National College of Ireland

# Configuration Manual

MSc Research Project
Cyber Security

Nishant Premnath Chavan
Student ID: 23277548

School of Computing
National College of Ireland

Supervisor: Liam Mccabe

**National College of Ireland**

**MSc Project Submission Sheet**

**School of Computing**

| | |
|---|---|
| **Student Name:** | Nishant Premnath Chavan |
| **Student ID:** | 23277548 |
| **Programme:** | MSc Cyber Security **Year:** 2025 |
| **Module:** | Practicum 2 |
| **Lecturer:** | Liam Mccabe |
| **Submission Due Date:** | 15/09/2025 |
| **Project Title:** | ESN-SVM Hybrid Architecture for Free-Text Keystroke Authentication |
| **Word Count:** | 1230 **Page Count:** 26 |

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

| | |
|---|---|
| **Signature:** | Nishant Premnath Chavan |
| **Date:** | 12/09/2025 |

**PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST**

| | |
|---|---|
| Attach a completed copy of this sheet to each project (including multiple copies) | □ |
| **Attach a Moodle submission receipt of the online project submission,** to each project (including multiple copies). | □ |
| **You must ensure that you retain a HARD COPY of the project**, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer. | □ |

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

| **Office Use Only** | |
|---|---|
| Signature: | |
| Date: | |
| Penalty Applied (if applicable): | |

# Table of Contents

## List of Acronyms

| Acronym | Definition |
|---|---|
| **DD** | Down-Down (keystroke timing) |
| **DU** | Down-Up (keystroke timing) |
| **EER** | Equal Error Rate |
| **ESN** | Echo State Networks |
| **GUI** | Graphical User Interface |
| **KD** | Keystroke Dynamics |
| **PCA** | Principal Component Analysis |
| **PyQt6** | Python GUI Framework |
| **ROC** | Receiver Operating Characteristic |
| **SVM** | Support Vector Machine |
| **UD** | Up-Down (keystroke timing) |
| **UU** | Up-Up (keystroke timing) |

# Configuration Manual

Nishant Premnath Chavan
23277548

# 1 Project Overview

This configuration manual provides detailed guidance for configuring and implementing the Echo State Network (ESN) and Support Vector Machine (SVM) hybrid architecture for free-text keystroke authentication system. Additionally, the manual also contains all the important details that are necessary to build a free-text keystroke dynamics GUI application using the hybrid ESN-SVM model.

# 2 System Requirements

The ESN-SVM hybrid model was tested on following hardware specification:

- Processor (CPU): AMD Ryzen 7 5825U, 8 cores, 16 threads.
- Memory (RAM): 16GB DDR4
- Storage: 512GB NVMe SSD
- GPU: Integrated AMD Radeon Graphics
- Operating System: Windows 11 Pro 64-bit

# 3 Software Requirements

The ESN-SVM hybrid model was tested on following software specification:

## 3.1 Programming Language & IDE

- Python 3.10 or higher
- Jupyter Notebook/Lab or Visual Studio Code with Python extension

## 3.2 Python Libraries

**Data Processing and Development Libraries:**

- numpy (2.1.0) - Numerical computing and array operations
- pandas (2.3.0) - Data manipulation and analysis
- ipykernel (6.29.5) - Jupyter kernel for Python

**Machine Learning & Visualizations Libraries:**

- scikit-learn (1.7.0) - SVM classification, preprocessing, and evaluation metrics
- matplotlib (3.10.3) - Data visualization and plotting
- torch (2.7.1) - PyTorch deep learning framework

**Graphical User Interface (GUI) Framework Libraries:**

- PyQt6 - Cross-platform GUI application framework
- keyring - Secure credential storage
- cryptography - Encryption and security functions

# 4 Dataset Details

## 4.1 Free-Text Dataset

The free-text dataset used in this research is from KeyRecs which is available on the (Dias et al., 2023). The dataset is in csv format, named as free-text.csv, and md5 hash value of a5ca6fcb0970cfdcd8eb958b3fe9f22a. The KeyRecs free-text dataset contains 99 participants with 2 sessions each. It has five core temporal features which consist of inter-key latencies measured by the time between each key press and release. It also has demographic information for the 99 participants which can be very helpful to generalize users.

## 4.2 Gibberish Detection Dataset

The gibberish detection system uses the dataset that is derived from *Google Web Trillion Corpus* as detailed by Franz and Brants (2006), which is available on Github as of now (Josh Kaufman, 2014). The gibberish detection dataset contains a list of 20,000 most common English words in ordered frequency which is determined by n-gram by Google's Trillion Word Corpus (Google, 2024).

# 5 Data Setup & Configuration

## 5.1 Data Analysis and Cleaning

Import necessary libraries for data analysis and cleaning and list out total number of participants and keystroke records for further analysis.

```
import pandas as pd
import numpy as np
```

Prints all the participants

```
print(f"Total participants: {df['participant'].nunique()}")
print(f"Total records: {len(df)}")
```

```
Total participants: 99
Total records: 562583
```

List out all the average participants and sessions per participants.

```
# Calculate participant statistics
stats = df.groupby('participant').agg(
    keystrokes=('session', 'count'),
    sessions=('session', 'nunique')
).sort_values(by='keystrokes')

# Summary statistics
print(f"Average keystrokes per participant: {stats['keystrokes'].mean():.0f}")
print(f"Average sessions per participant: {stats['sessions'].mean():.1f}")
```

```
Average keystrokes per participant: 5683
Average sessions per participant: 2.0
```

Remove participants with insufficient data, which can cause problems later in preprocessing.

```
# Remove participant 'p004'
participants_before = df['participant'].nunique()
records_before = len(df)
df = df[df['participant'] != 'p004'].copy()
print(f"Removed participant 'p004'.")
print(f"Participants changed from {participants_before} to {df['participant'].nunique()}.")
print(f"Records changed from {records_before} to {len(df)}.\n")
```

```
Removed participant 'p004'.
Participants changed from 99 to 98.
Records changed from 562583 to 557845.
```

If records are more per user, cap individual sessions. It is recommended to cap minimum of 500 keystrokes for good performance, as per the studies done by Wyciślik et al. (2024).

```
# Cap sessions at 500 keystrokes
print("Capping sessions at 500 keystrokes...")
df_capped = df.groupby(['participant', 'session'], group_keys=False).head(500)
```

```
Capping sessions at 500 keystrokes...
```

Verify the capping by comparing it to original records and capped records.

```
# Verify the capping
session_sizes = df_capped.groupby(['participant', 'session']).size()
print("Capping verification:")
print(f"  Max session size after capping: {session_sizes.max()}")
print(f"  Original records: {len(df)}")
print(f"  Capped records:   {len(df_capped)}")
print(f"  Records removed:  {len(df) - len(df_capped)}\n")
```

```
Capping verification:
  Max session size after capping: 500
  Original records: 557845
  Capped records:   98000
  Records removed:  459845
```

Save the cleaned data.

```
df_capped.to_csv('../data/processed/keystroke_capped.csv', index=False)
print("Capped data saved to '../data/processed/keystroke_capped.csv'")
```

```
Capped data saved to '../data/processed/keystroke_capped.csv'
```

Check if there are any invalid characters that could disturb the model's capability to identify real keystrokes.

```
# Check for invalid characters
valid_chars = set('abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789')
valid_punct = set('.,!?;:\'"()[]{}/-_=+*&%$#@|\\<>`~')
valid_special = set('«º') | set('äçìóõů')
valid_control = {'Space', 'Enter', 'Tab', 'Backspace', 'Delete', 'Shift', 'Control',
                 'Alt', 'AltGraph', 'Meta', 'CapsLock', 'Insert', 'Home', 'End',
                 'ArrowUp', 'ArrowDown', 'ArrowLeft', 'ArrowRight'}
all_valid = valid_chars | valid_punct | valid_special | valid_control
```

```
# Check for invalid characters
invalid_mask = True
for col in ['key1', 'key2']:
    invalid_mask = invalid_mask & df[col].apply(lambda x: str(x) in all_valid if pd.notna(x) else True)

invalid_count = (~invalid_mask).sum()
print(f"Records with invalid characters found: {invalid_count}")
```

```
Records with invalid characters found: 21
```

Remove any corrupted or any characters that are invalid and save the cleaned file.

```
records_before = len(df)

# Remove corrupted and invalid character records
df = df[~corrupted_mask & invalid_mask].copy()

records_after = len(df)
print(f"Removed {records_before - records_after} records.")
print(f"Dataset shape after initial cleaning: {df.shape}")
print("Corrupted and invalid character records have been removed.")

# Save the intermediate cleaned file
df.to_csv('../data/processed/keystroke_context_cleaned.csv', index=False)
print(f"Saved intermediate file to 'keystroke_context_cleaned.csv'")
```

```
Removed 21 records.
Dataset shape after initial cleaning: (97979, 9)
Corrupted and invalid character records have been removed.
Saved intermediate file to 'keystroke_context_cleaned.csv'
```

Detect negative timing intervals.

```
total_negative = 0
for col in TIMING_COLS:
    negative_count = (df[col] < 0).sum()
    if negative_count > 0:
        print(f"Found {negative_count:,} negative values in '{col}'.")
        total_negative += negative_count

print(f"Total negative intervals found: {total_negative:,}")
if total_negative > 0:
    print("Negative timings are physically impossible and must be corrected.")
else:
    print("No negative intervals found.")
```

```
Found 1,108 negative values in 'DD.key1.key2'.
Found 5 negative values in 'DU.key1.key2'.
Found 16,469 negative values in 'UD.key1.key2'.
Found 5 negative values in 'UU.key1.key2'.
Total negative intervals found: 17,587
Negative timings are physically impossible and must be corrected.
```

Correct negative interval timings by clipping them to psysiological bounds and verify it if its fixed.

```
# Fix negatives first
for col in TIMING_COLS:
    df[col] = df[col].clip(lower=0.01)
```

```
# Apply specific caps per column
caps = {'DD.key1.key2': 1.47, 'DU.key1.key2': 0.50, 'UD.key1.key2': 1.38, 'UU.key1.key2': 1.40, 'DU.key1.key1': 0.26,}
for col, cap in caps.items():
    df[col] = df[col].clip(upper=cap)

print("All 5 timing features clipped with specific physiological bounds.")
```

```
All 5 timing features clipped with specific physiological bounds.
```

```
# VERIFICATION
total_negative_after = sum((df[col] < 0).sum() for col in TIMING_COLS)
print(f"Negative intervals remaining: {total_negative_after}")
if total_negative_after == 0:
    print("Successfully corrected all negative intervals.")
```

```
Negative intervals remaining: 0
Successfully corrected all negative intervals.
```

Detect temporal consistency, as low consistency shows system errors or measurement artifacts, that should be corrected.

```
tolerance = 0.050

theoretical_uu = df['UD.key1.key2'] + df['DD.key1.key2']
error = np.abs(theoretical_uu - df['UU.key1.key2'])
inconsistent_mask = error > tolerance

consistency_rate = (len(df) - inconsistent_mask.sum()) / len(df) * 100

print(f"Found {inconsistent_mask.sum():,} records where the temporal relationship is violated.")
print(f"Current consistency rate: {consistency_rate:.1f}%")
print("The low consistency rate indicates measurement artifacts and must be corrected.")
```

```
Found 66,066 records where the temporal relationship is violated.
Current consistency rate: 32.6%
The low consistency rate indicates measurement artifacts and must be corrected.
```

Fix temporal consistency by enforcing UU = UD + DD for inconsistent records & save file.

```
# For inconsistent records, set UU = UD + DD
df.loc[inconsistent_mask, 'UU.key1.key2'] = theoretical_uu[inconsistent_mask]

# Re-apply the physiological cap to the corrected values
df['UU.key1.key2'] = df['UU.key1.key2'].clip(lower=0.01, upper=2.0)
print("Enforced UU = UD + DD for inconsistent records.")
```

```
Enforced UU = UD + DD for inconsistent records.
```

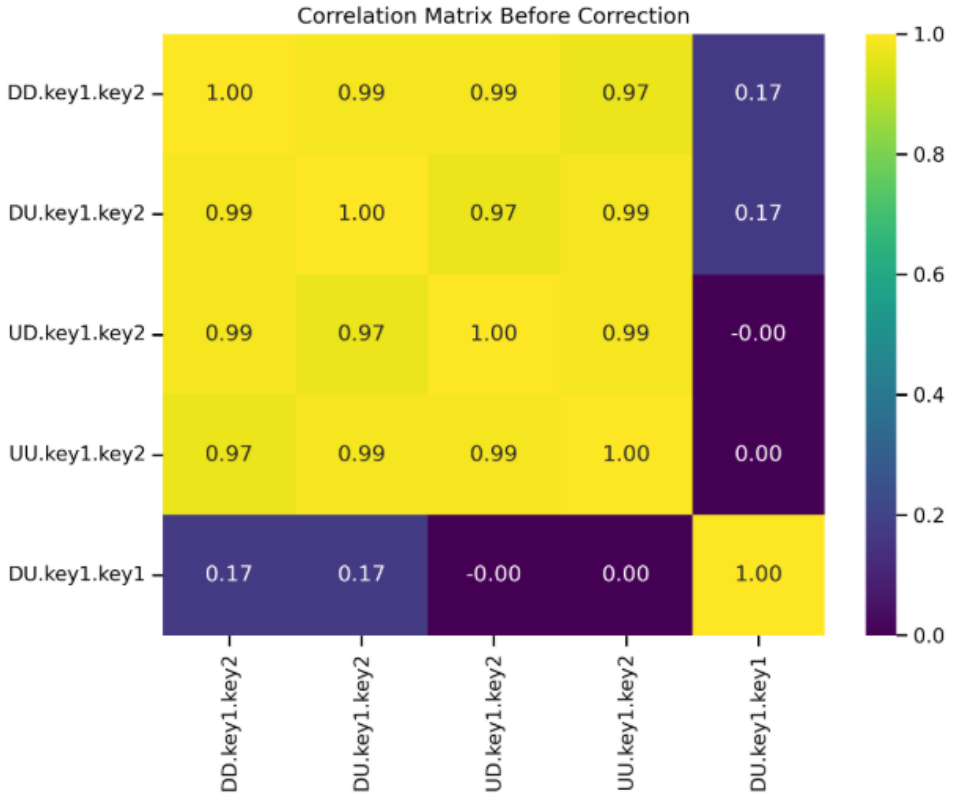
```
# VERIFICATION
new_error = np.abs(df['UD.key1.key2'] + df['DD.key1.key2'] - df['UU.key1.key2'])
final_consistency = (new_error <= tolerance).sum() / len(df) * 100
print(f"Final consistency rate after correction: {final_consistency:.1f}%")
```

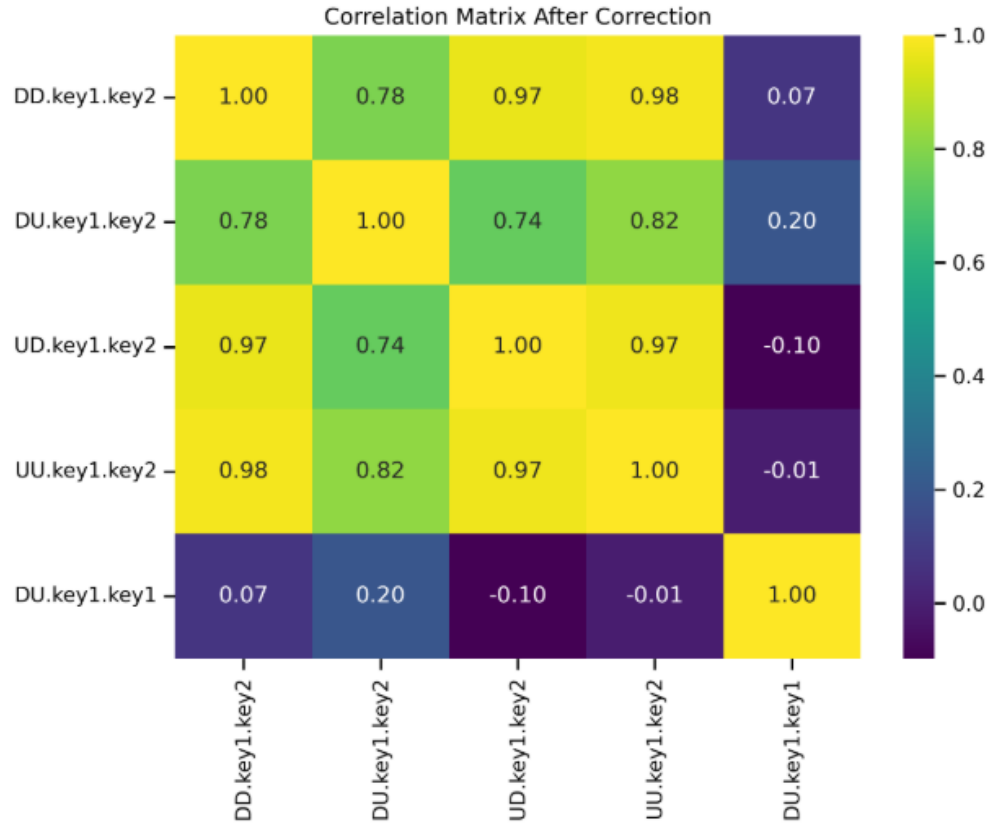
```
Final consistency rate after correction: 97.9%
```

```
# Save the temporally corrected file, as it's a major milestone
df.to_csv('../data/processed/keystroke_basic_interval_fix.csv', index=False)
df.to_csv('../data/processed/keystroke_temporal_fix.csv', index=False)
print("Saved temporally corrected data to 'keystroke_temporal_fix.csv'")
```

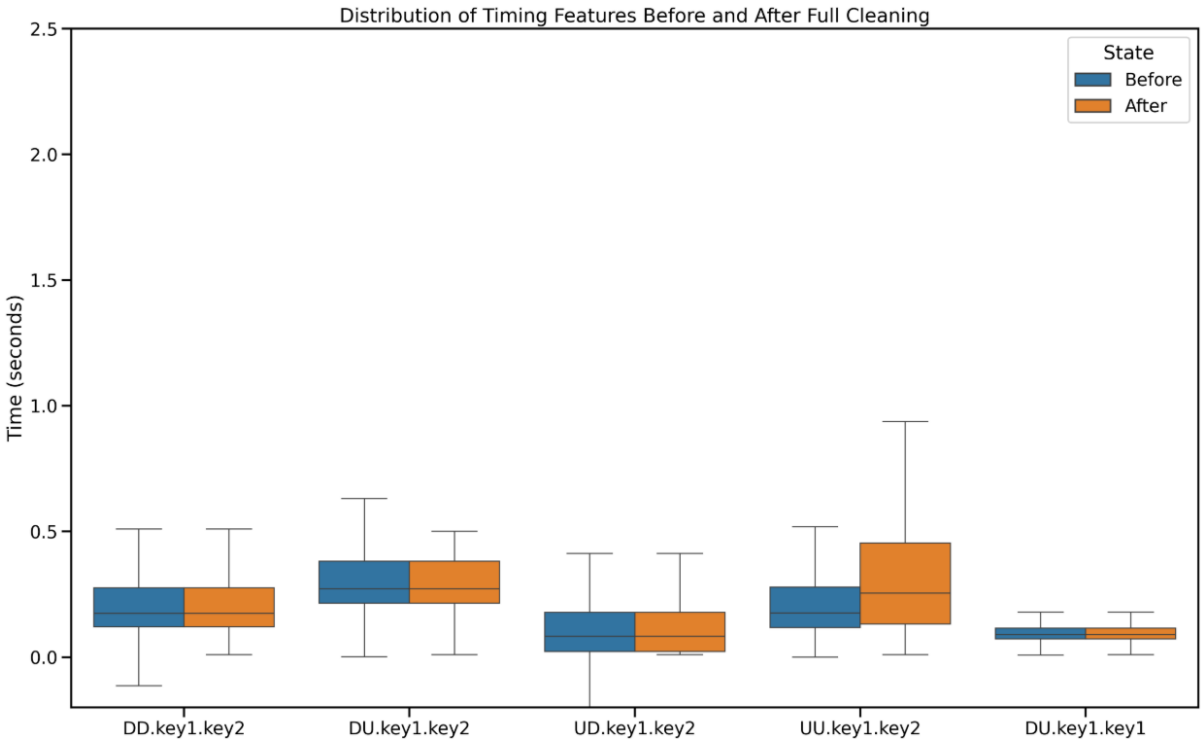
```
Saved temporally corrected data to 'keystroke_temporal_fix.csv'
```

Visualize corelation matrix as it can show how close timing features are related to eachother.



Visualize distribution of timing features before and after cleaning, as it can show feature spread and detect noise.




## 5.2 Data Splitting

Import necessary libraries for dataset splitting, and configure input file and output paths.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
import json
from pathlib import Path
```

```
# Configuration
INPUT_FILE = '../data/processed/keystroke_temporal_fix.csv'
OUTPUT_DIR = '../data/splits/'
RANDOM_SEED = 42
TIMING_FEATURES = ['DU.key1.key1', 'DD.key1.key2', 'DU.key1.key2', 'UD.key1.key2', 'UU.key1.key2']
```

Verify session balance ratio.

```
# Verfying Session Balance
session_balance = df.groupby(['participant', 'session']).size().unstack(fill_value=0)
session_balance['ratio'] = session_balance[2] / session_balance[1].replace(0, 1) # Avoid division by zero
imbalanced = session_balance[(session_balance['ratio'] < 0.8) | (session_balance['ratio'] > 1.2)]

if not imbalanced.empty:
    print(f"Found {len(imbalanced)} users with imbalanced sessions.")
else:
    print("All users have reasonably balanced sessions (ratio between 0.8 and 1.2).")
```

```
All users have reasonably balanced sessions (ratio between 0.8 and 1.2).
```

Check cross session consistency using average Pearson's r, if Pearson's r is high, that means a persons typing patterns are stable over time.

```
# Cross-Session Consistency Check
consistency_scores = []
for participant, group in df.groupby('participant'):
    s1 = group[group['session'] == 1][TIMING_FEATURES].mean()
    s2 = group[group['session'] == 2][TIMING_FEATURES].mean()
    if not s1.isna().any() and not s2.isna().any():
        corr, _ = stats.pearsonr(s1, s2)
        consistency_scores.append(corr)

avg_consistency = np.mean(consistency_scores) if consistency_scores else 0
if avg_consistency > 0.8:
    print(f"High cross-session consistency found (Average Pearson's r: {avg_consistency:.3f}).")
else:
    print(f"Low cross-session consistency (Average Pearson's r: {avg_consistency:.3f}).")
```

```
High cross-session consistency found (Average Pearson's r: 0.992).
```

Split the dataset into training, validation and test set, using User-Disjoint Split, and ensure no overlaps are there between the training, validation and test set.

```
# Ensure the output directory exists
output_path = Path(OUTPUT_DIR)
output_path.mkdir(parents=True, exist_ok=True)
```

```
# Get unique participants and shuffle them for random assignment
np.random.seed(RANDOM_SEED)
participants = sorted(df['participant'].unique())
participants_shuffled = np.random.permutation(participants)
```

```
# Define split boundaries and create participant lists
train_participants = participants_shuffled[:60].tolist()
val_participants = participants_shuffled[60:79].tolist()
test_participants = participants_shuffled[79:98].tolist()

print(f"Participant Allocation: Train={len(train_participants)}, Val={len(val_participants)}, Test={len(test_participants)}")
```

```
Participant Allocation: Train=60, Val=19, Test=19
```

```
# Verify no overlaps between sets
assert len(set(train_participants) & set(val_participants) & set(test_participants)) == 0, "FATAL: Overlap detected in splits!"
print("No user overlaps.")
```

```
No user overlaps.
```

Verify each split participants and records and create a data frame for each split.

```
# Create the dataframes for each split
train_df = df[df['participant'].isin(train_participants)].copy()
val_df = df[df['participant'].isin(val_participants)].copy()
test_df = df[df['participant'].isin(test_participants)].copy()

print(f"Data Split Sizes:")
print(f"Train: {len(train_df):,} records ({train_df['participant'].nunique()} users)")
print(f"Val:   {len(val_df):,} records ({val_df['participant'].nunique()} users)")
print(f"Test:  {len(test_df):,} records ({test_df['participant'].nunique()} users)")
```

```
Data Split Sizes:
Train: 59,987 records (60 users)
Val:   18,996 records (19 users)
Test:  18,996 records (19 users)
```

Save the train, val and  test split to a folder using the dataframe that created in previous step.

```
# Save the split dataframes
train_df.to_csv(output_path / 'train_split.csv', index=False)
val_df.to_csv(output_path / 'val_split.csv', index=False)
test_df.to_csv(output_path / 'test_split.csv', index=False)
print(f"Split CSV files saved to '{OUTPUT_DIR}'")
```

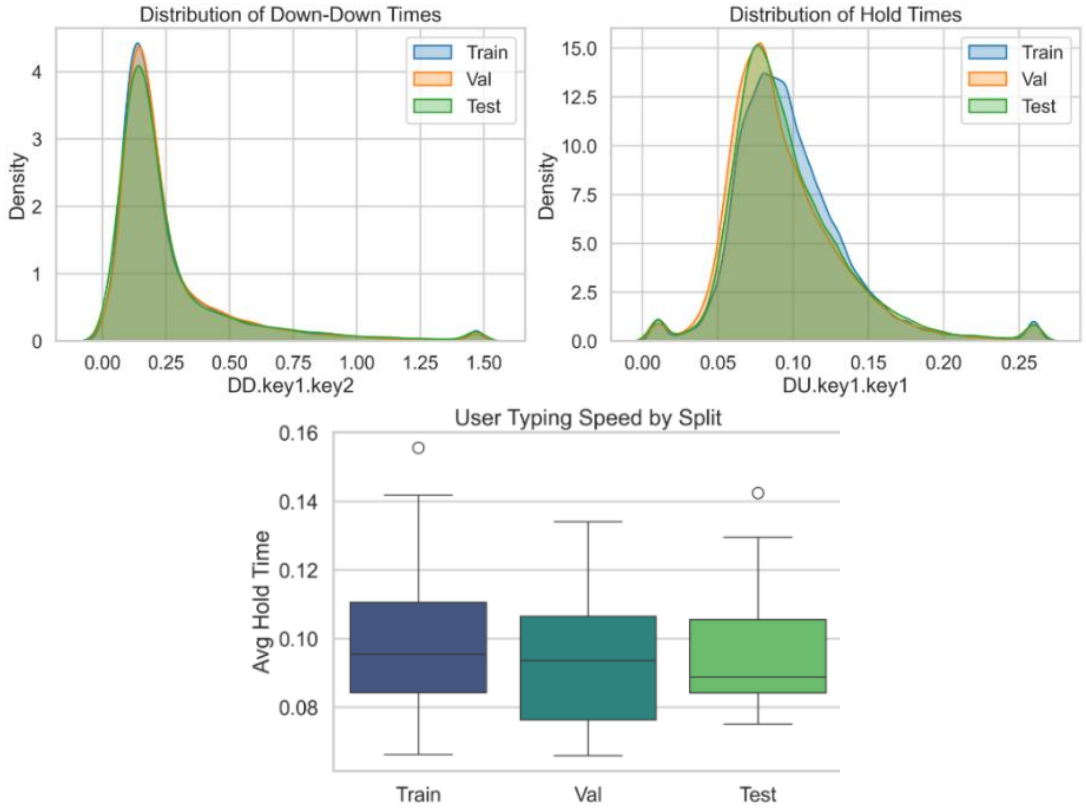
```
Split CSV files saved to '../data/splits/'
```

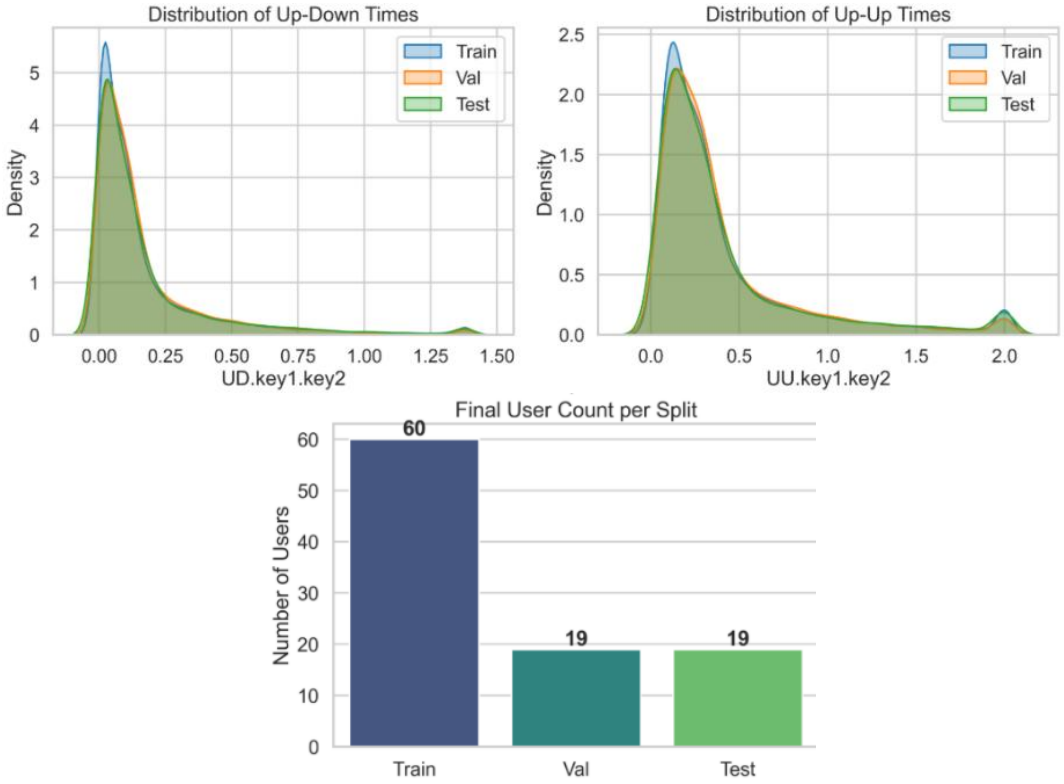
Check if the splits have similar statistical distribution using the Kolmogorov Smirnov (KS) test.

```
# Testing if splits have similar statistical distribution
splits = {'Train': train_df, 'Val': val_df, 'Test': test_df}
print("Testing if splits have similar statistical distributions...")

pvalues = [
    stats.ks_2samp(train_df[feature].dropna(), test_df[feature].dropna()).pvalue
    for feature in TIMING_FEATURES
]

avg_pvalue = np.mean(pvalues)
print(f"Average p-value from KS tests (Train vs. Test): {avg_pvalue:.3f}")

print("The splits are statistically similar (p > 0.05). This is good!"
      if avg_pvalue > 0.05
      else "The splits show some statistical differences (p < 0.05). This is acceptable but worth noting.")
```

```
Testing if splits have similar statistical distributions...
Average p-value from KS tests (Train vs. Test): 0.022
The splits show some statistical differences (p < 0.05). This is acceptable but worth noting.
```

Visualize the distribution of features post validation, user timing and user count by split.






## 5.3 Feature Analysis

As the study is based upon using only 5 core temporal features, we are not going to engineer features as ESNs have natural temporal capabilities which will create multiple feature states in the reservoir (Sun et al., 2023).

Load the configuration and 5 core features.

```
# Configuration
SPLITS_DIR = '../data/splits/'
FEATURES_DIR = '../data/features/'
CORE_FEATURES = ['DU.key1.key1', 'DD.key1.key2', 'DU.key1.key2', 'UD.key1.key2', 'UU.key1.key2']
```

Make sure that 5 features do not have negative values.

```
# Analyze Core Features Quality
all_data = pd.concat([train_df, val_df, test_df], ignore_index=True)
```

```
# Quality Check
has_issues = False
for feature in CORE_FEATURES:
    neg_count = (all_data[feature] < 0).sum()
    if neg_count > 0:
        print(f"Quality Issue: Found {neg_count} negative values in {feature}!")
        has_issues = True

if not has_issues:
    print("No negative values found in core features.")
```

```
No negative values found in core features.
```

## 5.4 ESN Baseline Implementation

Import necessary libraries for Echo State Network Baseline Implementation

```
# 1. Imports
import numpy as np
import pandas as pd
from scipy.linalg import eig
from scipy.spatial.distance import euclidean
from sklearn.metrics import roc_curve
```

Load data for Echo State Networks

```
def load_data():
    features = ['DU.key1.key1', 'DD.key1.key2', 'DU.key1.key2', 'UD.key1.key2', 'UU.key1.key2']
    def process(df):
        sequences, users = [], []
        for p in df['participant'].unique():
            for s in df[df['participant'] == p]['session'].unique():
                seq = df[(df['participant'] == p) & (df['session'] == s)][features].values
                seq = np.nan_to_num(seq)
                if len(seq) >= 50:
                    sequences.append(seq)
                    users.append(p)
        return sequences, users

    train_data, train_users = process(pd.read_csv('../data/features/train_features.csv'))
    val_data, val_users = process(pd.read_csv('../data/features/val_features.csv'))
    test_data, test_users = process(pd.read_csv('../data/features/test_features.csv'))
```

The below class builds a sparse matrix, randomly connected reservoir with controlled spectral radius. Seed 42 is used for reproducibility.

```
class ESN:
    def __init__(self, size, sr, scale, conn, wash, seed=42):
        np.random.seed(seed)
        self.size, self.wash = size, wash
        self.W_in = np.random.uniform(-scale, scale, (size, 5))

        W_res = np.zeros((size, size))
        for _ in range(int(size * size * conn)):
            i, j = np.random.randint(0, size, 2)
            W_res[i, j] = np.random.normal()

        if np.any(W_res):
            eigenvals = eig(W_res)[0]
            current_sr = np.max(np.abs(eigenvals))
            if current_sr > 0:
                W_res *= sr / current_sr
        self.W_res = W_res
```

Generate genuine and imposter pairs for ESN evaluation

```
    # Genuine pairs (same user)
    for user, feats in user_feats.items():
        if len(feats) >= 2:
            for i in range(len(feats)):
                for j in range(i+1, len(feats)):
                    genuine.append(euclidean(feats[i], feats[j]))

    # Impostor pairs (different users)
    users_list = list(user_feats.keys())
    for i, user1 in enumerate(users_list):
        for user2 in users_list[i+1:]:
            for feat1 in user_feats[user1]:
                for feat2 in user_feats[user2]:
                    impostor.append(euclidean(feat1, feat2))
```

Add guassian noise and make configurations for the ESN reservoir.

```
def add_noise(sequences, noise_std=0.01):
    return [seq + np.random.normal(0, noise_std, seq.shape) for seq in sequences]

# 5: MAIN EXECUTION
configs = [
    (800, 0.975, 0.90, 0.14, 40),
    (1000, 0.98, 0.95, 0.12, 30),
    (600, 0.95, 0.85, 0.15, 50),
    (1200, 0.99, 1.0, 0.10, 20),
    (400, 0.90, 0.70, 0.20, 60),
]
```

Select the best ESN configurations which will be tested on validation set.

```
results.sort()
best_score, best_orig, best_noisy, best_idx, best_params = results[0]

print(f"\nBest: Config {best_idx} (Score: {best_score:.4f})")
print(f"Original EER: {best_orig:.4f}, Noisy EER: {best_noisy:.4f}")
```

Validate the ESN results which is tested on validation set using Equal Error Rate (EER).

```
Train: 120, Val: 38, Test: 38

Testing configs with noise validation...
Training pairs: 19 genuine, 684 impostor
Config 1: Original=0.1082, Noisy=0.1082, Score=0.1082
Config 2: Original=0.1096, Noisy=0.1118, Score=0.1129
Config 3: Original=0.1023, Noisy=0.1016, Score=0.1027
Config 4: Original=0.1104, Noisy=0.1104, Score=0.1104
Config 5: Original=0.1126, Noisy=0.1140, Score=0.1148

Best: Config 3 (Score: 0.1027)
Original EER: 0.1023, Noisy EER: 0.1016
```

## 5.5 ESN-SVM Hybrid Model Creation

Import necessary libraries to create ESN and SVM hybrid model.

```
import numpy as np
import pandas as pd
from scipy.linalg import eig
from sklearn.metrics import roc_curve, precision_recall_fscore_support, accuracy_score
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
```

Load the train, validation, and test split.

```
(train_data, train_masks, train_users), \
(val_data, val_masks, val_users), \
(test_data, test_masks, test_users) = load_data_with_splits()

print(f"Train: {len(train_data)} sequences from {len(set(train_users))} users")
print(f"Validation: {len(val_data)} sequences from {len(set(val_users))} users")
print(f"Test: {len(test_data)} sequences from {len(set(test_users))} users")
```

Fit the scaler only on the train set to learn the core features well.

```
input_scaler = StandardScaler()
input_scaler.fit(valid_train_data)

# Apply the correctly fitted scaler to all datasets
# The transform function is applied to the full sequences, which is correct.
train_data_norm = [input_scaler.transform(seq) for seq in train_data]
val_data_norm = [input_scaler.transform(seq) for seq in val_data]
test_data_norm = [input_scaler.transform(seq) for seq in test_data]
```

Create ESN Reservoir

```
def create_reservoir(reservoir_size, spectral_radius, input_scaling, connectivity, seed=42):
    np.random.seed(seed)

    # Create input-to-reservoir weights
    W_in = np.random.uniform(-input_scaling, input_scaling, (reservoir_size, 5))

    # Create sparse reservoir-to-reservoir weights
    total_connections = reservoir_size * reservoir_size
    num_connections = int(total_connections * connectivity)
```

Extract the learned temporal features from the ESN reservoir state.

```
def extract_esn_features(sequence, mask, W_in, W_res, washout_period, leak_rate=1.0):
    sequence_length = len(sequence)
    reservoir_size = W_res.shape[0]
```

Create genuine and imposter pairs and train SVM classifier on that genuine and imposter pairs. Also define 5 ESN configuration parameters for testing. Create augmentation for validation data only

```
Creating augmented validation set...
Original validation: 38 sequences from 19 users
Augmented validation: 152 sequences from 19 users
```

Test the ESN configurations on the validation set.

```
Testing configurations on validation sets...

Testing Configuration 1:
  Original Val - EER: 0.0000, F1: 1.0000
  Augmented Val - EER: 0.0226, F1: 0.9775
  Robustness Gap: 0.0226

Testing Configuration 2:
  Original Val - EER: 0.0000, F1: 1.0000
  Augmented Val - EER: 0.0226, F1: 0.9774
  Robustness Gap: 0.0226

Testing Configuration 3:
  Original Val - EER: 0.0000, F1: 1.0000
  Augmented Val - EER: 0.0329, F1: 0.9671
  Robustness Gap: 0.0329

Testing Configuration 4:
  Original Val - EER: 0.0000, F1: 1.0000
  Augmented Val - EER: 0.0207, F1: 0.9793
  Robustness Gap: 0.0207

Testing Configuration 5:
  Original Val - EER: 0.0000, F1: 1.0000
  Augmented Val - EER: 0.0291, F1: 0.9708
  Robustness Gap: 0.0291

Completed testing 5 configurations
```

Best ESN configuration would be automatically selected by the model on the robustness gap. Validate the results using EER, robustness gap and combined score.

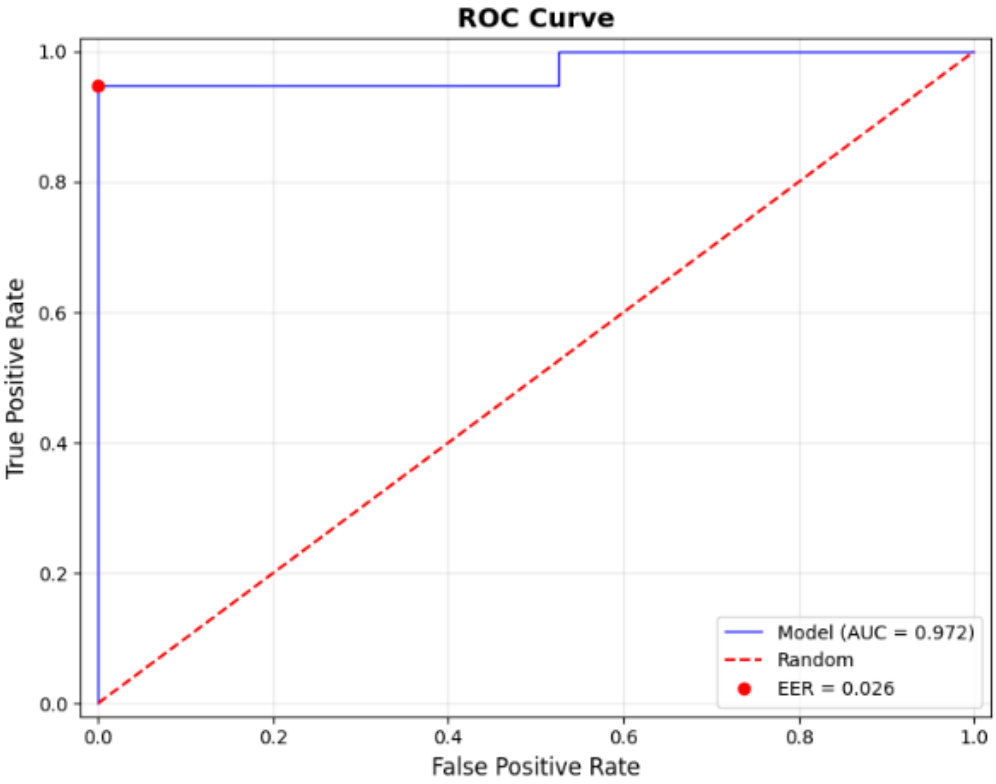
```
Validation Results Analysis:
Config | Original EER | Augmented EER | Robustness Gap | Combined Score
----------------------------------------------------------------------
  1    |    0.0000    |    0.0226     |     0.0226     |    0.0338
  2    |    0.0000    |    0.0226     |     0.0226     |    0.0338
  3    |    0.0000    |    0.0329     |     0.0329     |    0.0493
  4    |    0.0000    |    0.0207     |     0.0207     |    0.0310
  5    |    0.0000    |    0.0291     |     0.0291     |    0.0437

Best Configuration Analysis:
Selected Config 4 based on robustness
Original Validation EER: 0.0000
Augmented Validation EER: 0.0207
Robustness Gap: 0.0207
Combined Score: 0.0310
Best params: Size=1800, SR=1.01, InputScale=0.96, Connectivity=8%, Washout=20, Leak=0.85
```

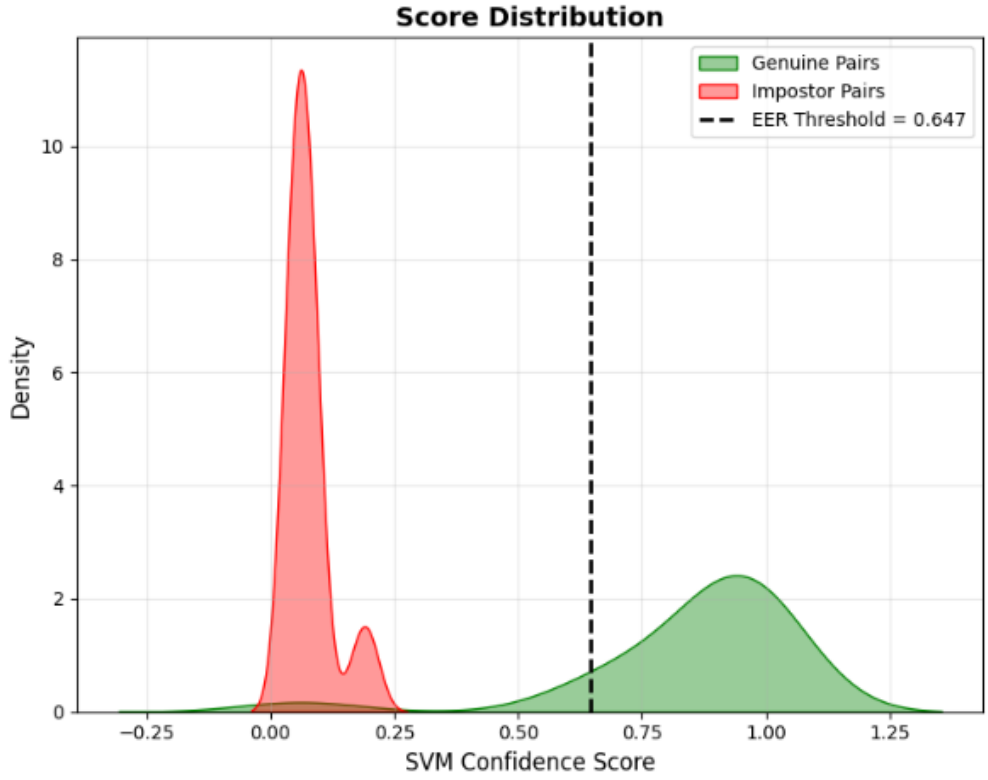
Train the final model only on the training data.

```
Training final model with best robust configuration...
Final model trained successfully
Created templates for 60 users
Training pairs: 60 genuine, 60 impostor
Note: Model trained ONLY on training data, no validation data used
```

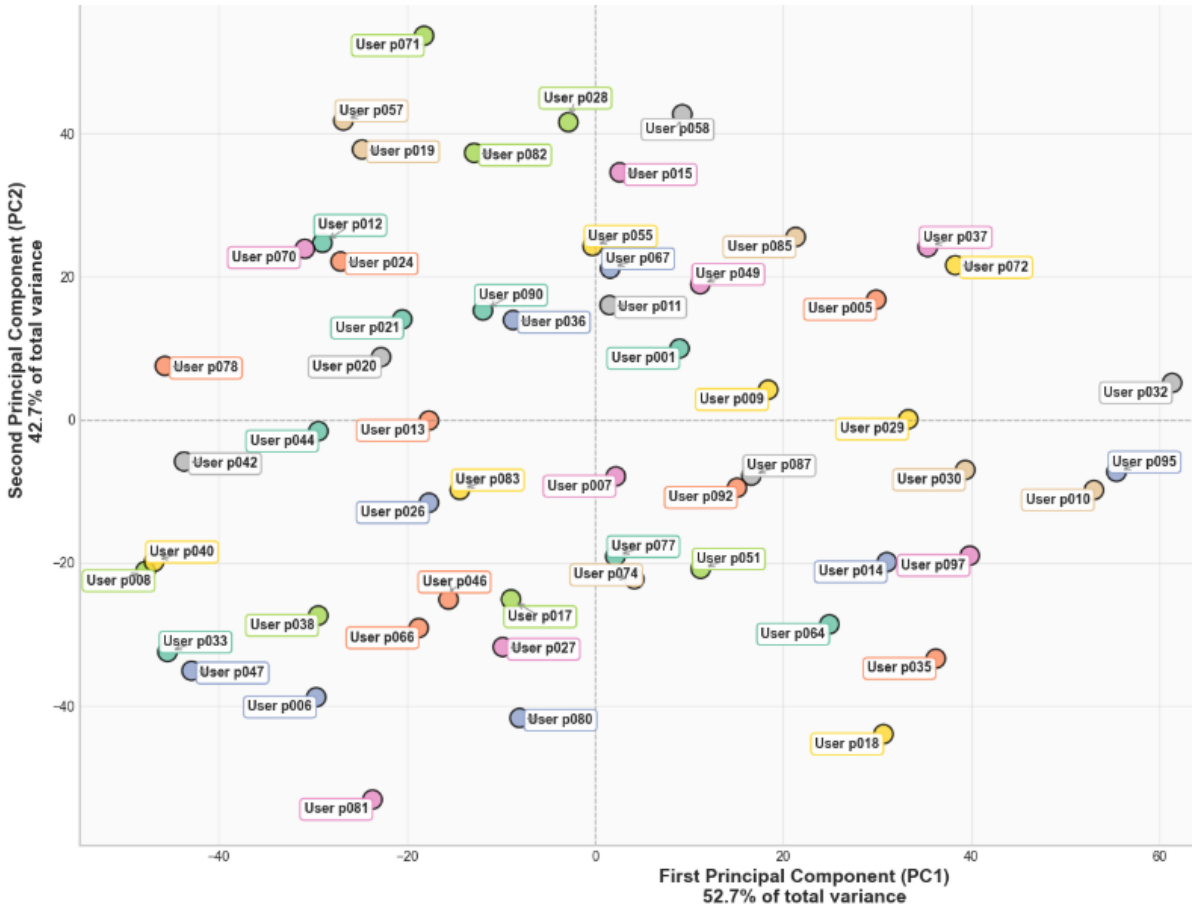
Evaluate and visualize the final model which is tested on test data, using EER and original, validation and generalization gap.

```
FINAL TEST RESULTS:
Equal Error Rate (EER): 0.0263
F1 Score: 0.9730
Optimal Threshold: 0.6468
esn_svm_model.pkl model saved!

ROBUSTNESS ANALYSIS:
Best config showed robustness gap of 0.0207 on validation
Original validation EER: 0.0000
Augmented validation EER: 0.0207
Final test EER: 0.0263
Generalization gap (val to test): 0.0263
Moderate generalization - acceptable gap
```


ROC Curve
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
Model (AUC = 0.972)
Random
EER = 0.026


Score Distribution
Density
SVM Confidence Score
Genuine Pairs
Impostor Pairs
EER Threshold = 0.647
0
2
4
6
8
10
−0.25
0.00
0.25
0.50
0.75
1.00
1.25

Visualize Principal Component Analysis (PCA) distribution of user templates and validate them.




## 5.6 GUI Simulation for Authentication Analysis

Layout the security configurations for the GUI simulation that are necessary for authentication.

```
CONFIDENCE_FLOOR = 0.68
SUSPICIOUS_SCORE_THRESHOLD = 0.75
MAX_WINDOW_SIZE = 10
ANCHOR_WEIGHT = 0.7
WINDOW_WEIGHT = 0.3
DRIFT_SESSIONS_FOR_REANCHOR = 5
CONSECUTIVE_ANOMALY_LIMIT = 3
MIN_SECURE_SVM_THRESHOLD = 0.70
MIN_SECURE_COSINE_THRESHOLD = 0.90
```

Enroll all unseen users from validation set and test set because they were not used in model training which totals to 38 unique users.

```
all_unseen_users_list = val_users + test_users
all_unseen_data = val_data_norm + test_data_norm
all_unseen_masks = val_masks + test_masks
enrolled_user_ids = sorted(list(set(all_unseen_users_list)))
enrolled_user_thresholds = {}
live_profiles = {}
```

Extract ESN features of each user and calculate personalized thresholds such as SVM, cosine similarity, Euclidean distance.

```
for user_id in enrolled_user_ids:
    user_indices = [i for i, u in enumerate(all_unseen_users_list) if u == user_id]
    user_vectors = []
    for idx in user_indices:
        features = extract_esn_features(all_unseen_data[idx], all_unseen_masks[idx], W_input_final, W_reservoir_final, washout,
        user_vectors.append(feature_scaler_final.transform(features.reshape(1, -1)).flatten())
    svm_scores, cos_sims, euc_dists = [], [], []
    if len(user_vectors) >= 2:
        for v1, v2 in combinations(user_vectors, 2):
            svm_scores.append(svm_classifier_final.predict_proba(np.abs(v1 - v2).reshape(1, -1))[0, 1])
            cos_sims.append(cosine_similarity([v1], [v2])[0, 0])
            euc_dists.append(euclidean_distances([v1], [v2])[0, 0])
    enrolled_user_thresholds[user_id] = {
        "svm": max(MIN_SECURE_SVM_THRESHOLD, (np.percentile(svm_scores, 5) - 0.08) if svm_scores else MIN_SECURE_SVM_THRESHOLD),
        "cos": (np.percentile(cos_sims, 5) - 0.02) if cos_sims else MIN_SECURE_COSINE_THRESHOLD,
        "dist": (np.percentile(euc_dists, 95) + 1.0) if euc_dists else 10.0
    }
```

Profile management via re anchoring and rolling window.

```
    initial_window_sample = [user_vectors[-1]] if user_vectors else []
    anchor_samples = user_vectors[:-1] if len(user_vectors) > 1 else initial_window_sample
    live_profiles[user_id] = {
        'esn_anchor': np.mean(anchor_samples, axis=0) if anchor_samples else np.zeros_like(user_vectors[0]),
        'rolling_window': initial_window_sample, 'quarantined_samples': [], 'drift_counter': 0, 'consecutive_anomaly_count': 0
    }
```

Feature extraction for user authentication

```
live_features_genuine = extract_esn_features(sequence, mask, W_input_final, W_reservoir_final, washout, leak_rate)
live_vector_genuine = feature_scaler_final.transform(live_features_genuine.reshape(1, -1))
```

Anchor template matching

```
profile = live_profiles[user_id]
thresholds = enrolled_user_thresholds[user_id]
svm_anchor = svm_classifier_final.predict_proba(np.abs(live_vector_genuine - profile['esn_anchor']))[0, 1]
cos_anchor = cosine_similarity(live_vector_genuine, [profile['esn_anchor']])[0, 0]
euc_anchor = euclidean_distances(live_vector_genuine, [profile['esn_anchor']])[0, 0]
```

Adaptive template matching

```
adaptive_template = (ANCHOR_WEIGHT * profile['esn_anchor']) + (WINDOW_WEIGHT * np.mean(profile['rolling_window'], axis=0)) if profile['rolling_window'] else profile['esn_anchor']
svm_adaptive = svm_classifier_final.predict_proba(np.abs(live_vector_genuine - adaptive_template))[0, 1]
cos_adaptive = cosine_similarity(live_vector_genuine, [adaptive_template])[0, 0]
```

Profile update logic

```
    disposition = "QUARANTINED"
    if svm_adaptive >= SUSPICIOUS_SCORE_THRESHOLD and is_anchor_match:
        disposition = "TRUSTED"
    elif svm_adaptive >= SUSPICIOUS_SCORE_THRESHOLD and not is_anchor_match:
        disposition = "DRIFT"
    if disposition in ["TRUSTED", "DRIFT"]:
        profile['rolling_window'].append(live_vector_genuine.flatten())
        if disposition == "TRUSTED" and profile['quarantined_samples']:
            profile['rolling_window'].extend(profile['quarantined_samples'])
            profile['quarantined_samples'].clear()
        profile['consecutive_anomaly_count'] = 0
    else:
        profile['quarantined_samples'].append(live_vector_genuine.flatten())
        profile['consecutive_anomaly_count'] += 1
```

Drift detection and reanchoring

```
    if is_authenticated:
        if is_adaptive_match and not is_anchor_match:
            profile['drift_counter'] += 1
        else:
            profile['drift_counter'] = 0
    if profile['drift_counter'] >= DRIFT_SESSIONS_FOR_REANCHOR and profile['rolling_window']:
        profile['esn_anchor'] = np.mean(profile['rolling_window'], axis=0)
        profile['rolling_window'], profile['drift_counter'] = [], 0
    if profile['consecutive_anomaly_count'] >= CONSECUTIVE_ANOMALY_LIMIT and profile['quarantined_samples']:
        profile['esn_anchor'] = np.mean(profile['quarantined_samples'], axis=0)
        profile['quarantined_samples'], profile['consecutive_anomaly_count'] = [], 0
    profile['rolling_window'] = profile['rolling_window'][-MAX_WINDOW_SIZE:]
```

Imposter attack simulation

```
other_users = [uid for uid in enrolled_user_ids if uid != user_id]
target_user_id = random.choice(other_users)
target_profile = live_profiles[target_user_id]
```

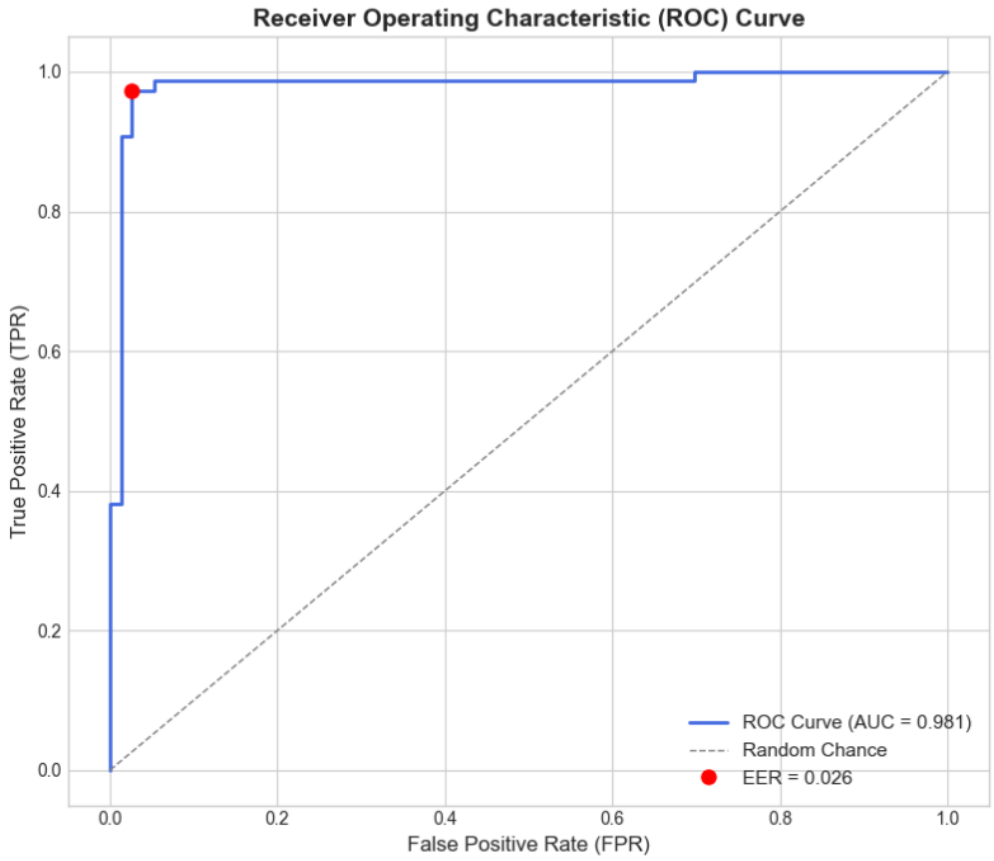
ROC curve generation

```
y_true = np.array([1] * len(genuine_attempts) + [0] * len(impostor_attempts))
y_scores = np.concatenate([genuine_attempts['SVM_Score'].values, impostor_attempts['SVM_Score'].values])
fpr, tpr, thresholds = roc_curve(y_true, y_scores)
auc = roc_auc_score(y_true, y_scores)
fnr = 1 - tpr
eer_index = np.nanargmin(np.abs(fpr - fnr))
eer = (fpr[eer_index] + fnr[eer_index]) / 2
fig, (ax_roc, ax_det) = plt.subplots(1, 2, figsize=(16, 7))
```

Evaluate the results from the simulation log via 3 scores which are seen in the logs.

```
--- Detailed Simulation Log ---
            User      Type  Disposition Auth_Result  SVM_Score  Cos_Score  Euc_Score  SVM_Thresh  Cos_Thres
h  Euc_Thresh
0           p034   Genuine      TRUSTED        PASS     0.9546     0.9990     1.3137      0.8890      0.967
2     5.3789
1    p034 -> p052  Impostor     Rejected        FAIL     0.0634    -0.4997    52.4714      0.7864      0.909
9    16.1603
2           p034   Genuine      TRUSTED        PASS     0.9668     0.9905     3.7221      0.8890      0.967
2     5.3789
3    p034 -> p048  Impostor     Rejected        FAIL     0.4345    -0.1170    24.6007      0.8843      0.155
4    15.0639
4           p043   Genuine      TRUSTED        PASS     0.9526     0.9999     1.2617      0.8836      0.979
1     5.2058
5    p043 -> p069  Impostor     Rejected        FAIL     0.0937     0.3837    66.6225      0.7000      0.262
1    22.2689
6           p043   Genuine      TRUSTED        PASS     0.9621     0.9994     3.5749      0.8836      0.979
1     5.2058
7    p043 -> p084  Impostor     Rejected        FAIL     0.0974    -0.7721   103.9997      0.8968      0.944
0    12.5740
8           p045   Genuine      TRUSTED        PASS     0.9651     0.9975     2.9463      0.8965      0.954
7    10.8209
9    p045 -> p076  Impostor     Rejected        FAIL     0.1454    -0.3529    67.1634      0.8169      0.836
7    24.4490
```

Evaluate the final resuly of the GUI simulation using the Receiver Operating Characteristic (ROC) curve and other metrics.

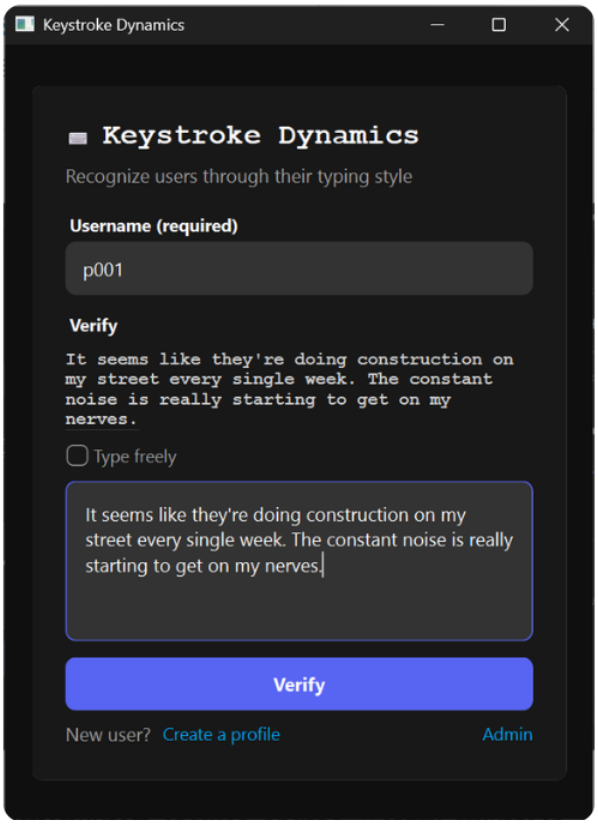


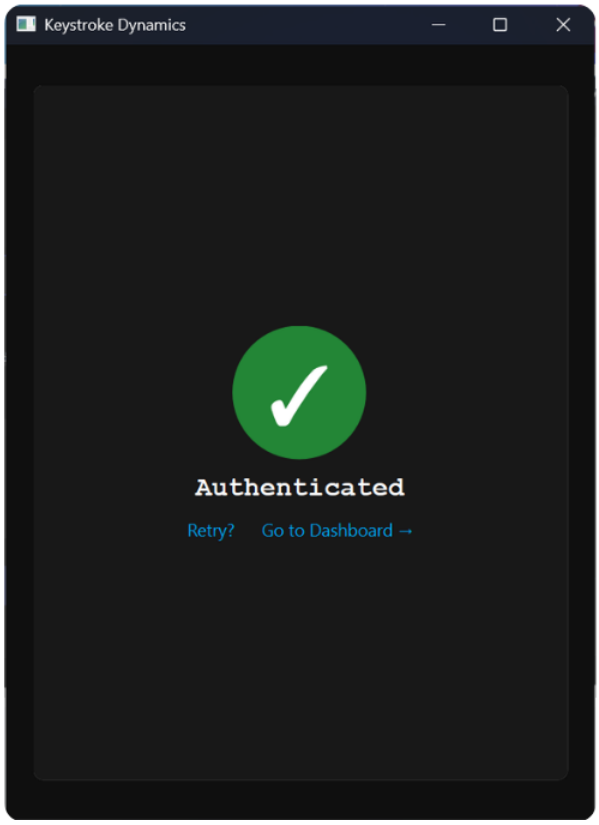

Other metrics for evaluation that signifies the effectives of the ESN-SVM hybrid model.

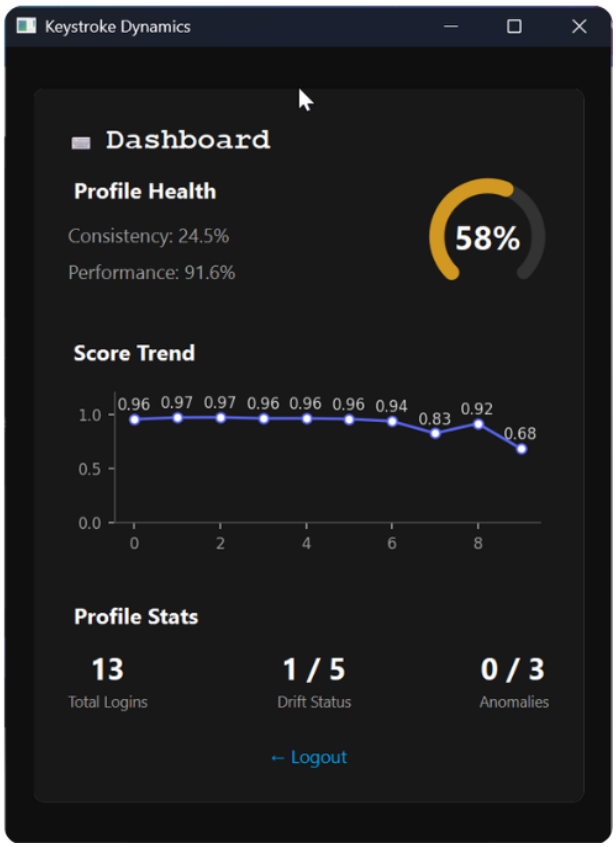
```
--- Biometric Performance Metrics ---
Area Under Curve (AUC): 0.9815
Equal Error Rate (EER): 0.0263
False Match Rate (FMR) at Current Thresholds: 2.63%
False Non-Match Rate (FNMR) at Current Thresholds: 1.32%
```

## 5.7 GUI Application for Free-Text Keystroke-Authentication

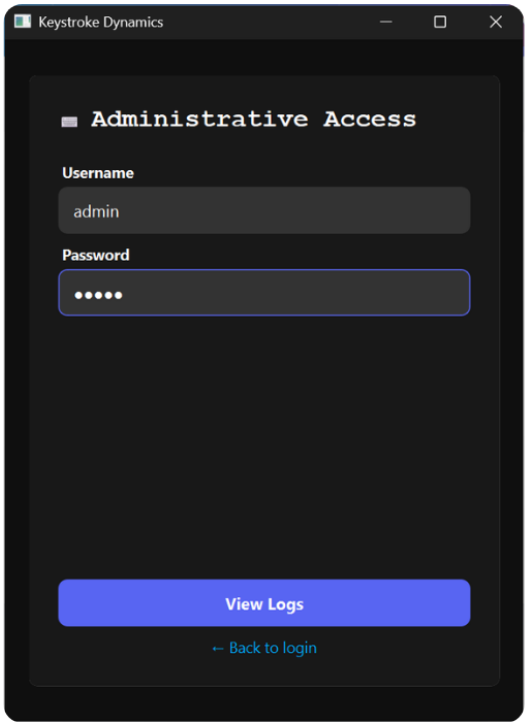
The GUI free-text keystroke dynamics application uses the same implementation as simulation test. As verification, authentication and dashboard panel can be seen in the below image, along with tick mark showing that the user is successfully authenticated, also showing SVM score trend which denotes how well your typing patterns are similar with the stored template for that specific user.


Keystroke Dynamics
Keystroke Dynamics
Recognize users through their typing style
Username (required)
p001
Verify
It seems like they're doing construction on my street every single week. The constant noise is really starting to get on my nerves.
Type freely
It seems like they're doing construction on my street every single week. The constant noise is really starting to get on my nerves.
Verify
New user? Create a profile
Admin


Keystroke Dynamics
Authenticated
Retry?
Go to Dashboard →


Keystroke Dynamics
Dashboard
Profile Health
Consistency: 24.5%
Performance: 91.6%
58%
Score Trend
0.96 0.97 0.97 0.96 0.96 0.96 0.94 0.83 0.92 0.68
1.0
0.5
0.0
0
2
4
6
8
Profile Stats
13
Total Logins
1 / 5
Drift Status
0 / 3
Anomalies
← Logout

The below image shows administrative access and detailed logs for each user and other authentication rules.





# References

Dias, T., Vitorino, J., Maia, E., Sousa, O., Praça, I., 2023. KeyRecs: A keystroke dynamics and typing pattern recognition dataset. Data in Brief 50, 109509. https://doi.org/10.1016/j.dib.2023.109509

Franz, A., Brants, T., 2006. All Our N-gram are Belong to You. https://research.google/blog/all-our-n-gram-are-belong-to-you/

Google, B., 2024. Google Books Ngram Viewer, https://books.google.com/ngrams/info

Josh Kaufman, 2014. google-10000-english/20k.txt at master · first20hours/google-10000-english, GitHub. https://github.com/first20hours/google-10000-english/blob/master/20k.txt

Sun, J., Li, L., Peng, H., 2023. Sequence Prediction and Classification of Echo State Networks. Mathematics 11, 4640. https://doi.org/10.3390/math11224640

Wyciślik, Ł., Wylężek, P., Momot, A., 2024. The Improved Biometric Identification of Keystroke Dynamics Based on Deep Learning Approaches. Sensors (Basel) 24, 3763. https://doi.org/10.3390/s24123763