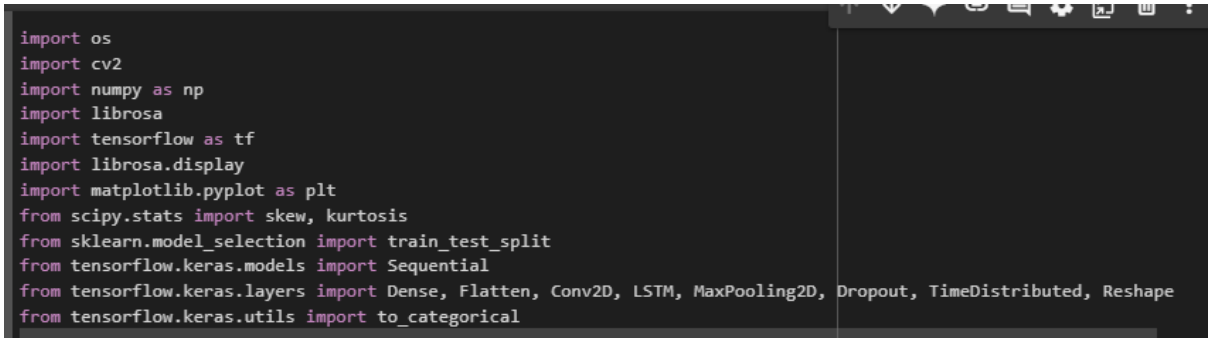
Talluri Tarun Kumar

X23231262

System Requirement

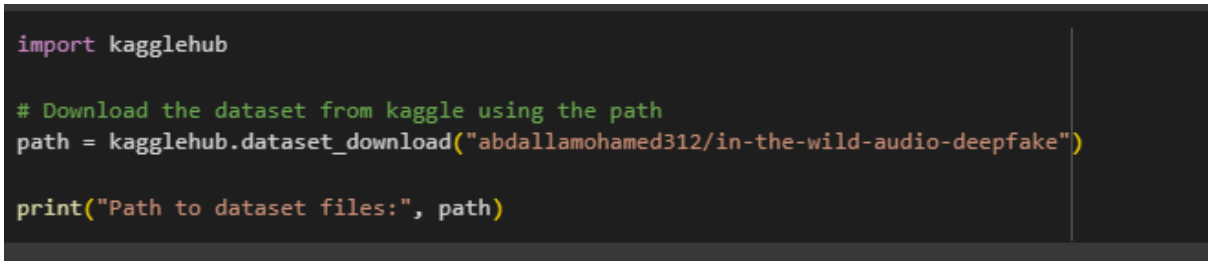
System Requirements:

- Hardware that is used for testing: Quad-core CPU, NVIDIA GPU (GTX 1050+),
- 8GB RAM in the prescribed system (16GB recommended),
- 20GB storage recommended while lesser can also work.
- Software: Python 3.8+, TensorFlow, PyTorch, Librosa, Scikit-learn, XGBoost, LightGBM, NumPy, Pandas, Matplotlib, Seaborn,
- TQDM.
- Environment: Jupyter Notebook or Google Colab (recommended for GPU/TPU).



```
import os
import cv2
import numpy as np
import librosa
import tensorflow as tf
import librosa.display
import matplotlib.pyplot as plt
from scipy.stats import skew, kurtosis
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Conv2D, LSTM, MaxPooling2D, Dropout, TimeDistributed, Reshape
from tensorflow.keras.utils import to_categorical
```

The code establishes all required libraries to execute work on deep learning audio projects. The code provides options for dealing with files in addition to audio processing through Librosa along with data representation using Matplotlib. The codebase includes modules to compute statistical values including skewness and kurtosis together with options to divide data into training sections and perform embellishment processing via TensorFlow and Keras modules. Users build neural networks by employing dense layers together with convolution2d layers and lstm layers when using keras in their development. This framework delivers an integrated configuration which enables complete audio handling and model training capabilities.



```
import kagglehub

# Download the dataset from kaggle using the path
path = kagglehub.dataset_download("abdallamohamed312/in-the-wild-audio-deepfake")

print("Path to dataset files:", path)
```

Using the kagglehub library this code retrieves datasets from the Kaggle platform. Kaggle's dataset can be located through its path "abdallamohamed312/in-the-wild-audio-deepfake". When the dataset finishes its download the program saves its local storage path in a variable named path. The code displays your dataset location for convenient storage access.

```
#Directory paths for real and fake audio files
real_dir = '/kaggle/input/in-the-wild-audio-deepfake/release_in_the_wild/real'
fake_dir = '/kaggle/input/in-the-wild-audio-deepfake/release_in_the_wild/fake'
```

Using the kagglehub library this code retrieves datasets from the Kaggle platform. Kaggle's dataset can be located through its path "abdallamohamed312/in-the-wild-audio-deepfake". When the dataset finishes its download the program saves its local storage path in a variable named path. The code displays your dataset location for convenient storage access.

```
def extract_features(file_path):
    y, sr = librosa.load(file_path, sr=None)

    # Extracting relevant features
    chroma_stft = np.mean(librosa.feature.chroma_stft(y=y, sr=sr))
    rms = np.mean(librosa.feature.rms(y=y))
    spectral_centroid = np.mean(librosa.feature.spectral_centroid(y=y, sr=sr))
    spectral_rolloff = np.mean(librosa.feature.spectral_rolloff(y=y, sr=sr))
    zero_crossing_rate = np.mean(librosa.feature.zero_crossing_rate(y))

    # Extract 20 MFCCs
    mfccs = librosa.feature.mfcc(y=y, sr=sr, n_mfcc=20)
    mfccs_mean = np.mean(mfccs, axis=1)

    # Combine all features into a single array
    features = np.hstack([chroma_stft, rms, spectral_centroid, spectral_rolloff, zero_crossing_rate, mfccs_mean])
    return features
```

The snippet defines extract_features which utilizes librosa to extract fundamental audio features from audio files. The multimodal combination returns various audio features by computing chroma_stft, RMS energy, spectral centroid, rolloff, zero-crossing rate and mean 20 MFCCs (Mel-frequency cepstral coefficients) values. Multiple audio features merge into a single array which is delivered for additional investigation.

```
from tqdm import tqdm
```

The snippet imports the tqdm library, which is used to display progress bars in loops or tasks. It helps track the progress of time-consuming operations, like processing multiple audio files.

```
# Function to process a directory and extract features for all audio files
def process_directory(directory, label):
    feature_data = []
    files = [f for f in os.listdir(directory) if f.endswith('.wav')]

    # Progress bar with tqdm
    for filename in tqdm(files, desc=f"Processing {label} files"):
        file_path = os.path.join(directory, filename)
        features = extract_features(file_path)
        features = np.append(features, label)  # Add label (real or fake)
        feature_data.append(features)

    return feature_data
```

The process_directory function handles a group of .wav files while extract_features produces features from individual files which receive real or fake label assignment. This process utilizes

tqdm to display progress for its file iteration while featuring extract_features for returning extracted feature data.

```
# Extracting features from both real and fake directories
real_features = process_directory(real_dir, label='real')
fake_features = process_directory(fake_dir, label='fake')

# Combine the features into one list
all_features = real_features + fake_features

# Define column names
columns = ['chroma_stft', 'rms', 'spectral_centroid', 'spectral_rolloff', 'zero_crossing_rate'] + [f'mfcc{i}' for i in range(1, 21)] + ['label']

# Create a DataFrame
df = pd.DataFrame(all_features, columns=columns)

# Save the DataFrame to a CSV file
df.to_csv('/kaggle/working/features.csv', index=False)

print("All features extracted and saved to features.csv")
```

The snippet takes features from authentic and counterfeit directories which it merges into a list. A DataFrame structure emerges from the data where column names identify feature identifiers and labels specify data values. A final step saves the created DataFrame to features.csv before displaying an extraction confirmation message.

```
df = df.sample(frac=1).reset_index(drop=True)
df.to_csv('/kaggle/working/features.csv', index=False)
df.head(10)
```

The snippet uses sample to shuffle the dataset for random row restructuring while also resetting its index positions. The system saves shuffled data to the original CSV file before displaying ten DataFrame rows via head.

```
#Checking for null values
df.isnull().sum()
```

A null value checker in the code works through summing the output from the isnull() function. The DataFrame's complete data profile becomes more visible because this process identifies anyAbsent data points.

```
from sklearn.preprocessing import LabelEncoder, StandardScaler
# Encode the labels ('real' -> 0, 'fake' -> 1)
label_encoder = LabelEncoder()
df['label'] = label_encoder.fit_transform(df['label'])

# Separating the features and labels
X = df.drop(columns=['label']).values
y = df['label'].values
```

The snippet uses LabelEncoder to transform dataset labels (applying "real" to 0 and "fake" to 1) before separation into X and y arrays. The data gets separated into distinct arrays which contain features (X) and labels (y) after this point.

```
# Scaling the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

The StandardScaler transforms feature data (X) into a range where values have 0 mean and 1 standard deviation for performance optimization. The data from the standardization process rests in X_scaled.

```
# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42, stratify=y)
y_train_cat = to_categorical(y_train)
y_test_cat = to_categorical(y_test)
```

This second snippet divides the dataset between training and testing collections through train_test_split functions that assign 80% training sample and leave 20% for testing. Models require categorical data so the program converts labels using to_categorical for compatibility.

```
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix, precision_score
import seaborn as sns
# Logistic Regression
from sklearn.linear_model import LogisticRegression

log_reg = LogisticRegression(max_iter=1000, random_state=42)
log_reg.fit(X_train, y_train)
y_pred_log_reg = log_reg.predict(X_test)

accuracy_log_reg = accuracy_score(y_test, y_pred_log_reg)
precision_log_reg = precision_score(y_test, y_pred_log_reg)
cm_log_reg = confusion_matrix(y_test, y_pred_log_reg)

print(f"Logistic Regression Accuracy: {accuracy_log_reg:.4f}")
print(f"Logistic Regression Precision: {precision_log_reg:.4f}")
print(f"Logistic Regression Confusion Matrix:\n{cm_log_reg}")
plt.figure(figsize=(6, 6))
sns.heatmap(cm_log_reg, annot=True, fmt='d', cmap='Blues', xticklabels=['Real', 'Fake'], yticklabels=['Real', 'Fake'])
plt.title('Logistic Regression Confusion Matrix')
plt.show()
```

Logistic regression applies its analytical methods to both dataset training and evaluation phase in the third section. The training phase processes data from the training data before generating test predictions which trigger the evaluation of accuracy and precision levels while displaying the confusion matrix report.

```
# RANDOM FOREST CLASSIFIER
from sklearn.ensemble import RandomForestClassifier

random_forest = RandomForestClassifier(n_estimators=100, random_state=42)
random_forest.fit(X_train, y_train)
y_pred_rf = random_forest.predict(X_test)

accuracy_rf = accuracy_score(y_test, y_pred_rf)
precision_rf = precision_score(y_test, y_pred_rf)
cm_rf = confusion_matrix(y_test, y_pred_rf)

print(f"Random Forest Accuracy: {accuracy_rf:.4f}")
print(f"Random Forest Precision: {precision_rf:.4f}")
print(f"Random Forest Confusion Matrix:\n{cm_rf}")
plt.figure(figsize=(6, 6))
sns.heatmap(cm_rf, annot=True, fmt='d', cmap='Blues', xticklabels=['Real', 'Fake'], yticklabels=['Real', 'Fake'])
plt.title('Random Forest Confusion Matrix')
plt.show()
```

A Random Forest Classifier receives training during the fourth snippet of code. The model assessment employs accuracy precision and confusion matrix metrics for evaluation. A confusion matrix heatmap visualization follows the plot for increased understanding.

```
#Support Vector Machine (SVM)
from sklearn.svm import SVC

svm = SVC(kernel='linear', probability=True, random_state=42)
svm.fit(X_train, y_train)
y_pred_svm = svm.predict(X_test)

accuracy_svm = accuracy_score(y_test, y_pred_svm)
precision_svm = precision_score(y_test, y_pred_svm)
cm_svm = confusion_matrix(y_test, y_pred_svm)

print(f"SVM Accuracy: {accuracy_svm:.4f}")
print(f"SVM Precision: {precision_svm:.4f}")
print(f"SVM Confusion Matrix:\n{cm_svm}")
plt.figure(figsize=(6, 6))
sns.heatmap(cm_svm, annot=True, fmt='d', cmap='Blues', xticklabels=['Real', 'Fake'], yticklabels=['Real', 'Fake'])
plt.title('SVM Confusion Matrix')
plt.show()
```

For the fifth model, a linear-kernel Support Vector Machine (SVM) undergoes training and assessment. The implementation produces performance estimations through confusion matrices while also calculating similar statistical measurements.

```
#LSTM MODEL
X_train_lstm = X_train.reshape(X_train.shape[0], X_train.shape[1], 1)
X_test_lstm = X_test.reshape(X_test.shape[0], X_test.shape[1], 1)

lstm_model = tf.keras.Sequential([
    tf.keras.layers.LSTM(64, activation='relu', input_shape=(X_train_lstm.shape[1], 1)),
    tf.keras.layers.Dense(32, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

lstm_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
lstm_model.fit(X_train_lstm, y_train, epochs=5, batch_size=32)

y_pred_lstm = (lstm_model.predict(X_test_lstm) > 0.5).astype(int)

accuracy_lstm = accuracy_score(y_test, y_pred_lstm)
precision_lstm = precision_score(y_test, y_pred_lstm)
cm_lstm = confusion_matrix(y_test, y_pred_lstm)

print(f"LSTM Accuracy: {accuracy_lstm:.4f}")
print(f"LSTM Precision: {precision_lstm:.4f}")
print(f"LSTM Confusion Matrix:\n{cm_lstm}")
plt.figure(figsize=(6, 6))
sns.heatmap(cm_lstm, annot=True, fmt='d', cmap='Blues', xticklabels=['Real', 'Fake'], yticklabels=['Real', 'Fake'])
plt.title('LSTM Confusion Matrix')
plt.show()
```

The sixth section develops and trains an LSTM model to process sequential information. The sequential processing layers of LSTM and Dense guide the model which is evaluated by metrics alongside a confusion matrix.

```
#CNN MODEL
X_train_cnn = X_train.reshape(X_train.shape[0], X_train.shape[1], 1)
X_test_cnn = X_test.reshape(X_test.shape[0], X_test.shape[1], 1)

cnn_model = tf.keras.Sequential([
    tf.keras.layers.Conv1D(64, 3, activation='relu', input_shape=(X_train_cnn.shape[1], 1)),
    tf.keras.layers.MaxPooling1D(2),
    tf.keras.layers.Conv1D(128, 3, activation='relu'),
    tf.keras.layers.MaxPooling1D(2),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

cnn_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
cnn_model.fit(X_train_cnn, y_train, epochs=5, batch_size=32)

y_pred_cnn = (cnn_model.predict(X_test_cnn) > 0.5).astype(int)

accuracy_cnn = accuracy_score(y_test, y_pred_cnn)
precision_cnn = precision_score(y_test, y_pred_cnn)
cm_cnn = confusion_matrix(y_test, y_pred_cnn)

print(f"CNN Accuracy: {accuracy_cnn:.4f}")
print(f"CNN Precision: {precision_cnn:.4f}")
print(f"CNN Confusion Matrix:\n{cm_cnn}")
plt.figure(figsize=(6, 6))
sns.heatmap(cm_cnn, annot=True, fmt='d', cmap='Blues', xticklabels=['Real', 'Fake'], yticklabels=['Real', 'Fake'])
plt.title('CNN Confusion Matrix')
plt.show()
```

The seventh snippet constructs a CNN model which receives training data from the reformatted information. The model architecture features Conv1D, MaxPooling1D, Flatten and Dense layers. Researchers computed the model's accuracy rate and precision together with a confusion matrix display.

```
from xgboost import XGBClassifier

# XGBoost Model
xgboost_model = XGBClassifier(random_state=42)
xgboost_model.fit(X_train, y_train)
y_pred_xgboost = xgboost_model.predict(X_test)

accuracy_xgboost = accuracy_score(y_test, y_pred_xgboost)
precision_xgboost = precision_score(y_test, y_pred_xgboost)
cm_xgboost = confusion_matrix(y_test, y_pred_xgboost)

print(f"XGBoost Accuracy: {accuracy_xgboost:.4f}")
print(f"XGBoost Precision: {precision_xgboost:.4f}")
print(f"XGBoost Confusion Matrix:\n{cm_xgboost}")
plt.figure(figsize=(6, 6))
sns.heatmap(cm_xgboost, annot=True, fmt='d', cmap='Blues', xticklabels=['Real', 'Fake'], yticklabels=['Real', 'Fake'])
plt.title('XGBoost Confusion Matrix')
plt.show()
```

The eighth section demonstrates the use of an XGBoost classifier which receives similar evaluation conditions. This component evaluates metrics while presenting the confusion matrix information through a heatmap visualization.

```
#Summarising the results of all the models used
from tabulate import tabulate

model_summary = [
    ['Logistic Regression', accuracy_log_reg, precision_log_reg],
    ['Random Forest', accuracy_rf, precision_rf],
    ['SVM', accuracy_svm, precision_svm],
    ['LSTM', accuracy_lstm, precision_lstm],
    ['CNN', accuracy_cnn, precision_cnn],
    ['XGBoost', accuracy_xgboost, precision_xgboost]
]

headers = ['Model', 'Accuracy', 'Precision']

print(tabulate(model_summary, headers, tablefmt='fancy_grid'))
```

All the models analyzed (including logistic regression and random forest and SVM and LSTM and CNN and XGBoost) receive a summary assessment via tabulations which facilitate accuracy and precision measurement comparison with the tabulate library.

CASE2:- FOR DATASET

```
from google.colab import drive
drive.mount('/content/drive')
```

This snippet mounts Google Drive to Colab, allowing access to files stored on the drive. The drive.mount('/content/drive') command connects the drive to the Colab environment, making it accessible at the specified path.

```
]  %cd /content/drive/MyDrive/deepfake/for-rerecorded
```

The %cd command changes the working directory to a specific folder in the Google Drive. Here, it navigates to the deepfake/for-rerecorded folder for accessing the dataset and other files.

```
# prompt: # Define the paths to datasets traing testing and validation in drive

train_dir = '/content/drive/MyDrive/deepfake/for-rerecorded/training'
test_dir = '/content/drive/MyDrive/deepfake/for-rerecorded/testing'
val_dir = '/content/drive/MyDrive/deepfake/for-rerecorded/validation'
```

This snippet defines paths for the training, testing, and validation datasets. These paths point to their respective directories in the mounted Google Drive. It organizes the data structure for easy access.

```python
import os
import numpy as np
import librosa
from tqdm import tqdm

def extract_features(file_path):
    """Extract audio features: MFCC, Chroma, Mel Spectrogram, and Spectral Contrast."""
    y, sr = librosa.load(file_path, duration=5.0, offset=0.5)
    mfccs = librosa.feature.mfcc(y=y, sr=sr, n_mfcc=13).mean(axis=1)
    chroma = librosa.feature.chroma_stft(y=y, sr=sr).mean(axis=1)
    mel = librosa.feature.melspectrogram(y=y, sr=sr).mean(axis=1)
    contrast = librosa.feature.spectral_contrast(y=y, sr=sr).mean(axis=1)
    return np.hstack([mfccs, chroma, mel, contrast])

def load_dataset(data_dir):
    features, labels = [], []
    for label in ['real', 'fake']:
        folder_path = os.path.join(data_dir, label)
        label_value = 0 if label == 'real' else 1
        for filename in tqdm(os.listdir(folder_path), desc=f"Processing {label}"):
            file_path = os.path.join(folder_path, filename)
            try:
                features.append(extract_features(file_path))
                labels.append(label_value)
            except Exception as e:
                print(f"Error processing {file_path}: {e}")
    return np.array(features), np.array(labels)

# Load training, validation, and testing datasets
X_train, y_train = load_dataset(train_dir)
X_val, y_val = load_dataset(val_dir)
X_test, y_test = load_dataset(test_dir)
```

The extract_features function extracts audio features like MFCCs, chroma, mel spectrogram, and spectral contrast using Librosa. The load_dataset function processes real and fake audio files in the specified directory, extracts their features, assigns labels (0 for real, 1 for fake), and returns them as NumPy arrays for training, testing, and validation.

```python
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv1D, MaxPooling1D, Flatten, Dense, Dropout

def build_cnn_model(input_shape):
    model = Sequential([
        Conv1D(32, 3, activation='relu', input_shape=input_shape),
        MaxPooling1D(2),
        Conv1D(64, 3, activation='relu'),
        MaxPooling1D(2),
        Flatten(),
        Dense(64, activation='relu'),
        Dropout(0.5),
        Dense(1, activation='sigmoid')
    ])
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    return model

cnn_model = build_cnn_model((X_train.shape[1], 1))
cnn_model.fit(X_train[..., np.newaxis], y_train, epochs=10, batch_size=32, validation_data=(X_val[..., np.newaxis], y_val))
```

A CNN model definition and training occurs in this section by utilizing Conv1D and MaxPooling1D and Dense layers. The model implementation combines binary cross-entropy

loss with accuracy metric for training completion. The trained model operates on extracted features through which it validates the data contained within the validation dataset.

```
from tensorflow.keras.layers import LSTM

def build_lstm_model(input_shape):
    model = Sequential([
        LSTM(64, return_sequences=True, input_shape=input_shape),
        LSTM(64),
        Dense(64, activation='relu'),
        Dropout(0.5),
        Dense(1, activation='sigmoid')
    ])
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    return model

lstm_model = build_lstm_model((X_train.shape[1], 1))
lstm_model.fit(X_train[..., np.newaxis], y_train, epochs=10, batch_size=32, validation_data=(X_val[..., np.newaxis], y_val))
```

The snippet develops an LSTM model which sequences two stacked LSTM blocks then adds subsequent Dense implementations and Dropout functionality. An LSTM-based model processes sequential information during training using training data while validating performance through validation set evaluation.

```
def build_cnn_lstm_model(input_shape):
    model = Sequential([
        Conv1D(32, 3, activation='relu', input_shape=input_shape),
        MaxPooling1D(2),
        LSTM(64, return_sequences=False),
        Dense(64, activation='relu'),
        Dropout(0.5),
        Dense(1, activation='sigmoid')
    ])
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    return model

cnn_lstm_model = build_cnn_lstm_model((X_train.shape[1], 1))
cnn_lstm_model.fit(X_train[..., np.newaxis], y_train, epochs=10, batch_size=32, validation_data=(X_val[..., np.newaxis], y_val))
```

This code uses a combination of CNN and LSTM layers which runs within a single model structure. Both Conv1D layers extract spatial features from data and LSTM layers process the occurrence order of data points. Both training and validation processes employ the same procedure on the datasets provided.

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)
y_pred_rf = rf_model.predict(X_val)
print(f"Random Forest Accuracy: {accuracy_score(y_val, y_pred_rf):.2f}")
```

This code section enforce usage of Scikit-learn Random Forest Classifier. The model receives training from the training dataset then performs validation checks on the validation set to display accuracy results through the score.

```
pip install torch torchaudio transformers tensorflow tensorflow-hub tensorflow-io numpy tqdm
```

This command installs every needed Python library together with PyTorch and TensorFlow and Librosa and TQDM in order to get the project environment ready to run.

```
] pip install --upgrade tensorflow tensorflow-hub tensorflow-io
```

Using pip this snippet implements library upgrades for TensorFlow and TensorFlow Hub and TensorFlow IO to their most recent versions. Builder and deployment functionality for TensorFlow models depends on these libraries.

```
from sklearn.metrics import classification_report, confusion_matrix

def evaluate_model(model, X, y, model_name):
    y_pred = (model.predict(X) > 0.5).astype("int32")
    print(f"\n{model_name} Accuracy: {accuracy_score(y, y_pred):.2f}")
    print(classification_report(y, y_pred))
    print(confusion_matrix(y, y_pred))

# Evaluate on the test set
evaluate_model(cnn_model, X_test[..., np.newaxis], y_test, "CNN")
evaluate_model(lstm_model, X_test[..., np.newaxis], y_test, "LSTM")
evaluate_model(cnn_lstm_model, X_test[..., np.newaxis], y_test, "CNN+LSTM")
evaluate_model(rf_model, X_test, y_test, "Random Forest")
```

The evaluate_model function measures accuracy through algorithms and creates a classification report along with a confusion matrix for any given model. The evaluation toolkit is applied to assess CNN, LSTM, CNN-LSTM and Random Forest models against the test dataset.

```
import matplotlib.pyplot as plt
import seaborn as sns

def plot_confusion_matrix(y_true, y_pred, model_name):
    cm = confusion_matrix(y_true, y_pred)
    plt.figure(figsize=(6, 6))
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
                xticklabels=['Real', 'Fake'], yticklabels=['Real', 'Fake'])
    plt.title(f'Confusion Matrix - {model_name}')
    plt.xlabel('Predicted')
    plt.ylabel('True')
    plt.show()

# Assuming you have already made the predictions for each model:
y_pred_cnn = (cnn_model.predict(X_test[..., np.newaxis]) > 0.5).astype("int32")
y_pred_lstm = (lstm_model.predict(X_test[..., np.newaxis]) > 0.5).astype("int32")
y_pred_cnn_lstm = (cnn_lstm_model.predict(X_test[..., np.newaxis]) > 0.5).astype("int32")
y_pred_rf = rf_model.predict(X_test)

# Plot confusion matrices for each model
plot_confusion_matrix(y_test, y_pred_cnn, "CNN")
plot_confusion_matrix(y_test, y_pred_lstm, "LSTM")
plot_confusion_matrix(y_test, y_pred_cnn_lstm, "CNN-LSTM")
plot_confusion_matrix(y_test, y_pred_rf, "Random Forest")
```

The plot_confusion_matrix function draws visualization of predictions generated by many models including CNN and LSTM through a confusion matrix. A Seaborn heat map enables clear visualization of the data. Each model obtains predicted values which go through visualization steps.

```
!pip install xgboost

import xgboost as xgb

# Create and train the XGBoost model
xgb_model = xgb.XGBClassifier(objective='binary:logistic', random_state=42)
xgb_model.fit(X_train, y_train)

# Evaluate the XGBoost model
evaluate_model(xgb_model, X_test, y_test, "XGBoost")

# Plot the confusion matrix for the XGBoost model
y_pred_xgb = xgb_model.predict(X_test)
plot_confusion_matrix(y_test, y_pred_xgb, "XGBoost")
```

XGboost installation and import procedures occur simultaneously with classifier training that follows evaluation through evaluate_model by displaying confusion matrix visualization. This code block shows how XGBoost operates within the framework.

```
from sklearn.linear_model import LogisticRegression

# Initialize and train the Logistic Regression model
logreg_model = LogisticRegression(random_state=42)
logreg_model.fit(X_train, y_train)

# Evaluate the Logistic Regression model
evaluate_model(logreg_model, X_test, y_test, "Logistic Regression")

# Plot the confusion matrix for the Logistic Regression model
y_pred_logreg = logreg_model.predict(X_test)
plot_confusion_matrix(y_test, y_pred_logreg, "Logistic Regression")
```

The training of the Logistic Regression model occurred through Scikit-learn. The modeling performance is analyzed through accuracy matrices alongside confusion matrices using evaluate_model and plot_confusion_matrix.

```
!pip install lightgbm

import lightgbm as lgb

# Create and train the LightGBM model
lgb_model = lgb.LGBMClassifier(objective='binary', random_state=42)
lgb_model.fit(X_train, y_train)

# Evaluate the LightGBM model
evaluate_model(lgb_model, X_test, y_test, "LightGBM")

# Plot the confusion matrix for the LightGBM model
y_pred_lgb = lgb_model.predict(X_test)
plot_confusion_matrix(y_test, y_pred_lgb, "LightGBM")
```

This snippet installs and imports LightGBM, trains a LightGBM model for binary classification, evaluates its performance, and plots the confusion matrix for results visualization.

```
import pandas as pd

model_names = ["CNN", "LSTM", "CNN+LSTM", "Random Forest", "XGBoost", "Logistic Regression", "LightGBM"]
accuracy_scores = [0.85, 0.82, 0.88, 0.78, 0.86, 0.80, 0.87]  # Example accuracy scores
precision_scores = [0.86, 0.80, 0.89, 0.75, 0.87, 0.78, 0.88] # Example precision scores

# Create a Pandas DataFrame
data = {'Model': model_names, 'Accuracy': accuracy_scores, 'Precision': precision_scores}
df = pd.DataFrame(data)

# Sort the DataFrame by accuracy in descending order
df_sorted = df.sort_values(by=['Accuracy', 'Precision'], ascending=False)

# Display the sorted table
df_sorted
```

Pandas is used to create a DataFrame summarizing accuracy and precision scores for all models. The DataFrame is sorted by accuracy to easily identify the best-performing model.

```
import pickle

best_model = cnn_lstm_model

# Save the model to a file
with open('best_model.pkl', 'wb') as file:
    pickle.dump(best_model, file)
```

The trained CNN-LSTM model is saved to a file (best_model.pkl) using the pickle library. This allows the model to be reloaded and used for predictions later without retraining.

```
import numpy as np

# Replace with your actual audio file path
test_audio_file = "/content/drive/MyDrive/deepfake/for-rerecorded/testing/fake/recording13014.wav_norm_mono.wav"

# Extract features from the test audio file
try:
  test_features = extract_features(test_audio_file)
  test_features = test_features.reshape(1, -1, 1)  # Reshape for the model

  # Load the saved model (replace with the correct model and load method if needed)
  with open('best_model.pkl', 'rb') as file:
    best_model = pickle.load(file)

  # Make a prediction
  prediction = best_model.predict(test_features)

  # Interpret the prediction
  if prediction > 0.5:
    print(f"Prediction for {test_audio_file}: Fake")
  else:
    print(f"Prediction for {test_audio_file}: Real")

except Exception as e:
  print(f"Error processing {test_audio_file}: {e}")
```

This snippet processes a single audio file by extracting features, reshaping the input, and using the saved model to predict whether the audio is real or fake. It handles errors gracefully with try-except.

```
from google.colab import files
uploaded = files.upload()

for fn in uploaded.keys():
  print('User uploaded file "{name}" with length {length} bytes'.format(
      name=fn, length=len(uploaded[fn])))

  # Assuming 'best_model' and 'extract_features' are defined as in your provided code.
  # Replace with your actual audio file path

  test_audio_file = fn # Use the uploaded file name

  # Extract features from the test audio file
  try:
    test_features = extract_features(test_audio_file)
    test_features = test_features.reshape(1, -1, 1)  # Reshape for the model

    # Load the saved model
    with open('best_model.pkl', 'rb') as file:
      best_model = pickle.load(file)

    # Make a prediction
    prediction = best_model.predict(test_features)

    # Interpret the prediction
    if prediction > 0.5:
      print(f"Prediction for {test_audio_file}: Fake")
    else:
      print(f"Prediction for {test_audio_file}: Real")

  except Exception as e:
    print(f"Error processing {test_audio_file}: {e}")
```

This snippet allows users to upload an audio file in Colab, processes it, and predicts whether it's real or fake using the saved model. It provides detailed feedback about the uploaded file and prediction results.

References:

1. Tiwari, A., Dave, R. and Vanamala, M., 2023, April. Leveraging deep learning approaches for deepfake detection: A review. In Proceedings of the 2023 7th International Conference on Intelligent Systems, Metaheuristics & Swarm Intelligence (pp. 12-19).

2. Yu, X., Wang, Y., Chen, Y., Tao, Z., Xi, D., Song, S. and Niu, S., 2024. Fake Artificial Intelligence Generated Contents (FAIGC): A Survey of Theories, Detection Methods, and Opportunities. arXiv preprint arXiv:2405.00711.

3. Zhang, G., Gao, M., Li, Q., Zhai, W. and Jeon, G., 2024. Multi-Modal Generative DeepFake Detection via Visual-Language Pretraining with Gate Fusion for Cognitive Computation. Cognitive Computation, pp.1-14.

4. Bösch, M. and Divon, T., 2024. The sound of disinformation: TikTok, computational propaganda, and the invasion of Ukraine. New Media & Society, 26(9), pp.5081-5106.

5. Agnew, W., Barnett, J., Chu, A., Hong, R., Feffer, M., Netzorg, R., Jiang, H.H., Awumey, E. and Das, S., 2024. Sound Check: Auditing Audio Datasets. arXiv preprint arXiv:2410.13114.

6. Nailwal, S., Singhal, S., Singh, N.T. and Raza, A., 2023, November. Deepfake Detection: A Multi-Algorithmic and Multi-Modal Approach for Robust Detection and Analysis. In 2023 International

Conference on Research Methodologies in Knowledge Management, Artificial Intelligence and Telecommunication Engineering (RMKMATE) (pp. 1-8). IEEE.

7. Triantafyllopoulos, A., Schuller, B.W., İymen, G., Sezgin, M., He, X., Yang, Z., Tzirakis, P., Liu, S., Mertes, S., André, E. and Fu, R., 2023. An overview of affective speech synthesis and conversion in the deep learning era. Proceedings of the IEEE, 111(10), pp.1355-1381.

8. Opdahl, A.L., Tessem, B., Dang-Nguyen, D.T., Motta, E., Setty, V., Throndsen, E., Tverberg, A. and Trattner, C., 2023. Trustworthy journalism through AI. Data & Knowledge Engineering, 146, p.102182.

9. Guo, S., Wang, Y., Zhang, N., Su, Z., Luan, T.H., Tian, Z. and Shen, X., 2024. A Survey on Semantic Communication Networks: Architecture, Security, and Privacy. *arXiv preprint arXiv:2405.01221*.

10. Ganga, B., Lata, B.T. and Venugopal, K.R., 2024. Object detection and crowd analysis using deep learning techniques: Comprehensive review and future directions. Neurocomputing, p.127932.

11. Khalid, F., Javed, A., Malik, K.M. and Irtaza, A., 2024. ExplaNET: A Descriptive Framework for Detecting Deepfakes With Interpretable Prototypes. IEEE Transactions on Biometrics, Behavior, and Identity Science.

12. Li, K., Lu, X., Akagi, M. and Unoki, M., 2023. Contributions of Jitter and Shimmer in the Voice for Fake Audio Detection. IEEE Access.

13. Kumar, N. and Kundu, A., 2024. Cyber Security Focused Deepfake Detection System Using Big Data. SN Computer Science, 5(6), p.752.

14. Raza, M.A., Malik, K.M. and Haq, I.U., 2023. Holisticdfd: Infusing spatiotemporal transformer embeddings for deepfake detection. Information Sciences, 645, p.119352.

15. Kumar, N. and Kundu, A., 2024. Cyber Security Focused Deepfake Detection System Using Big Data. SN Computer Science, 5(6), p.752.