Cloudy's OS:
AUDIO EMOTION ANALYSIS

ITCS225 Principles of Operating System – Group Project

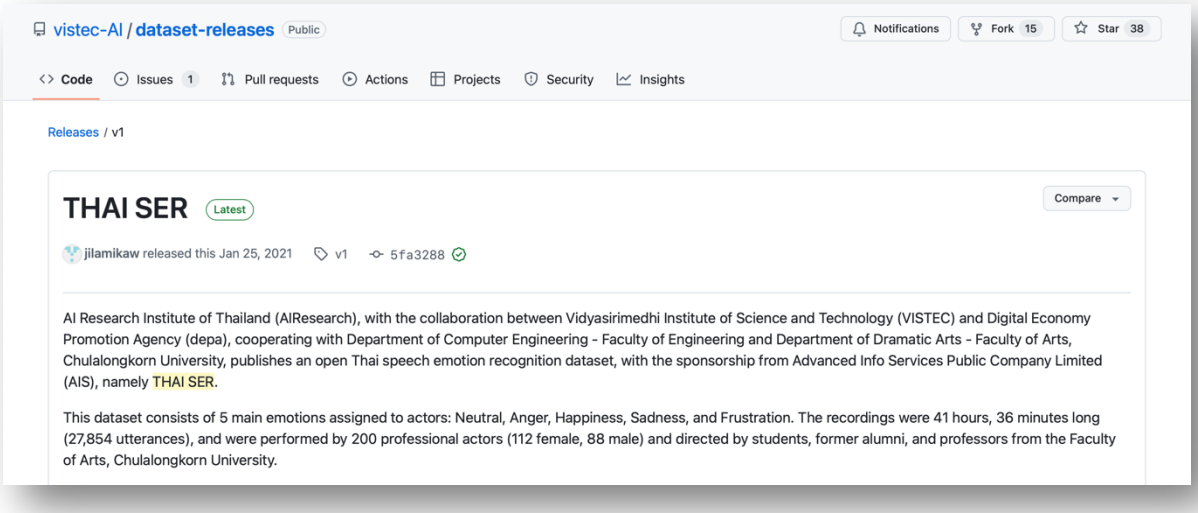
- 6688077 Bhumipat Pengpaiboon
- 6688108 Napas Siripaskrittakul
- 6688142 Krerkkiat Wattanaporn

1. Introduction

1.1. This report is focused on the implementation and performance analysis of a system designed to classify emotions from spoken Thai language. The main objective key is developing a machine learning to learn and identifying five emotional by the sound via dataset with 5 different classes such as angry, frustrated, happy, neutral, and sad from THAI SER dataset. We are trying to examine the performance and monitor the result by looking on involving data loading, feature extraction, model training, and evaluation as a measured time. This report will evaluate the predict accuracy and the performance between first-run version and enhancement version.

2. Methodology

2.1. Dataset: The THAI SER dataset was developed by AI Research Thailand in collaboration with Chulalongkorn University and AIS. In the dataset, there are 41.6 hours (25,185 audio files used in this analysis) of Thai speech audio recordings from 200 actors, annotated with the five target emotions.



2.2. Feature Extraction: We are using Mel-Frequency Cepstral Coefficients (MFCCs) to analyze the audio for the computer to understand by transferring from sound as a frequency in Hz to pure data by capturing information about the shape of the vocal. Each audio file sampled at 44.1 kHz with 13 MFCCs were extracted using the **librosa library**. To transform the input data to machine learning part, we are needed to checking input dimensionality for the machine learning model so the resulting feature sequences were standardized to a fixed length of 500-time steps by padding shorter sequences. The 2D feature array (500 steps x 13 MFCCs) was flattened into a 1D vector of 6500 features per audio sample.

2.3. Model: A Multi-Layer Perceptron (MLP) classifier is implemented by using **scikit-learn (MLPClassifier).** We are also splitting the data for training (70%), validation (15%), and test (15%) sets, followed by feature scaling using by **StandardScaler**.

2.4. The versions: We have two versions of code on this implementation to compare the computational efficiency on performance and result.

- The first version: This version reading the dataset sequentially and doing the Feature extraction that keep iterating through each audio file path in the metadata and transform it into the DataFrame from NumPy library and calling the **extract_features function** for each file within a standard for loop. This make the code to utilizes a single CPU core for the feature extraction process which is slow.

```
X_list = []
# ...
for index, row in tqdm(df_meta.iterrows()):
    features = extract_features(row['path'])
    X_list.append(features.flatten())
X = np.array(X_list)
```

- The final version: This version is using the parallel computation which is enable the code to run by the parallel processing with hardware accelerate from **joblib library** and **joblib.Parallel utility** so these function is calling the **extract_features** function to using multiple available CPU cores as (n_jobs=-1) to processing multiple audio files **concurrently**.

```
filepaths = df_meta['path'].tolist()
results = Parallel(n_jobs=-1, backend='loky')(
    delayed(extract_features)(fp) for fp in tqdm(filepaths)
)
X_list = [res.flatten() for res in results]
X = np.vstack(X_list)
```

For metadata processing, preprocessing, training, and evaluation is remained sequential in both implementations because it does not affect to significant of the performance and result.

2.5. Why we chosen this method: We have trying to use the GPU-hardware accelerate from CUDA toolkit, python threading library and python multiprocessing library but it does not work because of the incompatibility.

- CUDA (GPU) Incompatibility: The primary libraries used in project are librosa (for MFCC extraction) and scikit-learn (for the MLP classifier) but it does not offer native GPU acceleration for these specific operations via CUDA. It does mainly support PyTorch and TensorFlow.
- Python's Global Interpreter Lock (GIL): GIL is restricting the ability of standard to run on parallelism for CPU-bound.The speed of audio loading and MFCC are not faster when of loading audio files because the main library is loading the audio file as sequential algorithm.
- Python multiprocessing: The joblib library have the ability to computing the multiprocessing task for training the audio file as machine learning type with scikit-learn.

3. **Computational Performance Analysis**: Execution time and resource utilization were recorded while training in both implementations by using **time.time() and psutil**

   3.1. Feature Extraction Comparison: This stage is the most significant performance difference:

   - **First Version Time: 615.14 seconds**
   - **Enhanced Version Time: 115.40 seconds**

   The enhanced version achieved an approximate **5.3x speedup** for the feature extraction process by effectively utilizing multiple CPU cores to process files concurrently.

   3.2. Overall Execution Time Comparison:

   - **First Version Total Time: 642.60 seconds (~10.7 minutes)**
   - **Enhanced Version Total Time: 147.65 seconds (~2.5 minutes)**

     The parallel version was completed the entire process approximately 4.35 times faster than the first version because of the computational bottleneck.

   3.3. Resource Usage (CPU & Memory): The enhanced version was significantly reduced the wall-clock time and its peak memory usage was slightly higher (Final RSS: ~3632 MB vs. ~3146 MB because of managing concurrent processes and data for better performance.

4. **Model Accuracy Analysis**: The MLP model was training by the dataset and the results is test set using the results from the latest runs.

    - Standard Version Accuracy: 47.14%
    - Parallel Version Accuracy: 46.56%

    The results confirm that the parallelization technique applied to feature extraction did **not significantly impact the final predictive accuracy of the model.** The changes difference shows that the accuracy is less than 0.6% is with this type of model training.

    - Neutral emotion achieved the highest F1-score (0.55), suggesting it was the most distinguishable class for the model.
    - Sad emotion is getting the lowest F1-score (0.32) shows that it is greater difficulty in its classification.
    - Other classes (Angry, Frustrated, Happy) showed moderate F1-scores (0.43-0.47).

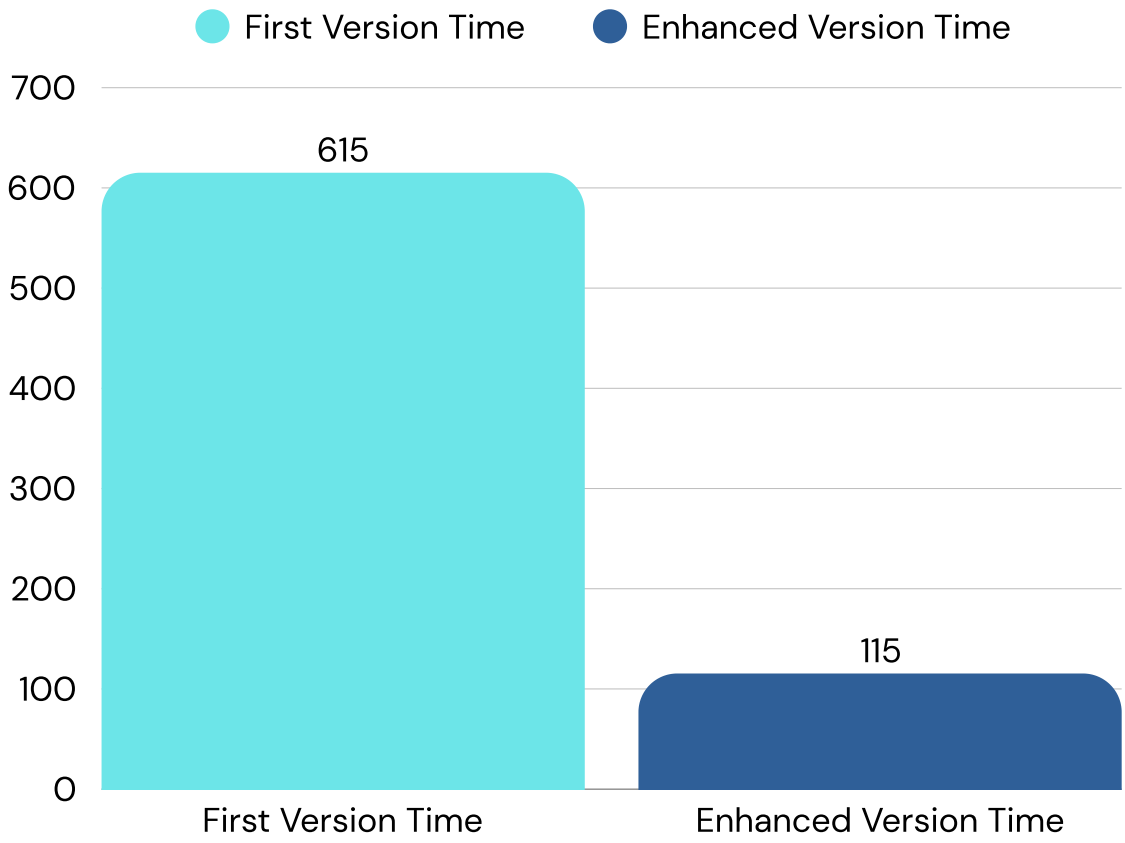
    The overall weighted average F1-score was 0.46-0.47 and it is moderate success in this multi-class classification task with these features and architecture.

5. **Optimization and Performance Analysis**
    - **Visualization of Optimization Comparison**: While direct GPU acceleration with CUDA was available due to library incompatibilities. The primary optimization will focus on parallelizing the most with the feature extraction. The comparison below shows that the highlights of the performance is very difference between the original sequential CPU execution and the parallel CPU execution by using joblib.

Metric	**First Version (Sequential CPU)**	**Enhanced Version (Parallel CPU)**	**Speedup**
Feature Extraction Time	615.14 seconds	115.40 seconds	~5.3x
Overall Execution Time	642.60 seconds	147.65 seconds	~4.35x
Peak Memory Usage	3146~ MB	3632~ MB	-

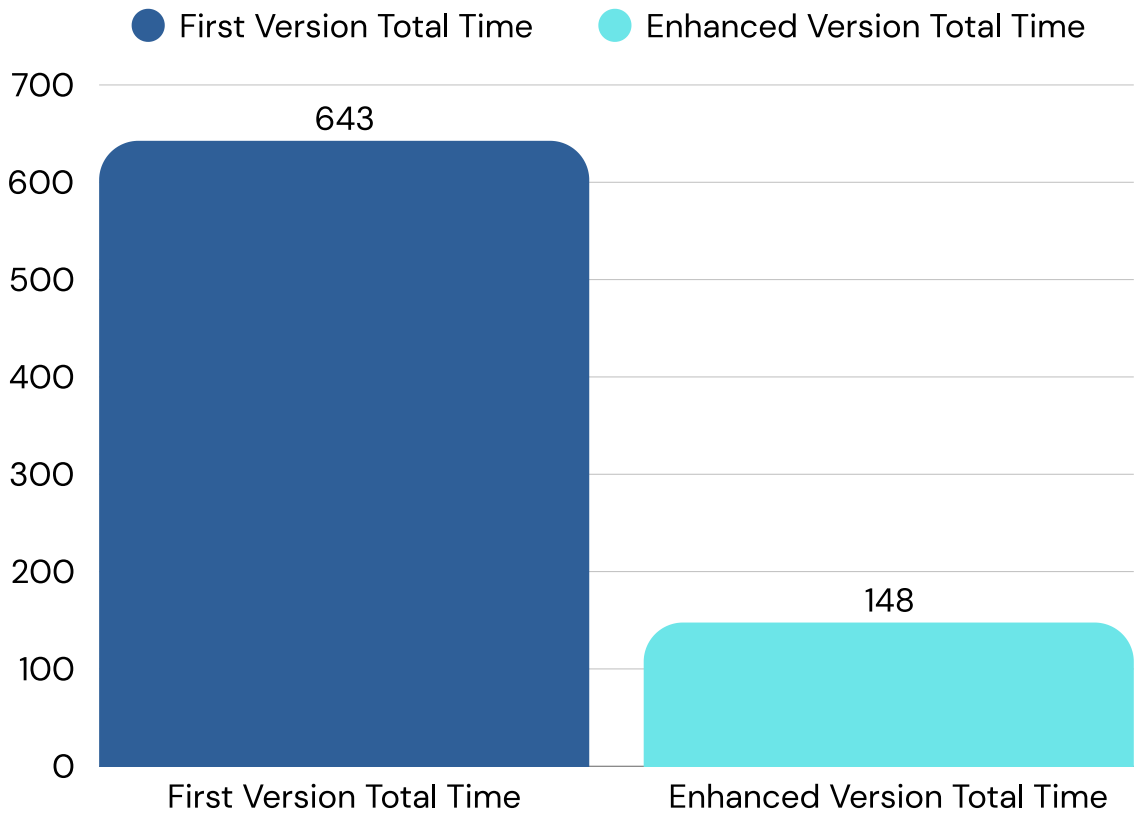
PERFORMANCE ANALYSIS

FEATURE EXTRACTION TIME COMPARISON



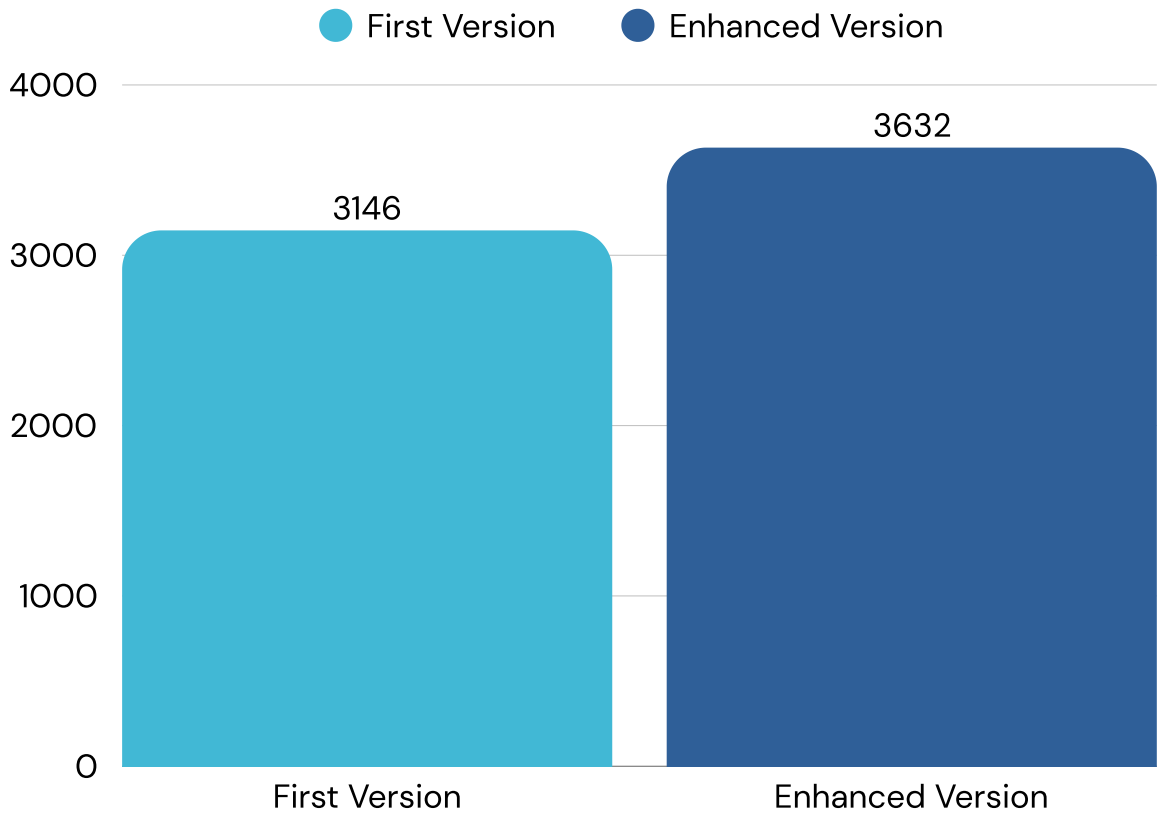
PERFORMANCE ANALYSIS

OVERALL EXECUTION TIME COMPARISON

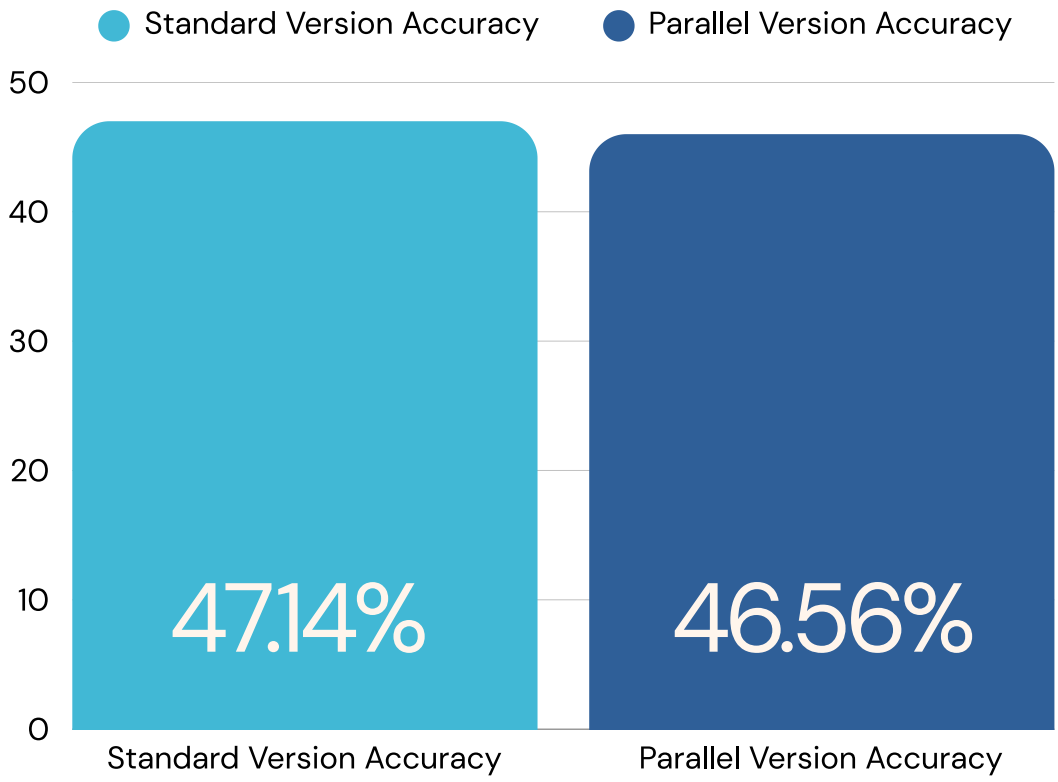


PERFORMANCE ANALYSIS

PEAK MEMORY USAGE



MODEL ACCURACY ANALYSIS



- **Selection of 13 MFCC Features**

  In this project, we chose to extract 13 Mel-Frequency Cepstral Coefficients (MFCCs) for each audio file because this number is a common standard in many speech and audio processing tasks as with a good balance model accuracy per optimization. We found that If we are extracting more than 20 so it will be hard to optimize and not necessary for the model accuracy because the model might capture finer details in vocal tract shape and it will affect to the optimization part.

  - **Increased Feature Vector Size:** Each MFCC coefficient will increases the size of the feature vector extracted per audio file from 500×13=6500 to 500×20=10000 features per sample.
  - **Impact on Processing Time:** A larger feature vector means more data needs to be processed during the feature extraction phase so the training phase of the machine learning model (MLP classifier) can lead to increased training time and memory consumption.
  - **Overfitting:** While more features could potentially improve accuracy by providing more information but they can also overfitting on the model training especially with a limited dataset size.

- **Optimization over Python's Global Interpreter Lock**

  One big thing that we had to figure out for optimization was dealing with Python's Global Interpreter Lock, or GIL. This became a real bottleneck for us because our feature extraction which uses libraries like librosa and numpy are using the CPU-heavy. Even if we tried using Python's threading to run things in parallel but the GIL will still limit us to just one thread running Python code. So, for these kinds of CPU-bound tasks, using threads doesn't really make things faster and sometimes it can even slow down a bit because of the extra work managing the threads. Our first version just went through each audio file one by one in a regular for loop, only using one CPU core for the feature extraction. Since we knew this part was taking the longest so we looked into how to do it in parallel. Standard Python threading wasn't going to cut it because of the GIL limiting those CPU execution. The joblib library can help us to do some specifically **joblib.Parallel** with **the backend='loky'**.

Instead of using threads. **The 'loky'** uses separate processes so they will have their own Python interpreter and memory space without waiting the GIL like threads do. This means multiple processes can run Python code on different CPU cores at the same time.

- By using **joblib.Parallel(n_jobs=-1, backend='loky')** to send the **extract_features function** to calls to multiple processes, we could use all the available CPU cores (n_jobs=-1) to work on different audio files all at once. This parallel methodology is for feature extraction which was the slowest part of our project. The result feature extraction time from 615.14 seconds to just 115.40 seconds.
- By the way, the GIL stopped us from getting true parallel speed with threads for our CPU work. But by switching to a multiprocessing method with joblib.

6. Tutorial Document

1. Download the Dataset:

- Go to the AI Research website: https://airesearch.in.th/releases/speech-emotion-dataset/
- Follow the instructions on their site to download the **THAI-SER Dataset**. This dataset contains Thai speech audio files (.flac) and a metadata file (emotion_label.json).

2. Set Up the Dataset Folder:

- create a subfolder named “Dataset”.
- Place the downloaded emotion_label.json file inside the Dataset folder.
- Unzip or extract the downloaded audio files.
  The notebook expects a structure like this:
  - Dataset/studio1-10/studio001/mic1/s001_mic1_001.flac
  - Dataset/zoom1-10/zoom001/mic/z001_mic_001.flac
  - *(...and so on for all other files)*

3. Prepare the Notebook:

- Open the cloudy-os.ipynb notebook in your Jupyter environment like Jupyter Notebook, VS Code or Google Colab.
- Checking that the path will set to: data_dir = './Dataset/'
- **Check the Path:** Make sure this path (./Dataset/) correctly points to the Dataset folder, where your notebook file is saved

4. Run the Notebook: Run the notebook to see the ecpected output

7. Legible Codes

Enhanced Version:

```
import os
import json
import numpy as np
import pandas as pd
import librosa
import librosa.display
import matplotlib.pyplot as plt
import seaborn as sns
from tqdm import tqdm
import soundfile as sf
import time
import joblib
from joblib import Parallel, delayed
import psutil

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.neural_network import MLPClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report
process = psutil.Process(os.getpid())

data_dir = './Dataset/'
metadata_file = os.path.join(data_dir, 'emotion_label.json')
emotion_map = {'โกรธ': 'angry', 'เศร้า': 'sad', 'สุข': 'happy', 'หงุดหงิด': 'frustrated',
'ปกติ': 'neutral'}
target_emotions = ['angry', 'sad', 'happy', 'frustrated', 'neutral']
sample_rate = 44100
n_mfcc = 13
max_len = 500
model_type = 'MLP'

def get_filepath(filename_stem):
    try:
        parts = filename_stem.split('_')
        if len(parts) < 3: return None
        rec_id, mic_type = parts[0], parts[1]
        rec_type = rec_id[0]
        rec_num = int(rec_id[1:])
        group_start = ((rec_num - 1) // 10) * 10 + 1
        group_end = group_start + 9
        if rec_type == 's':
            group_dir = f"studio{group_start}-{group_end}"
```

```
            session_dir = f"studio{rec_num:03d}"
            relative_path = os.path.join(group_dir, session_dir, mic_type)
        elif rec_type == 'z':
            group_dir = f"zoom{group_start}-{group_end}"
            session_dir = f"zoom{rec_num:03d}"
            relative_path = os.path.join(group_dir, session_dir, 'mic')
        else: return None
        full_path = os.path.join(data_dir, relative_path, filename_stem)
        if not os.path.splitext(full_path)[1]: full_path += ".flac"
        return full_path
    except Exception: return None

def extract_features(filepath, n_mfcc=n_mfcc, max_len=max_len, sr=sample_rate):
    try:
        audio, rate = librosa.load(filepath, sr=sr, res_type='kaiser_fast')
        mfccs = librosa.feature.mfcc(y=audio, sr=rate, n_mfcc=n_mfcc)
        if mfccs.shape[1] > max_len: mfccs = mfccs[:, :max_len]
        else:
            pad_width = max_len - mfccs.shape[1]
            mfccs = np.pad(mfccs, pad_width=((0, 0), (0, pad_width)),
mode='constant')
        return mfccs.T
    except Exception: return np.zeros((max_len, n_mfcc))

def process_metadata(metadata_file):
    try:
        with open(metadata_file, 'r', encoding='utf-8') as f: raw_meta =
json.load(f)
    except Exception: raw_meta = {}
    file_rows = []
    target_emotions_lower = [e.lower() for e in target_emotions]
    if isinstance(raw_meta, dict):
        for filename_stem, entry_list in raw_meta.items():
            if isinstance(entry_list, list) and entry_list:
                emotion_dict = entry_list[0]
                if isinstance(emotion_dict, dict):
                    emotion_label = emotion_dict.get('majority_emo')
                    if emotion_label and emotion_label.lower() in
target_emotions_lower:
                        full_path = get_filepath(filename_stem)
                        if full_path and os.path.exists(full_path):
                            file_rows.append({'stem': filename_stem, 'path':
full_path, 'label': emotion_label})
    return pd.DataFrame(file_rows)

def create_model(model_type='MLP'):
     if model_type == 'MLP':
```

```
        return MLPClassifier(hidden_layer_sizes=(100,), activation='relu',
solver='adam', alpha=0.0001, batch_size='auto', learning_rate='adaptive',
learning_rate_init=0.001, max_iter=300, shuffle=True, random_state=42,
early_stopping=True, n_iter_no_change=10, verbose=False)
    elif model_type == 'SVM':
        return SVC(C=1.0, kernel='rbf', gamma='scale', random_state=42,
verbose=False)
    else: raise ValueError("Invalid model_type")

def print_resources(stage_name, start_time, end_time):
    cpu_times = process.cpu_times()
    mem_info = process.memory_info()
    print(f"{stage_name} Time: {end_time - start_time:.2f}s")
    print(f"  CPU Times @ end: user={cpu_times.user:.2f}s,
system={cpu_times.system:.2f}s")
    print(f"  Memory Usage @ end: RSS={mem_info.rss / (1024**2):.2f} MB")

overall_start = time.time()
init_cpu_times = process.cpu_times()
init_mem_info = process.memory_info()
print(f"Initial CPU Times: user={init_cpu_times.user:.2f}s,
system={init_cpu_times.system:.2f}s")
print(f"Initial Memory Usage: RSS={init_mem_info.rss / (1024**2):.2f} MB")

start_meta = time.time()
df_meta = process_metadata(metadata_file)
end_meta = time.time()
print_resources("Metadata", start_meta, end_meta)

X = np.array([])
y = np.array([])
start_feat = time.time()
if not df_meta.empty:
    filepaths = df_meta['path'].tolist()
    results = Parallel(n_jobs=-1, backend='loky', verbose=0)(
        delayed(extract_features)(fp) for fp in tqdm(filepaths, desc="Parallel
FeatExtract")
        )

    if results and all(isinstance(r, np.ndarray) for r in results):
         X_list = [res.flatten() for res in results]
         X = np.vstack(X_list)
         y = df_meta['label'].values
    else:
        print("Parallel feature extraction failed or returned unexpected results.")
        X, y = np.array([]), np.array([])
else:
```

```
    print("Metadata empty, skipping feature extraction.")
end_feat = time.time()
print_resources("Feature Extraction", start_feat, end_feat)


start_prep = time.time()
if X.size > 0:
    le = LabelEncoder()
    y_encoded = le.fit_transform(y)
    X_train, X_temp, y_train, y_temp = train_test_split(X, y_encoded, test_size=0.3,
random_state=42, stratify=y_encoded)
    X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5,
random_state=42, stratify=y_temp)
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_val_scaled = scaler.transform(X_val)
    X_test_scaled = scaler.transform(X_test)
else:
    print("No data to preprocess.")
end_prep = time.time()
print_resources("Preprocessing", start_prep, end_prep)


start_train = time.time()
model = None
if 'X_train_scaled' in locals() and X_train_scaled.size > 0:
    model = create_model(model_type)
    model.fit(X_train_scaled, y_train)
else:
    print("No data to train.")
end_train = time.time()
print_resources("Training", start_train, end_train)


start_eval = time.time()
accuracy = 0
report = "Evaluation skipped."
if model is not None and 'X_test_scaled' in locals() and X_test_scaled.size > 0:
    y_pred = model.predict(X_test_scaled)
    accuracy = accuracy_score(y_test, y_pred)
    report = classification_report(y_test, y_pred, target_names=le.classes_,
zero_division=0)
else:
    print("Evaluation skipped.")
end_eval = time.time()
print_resources("Evaluation", start_eval, end_eval)
```

```
overall_end = time.time()
print(f"\n--- Parallel Version Total Wall Time: {overall_end - overall_start:.2f}s ---")
final_cpu_times = process.cpu_times()
final_mem_info = process.memory_info()
print(f"Final CPU Times: user={final_cpu_times.user:.2f}s, system={final_cpu_times.system:.2f}s")
print(f"Final Memory Usage: RSS={final_mem_info.rss / (1024**2):.2f} MB")
print(f"\nAccuracy: {accuracy:.4f}")
print(report)
```

First Version:

```
import os
import json
import numpy as np
import pandas as pd
import librosa
import librosa.display
import matplotlib.pyplot as plt
import seaborn as sns
from tqdm import tqdm
import soundfile as sf
import time
import joblib
import psutil # Import psutil

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.neural_network import MLPClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report

process = psutil.Process(os.getpid())

data_dir = './Dataset/'
metadata_file = os.path.join(data_dir, 'emotion_label.json')
emotion_map = {'โกรธ': 'angry', 'เศร้า': 'sad', 'สุข': 'happy', 'หงุดหงิด': 'frustrated', 'ปกติ': 'neutral'}
target_emotions = ['angry', 'sad', 'happy', 'frustrated', 'neutral']
sample_rate = 44100
n_mfcc = 13
max_len = 500
model_type = 'MLP'

def get_filepath(filename_stem):
```

```
    try:
        parts = filename_stem.split('_')
        if len(parts) < 3: return None
        rec_id, mic_type = parts[0], parts[1]
        rec_type = rec_id[0]
        rec_num = int(rec_id[1:])
        group_start = ((rec_num - 1) // 10) * 10 + 1
        group_end = group_start + 9
        if rec_type == 's':
            group_dir = f"studio{group_start}-{group_end}"
            session_dir = f"studio{rec_num:03d}"
            relative_path = os.path.join(group_dir, session_dir, mic_type)
        elif rec_type == 'z':
            group_dir = f"zoom{group_start}-{group_end}"
            session_dir = f"zoom{rec_num:03d}"
            relative_path = os.path.join(group_dir, session_dir, 'mic')
        else: return None
        full_path = os.path.join(data_dir, relative_path, filename_stem)
        if not os.path.splitext(full_path)[1]: full_path += ".flac"
        return full_path
    except Exception: return None

def extract_features(filepath, n_mfcc=n_mfcc, max_len=max_len, sr=sample_rate):
    try:
        audio, rate = librosa.load(filepath, sr=sr, res_type='kaiser_fast')
        mfccs = librosa.feature.mfcc(y=audio, sr=rate, n_mfcc=n_mfcc)
        if mfccs.shape[1] > max_len: mfccs = mfccs[:, :max_len]
        else:
            pad_width = max_len - mfccs.shape[1]
            mfccs = np.pad(mfccs, pad_width=((0, 0), (0, pad_width)),
mode='constant')
        return mfccs.T
    except Exception: return np.zeros((max_len, n_mfcc))

def process_metadata(metadata_file):
    try:
        with open(metadata_file, 'r', encoding='utf-8') as f: raw_meta =
json.load(f)
    except Exception: raw_meta = {}
    file_rows = []
    target_emotions_lower = [e.lower() for e in target_emotions]
    if isinstance(raw_meta, dict):
        for filename_stem, entry_list in raw_meta.items():
            if isinstance(entry_list, list) and entry_list:
                emotion_dict = entry_list[0]
                if isinstance(emotion_dict, dict):
                    emotion_label = emotion_dict.get('majority_emo')
```

```
                    if emotion_label and emotion_label.lower() in
target_emotions_lower:
                        full_path = get_filepath(filename_stem)
                        if full_path and os.path.exists(full_path):
                            file_rows.append({'stem': filename_stem, 'path':
full_path, 'label': emotion_label})
    return pd.DataFrame(file_rows)

def create_model(model_type='MLP'):
     if model_type == 'MLP':
        return MLPClassifier(hidden_layer_sizes=(100,), activation='relu',
solver='adam', alpha=0.0001, batch_size='auto', learning_rate='adaptive',
learning_rate_init=0.001, max_iter=300, shuffle=True, random_state=42,
early_stopping=True, n_iter_no_change=10, verbose=False)
     elif model_type == 'SVM':
        return SVC(C=1.0, kernel='rbf', gamma='scale', random_state=42,
verbose=False)
     else: raise ValueError("Invalid model_type")

def print_resources(stage_name, start_time, end_time):
    cpu_times = process.cpu_times()
    mem_info = process.memory_info()
    print(f"{stage_name} Time: {end_time - start_time:.2f}s")
    print(f"  CPU Times @ end: user={cpu_times.user:.2f}s,
system={cpu_times.system:.2f}s")
    print(f"  Memory Usage @ end: RSS={mem_info.rss / (1024**2):.2f} MB")

overall_start = time.time()
init_cpu_times = process.cpu_times()
init_mem_info = process.memory_info()
print(f"Initial CPU Times: user={init_cpu_times.user:.2f}s,
system={init_cpu_times.system:.2f}s")
print(f"Initial Memory Usage: RSS={init_mem_info.rss / (1024**2):.2f} MB")


start_meta = time.time()
df_meta = process_metadata(metadata_file)
end_meta = time.time()
print_resources("Metadata", start_meta, end_meta)


X_list = []
y_list = []
start_feat = time.time()
if not df_meta.empty:
    for index, row in tqdm(df_meta.iterrows(), total=len(df_meta), desc="Standard
FeatExtract"):
        features = extract_features(row['path'])
```

```
        X_list.append(features.flatten())
        y_list.append(row['label'])
    X = np.array(X_list)
    y = np.array(y_list)
else:
    X, y = np.array([]), np.array([])
    print("Metadata empty, skipping feature extraction.")
end_feat = time.time()
print_resources("Feature Extraction", start_feat, end_feat)


start_prep = time.time()
if X.size > 0:
    le = LabelEncoder()
    y_encoded = le.fit_transform(y)
    X_train, X_temp, y_train, y_temp = train_test_split(X, y_encoded, test_size=0.3,
random_state=42, stratify=y_encoded)
    X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5,
random_state=42, stratify=y_temp)
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_val_scaled = scaler.transform(X_val)
    X_test_scaled = scaler.transform(X_test)
else:
    print("No data to preprocess.")
end_prep = time.time()
print_resources("Preprocessing", start_prep, end_prep)


start_train = time.time()
model = None
if 'X_train_scaled' in locals() and X_train_scaled.size > 0:
    model = create_model(model_type)
    model.fit(X_train_scaled, y_train)
else:
    print("No data to train.")
end_train = time.time()
print_resources("Training", start_train, end_train)


start_eval = time.time()
accuracy = 0
report = "Evaluation skipped."
if model is not None and 'X_test_scaled' in locals() and X_test_scaled.size > 0:
    y_pred = model.predict(X_test_scaled)
    accuracy = accuracy_score(y_test, y_pred)
    report = classification_report(y_test, y_pred, target_names=le.classes_,
zero_division=0)
```

```
else:
    print("Evaluation skipped.")
end_eval = time.time()
print_resources("Evaluation", start_eval, end_eval)


overall_end = time.time()
print(f"\n--- Standard Version Total Wall Time: {overall_end - overall_start:.2f}s ---")
final_cpu_times = process.cpu_times()
final_mem_info = process.memory_info()
print(f"Final CPU Times: user={final_cpu_times.user:.2f}s, system={final_cpu_times.system:.2f}s")
print(f"Final Memory Usage: RSS={final_mem_info.rss / (1024**2):.2f} MB")
print(f"\nAccuracy: {accuracy:.4f}")
print(report)
```

8. Expected Output

First Version

```
Initial CPU Times: user=3.09s, system=0.11s
Initial Memory Usage: RSS=209.91 MB
Metadata Time: 0.39s
  CPU Times @ end: user=3.44s, system=0.16s
  Memory Usage @ end: RSS=257.47 MB
Standard FeatExtract: 100%|██████████| 25185/25185
[10:14<00:00, 40.95it/s]
Feature Extraction Time: 615.14s
  CPU Times @ end: user=6484.82s, system=9.67s
  Memory Usage @ end: RSS=1700.66 MB
Preprocessing Time: 1.41s
  CPU Times @ end: user=6485.95s, system=9.94s
  Memory Usage @ end: RSS=3137.56 MB
Training Time: 25.62s
  CPU Times @ end: user=6752.82s, system=10.42s
  Memory Usage @ end: RSS=3142.91 MB
Evaluation Time: 0.04s
  CPU Times @ end: user=6753.26s, system=10.42s
  Memory Usage @ end: RSS=3145.54 MB

--- Standard Version Total Wall Time: 642.60s ---
Final CPU Times: user=6753.26s, system=10.42s
Final Memory Usage: RSS=3145.54 MB

Accuracy: 0.4714
              precision    recall  f1-score   support

       Angry       0.48      0.42      0.44       467
  Frustrated       0.43      0.51      0.47      1180
       Happy       0.44      0.45      0.44       610
     Neutral       0.56      0.54      0.55      1104
         Sad       0.39      0.28      0.32       417

    accuracy                           0.47      3778
   macro avg       0.46      0.44      0.45      3778
weighted avg       0.47      0.47      0.47      3778
```

Enchanced Version

```
Initial CPU Times: user=3.09s, system=0.12s
Initial Memory Usage: RSS=210.20 MB
Metadata Time: 0.35s
  CPU Times @ end: user=3.39s, system=0.17s
  Memory Usage @ end: RSS=257.81 MB
Parallel FeatExtract: 100%|██████████| 25185/25185
[01:53<00:00, 221.17it/s]
Feature Extraction Time: 115.40s
  CPU Times @ end: user=20.83s, system=2.46s
  Memory Usage @ end: RSS=2145.20 MB
Preprocessing Time: 1.56s
  CPU Times @ end: user=21.90s, system=2.95s
  Memory Usage @ end: RSS=3582.18 MB
Training Time: 30.29s
  CPU Times @ end: user=337.88s, system=3.23s
  Memory Usage @ end: RSS=3629.14 MB
Evaluation Time: 0.05s
  CPU Times @ end: user=338.42s, system=3.23s
  Memory Usage @ end: RSS=3631.64 MB

--- Parallel Version Total Wall Time: 147.65s ---
Final CPU Times: user=338.42s, system=3.23s
Final Memory Usage: RSS=3631.64 MB

Accuracy: 0.4656
              precision    recall  f1-score   support

       Angry       0.48      0.44      0.46       467
  Frustrated       0.45      0.42      0.43      1180
       Happy       0.44      0.41      0.43       610
     Neutral       0.51      0.62      0.56      1104
         Sad       0.38      0.31      0.34       417

    accuracy                           0.47      3778
   macro avg       0.45      0.44      0.44      3778
weighted avg       0.46      0.47      0.46      3778
```

9. Conclusion

A machine learning from Thai speech emotion recognition was successfully implemented using MFCC features and an MLP classifier with the test accuracy of approximately 47%. The sequential feature extraction process is doing a bottleneck computation. By implementing parallel feature extraction using joblib. The overall execution time was reduced by approximately 77% (from ~10.7 minutes to ~2.5 minutes) without affect with the model's predictive accuracy. This can help us to minimize the time consume by using the parallelization for improving the efficiency of computationally in machine learning of audio analysis and It can save more time on large datasets. While achieved accuracy on the complexity of the task. In the future work we could explore alternative feature sets and trying different model architectures like CNNs, LSTMs and hyperparameter optimization to potentially enhance classification for better performance.

10. GitHub repository for source code and performance result:

https://github.com/Cloudy-s-OS/Audio-Emotion-Analysis