In the name of God

Iran University of Science & Technology

**Mechanical Engineering School**

# The Great Artificial Intelligence Competition


**Team Members:**
**Ali Badrloo**
**Mahdi Alikhani**
**Mohammadjavad Ghazikhani**

**Professor's name:**
**Seyyed Hasan Zabihifar**

**Mentors' names:**
**Mohammad Mahdi Khademi**
**Mohammad Hosein Cheraghi**
**Negar Naghavian**
**Saba Abbaszadeh Montazeri**


**Fall 2024**

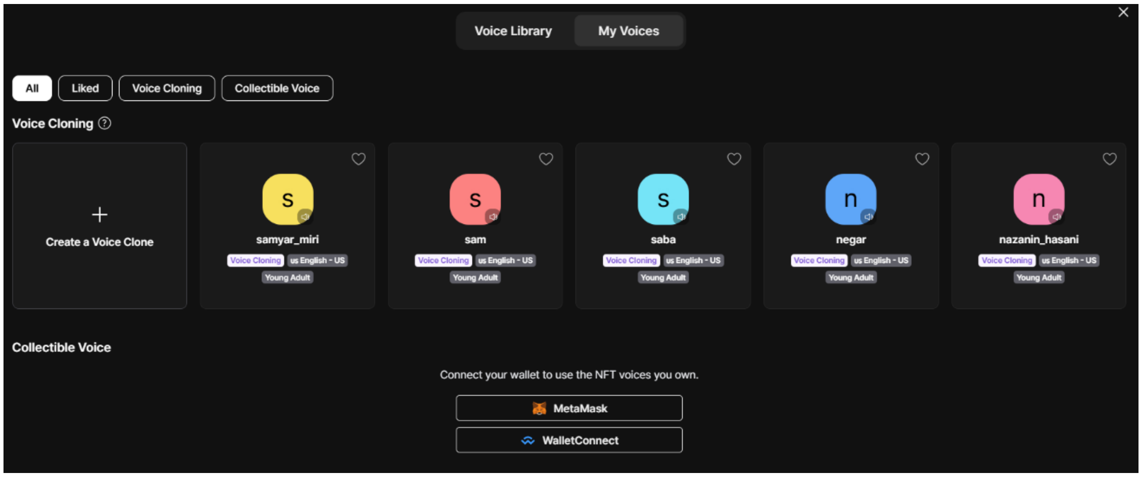
# The Great Artificial Intelligence Competition

The goal of this competition is to design and train a machine learning model for audio processing using the provided dataset. This dataset includes a collection of audio files recorded by different individuals, each uttering specific sentences. Your task is to develop a model that, upon receiving a new audio file from the same individuals (containing a different sentence), can accurately determine which person the voice belongs to. In other words, the model should be capable of identifying the speaker's identity (Speaker Identification) with high accuracy. At the end, the model that performs better in the evaluation metrics and achieves the highest accuracy will advance to the next stage of the competition.

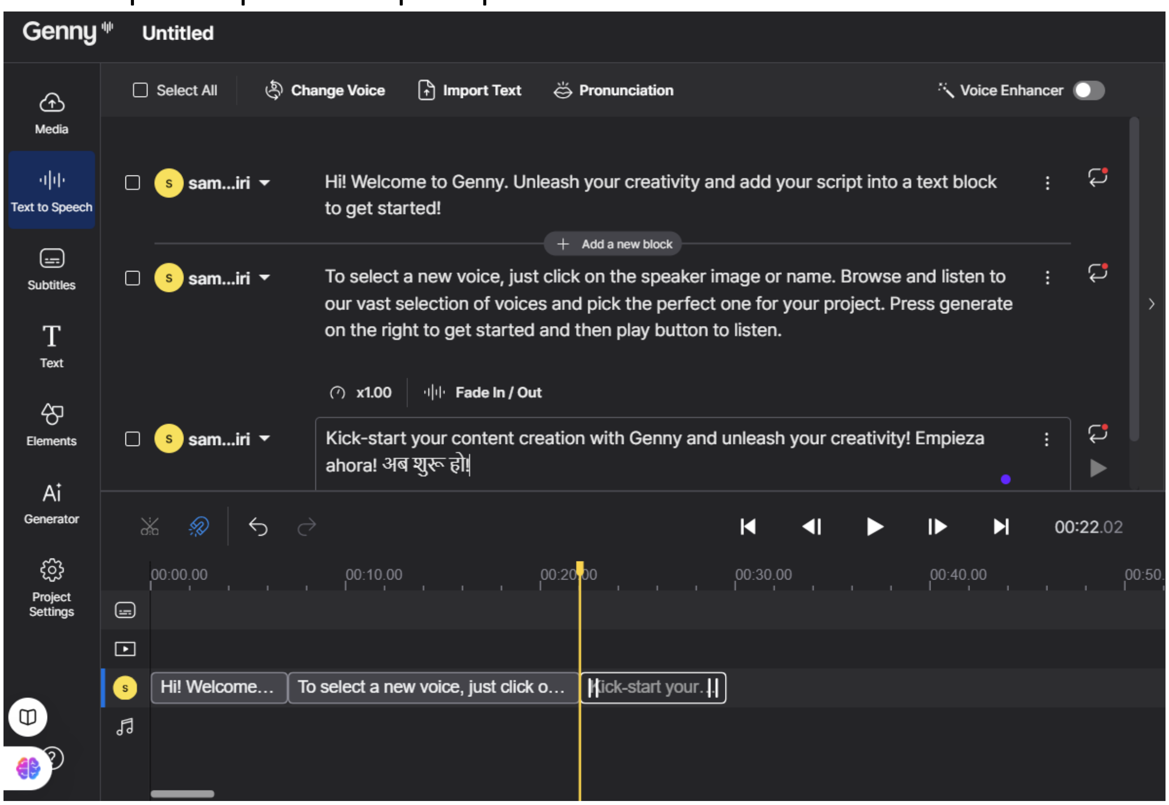
## Table of Contents

## Creating Deepfake Data

To increase the amount of data, we use deepfake techniques to generate similar audio files.




We generate deepfake data using **genny.lovo.ai**. In the **My Voices** section, we use the **Create a Voice Clone** feature to build a voice model for each individual. This process requires up to 4 sample input files to create the voice model.




In this section, we also utilize **text-to-speech (TTS)** technology. By using the voice models created earlier, the desired text is read aloud, generating synthetic audio files.

**Changing Data Format**

The dataset files for the competition are in **m4a** format, while the deepfake files are in **mp3** format. To utilize audio processing libraries effectively, we first need to convert the file formats to **wav**.

```
import os
from pydub import AudioSegment
from google.colab import drive

# Mount Google Drive
drive.mount('/content/drive', force_remount=True)

# Input and output folder paths
input_folder = '/content/drive/My Drive/Train_wav/'
output_folder = '/content/drive/My Drive/output_deepfaketoo/'

# Create output folder if not exists
os.makedirs(output_folder, exist_ok=True)

# Walk through all subdirectories and files
for root, dirs, files in os.walk(input_folder):
    for file_name in files:
        if file_name.endswith(".mp3"):
            # Build full input and output file paths
            input_file = os.path.join(root, file_name)
            relative_path = os.path.relpath(root, input_folder)
            output_subfolder = os.path.join(output_folder, relative_path)
            os.makedirs(output_subfolder, exist_ok=True)
            output_file = os.path.join(output_subfolder,
file_name.replace(".mp3", ".wav"))

            try:
                # Load MP3 and export as WAV
                audio = AudioSegment.from_mp3(input_file)
                audio.export(output_file, format="wav")
                print(f"Converted: {input_file} -> {output_file}")
            except Exception as e:
                print(f"Error converting {input_file}: {e}")
```

Using this code snippet, the file formats are converted and saved.

**Trimming Data**

To increase the amount of data and prevent the model from extracting more complex features, we convert the audio files into 1-second segments.

```
from google.colab import drive
import os

import os
from pydub import AudioSegment

# Mount Google Drive
drive.mount('/content/drive')



import os
from pydub import AudioSegment

# Function to trim audio files
def trim_audio_files(main_dir, output_dir, part_duration_ms):
    # Create output directory if it doesn't exist
    os.makedirs(output_dir, exist_ok=True)

    # Loop through each class folder in the main directory
    for class_folder in os.listdir(main_dir):
        class_path = os.path.join(main_dir, class_folder)

        # Only process directories (class folders)
        if os.path.isdir(class_path):
            # Create output folder for each class
            class_output_dir = os.path.join(output_dir, class_folder)
            os.makedirs(class_output_dir, exist_ok=True)

            # Loop through each audio file in the class folder
            for audio_file in os.listdir(class_path):
                audio_path = os.path.join(class_path, audio_file)

                # Only process .wav files
                if audio_file.endswith(".wav"):
                    print(f"Processing file: {audio_file}")
                    audio = AudioSegment.from_wav(audio_path)
                    total_duration = len(audio)  # Duration in
milliseconds

                    # Split the audio into parts of specified duration
```

```
                    for i in range(0, total_duration, part_duration_ms):
                        start_time = i
                        end_time = min(i + part_duration_ms, 
total_duration)  # Ensure last segment doesn't exceed audio length
                        part = audio[start_time:end_time]

                        # Generate output file name for each part
                        part_name = f"{audio_file.split('.')[0]}_part_{i 
// part_duration_ms}.wav"
                        part_path = os.path.join(class_output_dir, 
part_name)
                        part.export(part_path, format="wav")
                        print(f"Saved part: {part_name}")
    print("All audio files have been trimmed and saved.")

# Path to the main files
main_dir = '/content/drive/My Drive/merged data'

# Output folder
output_dir = '/content/drive/My Drive/merged data trimmed'

# Duration of each segment (in milliseconds)
part_duration = 1000  # 1000 ms = 1 s

# Run the trimming function
trim_audio_files(main_dir, output_dir, part_duration)
```

Using this code snippet, each audio file is divided into 1-second segments and saved.

## Determining Data Location and Classes

```
import os
import librosa
import numpy as np
import tensorflow as tf
from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D, Flatten, Dense
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from sklearn.model_selection import train_test_split
from tensorflow.keras.utils import to_categorical
from tensorflow.image import resize
```

Importing the required libraries.

```
# Define your folder structure
data_dir = 'trimmed_audio'
classes = ['ali_ghaderi', 'amin_taheri', 'faezeh_najafi', 'houman', 'kourosh',
            'mahdi', 'mahdi_joozdani', 'mani_hosseini', 'mehdi_gozali','mojtaba',
              'nazanin_hasani', 'negar', 'saba', 'sam', 'samyar_miri']
```

In this section, we assign the dataset path to the variable **data_dir** and list the classification class names in order in the **classes** list.

**Load and Preprocess Function**

```
import cv2

# Load and preprocess audio data
def load_and_preprocess_data(data_dir, classes, target_shape=(128, 128)):
    data = []
    labels = []

    for i, class_name in enumerate(classes):
        class_dir = os.path.join(data_dir, class_name)
        for filename in os.listdir(class_dir):
            if filename.endswith('.wav'):

                file_path = os.path.join(class_dir, filename)
                #Pre-Processing , trimming(scilence and noise)
                y, sr = librosa.load(file_path, sr=None)
                y_trimmed, _ = librosa.effects.trim(y, top_db=30)

                # Extract mel spectrogram
                S = librosa.feature.melspectrogram(y=y_trimmed, sr=sr, n_mels=128
* 5)
                S_db_mel = librosa.amplitude_to_db(S, ref=np.max)

                # Normalize the spectrogram
                S_db_mel_normalized = (S_db_mel - np.min(S_db_mel)) /
(np.max(S_db_mel) - np.min(S_db_mel))

                # Resize the spectrogram to fixed size
                fixed_height = 128
                fixed_width = 640
                S_db_mel_resized = cv2.resize(S_db_mel_normalized, (fixed_width,
fixed_height))

                mel_spectrogram = S_db_mel_resized
                mel_spectrogram = resize(np.expand_dims(mel_spectrogram, axis=-
1), target_shape)

                data.append(mel_spectrogram)
                labels.append(i)

    return np.array(data), np.array(labels)
```

In this section, a function is defined to load each file and perform preprocessing steps on the data, including removing silence and noise, extracting spectrograms, normalizing, and resizing.

## Loading and Splitting Data

```
# Split data into training and testing sets
data, labels = load_and_preprocess_data(data_dir, classes)
labels = to_categorical(labels, num_classes=len(classes))  # Convert labels to
one-hot encoding
X_train, X_test, y_train, y_test = train_test_split(data, labels, test_size=0.2,
random_state=42)
```

Now, the data and labels returned by the function are assigned to their respective variables, and the labels are transformed from the form 0, 1, 2, 3, ... into matrices of size 15x1, where all elements are zero except for the element at the index corresponding to the respective class number.

Then, using the train_test_split command, we split the data and labels into training and testing sets with a ratio of 0.2.

```
print("Shape of X_train:", X_train.shape)
print("Shape of X_test:", X_test.shape)
print("Shape of Y_train:", y_train.shape)
print("Shape of Y_test:", y_test.shape)
✓ 0.0s

Shape of X_train: (245, 128, 128, 1)
Shape of X_test: (62, 128, 128, 1)
Shape of Y_train: (245, 15)
Shape of Y_test: (62, 15)
```

```
print(y_train)
✓ 0.0s

[[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 1. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 1. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]
```

**Defining the Network**

```
# Create a neural network model
input_shape = X_train[0].shape
input_layer = Input(shape=input_shape)
x = Conv2D(32, (3, 3), activation='relu')(input_layer)
x = MaxPooling2D((2, 2))(x)
x = Conv2D(64, (3, 3), activation='relu')(x)
x = MaxPooling2D((2, 2))(x)
x = Flatten()(x)
x = Dense(64, activation='relu')(x)
output_layer = Dense(len(classes), activation='softmax')(x)
model = Model(input_layer, output_layer)
```

In this section, we define a CNN (Convolutional Neural Network) that includes a 3x3 filter, max pooling, another 3x3 filter, max pooling, followed by flattening, a hidden layer with 64 neurons, and an output layer with len(classes)=15 neurons and softmax activation function.

In the first two lines, we assign the shape of the 0th array in X_train to the variable input_shape, and in the next line, we use it as the shape for the input layer.

## Compile and Fit

```
model.compile(optimizer=Adam(learning_rate=0.001),
loss='categorical_crossentropy', metrics=['accuracy'])
```

In the compile section, we set the optimizer to **Adam**, the learning rate to **0.001**, the loss function to **categorical crossentropy**, and the metric to **accuracy**.

```
model.fit(X_train, y_train, epochs=12, batch_size=32, validation_data=(X_test,
y_test))
```

To fit the model, we set the number of epochs to **12** and the batch size to **32**.

```
model.fit(X_train, y_train, epochs=12, batch_size=32, validation_data=(X_test, y_test))
✓ 15.0s

Epoch 1/12
8/8 ━━━━━━━━━━━━━━━━━━━━ 3s 170ms/step - accuracy: 0.1116 - loss: 2.7078 - val_accuracy: 0.2419 - val_loss: 2.5586
Epoch 2/12
8/8 ━━━━━━━━━━━━━━━━━━━━ 1s 140ms/step - accuracy: 0.4128 - loss: 2.4001 - val_accuracy: 0.3710 - val_loss: 2.1925
Epoch 3/12
8/8 ━━━━━━━━━━━━━━━━━━━━ 1s 138ms/step - accuracy: 0.5821 - loss: 1.6258 - val_accuracy: 0.4194 - val_loss: 1.9824
Epoch 4/12
8/8 ━━━━━━━━━━━━━━━━━━━━ 1s 139ms/step - accuracy: 0.7910 - loss: 0.8515 - val_accuracy: 0.5000 - val_loss: 2.0258
Epoch 5/12
8/8 ━━━━━━━━━━━━━━━━━━━━ 1s 139ms/step - accuracy: 0.9093 - loss: 0.4053 - val_accuracy: 0.5000 - val_loss: 2.0552
Epoch 6/12
8/8 ━━━━━━━━━━━━━━━━━━━━ 1s 136ms/step - accuracy: 0.9569 - loss: 0.1946 - val_accuracy: 0.5484 - val_loss: 1.8927
Epoch 7/12
8/8 ━━━━━━━━━━━━━━━━━━━━ 1s 136ms/step - accuracy: 0.9837 - loss: 0.0828 - val_accuracy: 0.5645 - val_loss: 2.1374
Epoch 8/12
8/8 ━━━━━━━━━━━━━━━━━━━━ 1s 140ms/step - accuracy: 0.9936 - loss: 0.0644 - val_accuracy: 0.5484 - val_loss: 2.3757
Epoch 9/12
8/8 ━━━━━━━━━━━━━━━━━━━━ 1s 140ms/step - accuracy: 0.9973 - loss: 0.0350 - val_accuracy: 0.5161 - val_loss: 2.3611
Epoch 10/12
8/8 ━━━━━━━━━━━━━━━━━━━━ 1s 135ms/step - accuracy: 1.0000 - loss: 0.0375 - val_accuracy: 0.5645 - val_loss: 2.4926
Epoch 11/12
8/8 ━━━━━━━━━━━━━━━━━━━━ 1s 136ms/step - accuracy: 0.9788 - loss: 0.0543 - val_accuracy: 0.6935 - val_loss: 2.1723
Epoch 12/12
8/8 ━━━━━━━━━━━━━━━━━━━━ 1s 140ms/step - accuracy: 0.9952 - loss: 0.0209 - val_accuracy: 0.6290 - val_loss: 2.1280
```

**Final Accuracy**

```
Epoch 12/12
8/8 ━━━━━━━━━━━━━━━━━━━━ 1s 140ms/step - accuracy: 0.9952 - loss: 0.0209 - val_accuracy: 0.6290 - val_loss: 2.1280

<keras.src.callbacks.history.History at 0x2481601b3d0>
```

In the twelfth epoch, we achieve the observed values shown in the image above.

```
test_accuracy=model.evaluate(X_test,y_test,verbose=0)
print(test_accuracy[1])
✓ 0.1s

0.6290322542190552
```

**Machine Learning**

In the second code of our group, we used **XGBClassifier**, which is a machine learning algorithm.

The **XGBoost (eXtreme Gradient Boosting)** algorithm is one of the popular machine learning algorithms used for solving classification and regression problems. This algorithm is based on **Gradient Boosting** and has a high capability in handling large and complex datasets.

Gradient Boosting is a machine learning technique that iteratively combines simple models (usually decision trees) to create a more complex and accurate model. In each step, the new model is built on the errors of the previous models.

```
# Define the feature extractor (using the convolutional layers)
input_shape = X_train[0].shape
input_layer = Input(shape=input_shape)
x = Conv2D(32, (3, 3), activation='relu')(input_layer)
x = MaxPooling2D((2, 2))(x)
x = Conv2D(64, (3, 3), activation='relu')(x)
x = MaxPooling2D((2, 2))(x)
x = Flatten()(x)
feature_extractor = Model(inputs=input_layer, outputs=x)

# Extract features from training data
X_train_features = feature_extractor.predict(X_train)
X_test_features = feature_extractor.predict(X_test)

print("Extracted features (train):", X_train_features.shape

print("Extracted features (test):", X_test_features.shape)
```

First, we rewrite the previous network and extract the features of the data.

```
print("Extracted features (train):", X_train_features.shape)  # Example: (num_samples, num_features)
print("Extracted features (test):", X_test_features.shape)

8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 42ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 59ms/step
Extracted features (train): (245, 57600)
Extracted features (test): (62, 57600)
```

```
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score
import numpy as np

# Convert one-hot encoded labels to class indices
y_train_indices = np.argmax(y_train, axis=1)
y_test_indices = np.argmax(y_test, axis=1)

# Train XGBoost
xgb_model = XGBClassifier(n_estimators=200, max_depth=6, learning_rate=0.1,
use_label_encoder=False, eval_metric='mlogloss')
xgb_model.fit(X_train_features, y_train_indices)

# Predict and evaluate
y_pred = xgb_model.predict(X_test_features)
accuracy = accuracy_score(y_test_indices, y_pred)
print("XGBoost Test Accuracy:", accuracy)
```

We define the XGB model, set its arguments, fit it, and then evaluate it on the test data.

```
# Predict and evaluate
y_pred = xgb_model.predict(X_test_features)
accuracy = accuracy_score(y_test_indices, y_pred)
print("XGBoost Test Accuracy:", accuracy)

c:\Users\A\AppData\Local\Programs\Python\Python311\Li
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
XGBoost Test Accuracy: 0.7741935483870968
```