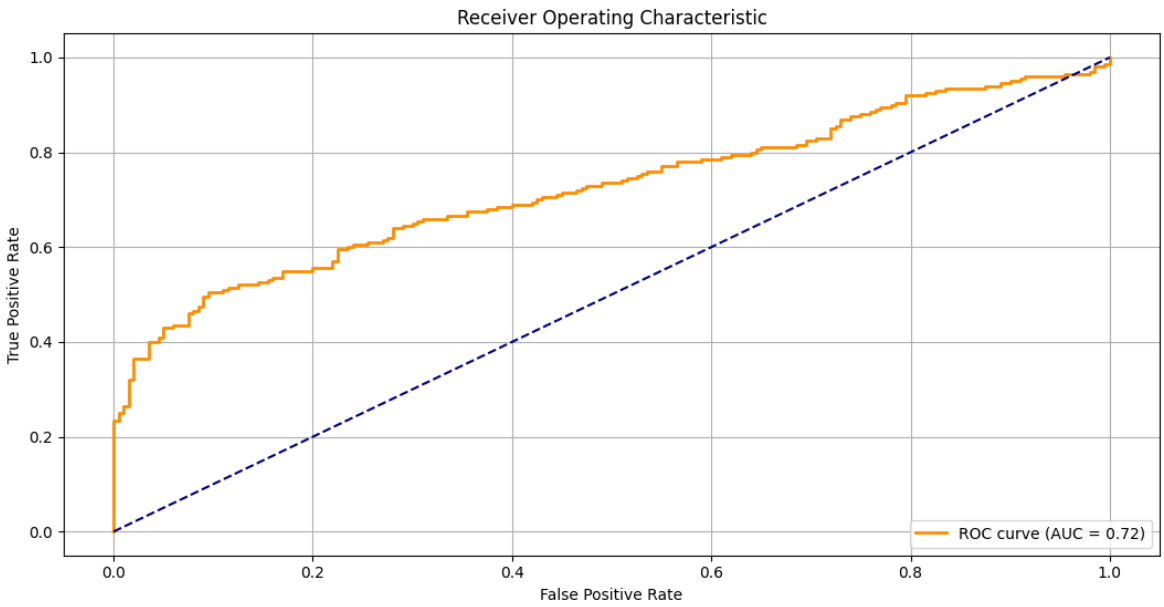

Fig. 1: ROC Curve for Non-Augmented Audios

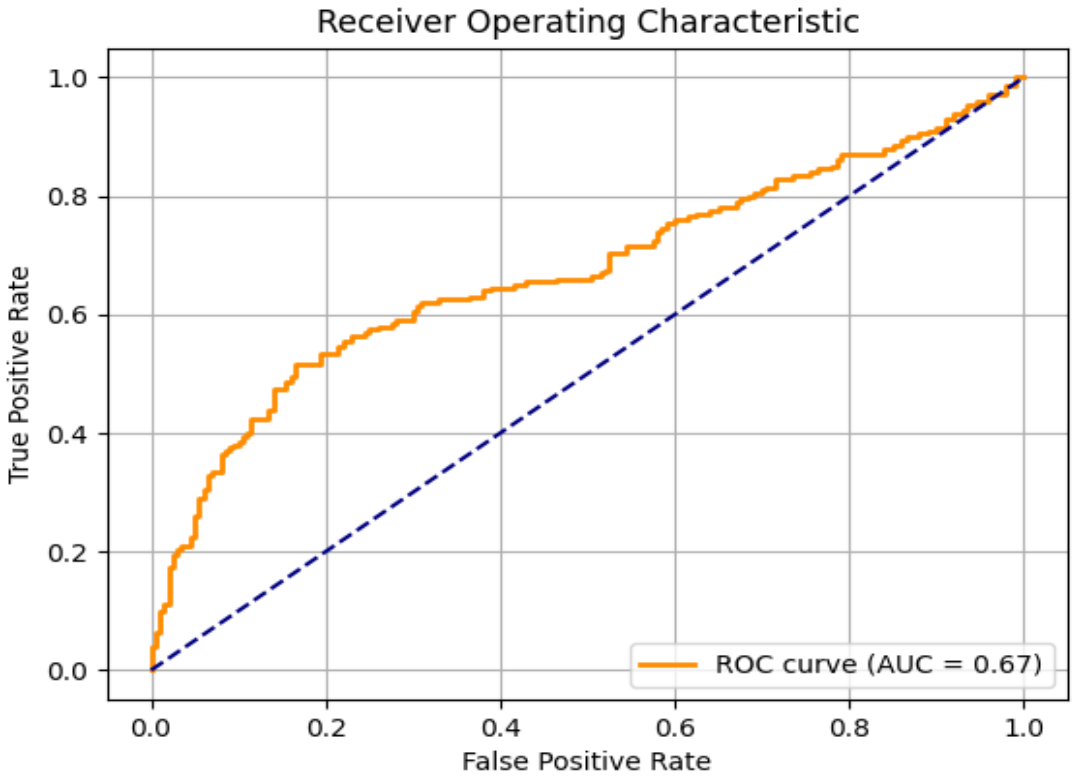

Fig. 2: ROC Curve for Augmented Audios

CHALLENGES FACED

- **Platform Compatibility:**
  The piper-sample-generator tool does not work on Windows. I had to shift to Google Colab to run it.
- **Torch Model Error:**
  A missing argument in a function caused the model loading to fail. I fixed it by modifying the script manually.
- **Low Model Accuracy Initially:**
  The first script used MFCC features, which were not compatible with the model. I had to study the original training pipeline and rewrite my script accordingly.
- **TensorFlow Installation:**
  I couldn't use tflite models due to TensorFlow not working on my laptop. So, I used the ONNX version instead.

WHAT I LEARNED

- I understood how wake word detection systems work and how they are evaluated.
- I learned how to use ONNX models and extract correct features for them.
- I improved my debugging skills by fixing code errors and understanding the repo deeply.
- I got hands-on experience with audio processing, performance evaluation, and model testing.
- I understood the importance of using the correct input features that match the training data.
- I also learned how to make scripts more reusable and flexible using argparse, which allowed users to pass values directly as command-line arguments.

CONCLUSION

This internship gave me great exposure to real-world machine learning workflows, especially in the field of voice technology. From reading open-source code to writing my own evaluation scripts, I learned a lot about how wake word systems function. The project helped me grow my problem-solving skills and made me more confident in working with pre-trained models and evaluation techniques.