Title: Setting Up a Linux-based Voice Analysis Server with Praat and a Flutter Interface on Windows (WSL2)

1 Overview

This guide outlines a step-by-step setup to configure a full-stack speech analysis environment on a Windows system using **WSL2** (Windows Subsystem for Linux). It includes the setup of a Linux-based server environment, **Praat** installation, **Python Flask API** configuration, and execution of a **Flutter-based front-end application**. The backend and frontend are hosted on the same machine using local loopback or WSL-accessible interfaces, making the system suitable for development and offline analysis workflows.

1 Prerequisites

- Windows 10 or 11 with WSL2 support
- VS Code with the following extensions:
  - Remote - WSL
  - Dart & Flutter
- Flutter SDK (installed on WSL Linux)
- Internet connection to install dependencies

2 Step-by-Step Setup Instructions

2.1 Enable WSL2 and Install Ubuntu

1. Open PowerShell as Administrator:

```
wsl --install
```

2. Restart the machine when prompted.
3. From Microsoft Store, install **Ubuntu 24.04 LTS**.
4. Launch Ubuntu and create a username and password.

2.2 Update Ubuntu and Install Required Packages

```
sudo apt update && sudo apt upgrade -y
```

```
sudo apt install -y build-essential cmake ninja-build clang libgtk-3-dev pkg-config curl git unzip python3-venv praat
```

2.3 Set Up Python Virtual Environment for Flask

```
mkdir -p ~/praat_server
```

```
cd ~/praat_server
```

```
python3 -m venv flask_env
```

```
source flask_env/bin/activate
```

```
pip install flask flask-cors
```

3 Flask API Development for Praat Automation

This chapter explains the development of a Flask-based web API to automate acoustic analysis of .wav files using a Praat script on a Linux server. The API is intended to be called by a Flutter application or other frontend client to analyze voice recordings and return metrics such as CPPS, F0, HNR, Jitter, and Shimmer.

3.1 Overview

The Flask API serves as a lightweight HTTP interface to run a Praat script on selected .wav files. It enables easy integration of acoustic analysis into cross-platform applications, particularly the Flutter frontend designed in this project.

3.2 Required Libraries and Installation

To implement the API, the following Python libraries were used:

Library	Purpose	Installation Command
Flask	Core web framework for building the API	pip install flask
Flask-CORS	Enables cross-origin requests (e.g., from Flutter)	pip install flask-cors
subprocess	Executes external Praat command-line scripts	(Built-in Python module)
os	Handles file paths and server-side file validation	(Built-in Python module)

Note: Flask and flask-cors were installed inside a virtual environment using python3 -m venv flask_env.

3.3 API Code Breakdown

3.3.1 Import and Initialize

```
from flask import Flask, request, jsonify
from flask_cors import CORS
import subprocess
import os
app = Flask(__name__)
CORS(app)
```

`Flask(__name__)`: Initializes the Flask app.

`CORS(app)`: Enables cross-origin requests, allowing the Flutter app (running on a different host or port) to access the Flask server.

3.3.2 Configuration Variables

```
PRAAT_SCRIPT_PATH = '/home/ahassanp/praat_server/praat_scripts/optimized_cpps.praat'
```

```
WAV_FOLDER = '/home/ahassanp/praat_server/static'
```

PRAAT_SCRIPT_PATH: Absolute path to the Praat .praat script.

WAV_FOLDER: Directory where .wav files are stored/shared.

3.3.3 Analyze API Endpoint

```
@app.route('/analyze', methods=['GET'])
def analyze():
```

Defines a GET endpoint /analyze, accepting a query parameter file=<filename.wav>.

- Step 1: Input Validation

```
filename = request.args.get('file')
if not filename or not filename.endswith('.wav'):
    return jsonify({'error': 'Missing or invalid .wav filename'}), 400
```

Ensures a valid .wav filename is passed in the request.

- Step 2: File Existence Check

```
wav_path = os.path.join(WAV_FOLDER, filename)
if not os.path.exists(wav_path):
    return jsonify({'error': f'File not found: {filename}'}), 404
```

Validates that the file exists in the designated folder.

- Step 3: Run the Praat Script

```
result = subprocess.run([
    'praat', '--run', PRAAT_SCRIPT_PATH,
    'wav', WAV_FOLDER, '60', '5000', '50', '60', '330', '1', '0.02', '0.0005', 'Straight'
], stdout=subprocess.PIPE, stderr=subprocess.PIPE, text=True)
```

Executes Praat in headless mode using CLI and passes arguments to the script.

Captures both stdout and stderr for debugging.

- Step 4: Error Handling

```
if "Error:" in result.stderr and "Done!" not in result.stderr:
    return jsonify({'error': f'Praat script failed: {result.stderr}'}), 500
```

Filters out false errors (e.g., “Error: Done!” which is benign).

- Step 5: Read Output File and Format Response

```
txt_path = wav_path.replace('.wav', '.txt')
if not os.path.exists(txt_path):
    return jsonify({'error': f'Praat output not found at {txt_path}'}), 500

```

```
with open(txt_path, 'r') as f:
    lines = f.readlines()
    header = lines[0].strip().split(',')
    values = lines[1].strip().split(',')
    return jsonify(dict(sorted(zip(header, values))))
```

Reads the output .txt file generated by the Praat script.

Converts the first line (header) and second line (values) into a dictionary and returns it as a JSON response.

Final Block: Application Entry Point

```
if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

Runs the Flask app, binding to all interfaces (0.0.0.0) to allow external access from the same network.

3.4 Example Usage

To trigger the Praat analysis via a GET request:

```
http://<WSL-IP>:5000/analyze?file=track24.wav
```

Example:

```
http://129.100.33.162:5000/analyze?file=track24.wav
```

The response will be in JSON format:


```
{
  "CPPS(dB)": "6.148720",
  "F0(Hz)": "65.814562",
  "HNR(dB)": "4.893712",
  "Jitter(%)": "2.819812",
  "Shimmer(dB)": "1.479816"
}
```


3.5 Integration with Flutter

This API was designed to be consumed by a Flutter frontend running either on desktop or web. The Flutter app sends a request to this Flask endpoint, receives the JSON, and displays the acoustic metrics in labeled text fields.

As illustrated in **Figure 1**, the Flutter application initially displays a dropdown for selecting a .wav file and an *Analyze* button before any processing occurs.

After pressing the *Analyze* button, the app communicates with the Flask API. Once the analysis is complete, the app populates the acoustic metrics, as shown in **Figure 2**.

Figure 1) The initial state of the app where the user selects a .wav file (here, track24.wav) before pressing the Analyze button. No acoustic features have been computed yet.

Figure 2) the Flutter-based GUI after analyzing the file track24.wav. The computed CPPS (6.15 dB) is displayed along with placeholders for F0, HNR, Jitter, and Shimmer, which will be populated once the analysis script returns complete results.

3.6 Security & Deployment Notes

This implementation uses Flask's development server (app.run()), which is not recommended for production.

In production, use a WSGI server like Gunicorn with Nginx or Apache as a reverse proxy.

Authentication and rate-limiting can be added using libraries like Flask-Login, Flask-Limiter, etc.

3.7 Sharing WAV Files Between Windows and WSL (Ubuntu Server)

To ensure that your Praat server running in WSL can access .wav files stored on your Windows filesystem:

Step 1: Use WSL's Auto-Mounted Drives

WSL auto-mounts Windows drives under /mnt. For example:

```
/mnt/c/Users/<YourName>/praat_wav_files/
```

If your .wav files are in "C:\Users\YourName\praat_wav_files", use that path. For example this the path in my username and PC:

```
\\wsl.localhost\Ubuntu\home\ahassanp\praat_server
```

Step 2: Set **WAV_FOLDER** Path in Flask

In your praat_api.py Flask script:

```
WAV_FOLDER = '/mnt/c/Users/<YourName>/praat_wav_files'
```

Or use a symbolic link:

```
ln -s /mnt/c/Users/<YourName>/praat_wav_files ~/praat_server/static
```

Step 3: Confirm File Access and Permissions

Ensure your .wav files are readable in WSL:

```
ls /mnt/c/Users/<YourName>/praat_wav_files/*.wav
chmod +r /mnt/c/Users/<YourName>/praat_wav_files/*.wav
```

3.8 Setup Flask Server API

- Place your "praat_api.py" file inside "~/praat_server"
- Ensure your Praat script and .wav files are under:
    - ~/praat_server/praat_scripts/
    - ~/praat_server/static/
- Run the Flask server:

```
source ~/flask_env/bin/activate
python3 praat_api.py
```

- The server runs at: http://127.0.0.1:5000

3.9 3.5 Install and Configure Flutter in WSL

```
git clone https://github.com/flutter/flutter.git -b stable ~/flutter
export PATH="$PATH:$HOME/flutter/bin"
flutter doctor
```

- Follow prompts to install dependencies if needed (e.g., libgtk-3-dev)
- Accept Android licenses if prompted:

```
flutter doctor --android-licenses
```

3.10 3.6 Create and Configure Flutter App

```
cd ~/praat_server
flutter create praat_flutter_app
cd praat_flutter_app
```

3.11 3.7 Add HTTP Dependency

Edit "pubspec.yaml":

```
dependencies:
  flutter:
    sdk: flutter
  http: ^0.13.5
```

Run:

```
flutter pub get
```

3.12 Update Flutter Code to Use Localhost Server

Ensure that your main.dart uses the local WSL server address:

```
final uri = Uri.parse('http://127.0.0.1:5000/analyze?file=$selectedFile');
```

3.13 Enable Linux Desktop Target for Flutter

Install missing build tools:

```
sudo apt install clang cmake ninja-build pkg-config libgtk-3-dev liblzma-dev
```

Then run:

```
flutter config --enable-linux-desktop
flutter doctor
```

3.14 Run the Flutter Desktop App

```
cd ~/praat_server/praat_flutter_app
```

flutter run -d linux

4 Development Notes

- All operations are performed **within WSL Ubuntu**.
- You must use VS Code with **Remote - WSL** to access and edit the Linux-side files.
- If you're running the Flutter app on another device or browser, ensure CORS is enabled in Flask (from flask_cors import CORS; CORS(app)).
- Ensure the Flask server is running before starting the Flutter app.

Open a second WSL terminal in VS Code or Ubuntu.
F1 + Connect to WSL in New Window, and choose the praat_server [wsl:Ubuntu]
In **WSL** VS Code terminal:
cd ~/praat_server
source flask_env/bin/activate
python3 praat_api.py
Then open new Terminal in **WSL** VS Code terminal:
source flask_env/bin/activate
cd ~/praat_server/praat_flutter_app
flutter run -d linux
Like below:
(flask_env) ahassanp@UWO-CSL20420:~/praat_server$ **source flask_env/bin/activate**
(flask_env) (flask_env) ahassanp@UWO-CSL20420:~/praat_server$ **cd ~/praat_server/praat_flutter_app**
(flask_env) (flask_env) ahassanp@UWO-CSL20420:~/praat_server/praat_flutter_app$ **flutter run -d linux**

5 Appendix

The following codes and scripts used in this setup are included in the appendix:

- Appendix A: Flask API (praat_api.py)
- Appendix B: FlutterApp_PraatAPI_v1 (main.dart)
- Appendix C: Praat Script for Analysis (optimized_cpps.praat)