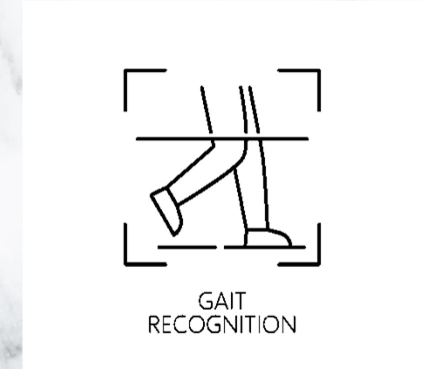
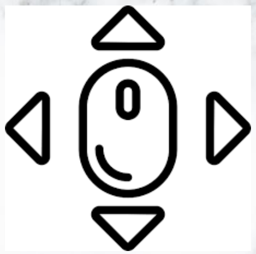
SecureAuth - Behavioral Biometrics for Enhanced Authentication Systems

24-25J-073





❑ Our Team

SUPERVISOR

Dr. Harinda Fernando

Assistant Professor

Faculty of Computing

CO-SUPERVISOR

Mr. Tharaniyawarma

Assistant Lecturer

Faculty of Computing

Madhubhashana H. N. D

IT21391668

CYBER SECURITY

Edirisinghe E.M.N

IT21340864

CYBER SECURITY

Anupama K.G. A

IT21345678

CYBER SECURITY

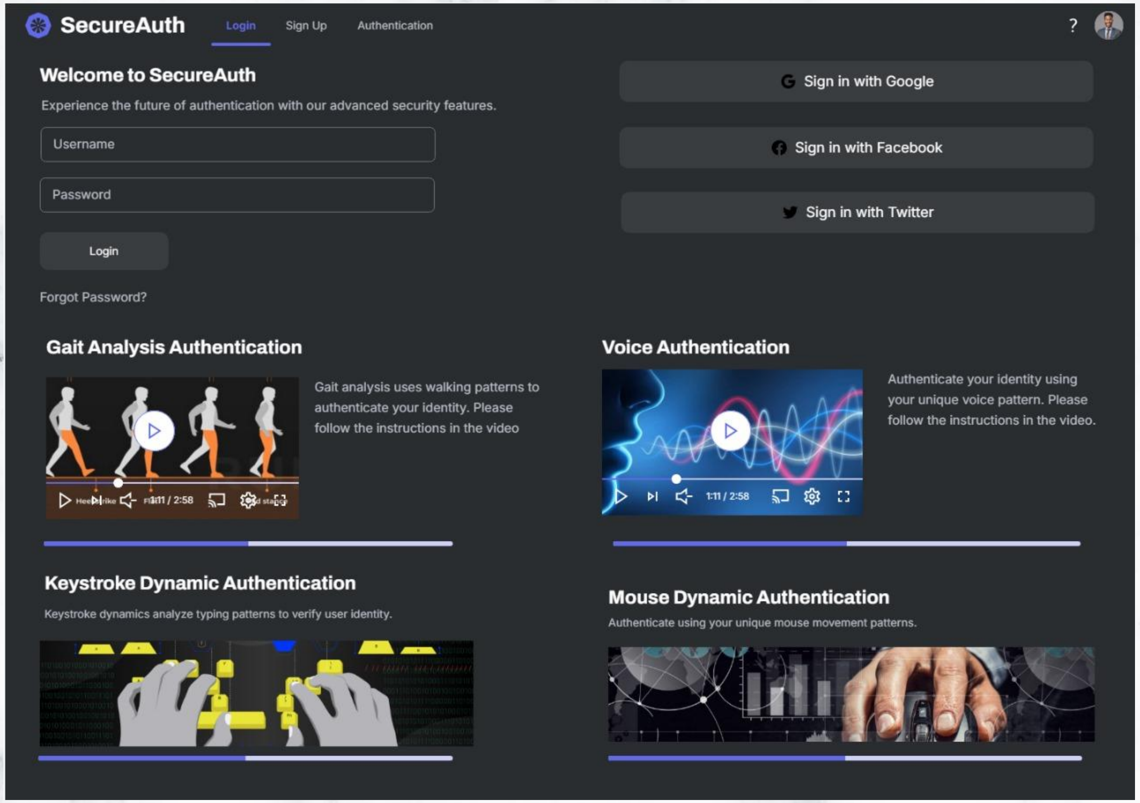
Rajapaksha R. P. K. D

IT21336072

CYBER SECURITY

SecureAuth

- **SecureAuth** is an advanced authentication solution that uses behavioral biometrics like gait, voice, typing patterns, and mouse movements to verify identity. With cutting-edge machine learning, it provides seamless, accurate, and secure access, making it ideal for protecting critical information and high-security environments.



❑ WHY **SecureAuth** IS IMPORTANT?

Enhanced Security: Protects sensitive data with multi-layered authentication.

Multi-Factor Authentication: Combines behavioral biometrics for stronger security.

Real-Time Authentication: Instant, seamless user verification.

Designed for High-Risk Environments: Ideal for government, finance, and healthcare.

Accurate & Reliable: Reduces false positives with hybrid authentication methods.

Future-Proof: Stays ahead of evolving cybersecurity threats.

❑ Research Objectives

✓ Primary Objective

To revolutionize user authentication by leveraging behavioral biometrics, offering a secure, seamless, and user-friendly alternative to traditional password systems.

❑ Research Objectives

✓ Secondary Objective

Gait Analysis

- Harness unique walking patterns to deliver an innovative and non-intrusive authentication method.

Mouse Dynamics

- Analyze natural mouse movements to enhance security without disrupting user experience.

Keystroke Dynamics

- Leverage typing patterns as an intuitive layer of identity verification.

Voice Biometric Authentication

- Utilize voice as a distinctive identifier, ensuring fast and reliable user verification..

Seamless Integration

- Provide an integrated solution that adapts to diverse environments and user needs.

Performance and Reliability

- Ensure the system performs consistently in real-world scenarios, offering high accuracy and resilience against breaches.

❑ Research Problem

Challenges with Traditional Biometric Authentication

- ➢ Vulnerable to spoofing and privacy concerns.
- ➢ Requires physical contact or proximity.

Limitations of Existing Gait Analysis Methods

- ➢ Often lack robustness and accuracy under diverse conditions.
- ➢ Need for improved feature extraction and modeling techniques.

Need for Robust and Accurate Behavioral Biometric Systems

- ➢ Behavioral biometrics offer non-intrusive and unique patterns.
- ➢ Potential to significantly enhance user authentication security.

❑ Research Question

How can behavioral biometrics, such as gait analysis, voice recognition, keystroke, and mouse dynamics, enhance the security and usability of authentication systems while maintaining user privacy and adaptability?

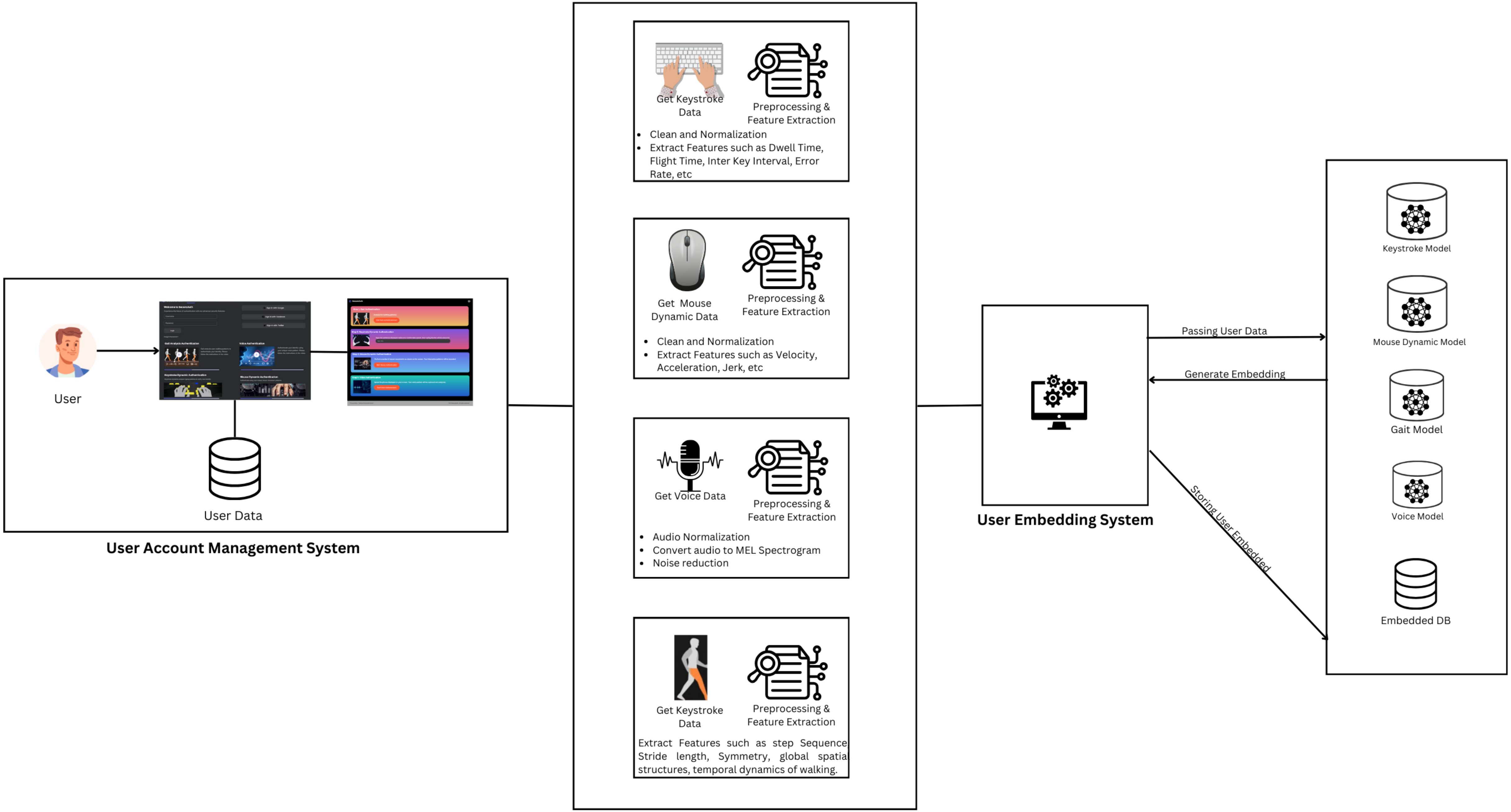
❑ Research Solution

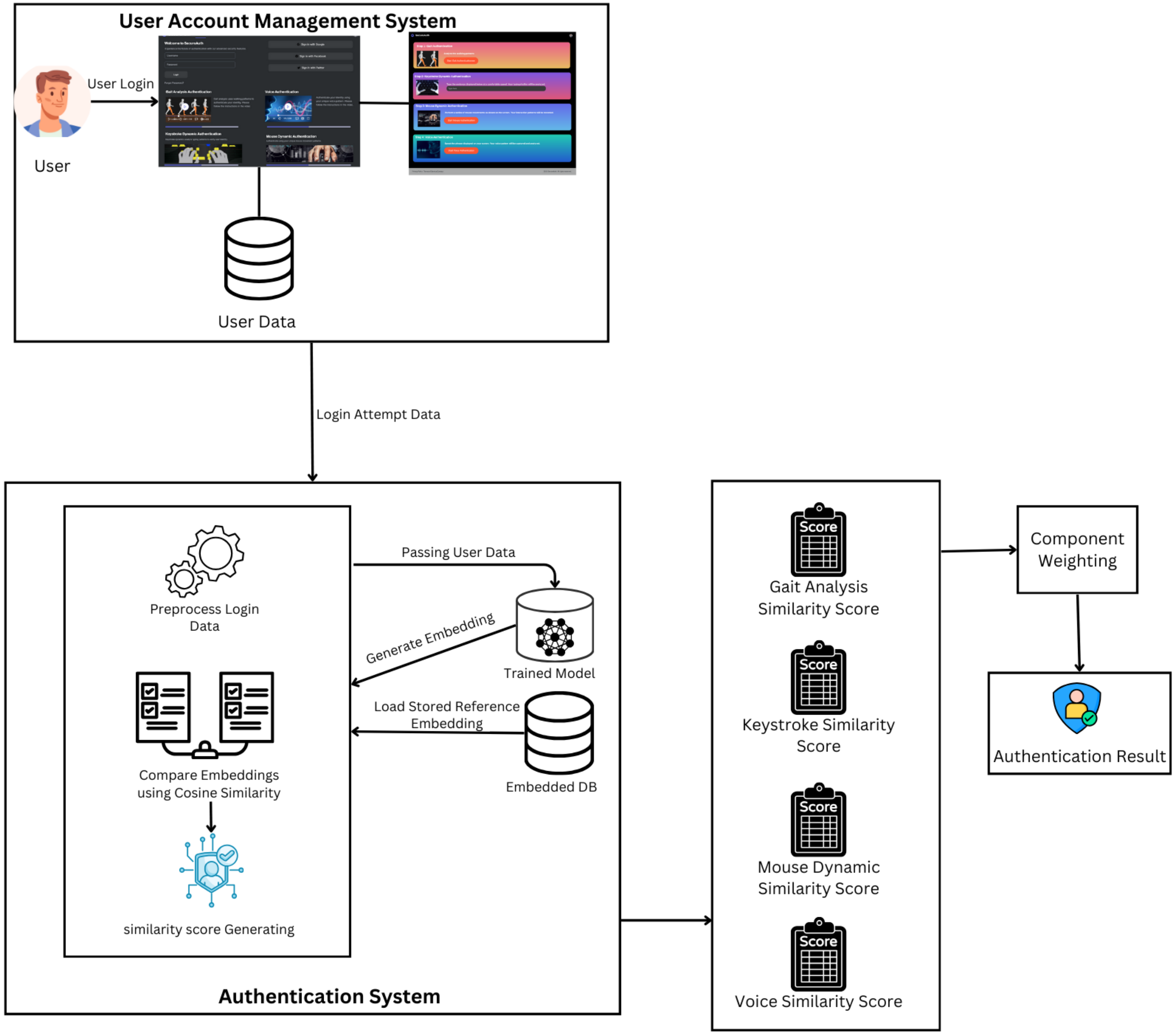
Develop **SecureAuth**, a multi-modal authentication system that combines gait, voice, keystroke, and mouse dynamics using advanced machine learning. It ensures robust security, user privacy, and seamless integration into high-security environments.

❑ Research Gap

Features/ Technologies	Scalability	Use of Online Datasets	Hybrid Model (CNN + RNN)	Specialized Hardware Required
Project X	✓	✗	✗	✗
Project Y	✓	✗	✗	✗
Project Z	✗	✗	✗	✓
SecureAuth	✓	✓	✓	✗

System Diagram


User
User Data
User Account Management System
Get Keystroke Data
Preprocessing & Feature Extraction
Clean and Normalization
Extract Features such as Dwell Time, Flight Time, Inter Key Interval, Error Rate, etc
Get Mouse Dynamic Data
Preprocessing & Feature Extraction
Clean and Normalization
Extract Features such as Velocity, Acceleration, Jerk, etc
Get Voice Data
Preprocessing & Feature Extraction
Audio Normalization
Convert audio to MEL Spectrogram
Noise reduction
Get Keystroke Data
Preprocessing & Feature Extraction
Extract Features such as step Sequence Stride length, Symmetry, global spatia structures, temporal dynamics of walking.
User Embedding System
Passing User Data
Generate Embedding
Storing User Embedded
Keystroke Model
Mouse Dynamic Model
Gait Model
Voice Model
Embedded DB


User Account Management System
User Login
User
User Data
Login Attempt Data
Preprocess Login Data
Passing User Data
Generate Embedding
Trained Model
Load Stored Reference Embedding
Embedded DB
Compare Embeddings using Cosine Similarity
similarity score Generating
Authentication System
Score
Gait Analysis Similarity Score
Score
Keystroke Similarity Score
Score
Mouse Dynamic Similarity Score
Score
Voice Similarity Score
Component Weighting
Authentication Result

IT21391668 | H.N.D. MADHUBHASHANA

BSc (Hons) in Information Technology Specialising in Cyber Security

Introduction to Gait Component

- Gait authentication uses unique walking patterns for secure, non-intrusive user verification. By analyzing these patterns with a CNN-GRU model, SecureAuth adds an extra layer of security. This system ensures both accuracy and privacy with advanced encryption, offering a seamless user experience for high-security applications.

Gait Component Objectives

Develop Hybrid Models: Combine CNN and GRU to capture both spatial and temporal features of walking.

Enhance Security: Use gait patterns for a unique, non-intrusive authentication method.

Improve Accuracy: Optimize model performance for real-world gait data.

Ensure Privacy: Maintain user data privacy with encrypted storage and processing.

Gait Component Sub-Objectives

Data Collection

- Capture high-quality Gait Energy Images (GEIs) for analysis.

Extract Features

- Improve techniques for extracting relevant gait features.

Train Model

- Fine-tune hybrid CNN-GRU models for faster processing and better accuracy.

Evaluate Model Performance

- Validate the model with diverse gait data to ensure robustness.

❑ Component Gap

Features/ Technologies	GEI	Use of Online Datasets	Hybrid Model (CNN + GRU)	Multi model integration
Project X	✓	✓	✗	✗
Project Y	✓	✗	✗	✗
Project Z	✗	✗	✗	✗
SecureAuth	✓	✓	✓	✓

Component Question

- How can a CNN-GRU model best extract spatial and temporal features from GEIs?
- What preprocessing ensures optimal GEI quality?
- How can we enhance accuracy in distinguishing unique gait patterns?

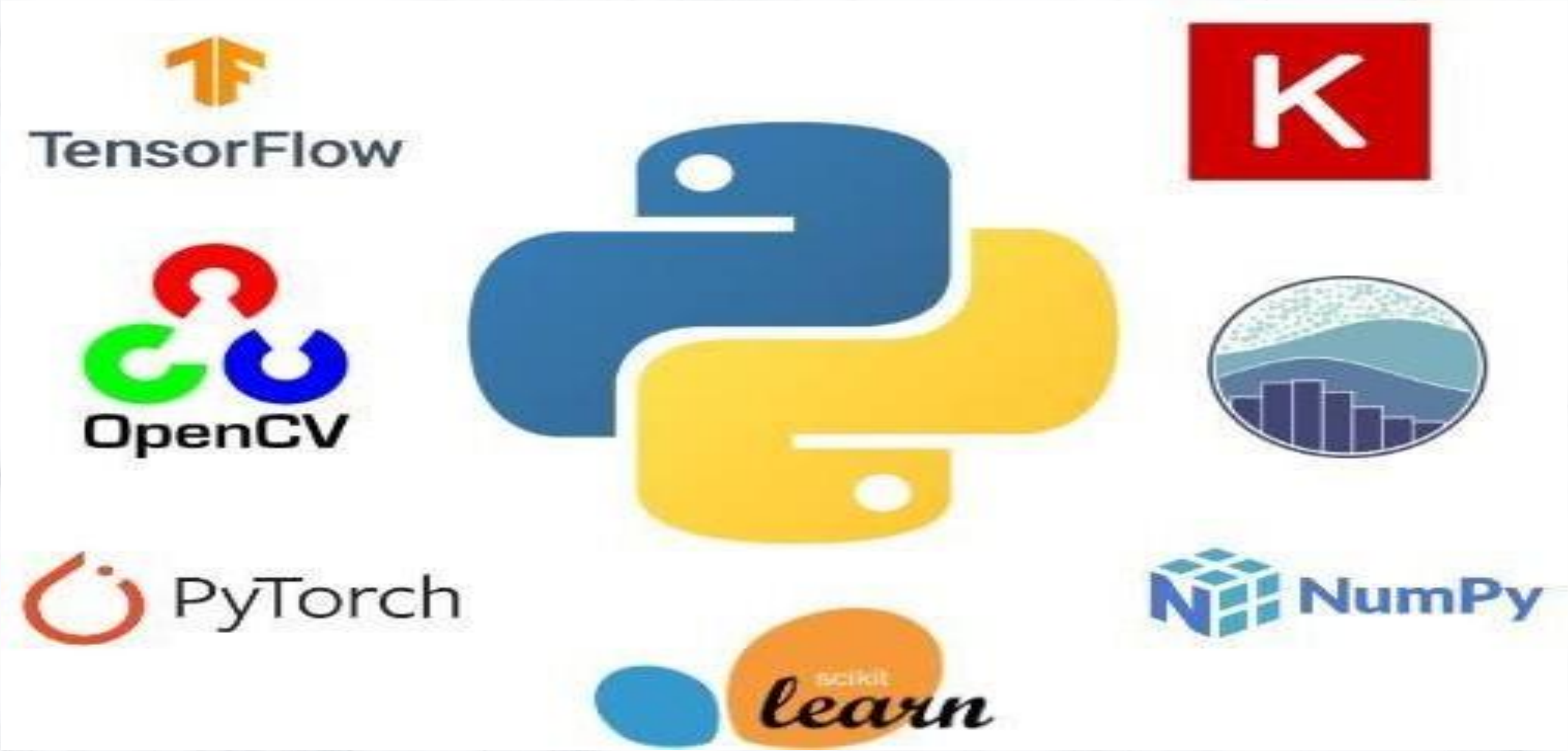
Component Solution

Combine CNN for spatial patterns and GRU for temporal sequences, trained jointly for precise gait recognition.

Align images, normalize intensities, and extract clean silhouettes to enhance data quality.

Integrate advanced feature extraction and hybrid model techniques.

Technologies

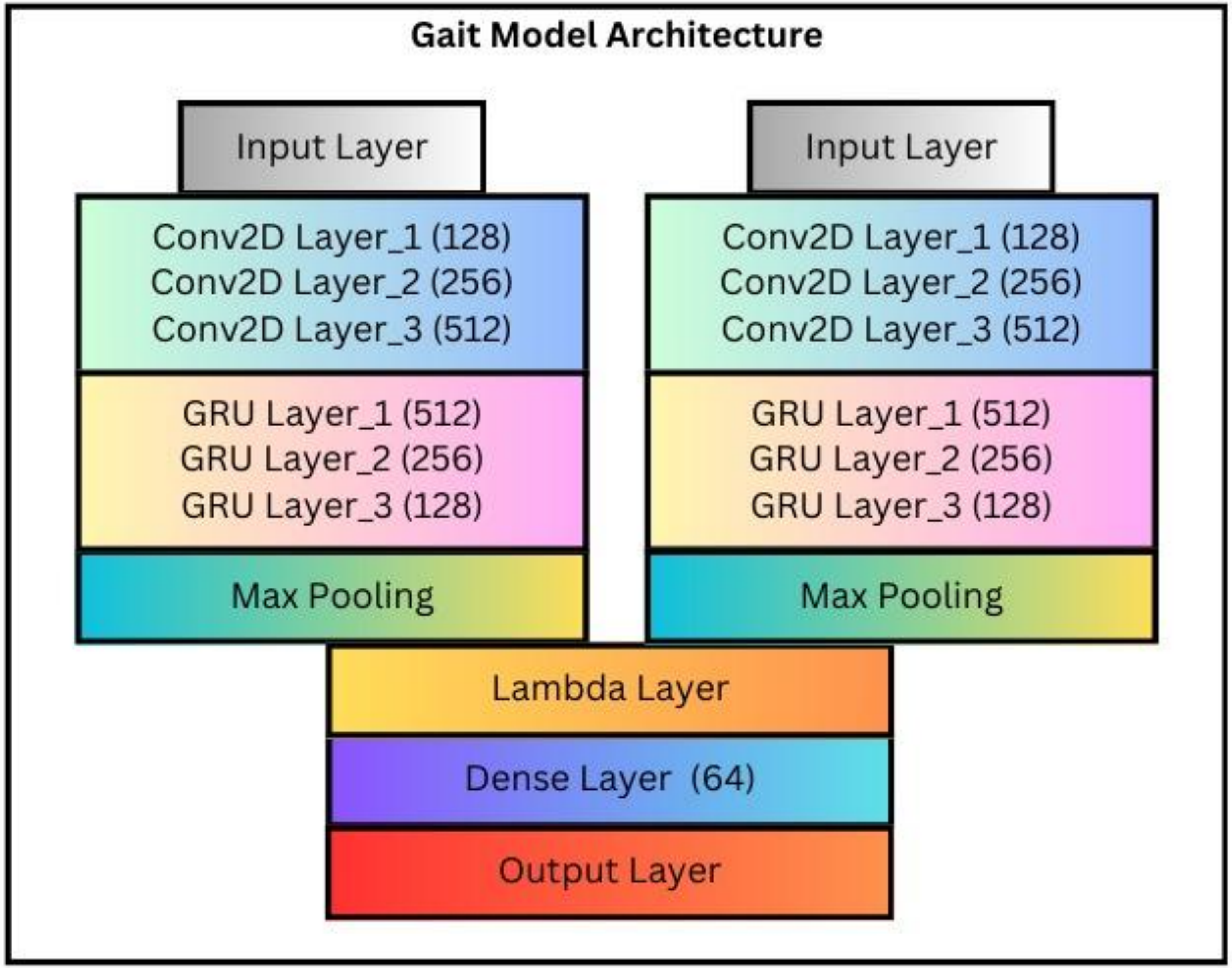


Methodology

Data Collection	Preprocessing	Model Development	Training and Validation	Evaluation
• Gather Gait Energy Images (GEIs) from online datasets.	• Normalize, resize, and augment images.	• Build a hybrid CNN-GRU model for spatial and temporal feature extraction.	• Train on GEIs and validate with cross-validation.	• Assess performance using metrics like accuracy and F1 score.

Novelty

- The gait authentication component introduces a hybrid CNN-GRU model to capture spatial and temporal features from Gait Energy Images (GEIs). This robust, non-intrusive approach adapts to variations like clothing or walking speed, making it ideal for real-time, high-security authentication.


Gait Model Architecture
Input Layer
Input Layer
Conv2D Layer_1 (128)
Conv2D Layer_2 (256)
Conv2D Layer_3 (512)
Conv2D Layer_1 (128)
Conv2D Layer_2 (256)
Conv2D Layer_3 (512)
GRU Layer_1 (512)
GRU Layer_2 (256)
GRU Layer_3 (128)
GRU Layer_1 (512)
GRU Layer_2 (256)
GRU Layer_3 (128)
Max Pooling
Max Pooling
Lambda Layer
Dense Layer (64)
Output Layer

Project Progress Completion

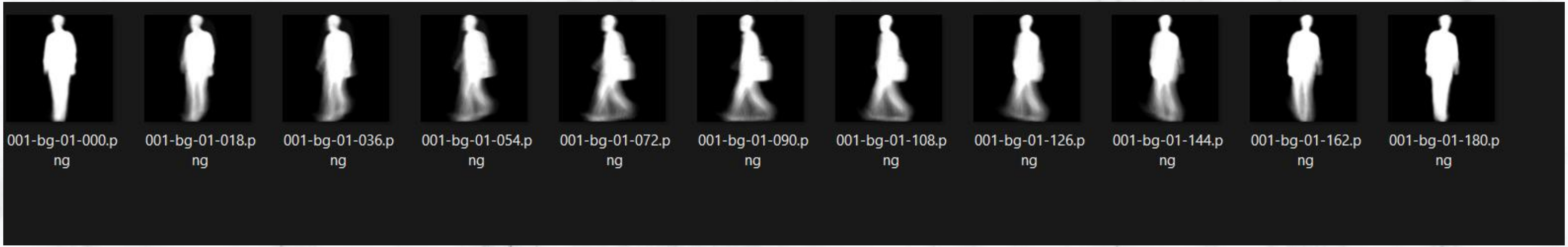
- Online Dataset

Name	Date modified	Type
001	9/25/2024 1:30 PM	File folder
002	9/25/2024 1:30 PM	File folder
003	9/25/2024 1:30 PM	File folder
004	9/25/2024 1:30 PM	File folder
005	9/25/2024 1:30 PM	File folder
006	9/25/2024 1:30 PM	File folder
007	9/25/2024 1:30 PM	File folder
008	9/25/2024 1:30 PM	File folder
009	9/25/2024 1:30 PM	File folder
010	9/25/2024 1:30 PM	File folder
011	9/25/2024 1:30 PM	File folder
012	9/25/2024 1:30 PM	File folder
013	9/25/2024 1:30 PM	File folder
014	9/25/2024 1:30 PM	File folder
015	9/25/2024 1:30 PM	File folder
016	9/25/2024 1:30 PM	File folder
017	9/25/2024 1:30 PM	File folder
018	9/25/2024 1:30 PM	File folder
019	9/25/2024 1:30 PM	File folder
020	9/25/2024 1:31 PM	File folder
021	9/25/2024 1:31 PM	File folder
022	9/25/2024 1:31 PM	File folder
023	9/25/2024 1:31 PM	File folder
024	9/25/2024 1:31 PM	File folder
025	9/25/2024 1:31 PM	File folder
026	9/25/2024 1:31 PM	File folder
027	9/25/2024 1:31 PM	File folder
028	9/25/2024 1:31 PM	File folder
029	9/25/2024 1:31 PM	File folder

bg-01	9/25/2024 1:30 PM	File folder
bg-02	9/25/2024 1:30 PM	File folder
cl-01	9/25/2024 1:30 PM	File folder
cl-02	9/25/2024 1:30 PM	File folder
nm-01	9/25/2024 1:30 PM	File folder
nm-02	9/25/2024 1:30 PM	File folder
nm-03	9/25/2024 1:30 PM	File folder
nm-04	9/25/2024 1:30 PM	File folder
nm-05	9/25/2024 1:30 PM	File folder
nm-06	9/25/2024 1:30 PM	File folder

Project Progress Completion

- Online Dataset



Project Progress Completion

- Generating Pairs

```python
# Shuffle and split
pairs_df = pairs_df.sample(frac=1).reset_index(drop=True)
train_pairs = pairs_df[:-1500]
val_pairs = pairs_df[-1500:]

# Preprocess and save data as .npz files
def preprocess_and_save(pairs_df, dataset_path, save_path):
    x1, x2, labels = [], [], []
    for _, row in pairs_df.iterrows():
        img1_path = os.path.join(dataset_path, row['image1_path'])
        img2_path = os.path.join(dataset_path, row['image2_path'])

        img1 = tf.keras.utils.load_img(img1_path, target_size=(128, 128), color_mode='grayscale')
        img2 = tf.keras.utils.load_img(img2_path, target_size=(128, 128), color_mode='grayscale')

        img1 = tf.keras.utils.img_to_array(img1) / 255.0
        img2 = tf.keras.utils.img_to_array(img2) / 255.0

        x1.append(img1)
        x2.append(img2)
        labels.append(row['label'])

    np.savez_compressed(save_path, x1=x1, x2=x2, labels=labels)

preprocess_and_save(train_pairs, GEI_DATASET_PATH, '/kaggle/working/train_data.npz')
preprocess_and_save(val_pairs, GEI_DATASET_PATH, '/kaggle/working/val_data.npz')
```

Project Progress Completion

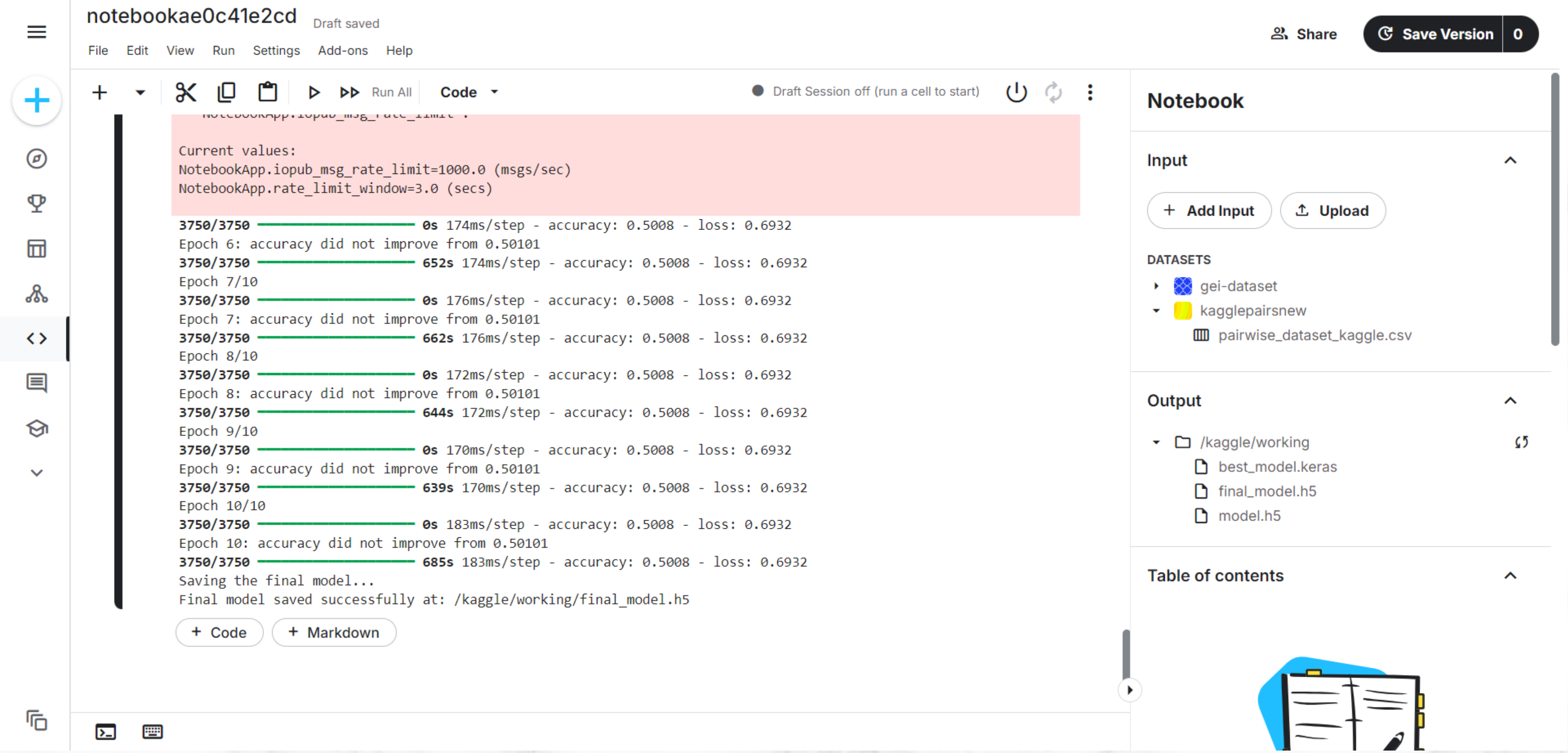
- Model Coding

```python
# CNN + GRU blocks
def create_cnn_gru_model():
    input_layer = Input(shape=(128, 128, 1))

    # CNN layers
    x = Conv2D(64, kernel_size=(3, 3), activation='relu', padding='same')(input_layer)
    x = MaxPooling2D(pool_size=(2, 2))(x)

    x = Conv2D(128, kernel_size=(3, 3), activation='relu', padding='same')(x)
    x = MaxPooling2D(pool_size=(2, 2))(x)

    x = Conv2D(128, kernel_size=(3, 3), activation='relu', padding='same')(x)
    x = MaxPooling2D(pool_size=(2, 2))(x)

    x = Conv2D(128, kernel_size=(3, 3), activation='relu', padding='same')(x)
    x = MaxPooling2D(pool_size=(2, 2))(x)

    x = Conv2D(256, kernel_size=(3, 3), activation='relu', padding='same')(x)
    x = MaxPooling2D(pool_size=(2, 2))(x)

    # Flatten and reshape for GRU
    x = Flatten()(x)
    x = Lambda(lambda tensor: tf.expand_dims(tensor, axis=1))(x)  # Add a time dimension

    # GRU layers
    x = GRU(256, return_sequences=True)(x)
    x = GRU(128, return_sequences=True)(x)
    x = GRU(64, return_sequences=True)(x)

    # Pooling after all GRU layers
    x = concatenate([GlobalMaxPooling1D()(x), GlobalAveragePooling1D()(x)])

    return Model(input_layer, x)
```

```python
# Create Siamese Network
def create_siamese_model():
    base_model = create_cnn_gru_model()

    input_a = Input(shape=(128, 128, 1))
    input_b = Input(shape=(128, 128, 1))

    processed_a = base_model(input_a)
    processed_b = base_model(input_b)

    # Distance calculation
    distance = Lambda(lambda tensors: tf.abs(tensors[0] - tensors[1]))([processed_a, processed_b])
    output = Dense(1, activation="sigmoid")(distance)

    return Model(inputs=[input_a, input_b], outputs=output)

# Compile the model
model = create_siamese_model()
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Training the model
batch_size = 32  # Adjust batch size if necessary
train_dataset = static_data_generator_with_weights(train_data, batch_size=batch_size)
val_dataset = static_data_generator_with_weights(val_data, batch_size=batch_size)

steps_per_epoch = len(train_data['labels']) // batch_size
validation_steps = len(val_data['labels']) // batch_size

early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)

history = model.fit(
    train_dataset,
    validation_data=val_dataset,
    steps_per_epoch=steps_per_epoch,
    validation_steps=validation_steps,
    epochs=20,
    callbacks=[early_stopping]
)

# Save model
model.save("/kaggle/working/siamese_model.h5")
```

Project Progress Completion

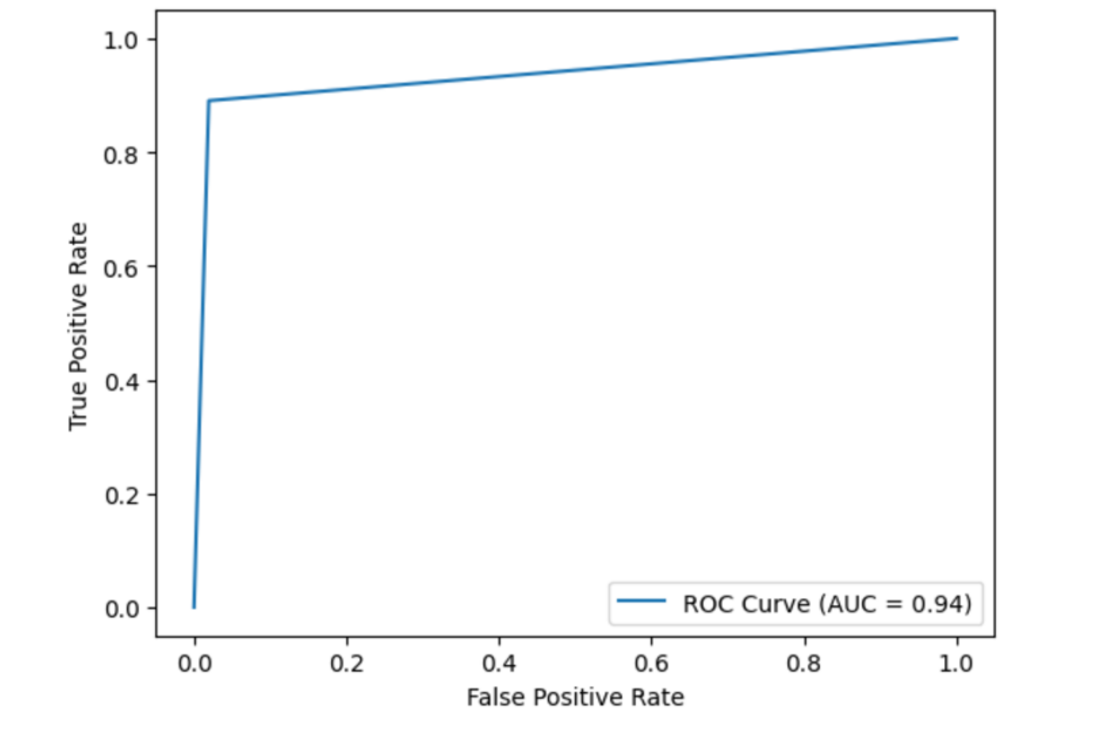
- Model Training



Project Progress Completion

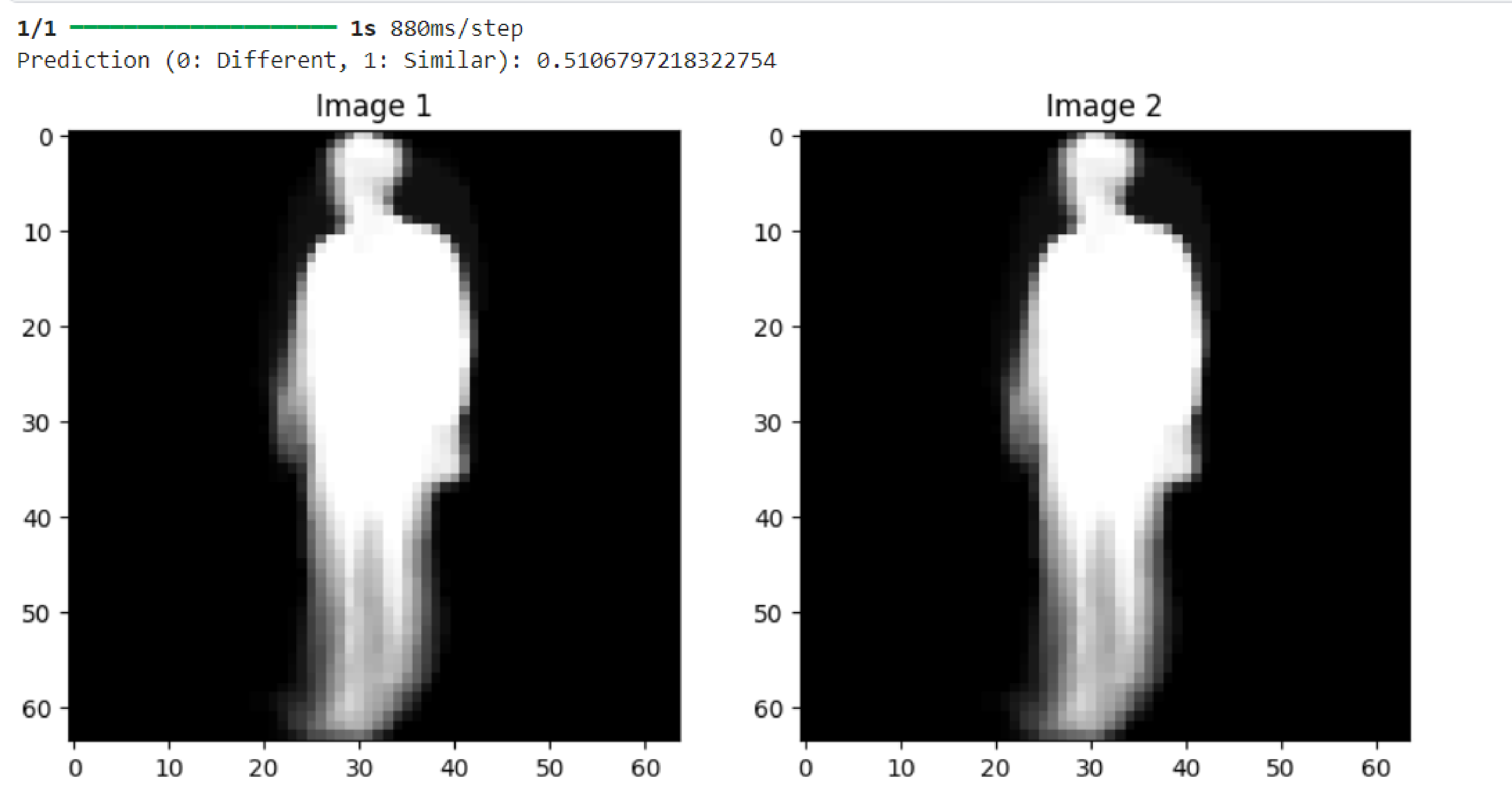
- Model Training

```
F1 Score: 0.9332434254888738
              precision    recall  f1-score   support

           0       0.89      0.98      0.93       723
           1       0.98      0.89      0.93       777

    accuracy                           0.93      1500
   macro avg       0.94      0.94      0.93      1500
weighted avg       0.94      0.93      0.93      1500

Confusion Matrix:
 [[709  14]
 [ 85 692]]
```



Project Progress Completion

- Model Evaluation



Progress

PP1 – 50%

- Dataset acquired and preprocessed.
- Model architecture coded (CNN-GRU hybrid).
- Model training initiated and initial results gathered.

PP2 – 90%

- Train model to achieve acceptable accuracy and F1 score.
- Validate model performance with real-world data.
- Integrate model output with other system components (e.g., voice, keystroke).

Final – 100%

- Complete frontend development and user interface.
- Finalize integration of all components.
- Compile and submit the final project report.

Future Interactions For 90% Phase

Model Optimization

- Fine-tune CNN-GRU for better accuracy and F1 score.

Validation

- Test with real-world data and integrate with other biometric models.

Frontend Preparation

- Plan and prepare for seamless frontend development.

REFERENCES

G. Giorgi, F. Martinelli, A. Saracino, and M. Sheikhalishahi, "Walking Through the Deep: Gait Analysis for User Authentication Through Deep Learning," *Inria*, [Online]. Available: https://inria.hal.science/hal-02023725/document. Accessed: Aug. 4, 2024.

I. Stylios, "Behavioral Biometrics for Continuous Authentication: Security and Privacy Issues," *ResearchGate*, Jan. 2023. [Online]. Available: https://www.researchgate.net/publication/369142299_Behavioral_Biometrics_for_Continuous_Authentication_Security_and_Privacy_Issues. Accessed: Aug. 4, 2024.

IT21340864 | E.M.N. EDIRISINGHE

BSc (Hons) in Information Technology Specializing in Cyber Security

Introduction to Keystroke Component

- SecureAuth leverages keystroke dynamics for secure, user authentication. By analyzing the unique patterns in a user's typing behavior, such as key press duration and typing speed, the system can identify and verify individuals. This method provides an additional layer of security without requiring any physical interaction, offering a seamless user experience while ensuring robust protection against unauthorized access. Keystroke biometrics integrates with other authentication techniques to enhance overall security in high-stakes applications.

Component Objectives

- **Develop Efficient One-Shot Learning**: Implement Siamese Network architecture for authentication using minimal data, ensuring quick enrollment and recognition
- **Achieve Robust and Real-Time Performance**: Design the system for fast, reliable, and scalable user authentication in live environments.
- **Ensure Privacy and Scalability**: Implement efficient user embedding storage and management for secure and scalable deployment

Component Sub-Objectives

Develop a Keystroke Dynamics Dataset:Create a comprehensive dataset of keystroke dynamics for training and evaluation purposes.

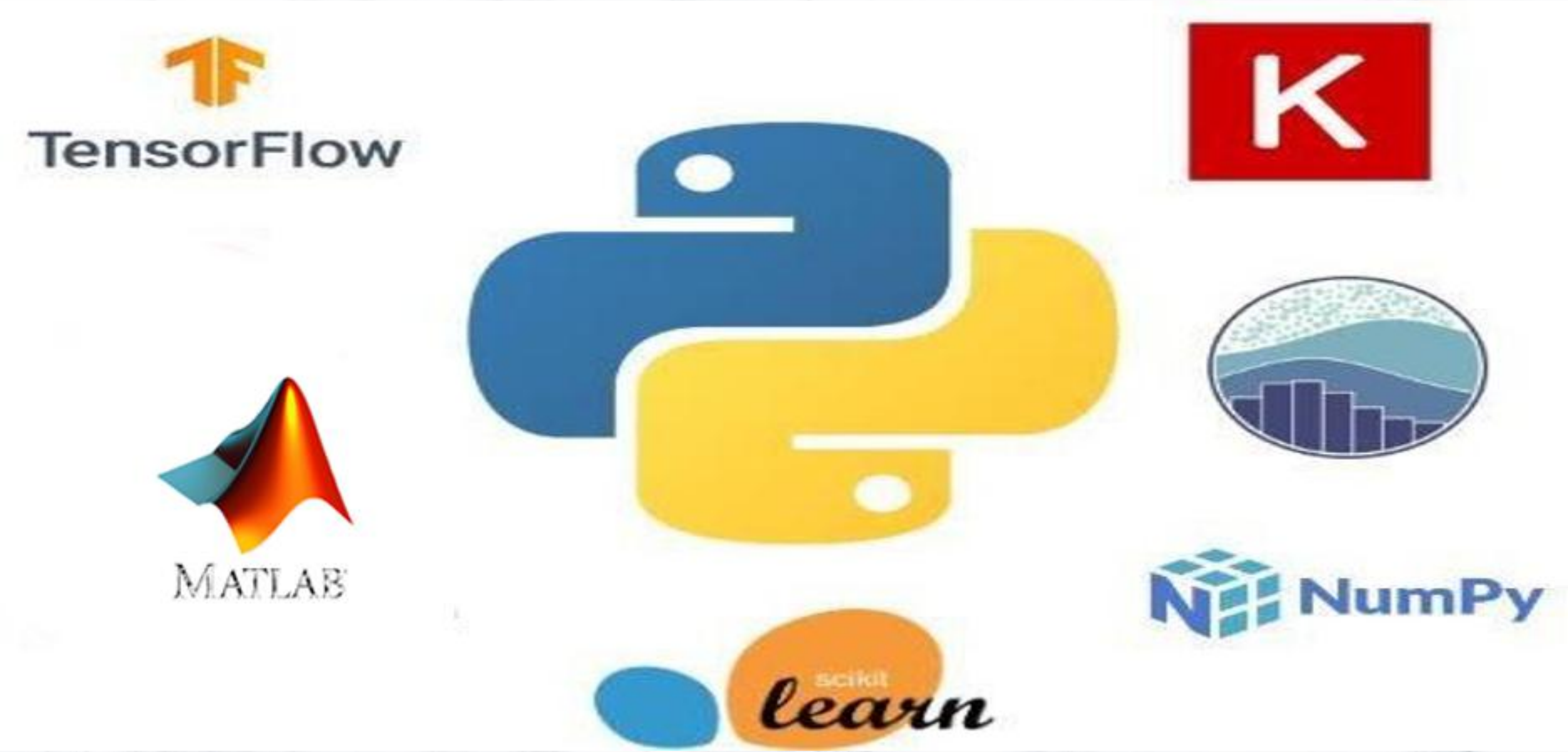
Extract and Analyze Keystroke Features:Identify and extract key features from typing patterns, such as dwell time and flight time,Error rate, IKI, ROR,etc.

Train and Validate RNN Models:Develop and validate Recurrent Neural Network (RNN) models to accurately capture the temporal dependencies in typing behavior.

Embedding Creation: Develop an efficient user embedding system for accurate authentication with minimal computational overhead.

Evaluate Authentication Performance:Assess the performance of the keystroke dynamics component in terms of accuracy, precision, recall, and overall robustness in authentication scenarios.

Technologies



❑ Component Gap

Features/ Technologies	Use of Online Datasets	Bidirectional LSTM for Sequence Modeling	One-Shot Siamese Network for Authentication	Cross-User Authentication via Embedding Matching
Project X	✓	✗	✗	✗
Project Y	✓	✓	✗	✓
Project Z	✗	✗	✗	✗
SecureAuth	✓	✓	✓	✓

Component Question

How can we ensure accurate authentication with minimal user enrollment data?

How does the system determine an appropriate threshold for authentication decisions?

How is user data secured during the embedding and authentication process?

Component Solution

Employing a one-shot Siamese network with Bidirectional LSTMs, the system captures unique typing patterns efficiently, enabling high accuracy even with minimal user input

Analyzing the cosine similarity scores between reference embeddings and login attempt embeddings, optimizing the balance between false positives and false negatives

The system employs encryption for embedding storage and transfer, ensuring data privacy.

Methodology

Data Collection

Use online datasets to collect keystroke data.

Data Preprocessing

Clean and normalize the collected data to ensure consistency and quality.

Feature Extraction

Extract Sequence Feature like Dwell time, Flight Time, Inter-Key Interval, Release Interval, etc

Preparing for Model Input

Sequence Analysis

Keystrokes are analyzed over time to capture typing rhythm, speed, and patterns

Use Advance technologies like Bidirectional LSTMs to learn the temporal dependencies between keystrokes.

Model Integration and Authentication Decision

Sequence analysis models to form a complete authentication system

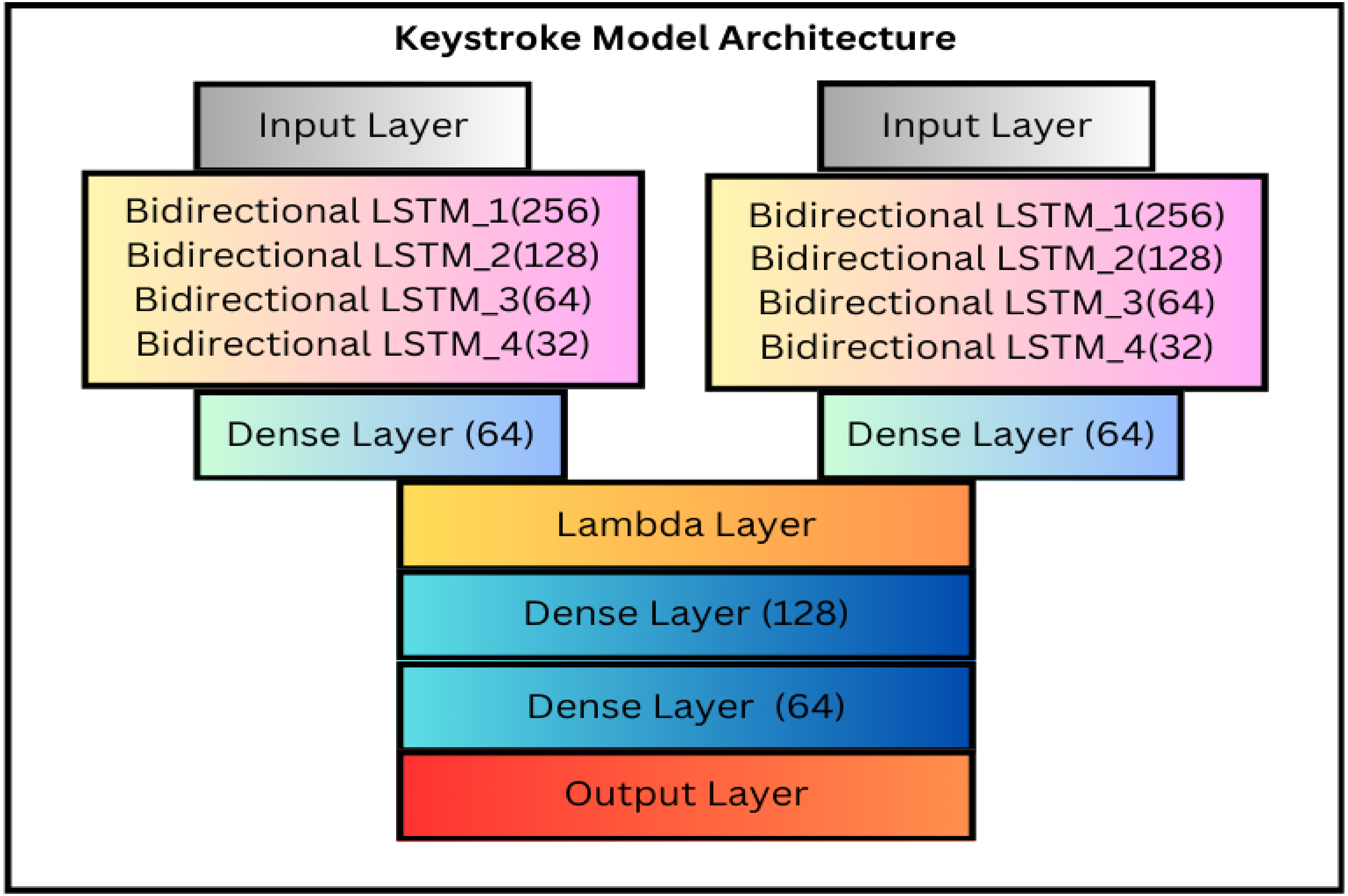
Generate embeddings for both reference user data and login attempt data.

Compare reference embeddings and login embeddings using similarity metrics like cosine similarity

Set a similarity threshold to decide whether the login attempt is authentic.

Novelty

- **Bidirectional LSTM Layers**: The use of Bidirectional LSTMs (Bi-LSTMs) helps capture both past and future dependencies in the keystroke sequence, making the model more adept at understanding the temporal aspects of typing behavior.
- **User-Specific Embedding Framework:** SecureAuth uses an advanced one-shot siamese network architecture to generate user-specific embeddings, ensuring personalized authentication accuracy.


Keystroke Model Architecture
Input Layer
Input Layer
Bidirectional LSTM_1(256)
Bidirectional LSTM_2(128)
Bidirectional LSTM_3(64)
Bidirectional LSTM_4(32)
Bidirectional LSTM_1(256)
Bidirectional LSTM_2(128)
Bidirectional LSTM_3(64)
Bidirectional LSTM_4(32)
Dense Layer (64)
Dense Layer (64)
Lambda Layer
Dense Layer (128)
Dense Layer (64)
Output Layer

Project Progress Completion

- Online Dataset

Participant_5	12/2/2024 15:14	File folder
Participant_7	12/2/2024 15:14	File folder
Participant_23	12/2/2024 15:14	File folder
Participant_24	12/2/2024 15:14	File folder
Participant_25	12/2/2024 15:14	File folder
Participant_30	12/2/2024 15:14	File folder
Participant_31	12/2/2024 15:14	File folder
Participant_32	12/2/2024 15:14	File folder
Participant_33	12/2/2024 15:14	File folder
Participant_35	12/2/2024 15:14	File folder
Participant_36	12/2/2024 15:14	File folder
Participant_38	12/2/2024 15:14	File folder
Participant_39	12/2/2024 15:14	File folder
Participant_40	12/2/2024 15:14	File folder
Participant_45	12/2/2024 15:14	File folder
Participant_49	12/2/2024 15:15	File folder

PARTICIPANT_I	TEST_SECTION_ID	PRESS_TIME	RELEASE_TIME	LETTER	KEYSTROKE_ID
10001	106696	1.47205E+12	1.47205E+12	SHIFT	5088570
10001	106696	1.47205E+12	1.47205E+12	H	5088575
10001	106696	1.47205E+12	1.47205E+12	e	5088580
10001	106696	1.47205E+12	1.47205E+12		5088581
10001	106696	1.47205E+12	1.47205E+12	p	5088583
10001	106696	1.47205E+12	1.47205E+12	l	5088609
10001	106696	1.47205E+12	1.47205E+12	a	5088612
10001	106696	1.47205E+12	1.47205E+12	y	5088616
10001	106696	1.47205E+12	1.47205E+12	e	5088618
10001	106696	1.47205E+12	1.47205E+12	d	5088621

ERROR_RATE	AVG_WPM_15	AVG_IKI	ECPC	KSPC	ROR
3.840472674	60.8829	169.3101457	0.045317221	1.152567976	0.4332
1.612903226	33.444	319.0930581	0.041420118	1.137573964	0.1671
0.735294118	40.7928	268.541052	0.03974359	1.105128205	0.1736
1.293900185	85.3952	124.2083817	0.038817006	1.136783734	0.4083
0.170357751	37.3318	267.2398387	0.042662116	1.208191126	0.3137
1.47601476	41.989	260.6474779	0.027777778	1.109259259	0.0599
4.320987654	22.8563	466.7668385	0.054574639	1.144462279	0.033
0.36900369	80.4561	135.974997	0.035120148	1.103512015	0.2251
0.3125	77.0218	131.1405147	0.071875	1.1796875	0.4892
0.299401198	33.7949	291.7303685	0.09924812	1.239097744	0.1978
0.304414003	26.6545	375.1684623	0.083969466	1.216793893	0.0435
0.866551127	71.9805	137.5283745	0.019097222	1.076388889	0.5033

Project Progress Completion

- Model Coding

```python
def siamese_lstm_block(input_layer):
    """A function to define the shared Bidirectional LSTM block."""

    # First Bidirectional LSTM layer (256 units)
    x = Bidirectional(LSTM(256, return_sequences=True, dropout=0.4, recurrent_dropout=0.4))(input_layer)

    # Second Bidirectional LSTM layer (128 units)
    x = Bidirectional(LSTM(128, return_sequences=True, dropout=0.4, recurrent_dropout=0.4))(x)

    # Third Bidirectional LSTM layer (64 units)
    x = Bidirectional(LSTM(64, return_sequences=True, dropout=0.4, recurrent_dropout=0.4))(x)

    # Fourth Bidirectional LSTM layer (32 units)
    x = Bidirectional(LSTM(32, return_sequences=False))(x)

    # Return the output of the LSTM block
    return x

# Input layers for paired sequences
input1 = Input(shape=(1, 12), name="input_sequence_1")  # Sequence 1
input2 = Input(shape=(1, 12), name="input_sequence_2")  # Sequence 2

# Apply the same LSTM block to both inputs (shared weights)
output1 = siamese_lstm_block(input1)
output2 = siamese_lstm_block(input2)

# Dense layer with 64 units
dense1 = Dense(64, activation='relu')(output1)
dense2 = Dense(64, activation='relu')(output2)

# Dropout layers after Dense layers
dropout1 = Dropout(0.4)(dense1)
dropout2 = Dropout(0.4)(dense2)
```

```python
# Lambda layer to compute the absolute difference between the two embeddings
def absolute_difference(tensors):
    return K.abs(tensors[0] - tensors[1])

lambda_layer = Lambda(absolute_difference)([dropout1, dropout2])

# Fully connected layers for final classification
fc1 = Dense(128, activation='relu')(lambda_layer)
dropout3 = Dropout(0.4)(fc1)

fc2 = Dense(64, activation='relu')(dropout3)
dropout4 = Dropout(0.4)(fc2)

# Final output layer (binary classification for similarity)
final_output = Dense(1, activation='sigmoid')(dropout4)

# Define the model
model = Model(inputs=[input1, input2], outputs=final_output)
```

Project Progress Completion

- Model Training

```python
# Define Exponential Decay schedule for learning rate
lr_schedule = ExponentialDecay(
    initial_learning_rate=0.001,   # Initial learning rate
    decay_steps=10000,              # Number of steps after which the learning rate decays
    decay_rate=0.96,                # Decay rate (learning rate is multiplied by decay_rate)
    staircase=True                  # If True, decay happens in discrete intervals
)


# Define early stopping to prevent overfitting
early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)

# Compile the model
optimizer = Adam(learning_rate=lr_schedule)  # Clip gradients to a max value of 1
model.compile(optimizer=optimizer, loss='binary_crossentropy', metrics=['accuracy'])
```

```python
# Train the updated model
history = model.fit(
    [X_train_1, X_train_2],
    y_train,
    validation_data=([X_test_1, X_test_2], y_test),
    epochs=50,
    batch_size=64,
    callbacks=[early_stopping],
    #verbose=1
)

# Check the training and validation accuracy
print("Training accuracy: ", history.history['accuracy'][-1])
print("Validation accuracy: ", history.history['val_accuracy'][-1])
```

Project Progress Completion

- Model Training

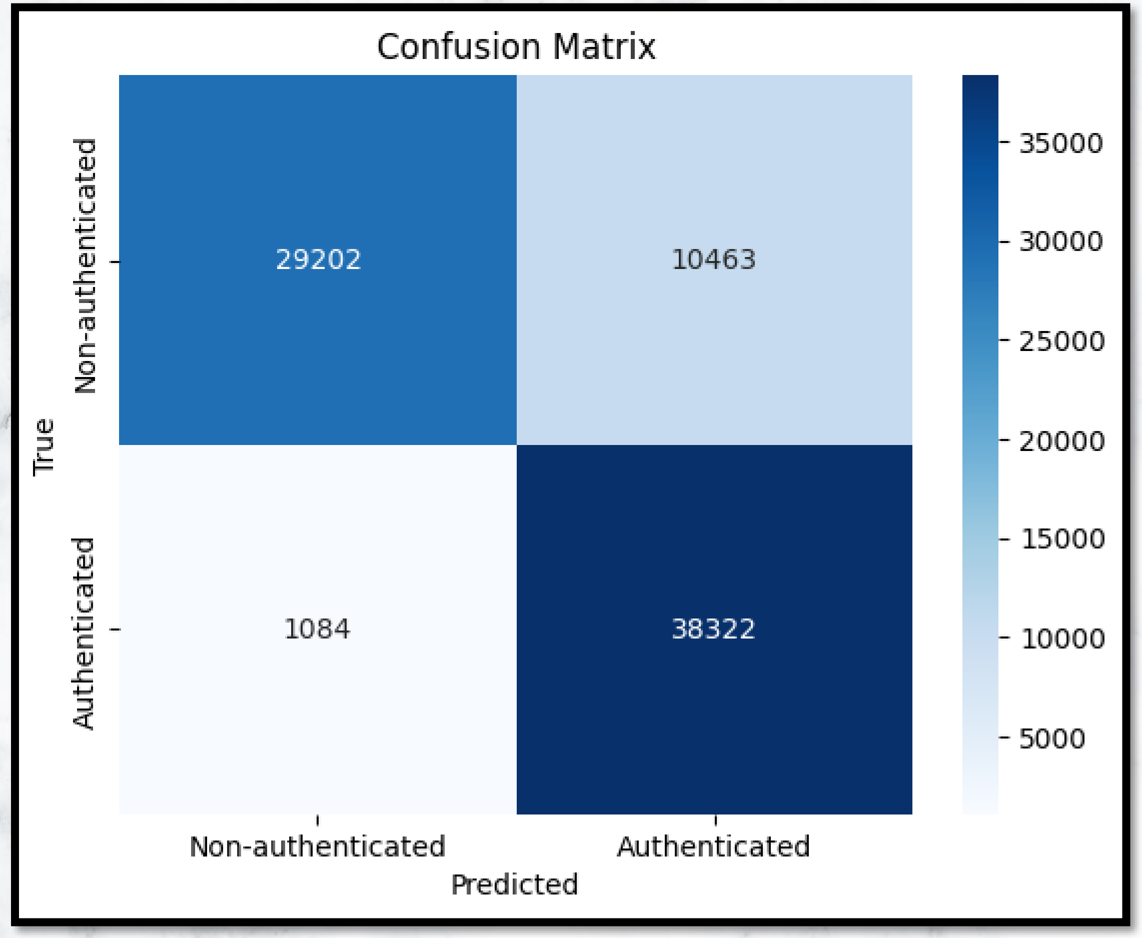
```
Epoch 1/50
4942/4942 ━━━━━━━━━━━━━━━━━━━━ 196s 35ms/step - accuracy: 0.5409 - loss: 0.6818 - val_accuracy: 0.5921 - val_loss: 0.6595
Epoch 2/50
4942/4942 ━━━━━━━━━━━━━━━━━━━━ 172s 35ms/step - accuracy: 0.5869 - loss: 0.6662 - val_accuracy: 0.6478 - val_loss: 0.6393
Epoch 3/50
4942/4942 ━━━━━━━━━━━━━━━━━━━━ 170s 34ms/step - accuracy: 0.6110 - loss: 0.6548 - val_accuracy: 0.6621 - val_loss: 0.6223
Epoch 4/50
4942/4942 ━━━━━━━━━━━━━━━━━━━━ 170s 34ms/step - accuracy: 0.6264 - loss: 0.6444 - val_accuracy: 0.7055 - val_loss: 0.5929
Epoch 5/50
4942/4942 ━━━━━━━━━━━━━━━━━━━━ 171s 35ms/step - accuracy: 0.6359 - loss: 0.6341 - val_accuracy: 0.7301 - val_loss: 0.5676
Epoch 6/50
4942/4942 ━━━━━━━━━━━━━━━━━━━━ 170s 34ms/step - accuracy: 0.6473 - loss: 0.6249 - val_accuracy: 0.7231 - val_loss: 0.5483
```

```
4942/4942 ━━━━━━━━━━━━━━━━━━━━ 175s 35ms/step - accuracy: 0.7426 - loss: 0.5051 - val_accuracy: 0.8459 - val_loss: 0.3501
Epoch 45/50
4942/4942 ━━━━━━━━━━━━━━━━━━━━ 173s 35ms/step - accuracy: 0.7414 - loss: 0.5039 - val_accuracy: 0.8472 - val_loss: 0.3474
Epoch 46/50
4942/4942 ━━━━━━━━━━━━━━━━━━━━ 175s 35ms/step - accuracy: 0.7451 - loss: 0.5015 - val_accuracy: 0.8468 - val_loss: 0.3524
Epoch 47/50
4942/4942 ━━━━━━━━━━━━━━━━━━━━ 176s 36ms/step - accuracy: 0.7451 - loss: 0.4993 - val_accuracy: 0.8540 - val_loss: 0.3415
Epoch 48/50
4942/4942 ━━━━━━━━━━━━━━━━━━━━ 176s 36ms/step - accuracy: 0.7464 - loss: 0.4993 - val_accuracy: 0.8526 - val_loss: 0.3428
Epoch 49/50
4942/4942 ━━━━━━━━━━━━━━━━━━━━ 175s 35ms/step - accuracy: 0.7464 - loss: 0.4968 - val_accuracy: 0.8528 - val_loss: 0.3426
Epoch 50/50
4942/4942 ━━━━━━━━━━━━━━━━━━━━ 178s 36ms/step - accuracy: 0.7501 - loss: 0.4927 - val_accuracy: 0.8495 - val_loss: 0.3430
```

Project Progress Completion

- Model Evaluation

```
1236/1236 ━━━━━━━━━━━━━━━━━━━━ 10s 8ms/step - accuracy: 0.8562 - loss: 0.3353
Test Loss:  0.33571961522102356
Test Accuracy:  0.8557878136634827
2471/2471 ━━━━━━━━━━━━━━━━━━━━ 20s 7ms/step
Accuracy: 0.8558
Precision: 0.7877
Recall: 0.9729
F1 Score: 0.8705
AUC: 0.9192
              precision    recall  f1-score   support

           0       0.96      0.74      0.84     39665
           1       0.79      0.97      0.87     39406

    accuracy                           0.86     79071
   macro avg       0.88      0.86      0.85     79071
weighted avg       0.88      0.86      0.85     79071
```



Project Progress Completion

- User Embedding

```python
# Define the absolute_difference function (with explicit output_shape)
def absolute_difference(tensors):
    return K.abs(tensors[0] - tensors[1])

# Create a custom object scope to handle the Lambda layer
def absolute_difference_output_shape(input_shape):
    # The output shape of the Lambda layer will be the same as the input shape (i.e., (No
    return input_shape[0],

# Load the trained model with custom objects
model = load_model(
    '/kaggle/input/kd-final-7/tensorflow2/default/1/keystroke_authentication_model.h5',
    custom_objects={
        'absolute_difference': absolute_difference,
        'absolute_difference_output_shape': absolute_difference_output_shape
    }
)
```

```python
from tensorflow.keras.models import Model

# Redefine the model to output embeddings from the Lambda layer
embedding_model = Model(inputs=model.input, outputs=model.get_layer('lambda').output)

# Step 1: Prepare the Input Data for Batch Prediction
# Reshape new_user_data to have shape (n_keystrokes, 1, 12)
new_user_data_reshaped = np.expand_dims(new_user_data, axis=1)  # Shape: (n_keystrokes, 1, 12)

# Step 2: Generate Embeddings for All Keystrokes at Once (Batch Processing)
keystroke_embeddings = embedding_model.predict([new_user_data_reshaped, new_user_data_reshaped])
```

```
24/24 ━━━━━━━━━━━━━━━━━━━━ 5s 123ms/step
```

```python
# Step 3: Aggregate Embeddings into a Single Reference Embedding
# Aggregate the embeddings (mean, for example)
reference_embedding = np.mean(keystroke_embeddings, axis=0)

# Step 4: Include User ID with the Reference Embedding
user_reference_embedding = {'user_id': user_id, 'embedding': reference_embedding}

# Print the user reference embedding
print(f"User ID: {user_reference_embedding['user_id']}")
print(f"Reference Embedding Shape: {user_reference_embedding['embedding'].shape}")
# print(f"Reference Embedding: {user_reference_embedding['embedding']}")

# Step 5: Save the Reference Embedding with User ID
np.save(f'/kaggle/working/reference_embedding_user_{user_id}.npy', user_reference_embedding)
print("Reference embedding with User ID saved!")

np.save(f'reference_embedding_user_{user_id}.npy', user_reference_embedding)
```

```
User ID: 1002
Reference Embedding Shape: (64,)
Reference embedding with User ID saved!
```

Project Progress Completion

- Authentication Process

```python
# Redefine the model to output embeddings from the Lambda layer
embedding_model = Model(inputs=model.input, outputs=model.get_layer('lambda').output)

# Function to load the reference embedding for a specific user
def load_reference_embedding(user_id):
    # Load the saved reference embedding file for the user
    try:
        reference_embedding = np.load(f'/kaggle/working/reference_embedding_user_{user_id}.npy', allow_pickle=True).item()
        print(f"Reference embedding for user {user_id} loaded successfully!")
        return reference_embedding
    except FileNotFoundError:
        print(f"Reference embedding for user {user_id} not found!")
        return None

# Function to generate the embedding for a given login attempt
def generate_login_embedding(login_data):
    # Assuming login_data is in the form of an ndarray with shape (n_keystrokes, 12 features)

     login_data_reshaped = np.expand_dims(login_data, axis=1)  # Reshape to (n_keystrokes, 1, 12)

    # Generate the embedding for the login attempt
    login_embedding = embedding_model.predict([login_data_reshaped, login_data_reshaped])

    # Aggregate the embeddings (mean, for example)
    login_embedding = np.mean(login_embedding,axis=0)

    return login_embedding
```

```python
# Function to compare embeddings using cosine similarity
def compare_embeddings(reference_embedding, login_embedding):
    # Print shapes for debugging
    print(f"Reference embedding shape: {reference_embedding.shape}")
    print(f"Login embedding shape: {login_embedding.shape}")

    # Check if the embeddings have the same shape
    if reference_embedding.shape != login_embedding.shape:
        print(f"Shape mismatch! Reference embedding shape: {reference_embedding.shape}, Login embedding shape: {login_embedding.shape}")

    # Compute cosine similarity
    similarity = cosine_similarity([reference_embedding], [login_embedding])[0][0]

    return similarity

# Set a similarity threshold for authentication
SIMILARITY_THRESHOLD = 0.80  # Adjust this threshold based on your validation
```

```python
    # Generate the login attempt embedding
    login_embedding = generate_login_embedding(login_data)

    # Step 3: Compare the Reference Embedding with the Login Attempt Embedding
    similarity_score,distance_score  = compare_embeddings(reference_embedding, login_embedding)
    print(f"Similarity Score: {similarity_score}\n")

    # Step 4: Authentication Decision
    if similarity_score >= SIMILARITY_THRESHOLD:
        print("Authentication successful!")
    else:
        print("Authentication failed!")
```

```
Reference embedding for user 1002 loaded successfully!
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step
Reference embedding shape: (64,)
Login embedding shape: (64,)
Similarity Score: 0.9685071706771851
```

Progress

PP1 – 50%

- Dataset acquired and preprocessed.
- Model architecture coded (Bi-LSTM).
- Model training initiated and initial results gathered (accuracy, precision, recall, F1-score, and AUC).
- User Embedding and Authentication Process Backend Coded

PP2 – 90%

- Train model to achieve acceptable accuracy ,F1 score and AUC.
- Validate model performance with real-world data.
- Securely store the User embedded and User data
- Integrate model output with other system components (e.g., voice, Gait, Mouse).

Final – 100%

- Complete frontend development and user interface.
- Finalize integration of all components.
- Compile and submit the final project report.

Future Interactions For 90% Phase

Model Optimization

- Fine-tune Bi-LSTM for better accuracy and F1 score.
- Securely store the User embedded and User data

Validation

- Test with real-world data and integrate with other biometric models.

Frontend Preparation

- Plan and prepare for seamless frontend development.

REFERENCES

- Aditya Arsh, Nirmalya Kar , and Subhrajyoti Deb , "Multiple Approaches Towards Authentication Using Keystroke Dynamics," 2024.

- Rashik Shadman, Ahmed Anu Wahab, Michael Manno, Matthew Lukaszewski, Daqing Hou, Faraz Hussain, “Keystroke Dynamics: Concepts, Techniques, and Applications” ,2024.

- Yutong Shi, Xiujuan Wang, Kangfeng Zheng, “User authentication method based on keystroke dynamics and mouse dynamics using HDA”, 2022.

IT21345678 |ANUPAMA K. G. A

BSc (Hons) in Information Technology Specialising in Cyber Security

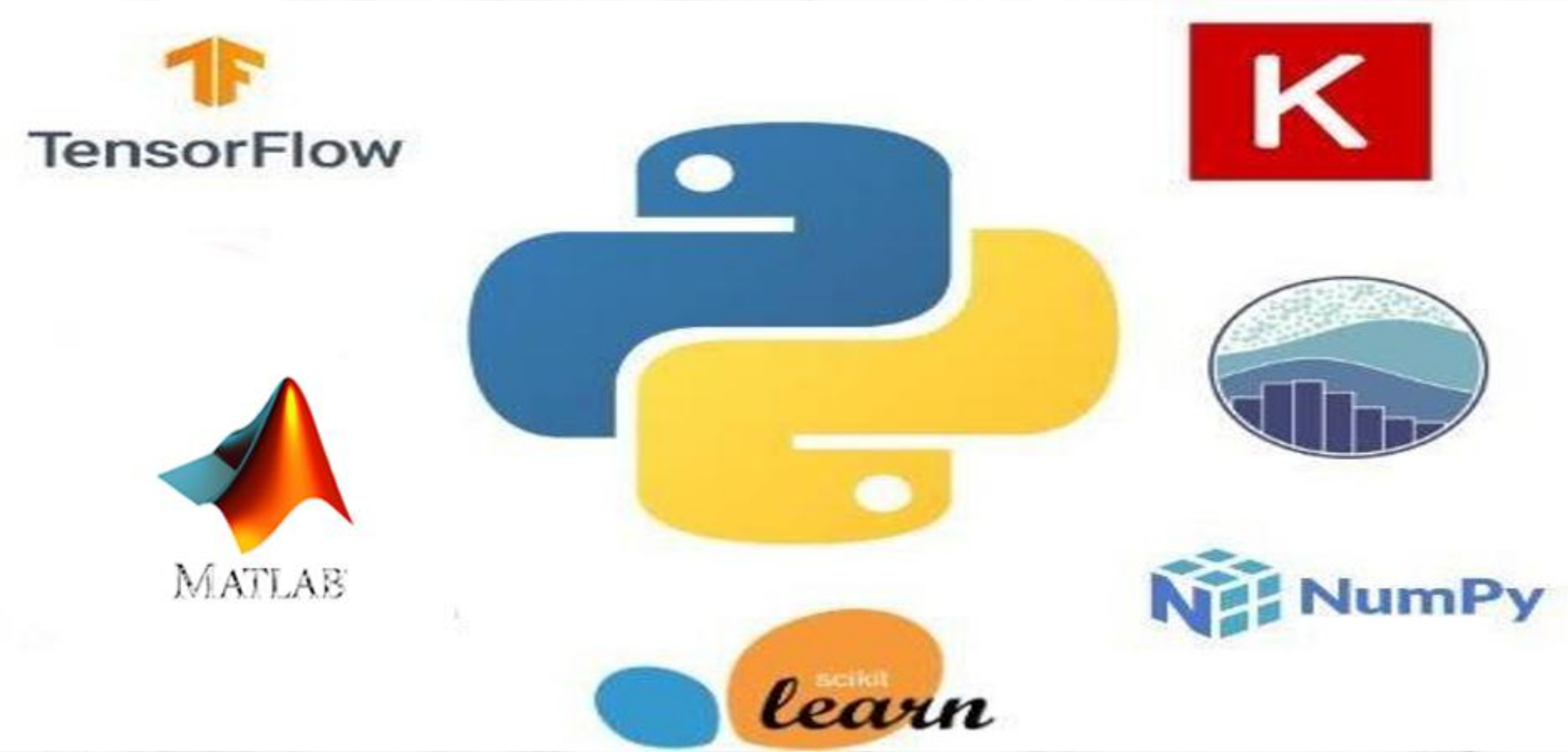
Introduction to Mouse movement Component

- Mouse movement explores behavioral authentication using mouse movement patterns, aiming to enhance security by leveraging unique user behaviors. Machine learning models, including Siamese networks with one-shot learning, will be used to analyze features like velocity, acceleration, and jerk to determine if two sessions belong to the same user. This approach enables accurate authentication with minimal user data, offering a robust solution for identity verification and fraud prevention.

Component Objectives

- **Develop Efficient One-Shot Learning**: Implement Siamese Network architecture for authentication using minimal data, ensuring quick enrollment and recognition
- **Authentication System Implementation**: Develop a system to authenticate users in real-time based on their mouse movement behavior.
- **Ensure Privacy and Scalability**: Implement efficient user embedding storage and management for secure and scalable deployment

Technologies



Component Sub-Objectives

Data Collection and Preprocessing : Collect and preprocess mouse movement data, extracting key features such as velocity, acceleration, jerk, and path efficiency.

Feature Engineering: Analyze and transform raw mouse movement data into meaningful inputs for the model, ensuring optimal feature representation.

Model Development: Design and implement a Siamese network with one-shot learning to compare user sessions and determine similarity.

Model Training and Evaluation :Train the model on session pairs and evaluate its accuracy in distinguishing between users.

Authentication System Implementation : Develop a system to authenticate users in real-time based on their mouse movement behavior.

❑ Component Gap

Features/ Technologies	Use of Online Datasets	LSTM for Sequence Modeling	One-Shot Siamese Network for Authentication	Cross-User Authentication via Embedding Matching
Project X	✓	✗	✗	✗
Project Y	✓	✓	✗	✓
Project Z	✗	✗	✗	✗
SecureAuth	✓	✓	✓	✓

Component Question

1. How can the system maintain high accuracy while authenticating users without affecting user experience?

2. How scalable is the system when deployed in environments with many users, and how does it perform as the user base grows?

3. How can the system maintain high accuracy while authenticating users?

Component Solution

Use lightweight models or optimize existing ones for real-time performance. Implement asynchronous authentication or batch processing to reduce latency. Dynamic thresholding based on context can balance accuracy and user experience.

The system can handle many users because it compares sessions to check for similarities. You don’t need to retrain the model for new users. As more users are added, some fine-tuning may be needed to keep the system accurate and avoid mistakes.

Adding contextual factors helps the model adjust its settings based on things like time or device type. This can reduce mistakes, like wrongly accepting or rejecting a user. Regular updates can improve the model’s performance in different situations.

Methodology

Data Collection

Collect comprehensive mouse movement data from diverse users across multiple sessions.

Data Preprocessing

Clean, normalize, and label the data to ensure consistency and readiness for model input.

Feature Extraction

Extract and transform key behavioral features such as velocity, acceleration, and jerk for effective analysis.

Model Development and Training

Design and implement a Siamese network to compare user sessions and measure similarity.

Train the model using labeled session pairs, optimizing with a suitable loss function like contrastive loss or binary cross-entropy.

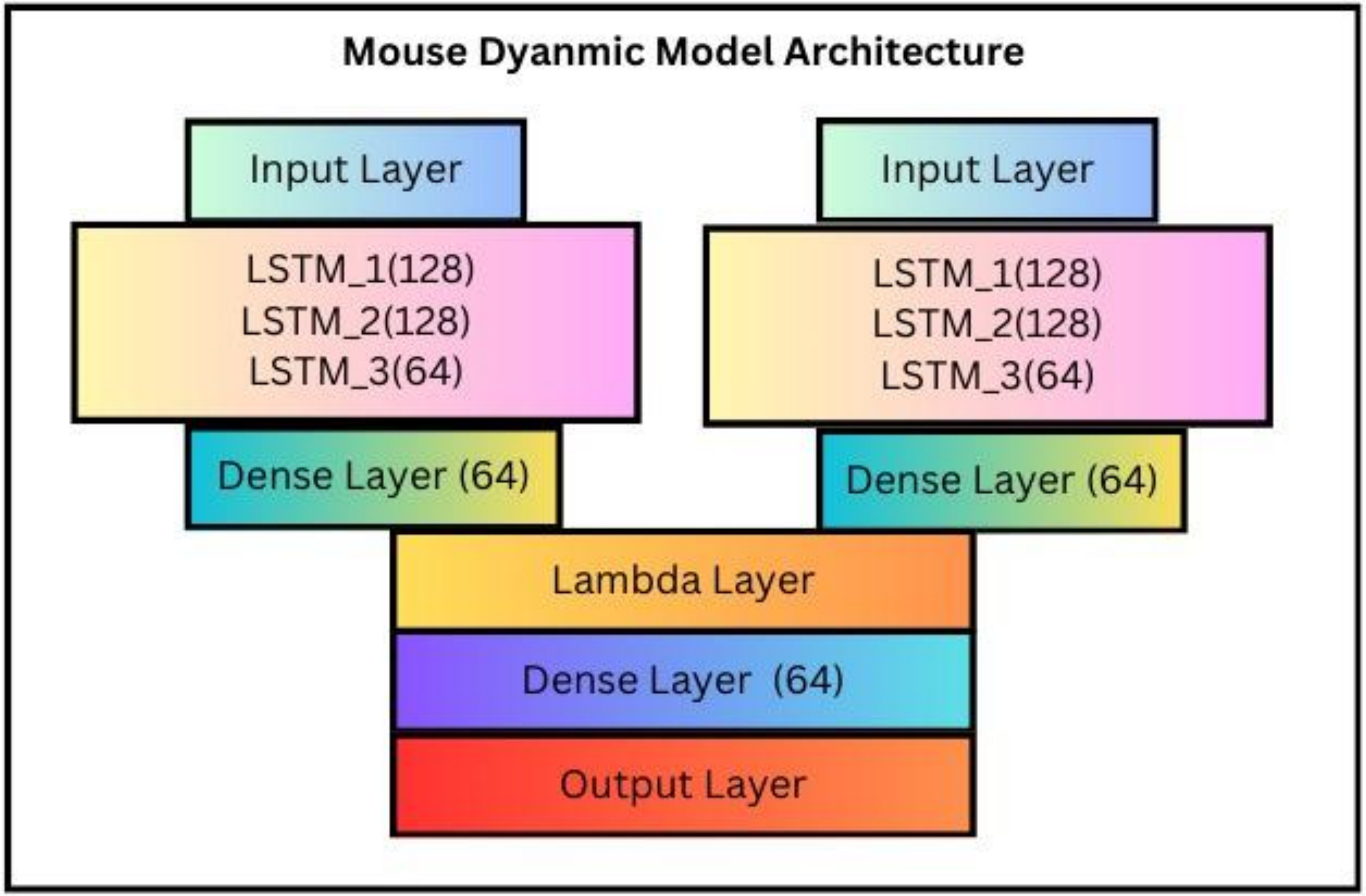
Evaluation and Implementation

Evaluate model performance using key metrics, including accuracy, precision, recall, and F1-score to ensure effectiveness.

Deploy the trained model for real-time, dynamic user authentication based on mouse movement behavior.

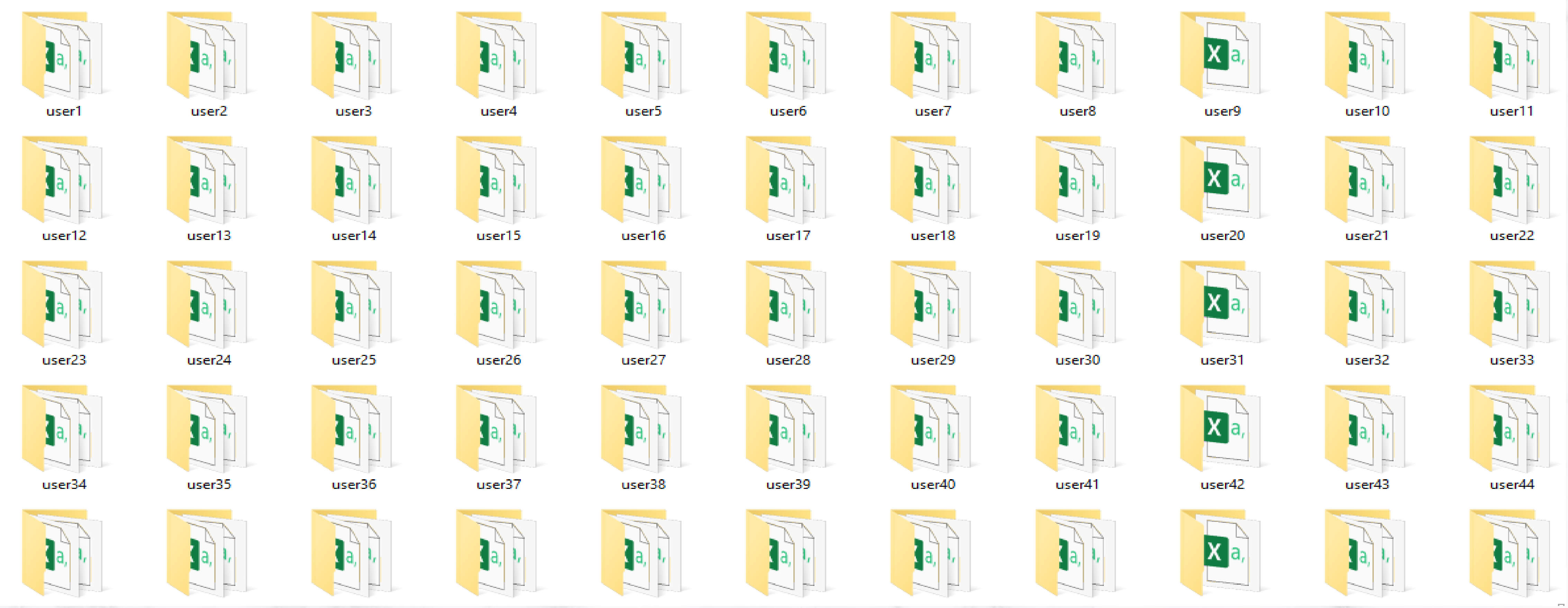
Novelty

- This Component brings a new idea by improving the one-shot Siamese model. It uses an adaptive system that adjusts the similarity threshold based on mouse movements. This helps the model make better decisions for each user, improving accuracy even with small data and changing user behavior.


Mouse Dyanmic Model Architecture
Input Layer
LSTM_1(128)
LSTM_2(128)
LSTM_3(64)
Dense Layer (64)
Input Layer
LSTM_1(128)
LSTM_2(128)
LSTM_3(64)
Dense Layer (64)
Lambda Layer
Dense Layer (64)
Output Layer

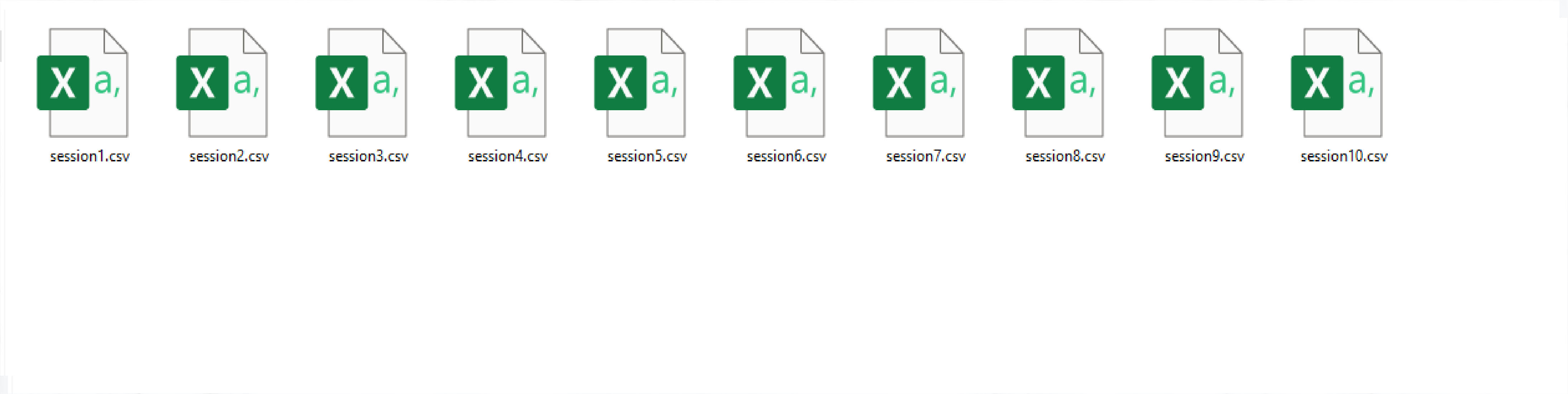
Project Progress Completion

- Online Dataset



Project Progress Completion

- Each user have Multiple Sessions



Project Progress Completion

client timestamp	button	state	x	y	distance moved	velocity_x	velocity_y	velocity	acceleration	path efficiency	jerk	angle	user
30933	NoButton	Move	740	608	13.6	-0.25	-0.81	0.85	-0.71	0.99	-38.856875	-107.1	1
30950	NoButton	Move	734	591	18.03	-0.35	-1	1.06	-0.9	1	-35.81764706	-109.44	1
30968	NoButton	Move	728	573	18.97	-0.33	-1	1.05	-0.88	1	-32.88222222	-108.43	1
30983	NoButton	Move	724	555	18.44	-0.27	-1.2	1.23	-0.95	1	-38.26333333	-102.53	1
31000	NoButton	Move	720	540	15.52	-0.24	-0.88	0.91	-0.72	1	-32.68941176	-104.93	1
31233	NoButton	Move	725	532	4.47	0.36	-0.18	0.41	0.18	0.95	-48.52909091	-26.57	1
31250	NoButton	Move	739	524	16.12	0.82	-0.47	0.95	0.39	0.87	-31.27117647	-29.74	1
31267	NoButton	Move	748	516	12.04	0.53	-0.47	0.71	0.16	0.85	-30.81411765	-41.63	1
31283	NoButton	Move	753	505	12.08	0.31	-0.69	0.76	-0.13	0.86	-32.258125	-65.56	1
31300	NoButton	Move	759	488	18.03	0.35	-1	1.06	-0.25	0.87	-29.72058824	-70.56	1
31317	NoButton	Move	764	475	13.93	0.29	-0.76	0.82	-0.16	0.88	-28.71529412	-68.96	1
31333	NoButton	Move	772	453	23.41	0.5	-1.38	1.46	-0.28	0.89	-29.705	-70.02	1
31350	NoButton	Move	776	425	28.28	0.24	-1.65	1.66	-0.61	0.91	-26.68294118	-81.87	1
31366	NoButton	Move	781	386	39.32	0.31	-2.44	2.46	-0.85	0.92	-26.615625	-82.69	1
31383	NoButton	Move	789	347	39.81	0.47	-2.29	2.34	-0.54	0.93	-22.73764706	-78.41	1
31399	NoButton	Move	793	317	30.27	0.25	-1.88	1.89	-0.5	0.94	-21.71875	-82.41	1
31416	NoButton	Move	795	289	28.07	0.12	-1.65	1.65	-0.48	0.94	-18.67529412	-85.91	1
31666	NoButton	Move	786	233	6.08	-0.35	0.06	0.36	-0.32	0.92	-13.66588235	170.54	1

Project Progress Completion

- Model Coding – Convert Data into NumPy array

```
# Load session data
def load_session_data(session_path):
    session_data = pd.read_csv(session_path)
    selected_columns = ['distance moved','velocity_x', 'velocity_y','velocity','acceleration','path efficiency', 'jerk', 'angle']
    session_data = session_data[selected_columns].values
    return session_data

# Create training data
def create_training_data(pair_csv_path, sequence_length=100):
    pair_data = pd.read_csv(pair_csv_path)
    session_1_data = []
    session_2_data = []
    labels = []

    for _, row in pair_data.iterrows():
        session_1 = load_session_data(row['session_1_path'])
        session_2 = load_session_data(row['session_2_path'])
        label = row['label']

        session_1_windows = [session_1[i:i + sequence_length] for i in range(0, len(session_1) - sequence_length + 1, sequence_length)]
        session_2_windows = [session_2[i:i + sequence_length] for i in range(0, len(session_2) - sequence_length + 1, sequence_length)]

        min_windows = min(len(session_1_windows), len(session_2_windows))
        session_1_data.extend(session_1_windows[:min_windows])
        session_2_data.extend(session_2_windows[:min_windows])
        labels.extend([label] * min_windows)

    session_1_data = tf.keras.preprocessing.sequence.pad_sequences(session_1_data, maxlen=sequence_length, dtype='float32', padding='post')
    session_2_data = tf.keras.preprocessing.sequence.pad_sequences(session_2_data, maxlen=sequence_length, dtype='float32', padding='post')
    labels = np.array(labels, dtype='float32')

    return session_1_data, session_2_data, labels
```

Project Progress Completion

- Model Coding

```python
# Define the Siamese network model
def build_siamese_model(sequence_length, feature_dim):
    """
    Builds and returns a Siamese neural network for behavioral authentication.
    """
    # Input layers for both sessions
    input_1 = Input(shape=(sequence_length, feature_dim), name="Input_Session_1")
    input_2 = Input(shape=(sequence_length, feature_dim), name="Input_Session_2")

 # Shared LSTM layers with separated dropout
    lstm_1 = LSTM(128, return_sequences=True, dropout=0.4, name="LSTM_1")  # LSTM with 128 units
    lstm_2 = LSTM(128, return_sequences=True, dropout=0.4, name="LSTM_2")  # LSTM with 128 units
    lstm_3 = LSTM(64, return_sequences=False, dropout=0.4, name="LSTM_3")  # LSTM with 64 units

    dense_layer = Dense(64, activation='relu', name="Dense_Layer")

    # Process the input through each layer
    lstm_1_output_1 = lstm_1(input_1)  # Apply lstm_1 to the first input
    lstm_2_output_1 = lstm_2(lstm_1_output_1)  # Apply lstm_2 to the output of lstm_1
    lstm_3_output_1 = lstm_3(lstm_2_output_1)  # Apply lstm_3 to the output of lstm_2
    processed_1 = dense_layer(lstm_3_output_1)  # Apply dense layer to the output of lstm_3

    lstm_1_output_2 = lstm_1(input_2)  # Apply lstm_1 to the second input
    lstm_2_output_2 = lstm_2(lstm_1_output_2)  # Apply lstm_2 to the output of lstm_1
    lstm_3_output_2 = lstm_3(lstm_2_output_2)  # Apply lstm_3 to the output of lstm_2
    processed_2 = dense_layer(lstm_3_output_2)  # Apply dense layer to the output of lstm_3
```

```python
 # Lambda layer to compute absolute difference between embeddings
def absolute_difference(tensors):
    return K.abs(tensors[0] - tensors[1])

distance = Lambda(absolute_difference)([processed_1, processed_2])

 # new dense layer after the Lambda layer
dense_after_lambda = Dense(64, activation='relu', name="Dense_After_Lambda")(distance)

# Output layer
output = Dense(1, activation='sigmoid', name="Output_Layer")(dense_after_lambda)

# Build the model
model = Model(inputs=[input_1, input_2], outputs=output)
return model
```

Project Progress Completion

- Model Architecture

```
# Compile the model
model.compile(optimizer=Adam(learning_rate=0.001), loss='binary_crossentropy', metrics=['accuracy'])

# Print model summary
print("Model Architecture Summary:")
model.summary()

# Train the model
history = model.fit(
    [train_session_1_data, train_session_2_data], train_labels,
    validation_split=0.1,  # Optionally use part of training data for validation
    epochs=20,
    batch_size=32
)

# Evaluate the model on test data
test_loss, test_accuracy = model.evaluate([test_session_1_data, test_session_2_data], test_labels)
print(f"Test Loss: {test_loss}, Test Accuracy: {test_accuracy}")

# Save the trained model
model_save_path = r"/kaggle/working/new_2.h5"
model.save(model_save_path)
print(f"Model saved as '{model_save_path}'")
```

Project Progress Completion

- Model Training

```
Total params: 259,521 (1013.75 KB)
Trainable params: 259,521 (1013.75 KB)
Non-trainable params: 0 (0.00 B)
Epoch 1/20
163/163 ━━━━━━━━━━━━━━━━━━━━ 66s 349ms/step - accuracy: 0.4856 - loss: 0.7440 - val_accuracy: 0.5285 - val_loss: 0.6926
Epoch 2/20
163/163 ━━━━━━━━━━━━━━━━━━━━ 56s 342ms/step - accuracy: 0.4792 - loss: 0.7422 - val_accuracy: 0.5026 - val_loss: 0.6930
Epoch 3/20
163/163 ━━━━━━━━━━━━━━━━━━━━ 56s 346ms/step - accuracy: 0.4880 - loss: 0.7401 - val_accuracy: 0.4560 - val_loss: 0.6948
Epoch 4/20
163/163 ━━━━━━━━━━━━━━━━━━━━ 57s 351ms/step - accuracy: 0.4901 - loss: 0.7398 - val_accuracy: 0.4560 - val_loss: 0.6942
Epoch 5/20
163/163 ━━━━━━━━━━━━━━━━━━━━ 56s 343ms/step - accuracy: 0.4785 - loss: 0.7400 - val_accuracy: 0.4560 - val_loss: 0.6942
Epoch 6/20
163/163 ━━━━━━━━━━━━━━━━━━━━ 82s 345ms/step - accuracy: 0.4645 - loss: 0.7396 - val_accuracy: 0.4577 - val_loss: 0.6936
Epoch 7/20
163/163 ━━━━━━━━━━━━━━━━━━━━ 56s 346ms/step - accuracy: 0.4811 - loss: 0.7390 - val_accuracy: 0.4560 - val_loss: 0.6943
Epoch 8/20
163/163 ━━━━━━━━━━━━━━━━━━━━ 55s 340ms/step - accuracy: 0.4805 - loss: 0.7396 - val_accuracy: 0.4560 - val_loss: 0.6937
Epoch 9/20
163/163 ━━━━━━━━━━━━━━━━━━━━ 56s 344ms/step - accuracy: 0.4893 - loss: 0.7393 - val_accuracy: 0.5130 - val_loss: 0.6930
Epoch 10/20
163/163 ━━━━━━━━━━━━━━━━━━━━ 56s 343ms/step - accuracy: 0.4870 - loss: 0.7391 - val_accuracy: 0.4560 - val_loss: 0.6935
Epoch 11/20
163/163 ━━━━━━━━━━━━━━━━━━━━ 56s 343ms/step - accuracy: 0.4758 - loss: 0.7393 - val_accuracy: 0.4560 - val_loss: 0.6934
Epoch 12/20
163/163 ━━━━━━━━━━━━━━━━━━━━ 56s 345ms/step - accuracy: 0.4700 - loss: 0.7398 - val_accuracy: 0.4991 - val_loss: 0.6932
Epoch 13/20
163/163 ━━━━━━━━━━━━━━━━━━━━ 56s 344ms/step - accuracy: 0.4820 - loss: 0.7398 - val_accuracy: 0.4680 - val_loss: 0.6933
Epoch 14/20
163/163 ━━━━━━━━━━━━━━━━━━━━ 56s 343ms/step - accuracy: 0.4902 - loss: 0.7381 - val_accuracy: 0.4560 - val_loss: 0.6938
```


Epoch 14/20
163/163 ━━━━━━━━━━━━━━━━━━━━ 56s 343ms/step - accuracy: 0.4902 - loss: 0.7381 - val_accuracy: 0.4560 - val_loss: 0.6938


```
Epoch 15/20
163/163 ━━━━━━━━━━━━━━━━━━━━ 56s 342ms/step - accuracy: 0.4849 - loss: 0.7397 - val_accuracy: 0.4870 - val_loss: 0.6932
Epoch 16/20
163/163 ━━━━━━━━━━━━━━━━━━━━ 56s 344ms/step - accuracy: 0.4824 - loss: 0.7402 - val_accuracy: 0.5147 - val_loss: 0.6931
Epoch 17/20
163/163 ━━━━━━━━━━━━━━━━━━━━ 56s 346ms/step - accuracy: 0.4824 - loss: 0.7399 - val_accuracy: 0.5440 - val_loss: 0.6928
Epoch 18/20
163/163 ━━━━━━━━━━━━━━━━━━━━ 55s 340ms/step - accuracy: 0.5021 - loss: 0.7400 - val_accuracy: 0.4421 - val_loss: 0.6932
Epoch 19/20
163/163 ━━━━━━━━━━━━━━━━━━━━ 57s 347ms/step - accuracy: 0.4658 - loss: 0.7395 - val_accuracy: 0.4560 - val_loss: 0.6939
Epoch 20/20
163/163 ━━━━━━━━━━━━━━━━━━━━ 56s 345ms/step - accuracy: 0.4918 - loss: 0.7401 - val_accuracy: 0.4663 - val_loss: 0.6932
```

Project Progress Completion

- Model Evaluation

```
Test Loss: 0.6922535300254822, Test Accuracy: 0.5161111354827881
Model saved as '/kaggle/working/new_6.h5'
57/57 ━━━━━━━━━━━━━━━━━━━━ 9s 140ms/step
F1 Score: 0.45596502186133664
Precision: 0.41809851088201605
Recall: 0.5013736263736264
Accuracy: 0.5161111111111111
              precision    recall  f1-score   support

         0.0       0.61      0.53      0.56      1072
         1.0       0.42      0.50      0.46       728

    accuracy                           0.52      1800
   macro avg       0.51      0.51      0.51      1800
weighted avg       0.53      0.52      0.52      1800
```

Project Progress Completion

- Authentication Process

```python
# Load the trained model with custom objects
model = load_model(
    '/kaggle/working/new_2.h5',
    custom_objects={'absolute_difference': absolute_difference}
)
model.summary()

print("Model Loaded")

# Function to perform authentication by comparing two session data
def authenticate_sessions(session_1_path, session_2_path, threshold=0.5):
    """
    Authenticate the sessions by comparing them using the Siamese network.
    If the similarity score is above the threshold, the sessions are considered authentic.
    """
    # Preprocess the session data
    session_1_data, session_2_data = preprocess_authentication_data(session_1_path, session_2_path)

    # Get the similarity score from the model
    similarity_score = model.predict([session_1_data, session_2_data])

    # Print the similarity score
    print(f"Similarity Score: {similarity_score[0][0]}")

    # Compare similarity score to threshold for authentication decision
    if similarity_score[0][0] > threshold:
        print("Authentication Successful: The sessions belong to the same user.")
    else:
        print("Authentication Failed: The sessions belong to different users.")

# Example usage:
session_1_path = r"/kaggle/input/userss/Userss2/user10/session1.csv"
session_2_path = r"/kaggle/input/userss/Userss2/user10/session2.csv"

# Perform authentication
authenticate_sessions(session_1_path, session_2_path)
```

Project Progress Completion

- Authentication Process – Same Users and Different Users

```
# Example usage: # session_1 mean legi user and session2 mean user who want to authentacation
session_1_path = r"/kaggle/input/userss/Userss2/user10/session1.csv"
session_2_path = r"/kaggle/input/userss/Userss2/user10/session2.csv"

# Perform authentication
authenticate_sessions(session_1_path, session_2_path)
```

```
# Example usage: # session_1 mean legi user and session2 mean user who want to authentacation
session_1_path = r"/kaggle/input/userss/Userss2/user11/session1.csv"
session_2_path = r"/kaggle/input/userss/Userss2/user10/session2.csv"

# Perform authentication
authenticate_sessions(session_1_path, session_2_path)
```

```
Total params: 259,523 (1013.77 KB)
Trainable params: 259,521 (1013.75 KB)
Non-trainable params: 0 (0.00 B)
Optimizer params: 2 (12.00 B)
Model Loaded
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
Similarity Score: 0.5003320574760437
Authentication Successful: The sessions belong to the same user.
```

```
Total params: 259,523 (1013.77 KB)
Trainable params: 259,521 (1013.75 KB)
Non-trainable params: 0 (0.00 B)
Optimizer params: 2 (12.00 B)
Model Loaded
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 884ms/step
Similarity Score: 0.4906914234161377
Authentication Failed: The sessions belong to different users.
```

Progress

PP1 – 50%

- Dataset acquired and preprocessed.
- Model architecture coded (Siamese Network and LSTM).
- Model training initiated and initial results gathered (accuracy, precision, recall, F1-score, and AUC).
- User Embedding and Authentication Process

PP2 – 90%

- Train model to achieve acceptable accuracy ,F1 score and AUC.
- Validate model performance with real-world data.
- Securely store the User embedded and User data
- Integrate model output with other system components (e.g., voice, Gait, Key bord).

Final – 100%

- Complete frontend development and user interface.
- Finalize integration of all components.
- Compile and submit the final project report.

Future Interactions For 90% Phase

Model Optimization

- Fine-tune LSTM for better accuracy , F1 score , recall and precision.
- Securely store the User embedded and User data

Validation

- Test with real-world data and integrate with other biometric models.

Frontend Preparation

- Plan and prepare for seamless frontend development.

IT21336072 | R.P.K.D RAJAPAKSHA

BSc (Hons) in Information Technology Specializing in Cyber Security

Introduction to Voice Component

- Voice authentication leverages the unique characteristics of an individual's voice for secure user identification. By analyzing features such as pitch, tone, and speaking patterns, the system can verify identities with high accuracy. Voice biometrics, combined with deep learning techniques like Siamese Networks, provides robust protection against spoofing attacks and ensures secure access to sensitive systems. By integrating voice authentication with other methods, this technology enhances security in applications demanding advanced and multifactor authentication.

Research Sub-Objectives

Data Preprocessing

- Develop an efficient pipeline for converting raw audio files into clean,denoised data suitable for analysis

Extract Features

- Improve techniques extracting relevant voice features

Model Design

- Evaluate the effectiveness of the network in distinguishing between similar and dissimilar voice samples

Pair generation

- Design an efficient mechanism for generating positive and negative spectrogram pairs for training

Technologies

Research Question

- How can we handle variations in voice caused by emotional states, health conditions, or environmental factors in a CNN-based voice authentication system?

- What is the impact of using spectrograms in voice authentication compared to traditional waveform analysis for CNN-based models?

Solution

- We preprocess audio to reduce noise and standardize speech, ensuring consistency despite emotional or health variations. The CNN is trained on diverse data, using augmentation techniques to simulate different conditions, making the system robust to these variations while maintaining high accuracy.

- Spectrograms capture time-frequency features like pitch and formants, which are crucial for accurate voice identification. By using spectrograms, the CNN model learns better spatial patterns, improving performance over traditional waveform analysis, especially in noisy environments.

Methodology

Data Collection

Collect a dataset of a 3000 user voice samples

1

Data Preprocessing

Utilize Librosa or PyDub for noise reduction, normalization, and segmentation of voice data.

2

Feature Extraction

Apply CNNs to the spectrograms to extract key features that represent the unique characteristics of each speaker's voice

3

Model Development

Develop the CNN model to process the spectrograms and classify the voice data, The model consists of multiple convolutional layers followed by pooling layers to capture hierarchical features from the spectrograms

4

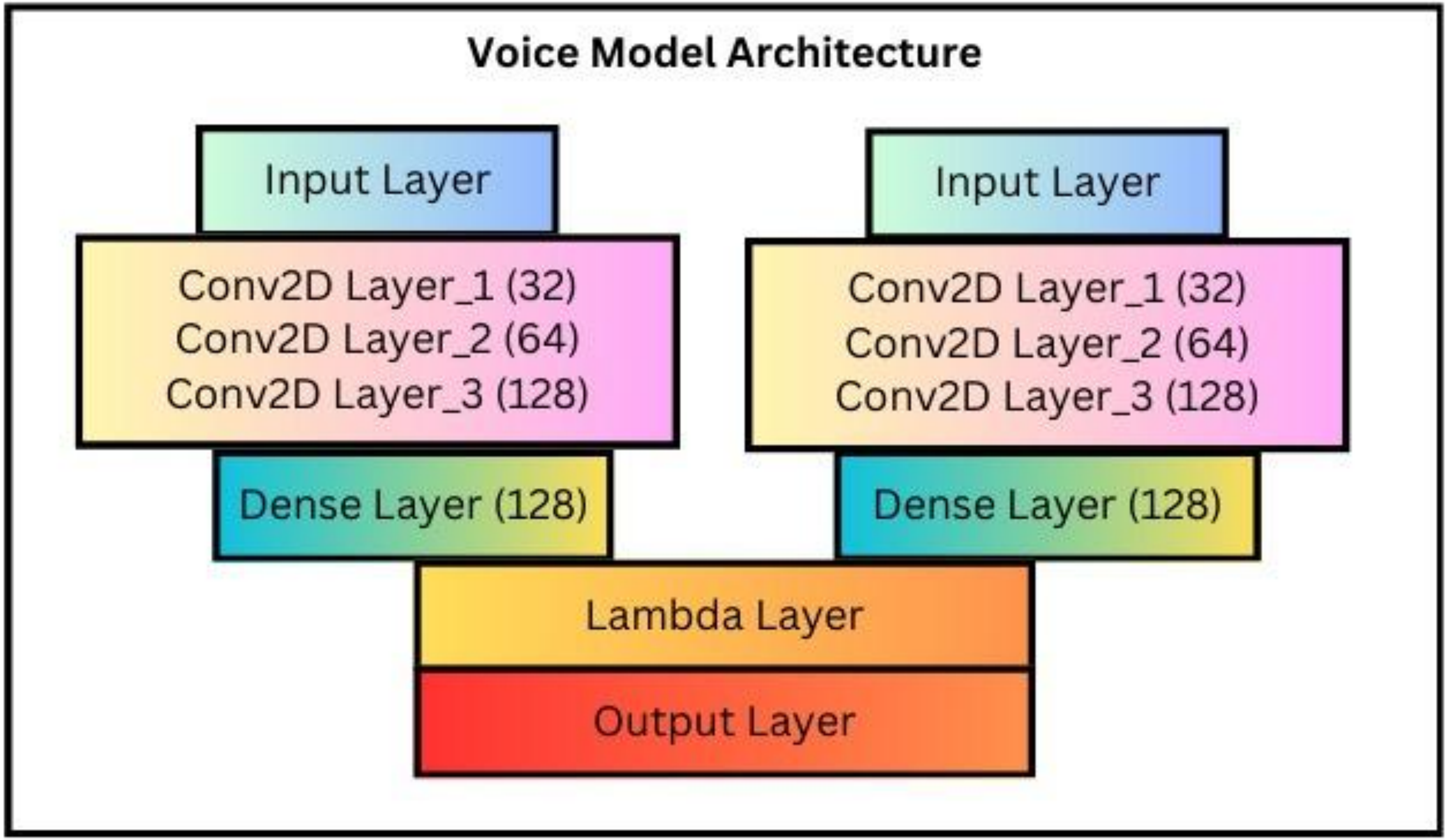
Training and Evaluation

Split the dataset into training, validation, and testing subsets. Use cross validation techniques to avoid overfitting and ensure the model generalizes well to unseen data

Evaluate model performance using metrics such as accuracy, F1 score.

Novelty

- This system combines Siamese Networks with spectrogram-based input for one-shot learning, enabling efficient voice authentication with minimal data. By analyzing detailed spatial-temporal features from spectrograms, it captures unique voice characteristics like pitch, tone, and cadence. The Siamese Network compares voice samples directly, learning to verify identity with just one sample per user, reducing training data requirements.


Voice Model Architecture
Input Layer
Conv2D Layer_1 (32)
Conv2D Layer_2 (64)
Conv2D Layer_3 (128)
Dense Layer (128)
Input Layer
Conv2D Layer_1 (32)
Conv2D Layer_2 (64)
Conv2D Layer_3 (128)
Dense Layer (128)
Lambda Layer
Output Layer

Project Progress Completion

- Online Dataset

Project Progress Completion

- Categorized Dataset

Name	Date modified	Type	Size
fc4bce46580c93612c59da24c6e2c927fbbc...	12/5/2024 10:42 AM	File folder	
fc6f18a6b9e460714135c514d5d840b4804e...	12/5/2024 10:42 AM	File folder	
fc78d7ff60e2d3c8e9fede0f95fcb172eca0f...	12/5/2024 10:43 AM	File folder	
fc80ec33efef9ae8982192f74824c1eeb6d03...	12/5/2024 10:40 AM	File folder	
fc89daa81cd9344fa6dce5eb7321ac8bf53f...	12/5/2024 10:41 AM	File folder	
fc806945d3a7aeecbb2bc8f92635c143953a...	12/5/2024 10:39 AM	File folder	
fcae33f0381c4597a35a3ab361fa75b6c58c...	12/5/2024 10:39 AM	File folder	
fcb70e1ba88be0566cf9ae7b7f7bcd49f011...	12/5/2024 10:40 AM	File folder	
fce08c366ebddbe5883c35b6ebee2779fe3...	12/5/2024 10:40 AM	File folder	
fcebea4213705d790661434a5c3830f1a448...	12/5/2024 10:39 AM	File folder	
fd01d1138eb0ded883f84f163e0b85b60288...	12/5/2024 10:41 AM	File folder	
fd8e03d401a67bf4ebcca2e34d097f32b62e...	12/5/2024 10:41 AM	File folder	
fd14bbb00f69adbe3f71a8dec7af2cde54cb...	12/5/2024 10:43 AM	File folder	
fd20d78f0fef65f21f25d6231e288adbe1278...	12/5/2024 10:41 AM	File folder	
fd55a1c8fbcd0bcf3f25f1b5cc22485436535...	12/5/2024 10:41 AM	File folder	
fd1157c727d71fa70a9e8cede2999f5138b8...	12/5/2024 10:39 AM	File folder	
fd57870e83610def546ff2badf5b0a3863e05...	12/5/2024 10:41 AM	File folder	
fdaff725b56a2a80ce6788a5ded5597e7ec3...	12/5/2024 10:41 AM	File folder	
fde8a1af60055cef902d48a6ce5593144106...	12/5/2024 10:41 AM	File folder	
fde9d29f8b0c9d5f41ff4f5db25b53b27a3c...	12/5/2024 10:42 AM	File folder	
fe0ee2621f87e02c7ee6563852a129538438...	12/5/2024 10:43 AM	File folder	
fe3bb1d8a183e83a54511203efce44b98a6f...	12/5/2024 10:41 AM	File folder	
fe4e32bcecc062b99ca24d67bc024aa9248...	12/5/2024 10:41 AM	File folder	
fe6b98d8ea847c513993dcd2801dcea863f7...	12/5/2024 10:42 AM	File folder	
fe7b652cab375b71c869d8aee9aea262a5d...	12/5/2024 10:42 AM	File folder	
fe12ec033cc38c767644d14f4346aa5ca09c...	12/5/2024 10:40 AM	File folder	
fe28b335a37abd9677ea946ba2fc3a79c156...	12/5/2024 10:39 AM	File folder	

Name	#	Title	Contributing artists	Album
common_voice_en_...				
common_voice_en_...				
common_voice_en_...				
common_voice_en_...				
common_voice_en_...				

Project Progress Completion

- Model Coding

```
import tensorflow as tf
from tensorflow.keras import layers, Model # type: ignore
from tensorflow.keras.utils import to_categorical # type: ignore
import numpy as np

def create_base_network(input_shape):
    """
    Creates the base network for feature extraction.
    :param input_shape: Tuple representing the shape of the input spectrograms.
    :return: A Keras model for feature extraction.
    """
    inputs = layers.Input(shape=input_shape)
    x = layers.Conv2D(32, (3, 3), activation='relu', padding='same')(inputs)
    x = layers.MaxPooling2D(pool_size=(2, 2))(x)
    x = layers.Conv2D(64, (3, 3), activation='relu', padding='same')(x)
    x = layers.MaxPooling2D(pool_size=(2, 2))(x)
    x = layers.Conv2D(128, (3, 3), activation='relu', padding='same')(x)
    x = layers.GlobalAveragePooling2D()(x)
    outputs = layers.Dense(128, activation='relu')(x)  # Feature vector
    return Model(inputs, outputs, name="BaseNetwork")

def siamese_network(input_shape):
    """
    Creates a Siamese network for one-shot learning.
    :param input_shape: Tuple representing the shape of the input spectrograms.
    :return: A compiled Siamese network model.
    """
    # Define the inputs
    input_1 = layers.Input(shape=input_shape, name="Input_1")
    input_2 = layers.Input(shape=input_shape, name="Input_2")

    # Create the base network for shared feature extraction
    base_network = create_base_network(input_shape)

    # Pass both inputs through the base network
    embedding_1 = base_network(input_1)
    embedding_2 = base_network(input_2)

    # Compute the absolute difference between embeddings
    difference = layers.Lambda(lambda tensors: tf.abs(tensors[0] - tensors[1]))([embedding_1, embedding_2])
```

Project Progress Completion

- Model Coding

```
    # Add a dense layer for classification
    outputs = layers.Dense(1, activation='sigmoid')(difference)

    # Define the Siamese network model
    model = Model(inputs=[input_1, input_2], outputs=outputs, name="SiameseNetwork")

    # Compile the model
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

    return model

if __name__ == "__main__":
    # Define input shape (same as target_shape in your preprocessing script)
    input_shape = (100, 100, 1)  # Spectrograms are grayscale

    # Create the Siamese network
    model = siamese_network(input_shape)

    # Print the model summary
    model.summary()

# Load the preprocessed data
input_1 = np.load('/kaggle/working/input_1.npy')
input_2 = np.load('/kaggle/working/input_2.npy')
labels = np.load('/kaggle/working/labels.npy')

# Expand dimensions if spectrograms are grayscale
input_1 = np.expand_dims(input_1, axis=-1)
input_2 = np.expand_dims(input_2, axis=-1)
```

Project Progress Completion

- Model Training

```
from sklearn.model_selection import train_test_split

# Split the data into training and validation sets (80% training, 20% validation)
input_1_train, input_1_val, input_2_train, input_2_val, labels_train, labels_val = train_test_split(
    input_1, input_2, labels, test_size=0.2, random_state=42)

# Train the model
history = model.fit([input_1_train, input_2_train], labels_train,
                    batch_size=64, epochs=10, validation_data=([input_1_val, input_2_val], labels_val))

# Check the training and validation accuracy
print("Training accuracy: ", history.history['accuracy'][-1])
print("Validation accuracy: ", history.history['val_accuracy'][-1])
```

```
Epoch 1/10
73/73 ━━━━━━━━━━━━━━━━━━━━ 109s 1s/step - accuracy: 0.5288 - loss: 0.6838 - val_accuracy: 0.7363 - val_loss: 0.5887
Epoch 2/10
73/73 ━━━━━━━━━━━━━━━━━━━━ 104s 1s/step - accuracy: 0.7023 - loss: 0.5494 - val_accuracy: 0.7517 - val_loss: 0.4801
Epoch 3/10
73/73 ━━━━━━━━━━━━━━━━━━━━ 104s 1s/step - accuracy: 0.7750 - loss: 0.4434 - val_accuracy: 0.8196 - val_loss: 0.3885
Epoch 4/10
73/73 ━━━━━━━━━━━━━━━━━━━━ 104s 1s/step - accuracy: 0.8137 - loss: 0.3857 - val_accuracy: 0.8342 - val_loss: 0.3594
Epoch 5/10
73/73 ━━━━━━━━━━━━━━━━━━━━ 106s 1s/step - accuracy: 0.8489 - loss: 0.3305 - val_accuracy: 0.8608 - val_loss: 0.3032
Epoch 6/10
73/73 ━━━━━━━━━━━━━━━━━━━━ 104s 1s/step - accuracy: 0.8581 - loss: 0.3010 - val_accuracy: 0.8771 - val_loss: 0.2776
Epoch 7/10
73/73 ━━━━━━━━━━━━━━━━━━━━ 103s 1s/step - accuracy: 0.8933 - loss: 0.2600 - val_accuracy: 0.8969 - val_loss: 0.2605
Epoch 8/10
73/73 ━━━━━━━━━━━━━━━━━━━━ 104s 1s/step - accuracy: 0.9069 - loss: 0.2364 - val_accuracy: 0.8943 - val_loss: 0.2487
Epoch 9/10
73/73 ━━━━━━━━━━━━━━━━━━━━ 143s 1s/step - accuracy: 0.9108 - loss: 0.2177 - val_accuracy: 0.8978 - val_loss: 0.2464
Epoch 10/10
73/73 ━━━━━━━━━━━━━━━━━━━━ 144s 1s/step - accuracy: 0.9235 - loss: 0.2086 - val_accuracy: 0.9227 - val_loss: 0.1997
Training accuracy:  0.9213917255401611
Validation accuracy:  0.9226804375648499
```

```
# Save the model after training
model.save('Voice_authentication_model.h5')
```

Project Progress Completion

- Model Training

```
37/37 ━━━━━━━━━━━━━━━━━━━━ 7s 198ms/step
Validation F1 Score:  0.9259868421052632
Validation Precision:  0.911003236245954 7
Validation Recall:  0.9414715719063546
Validation AUC:  0.9736769798031129
```

Progress

PP1 – 50%	PP2 – 90%	Final – 100%
• Dataset acquired and preprocessed. • Model architecture coded (CNN) • Model training initiated	• Train model to achieve acceptable accuracy and F1 score. • Validate model performance with real-world data. • Integrate model output with other system components (e.g., Gait, keystroke).	• Complete frontend development and user interface. • Finalize integration of all components. • Compile and submit the final project report.

Future Interactions For 90% Phase

Model Optimization

- Fine-tune the Siamese network to improve accuracy, focusing on reducing f1 score.

Validation

- Test with real-world data and integrate with other biometric models.

Frontend Preparation

- Plan and prepare for seamless frontend development.

REFERENCES

- Kinnunen, T., & Li, H. (2010). "An overview of text-independent speaker recognition: From features to supervectors". Speech Communication, 52(1), 12-40.
- Graves, A., et al. (2013). "Speech recognition with deep recurrent neural networks". IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP).
- Sainath, T. N., et al. (2015). "Convolutional, Long Short-Term Memory, fully connected Deep Neural Networks". IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP).