IN PARTNERSHIP WITH PLYMOUTH UNIVERSITY

Name: Walpola L Divyanjalee
Student Reference Number: 10953772

Module Code: PUSL3123	Module Name: AI and Machine Learning
Coursework Title: Acceleration-Based Continuous User Authentication Using Smartwatch Sensor Data	
Deadline Date: 20/11/2025	Member of staff responsible for coursework:
Programme: BSc (Hons) Data Science and Computer Science	

Please note that University Academic Regulations are available under Rules and Regulations on the University website www.plymouth.ac.uk/studenthandbook.

Group work: please list all names of all participants formally associated with this work and state whether the work was undertaken alone or as part of a team. Please note you may be required to identify individual responsibility for component parts.

- Walpola L Divyanjalee - 10953772
- Rathnasekara M Sumanarathne - 10953768
- Karunarathna Thejanga - 10953390
- Wanasinghe Wanasinghe - 10953715

We confirm that we have read and understood the Plymouth University regulations relating to Assessment Offences and that we are aware of the possible penalties for any breach of these regulations. We confirm that this is the independent work of the group.

Signed on behalf of the group: Tharushi

Individual assignment: ***I confirm that I have read and understood the Plymouth University regulations relating to Assessment Offences and that I am aware of the possible penalties for any breach of these regulations. I confirm that this is my own independent work.***

Signed :

Use of translation software: failure to declare that translation software or a similar writing aid has been used will be treated as an assessment offence.

I *have used/not used translation software.

If used, please state name of software.....

Overall mark _____% Assessors Initials _____ Date_____

Table of Content

Chapter 1 - Introduction

Smartphones and smartwatches have become essential tools for accessing banking services, digital payments, communication platforms, and other sensitive personal information. As these devices increasingly support security-critical tasks, ensuring that only authorized users interact with them is crucial. Conventional authentication methods such as passwords, PINs, fingerprints, and facial recognition verify identity only now of login. Once unlocked, they cannot guarantee that the same person continues using the device. These methods also have practical weaknesses: passwords may be forgotten, guessed, or leaked, while biometric sensors can be affected by lighting conditions, physical injuries, spoofing attacks, or hardware limitations. Such issues highlight the need for a more reliable, seamless, and continuous authentication mechanism.

Recent research has shown that behavior-based continuous authentication, particularly using accelerometer and gyroscope data, offers a promising alternative. Users naturally generate distinctive motion patterns through walking, wrist movement, device handling, or head motion, and these signals can be captured unobtrusively by everyday devices. Studies involving smartphones, smartwatches, ear-worn devices, and smart insoles demonstrate that these behavioral signatures can be modelled effectively, achieving classification accuracies above 90–99% even under cross-day evaluation. Machine learning models, including SVMs, Random Forests, CNN-LSTM hybrids, and spiking neural networks, have further strengthened the practicality of acceleration-based behavioral biometrics (Schöffel et al., 2022; Moreno-Pérez et al., 2023; Kumari et al., 2024).

This report investigates the use of smart device acceleration data for user authentication using supervised machine learning. It begins with a review of recent motion-based authentication methods, focusing on data acquisition, feature extraction, and classification techniques. The methodology section describes the preprocessing pipeline and handcrafted feature engineering applied to the dataset, followed by the development and training of a neural network classifier. The evaluation examines system performance using FAR, FRR, and EER metrics, while optimization experiments explore feature reduction and parameter tuning. The report concludes by discussing the practical applicability, limitations, and future potential of acceleration-based continuous authentication systems.

Chapter 2 - Background

Many recent studies look at how to keep track of users all the time using sensors in wearable devices like earphones, smartwatches, and smartphones. These devices have sensors such as accelerometers and gyroscopes that record movements like head turns, walking patterns, and hand gestures. Since these movements are unique to each person, they can be used to verify a person's identity without interrupting their daily activities.

The key features used for this are basic measurements taken from the movement signals, such as averages, changes, how much energy the movement has, and how often the signal crosses zero. (Liu *et al.,* 2023)Features from the frequency of the movements are also used, like the main frequency components and how complex or spread out the signals are. Different types of machine learning models like Support Vector Machines, Random Forests, and neural networks are used for classifying users. These models often reach accuracy rates above 90%, showing that sensor data can reliably identify people.

One specific focus is on gait, or how a person walks. Researchers have made new ways to analyse walking patterns, even when sensors are placed differently or when people walk differently on different days. (Lee *et al.,* 2022) These methods can achieve more than 99% accuracy in the same day and more than 94% accuracy across different days. Using more than one sensor together makes these results even better.

There are also small, portable devices that combine pressure sensors and motion sensors. These help analyse walking and movements outside labs, with high accuracy, and are useful for healthcare and daily life.(Manupibul *et al.,* 2023) Some systems gather data from multiple devices, like smartphones and tablets, at the same time. Using this combined data and advanced deep learning methods, they can identify users with about 99% accuracy, promoting secure continuous access to mobile devices.

In addition, studies show that using raw sensor data without cleaning sometimes gives better results than cleaned data. (Al-Mahadeen *et al.,* 2023) Deep learning models like Long Short-Term Memory (LSTM) networks get nearly perfect accuracy, which is important for keeping smartphones secure based on how users move naturally.

To handle the recognition of unknown users, some systems use special advanced models that can tell both known and unknown users apart, with about 95% accuracy. To protect user privacy, some systems also encrypt movement features so they can't be stolen or misused.(Moon *et al.,* 2022)

Smartwatches are also used for continuous user checking. They use many features from sensors and select the best ones for each person. These systems perform very well in real-life situations, with very low error rates.

For applications like mobile banking, combining movement and touch data into deep learning models can reach over 99.85% accuracy. This shows that these methods can be used safely in everyday life.(Uslu, Incel and Alptekin, 2023)

Energy-efficient models like Spiking Neural Networks (SNNs) use movement data to quickly and accurately verify users without extra effort from them. These new models offer fast and reliable security for mobile devices.(Li *et al.,* 2024)

Sensors in wearable devices combined with advanced computer programs can reliably identify and verify users, making everyday devices safer, more personal, and easier to use.

Even though there has been a lot of progress in using wearable sensors for always checking who the user is, some problems still exist. Current systems do not work smoothly across many devices, and they find it hard to adjust when people's behaviour changes over time. Protecting users' private data in small devices like watches is still difficult. Also, these systems need to work better in everyday situations where sensors can move or conditions change. Future work should focus on making these systems easier to use, more secure, able to learn from changing behaviours, and work well on different devices.

Paper Title	**Device**	**Sensor Data (SD) IMU (** Acceleration, Gyroscope)	**Classification**	**Featured Used** TD-Time Domain FD-Frequency Domain	**Preprocessing**	**Result %**
Secure User Verification and Continuous Authentication via Earphone IMU (2023)	Earphone-Embedded (IMU) sensor	IMU	SVM, Random Forest, Neural Networks	TD FD	Noise filtering Windowing Normalization Outlier removal	CCR y > 90
Integration of force and IMU sensors for developing low-cost portable gait measurement system in lower extremities (2023)	Wearable lower-limb system	IMU	Measurement system, not classifier based	TD FD	Kalman filtering Low-pass filtering	IMU CCR > 92, Force sensor CCR > 97

Gait-Based Continuous Authentication Using a Novel Sensor Compensation Algorithm and Geometric Features Extracted from Wearable Sensors (2022)	Wearable IMU sensors on wrist and thigh	IMU	Linear SVM	TD	Noise filtering Gait cycle segmentation Sensor compensation algorithm	Same Session (Day 1): 99.63 Cross Session 94.16-94.20
Open Set User Identification Using Gait Pattern Analysis Based on Ensemble Deep Neural Network (2022)	Sensor-equipped insoles	IMU	Ensemble DL + One-Class SVM	TD	Gaussian smoothing Cycle segmentation Interpolation	95.3 CCR
Continuous User Authentication on Multiple Smart Devices (2023)	Smartphone + Tablet	IMU	CNN + LSTM	TD	Butterworthfiltering Frequency sync Interpolation	CCR Smartphone 97.9 Tablet 96.3
Smartphone User Identification/Authentication Using Accelerometer and Gyroscope Data (2023)	Smartphone	IMU	LSTM, XGBoost	TD FD	Sliding windows Magnitude calculation FFT	99.7 CCR

User Recognition in Participatory Sensing Systems Using Deep Learning Based on Spectro-Temporal Representation of Accelerometer Signals (2024)	Smartphone	Accelerometer	LSTM (best), RF, SVM	TD FD	Sliding windows Magnitude FFT	CCR 89 (ID) 99.9 (Auth)
Gait-Based Privacy Protection for Smart Wearable Devices (2024)	Smart Wearable Devices	IMU	ABLSTM + SOT + Biometric Encryption	TD FD	Wavelet denoising Gait cycle detection with DCA	CCR 95.28
Real-world continuous smartwatch-based user authentication (2025)	Smartwatches	IMU	MLP Neural Network	TD FD	Filtering Interpolation Segmentation	EER 0.13 (controlled) 0.7 (real-world)
SNNAuth: Sensor-Based Continuous Authentication on Smartphones Using Spiking Neural Networks (2024)	Smartphone	IMU	ANN - SNN conversion + One-class KNN	TD	Normalization Slicing Positional encoding	CCR 97.89

Evaluation of Deep Learning Models for Continuous Authentication Using Behavioral Biometrics (2023)	Smartphone	IMU	MLP, LSTM, Bi-LSTM, Conv-LSTM	TD	Normalization SMOTE balancing	CCR 99.85

Table 1: - Summary of Recent Studies on Acceleration-Based Continuous User Authentication Using Wearable Sensors

Chapter 3 - Testing Methodology

This section outlines the experimental procedures, evaluation metrics, and validation strategy used to assess the acceleration-based authentication system. All steps related to data partitioning, threshold selection, and performance measurement are detailed to ensure methodological clarity and reproducibility.

3.1 Dataset Overview

The smartwatch acceleration dataset consists of recordings from 10 users collected over two sessions. First Day (FD) and Multiple Day (MD) at a sampling rate of 32 Hz. Each session includes roughly 12 minutes of walking data per user, containing both accelerometer and gyroscope measurements across six channels. After segmentation, 136 overlapping segments were extracted per session using 5-second windows with 50% overlap, resulting in a total of 2720 segments. Figure 1 shows examples of the raw acceleration signals, the segmentation framework, and a sample interpolated segment, illustrating the clean and periodic nature of the walking patterns.

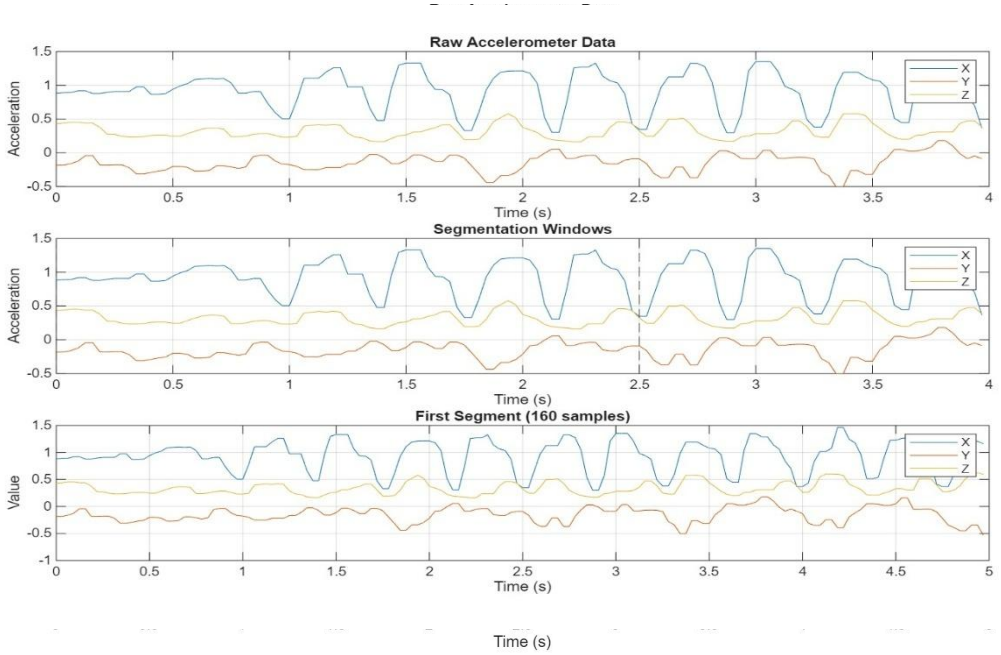
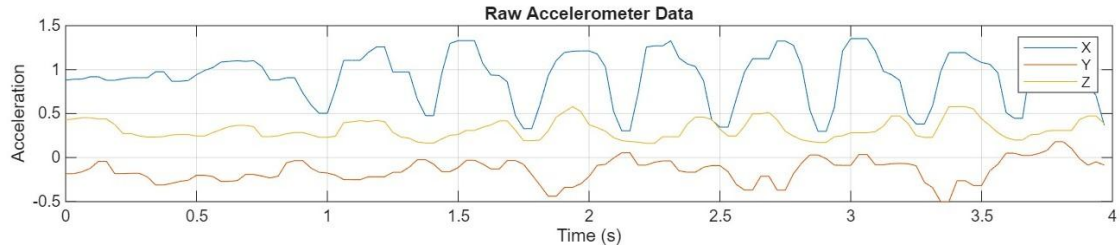


Figure 1: Preprocessing pipeline showing: (a) raw accelerometer data, (b) segmented 5s windows with 50% overlap, and (c) a normalized and interpolated segment (160 samples).

3.2 Preprocessing Pipeline

Preprocessing was designed to minimise sensor noise and standardise segment lengths. A median filter with a kernel size of 3 was applied to each channel to suppress spikes and motion artefacts. The data was then segmented into 5-second windows (160 samples) with 50% overlap, providing sufficient temporal representation of each gait cycle. Each segment was subsequently interpolated to maintain a fixed length and consistent sampling rate. Normalisation was performed to ensure all features remained within a comparable scale, reducing bias during model training. The before-and-after plots in Figure 2 illustrate the reduction in noise and the improved uniformity achieved across users.

Before Preprocessing



After Preprocessing

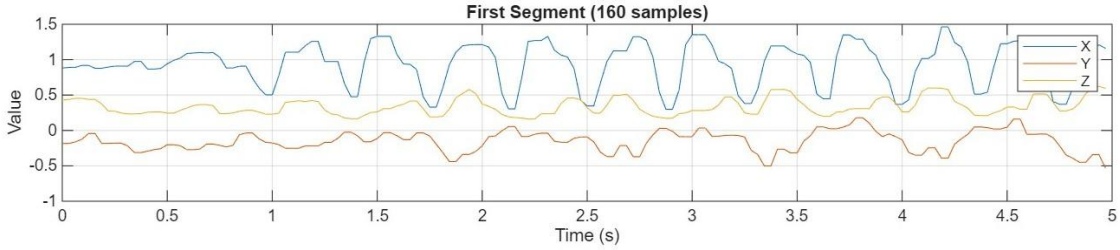


Figure 2. Comparison of raw vs. pre-processed acceleration signals, highlighting noise reduction and improved signal stability achieved through median filtering and interpolation.

3.3 Feature Extraction

A total of 105 handcrafted time-domain features were extracted from each segment using both accelerometer and gyroscope signals. These included:

- Statistical features: mean, variance, standard deviation, range, percentiles, skewness, and kurtosis.
- Signal features: RMS, energy, zero-crossing rate (ZCR), and mean absolute deviation (MAD).
- Global gait features: Signal Magnitude Area (SMA), Signal Magnitude Vector (SMV), and inter-axis correlations (XY, XZ, YZ).
- Autocorrelation features: maximum and mean autocorrelation across lags 0–20.

Together, these features capture both steady-state characteristics and dynamic motion variations unique to each user.

Feature category	Domain	Description	No.of Features	Importance
Basic Statistics	Time	Mean, Std, Var, Min, Max, Range	36	Capture amplitude and variability
Percentiles & IQR	Time	P25, P50, P75, IQR	24	Represent signal spread and stability
RMS, Energy,	Frequency	RMS and signal energy in the	18	Reflect periodic motion and gait strength

ZCR		frequency spectrum		
Gait and Correlation	Time	SMA, SMV, XY–YZ correlations	9	Represent inter-axis coordination
Autocorrelation	Time	Mean, Max lag correlation	6	Capture gait cycle periodicity
Spectral Power and Entropy	Frequency	Band energy and spectral entropy	12	Measure frequency complexity

Table 2: summarizes the feature categories and their analytical relevance.

3.4 Template Generation and Data Split

Each user's dataset was split using a cross-day approach to prevent data leakage. The FD session served as the training set for both genuine and impostor classification, while the MD session was used exclusively for testing. For each user, the model was trained and evaluated with 136 genuine samples and 1224 impostor samples following a 1-vs-rest setup. This strategy provides a realistic impostor distribution and supports reliable cross-day generalisation.

3.5 Classifier Design

A Feedforward Multilayer Perceptron (FFMLP) was used because of its ability to model nonlinear temporal patterns present in the handcrafted features. The network included a single hidden layer with 30 neurons and was trained using the Scaled Conjugate Gradient (SCG) algorithm. Internally, the data were partitioned into 70% training, 15% validation, and 15% testing subsets. The FFMLP was chosen over alternatives such as SVM or Random Forest due to its stronger capability for multi-feature integration and its improved generalisation under small inter-session variations.

3.6 Performance Metrics

Metric	**Definition**	**Description**	**Purpose**
FAR (False Acceptance Rate)	Ratio of impostor attempts incorrectly accepted as genuine users.	Proportion of impostor samples incorrectly accepted as genuine users. Reflects system security.	Measures system security robustness (lower is better)
FRR (False Rejection Rate)	Ratio of genuine user attempts incorrectly rejected.	Proportion of genuine samples incorrectly rejected. Reflects system usability.	Evaluates usability and convenience (lower is better)
EER (Equal Error Rate)	The point where FAR equals FRR.	Indicates the balance point between usability and security. Commonly used benchmark for	Represents overall model reliability

		continuous verification systems.	

Table 3: Summary of Performance Metrics

In summary, the use of FAR, FRR, and EER provides a balanced and interpretable assessment of system reliability, reflecting both security and user convenience. ROC and AUC analyses further quantify classifier discrimination under varying thresholds, ensuring the authentication model performs consistently across genuine and impostor scenarios.

Chapter 4 - Evaluation

In this study, the performance of an acceleration-based user authentication system was examined using data collected from ten participants. Models were trained on Day-1 (FD) walking recordings and evaluated on Day-2 (MD) to reflect natural variations in daily behaviour. Each walking trial was segmented into 5-second windows, resampled to 150 points, and represented using 105 time-domain features derived from accelerometer and gyroscope signals. A Feedforward Multi-Layer Perceptron (FFMLP) was developed for each user, treating the target subject as genuine and the remaining users as impostors. System performance was measured using Correct Classification Rate (CCR), False Acceptance Rate (FAR), False Rejection Rate (FRR), and Equal Error Rate (EER). Each user reviewed these results, and overall, supported by FAR–FRR curves, ROC analysis, and comparisons across sensor combinations.

User	**U1**	**U2**	**U3**	**U4**	**U5**	**U6**	**U7**	**U8**	**U9**	**U10**	**Overall**
CCR %	99.78	98.53	100.0	97.13	100.0	97.65	98.38	100.0	99.26	99.63	**99.04**
FAR %	0.25	1.47	0.00	2.86	0.00	2.37	1.63	0.00	0.74	0.33	-
FRR %	0.00	1.47	0.00	2.94	0.00	2.21	1.47	0.00	0.74	0.74	-
EER %	0.12	1.47	0.00	2.90	0.00	2.29	1.55	0.00	0.74	0.53	**0.96**

Table 4: Baseline System Performance of FFMLP

The table summarizes the baseline performance of the FFMLP classifier using time-domain features from accelerometer and gyroscope signals. The system shows consistently strong performance across all ten users, with an overall CCR of 99.04%, indicating clear separation between genuine and impostor samples under cross-day testing.

User-level variation is evident. Users 3, 5, and 8 achieved perfect results (CCR = 100%, EER = 0%), reflecting highly stable gait patterns. In contrast, User 4 recorded the highest error rates (FAR = 2.86%, FRR = 2.94%, EER = 2.90%), likely due to day-to-day differences in pace, footwear, sensor placement, or environmental factors.

The average EER of 0.96% demonstrates strong discriminative capability. Low FAR values indicate robust protection against impostor attempts, while low FRR values confirm good usability for genuine users.

Overall, the results show that the FFMLP effectively models user-specific gait behaviour under realistic cross-day conditions, though users with greater behavioural variability may require further optimization.

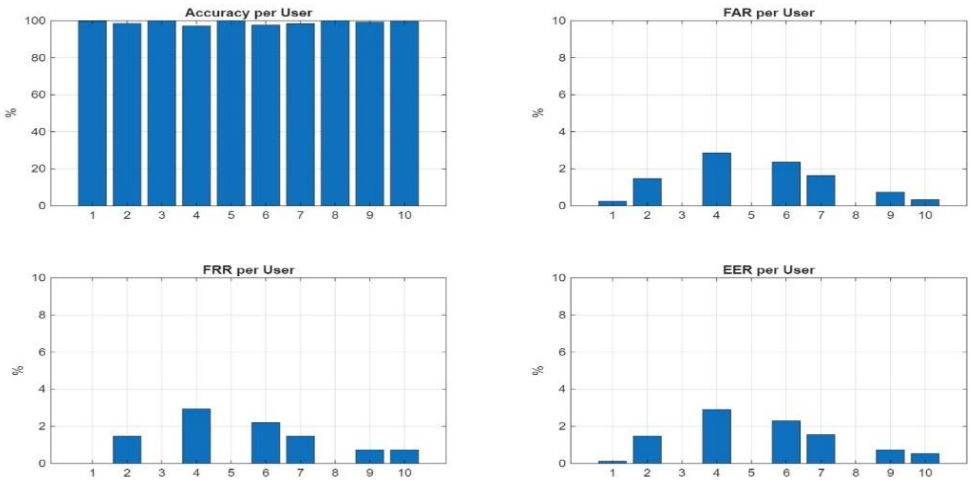


Figure 1: Accuracy, FAR, FRR, and EER per User

Figure 1 illustrates the distribution of CCR, FAR, FRR, and EER across all users, providing a clear visual comparison of performance variations. While the detailed values are listed in Table 1, the bar charts highlight differences in gait consistency among users and show the relative difficulty of each profile. Overall, the plots confirm that error rates remain low, indicating reliable separation between genuine and impostor samples.

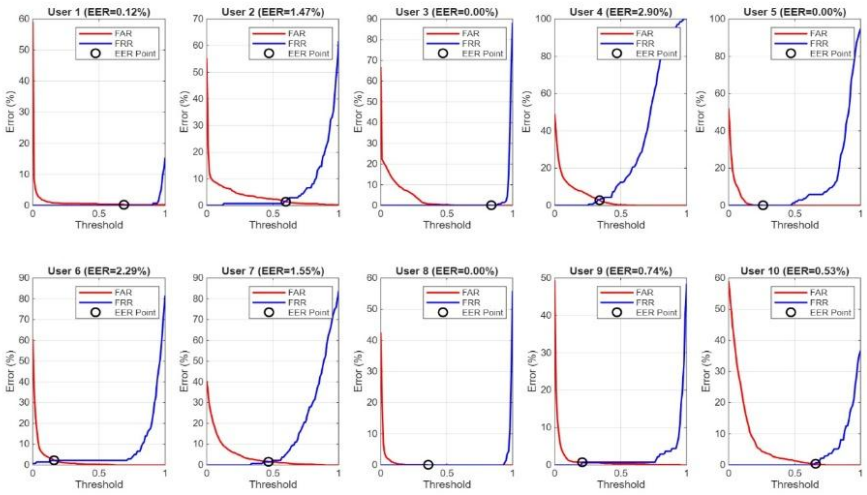


Figure 2: User-specific FAR–FRR Curves with EER Points

Figure 2 shows the FAR–FRR curves for all users. The intersection point representing the EER consistently appears at low error levels, indicating stable threshold behaviour across subjects. Users with curves that are closer or less separated (e.g., U4 and U6) exhibit greater overlap between genuine and impostor scores, corresponding to their relatively higher EER values.

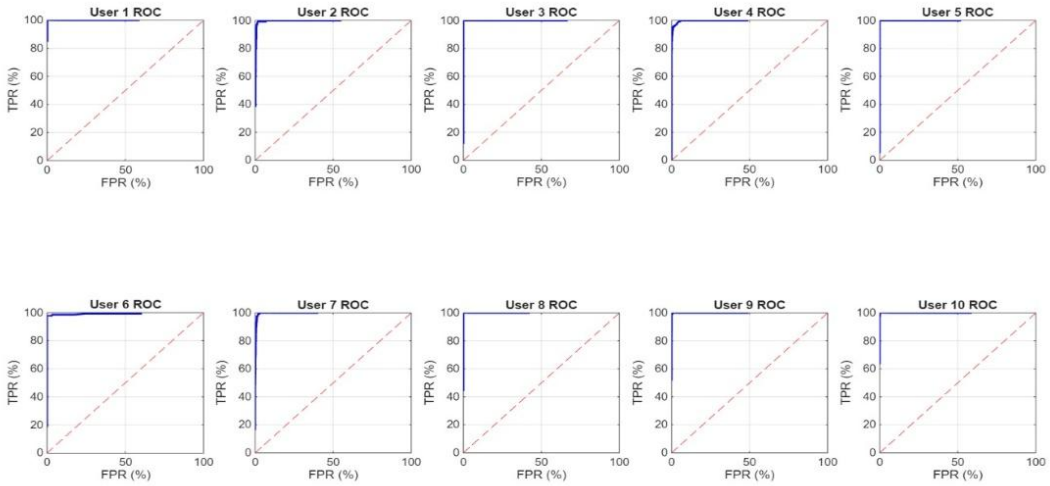


Figure 3: User-specific ROC Curves

Figure 3 displays the ROC curves, where most users show steep trajectories approaching the top-left corner, indicating strong true-positive performance at low false-positive rates. Flatter ROC curves correspond to users with higher EER values, visually confirming their lower discriminability.

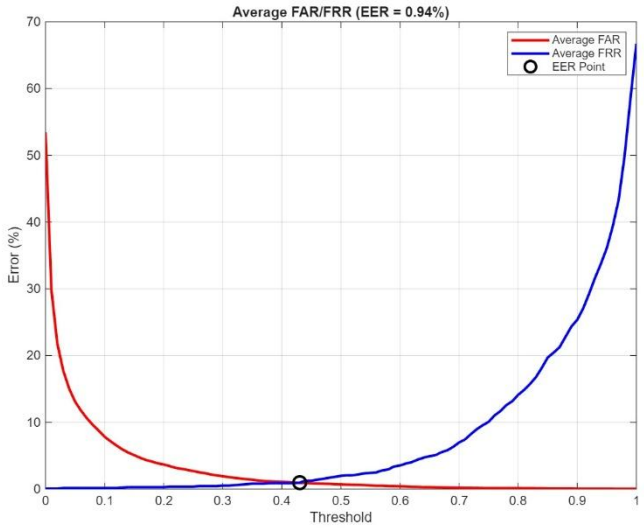


Figure 4: Average FAR–FRR Curve across All Users

Figure 4 presents the averaged FAR–FRR curve for the system. The global EER (~0.95%) closely matches the mean EER reported in the baseline results, confirming stable performance at the system level.

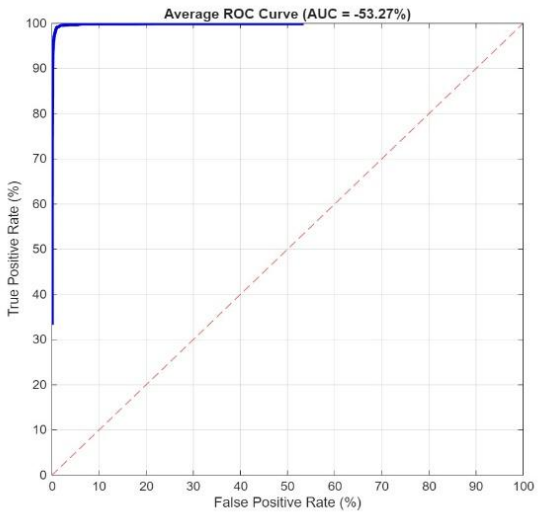


Figure 5: Average ROC Curve for All Users

Figure 5 shows the average ROC curve, which remains close to the ideal vertical-left profile, indicating strong overall separability. Although AUC is not the primary metric in biometrics, the near-ideal ROC shape further supports the effectiveness of the classification model.

Classifier	Accuracy	EER
FFMLP	99.18%	0.88%
SVM	98.18%	1.83%

Table 5: FFMLP vs SVM Performance

To benchmark the FFMLP against a classical approach, a linear SVM was tested using the same feature set and cross-day protocol. As shown in Table X, the FFMLP achieved higher accuracy (99.18%) and a lower EER (0.88%) than the SVM (98.18%, EER = 1.83%). These results suggest that the neural network captures non-linear gait characteristics more effectively, whereas the linear SVM has difficulty separating overlapping genuine and impostor samples. The performance gap is most evident for users with greater behavioural variability, where the FFMLP provides clearer discrimination.

The evaluation results highlight both the strengths and limitations of acceleration-based authentication under realistic cross-day conditions. The system demonstrated strong baseline performance (CCR = 99.04%, EER = 0.96%), showing that the FFMLP can reliably separate genuine and impostor gait patterns for most users. However, the per-user variation observed in Table 1 and Figures 1–5 points to several behavioural and model-related factors that influence performance.

User-dependent variability remains a major contributor to error. Users such as U3, U5, and U8 displayed highly stable gait behaviour, resulting in EER values of 0%. In contrast, U4 and U6 recorded higher EERs, likely due to day-to-day variations in walking pace, posture, footwear, fatigue, or inconsistent wrist-sensor placement. These behavioural fluctuations introduce noise and reduce the separation between genuine and impostor scores, as reflected in their flatter ROC curves.

Sensor characteristics also affect accuracy. Gyroscope-only authentication performed notably worse because rotational patterns vary more between sessions and are sensitive to small angular changes. Accelerometer signals, which capture impact forces and vertical displacement, offer more stable features and therefore better discriminative power. The combined Acc+Gyro configuration outperformed both single-sensor models, indicating that gyroscope data becomes useful when complemented by accelerometer information.

Model capacity provides further insight. Increasing hidden neurons improved performance up to a point, after which gains diminished due to early overfitting, where the network begins to learn session-specific noise. This suggests the need for balanced architecture rather than simply increasing model size.

Metric choice also influences interpretation. Although CCR values appear high, CCR alone can obscure the security–usability balance. EER offers a more reliable measure by identifying the point where FAR and FRR are equal and therefore better represents true authentication performance.

Finally, the cross-day evaluation naturally introduces performance reductions because user behaviour changes across sessions. While accuracy is slightly lower than in same-day testing, this setup better reflects real deployment scenarios.

Overall, the findings indicate that the system is robust, while also emphasizing the role of behavioural variation, sensor characteristics, and model design in shaping per-user performance.

Overall, the evaluation confirms that acceleration-based user authentication can achieve highly reliable performance under realistic cross-day conditions. The FFMLP effectively distinguished genuine and impostor gait patterns, achieving an average CCR of 99.04% and an EER of 0.96%. Visual analyses indicated strong score separation for most users, with slight variability observed in a few cases due to behavioural differences between sessions. Sensor comparisons showed that accelerometer features were more discriminative than gyroscope signals, while combining both improved overall robustness. Despite natural fluctuations in gait, the system maintained excellent security and usability, demonstrating strong potential for further optimization and real-world deployment.

Chapter 5 - Optimization

After getting the baseline system working, optimization experiments were run to improve performance. The baseline used combined sensors, 30 hidden neurons, a 70/30 split, and all 105 features, giving 98.04% accuracy with 1.94% EER. The goal was to improve these results while making the system more efficient for smartwatches.

5.1 Optimization Experiments

Six experiments tested different system components. The first compared sensor configurations - combined, accelerometer-only, and gyroscope-only - to see if both sensors are really needed. The second tested neural network sizes with 10, 20, 30, and 50 hidden neurons. Different train-test splits (60/40, 70/30, 80/20) were also tried to find the right balance.

For feature reduction, two methods were used. The simple method tested 105, 70, 50, and 30 features. The second used PCA with 90, 70, 50, and 30 components. Finally, the neural network was compared against a linear classifier to make sure it was necessary.

5.2 Results

The sensor comparison results are in Table 6. Combined sensors got 98.04% accuracy, accelerometer alone got 97.73%, and gyroscope alone only 90.39% (Figure 6). This shows that both sensors are needed because walking has straight movement that the accelerometer picks up and rotation that the gyroscope measures.

Sensor Mode	Accuracy	FAR	FRR	EER
Combined	98.04%	1.97%	1.91%	1.94%
Accelerometer-only	97.73%	2.28%	2.21%	2.24%
Gyroscope-only	90.39%	9.65%	9.26%	9.46%

Table 6: Sensor Configuration Results

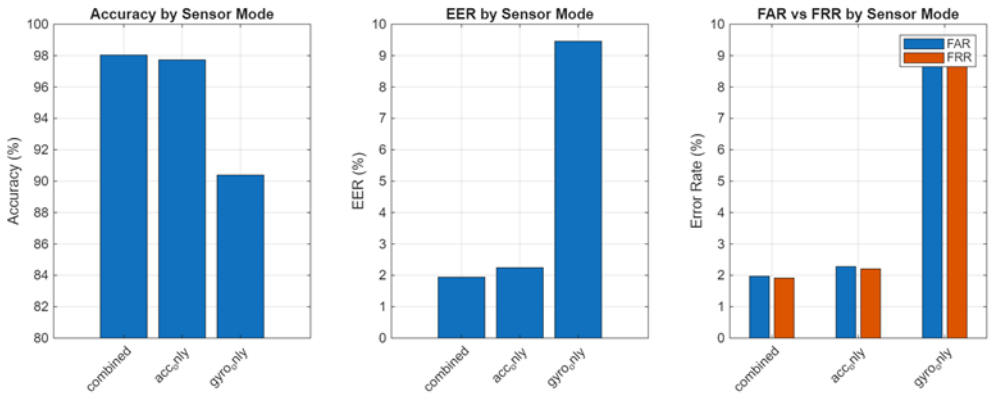


Figure 6: Performance comparison across different sensor configurations

For the neural network experiments, something surprising happened. Table 7 shows the results for different network sizes. The smallest network with 10 neurons worked best, getting 99.18% accuracy and 0.88% EER. This beat all the bigger networks (Figure 7). The reason for this is probably that bigger networks overfit - they memorize the training data too much instead of learning the general patterns that work on new data.

Hidden Neurons	Accuracy	EER
10	99.18%	0.88%
20	98.58%	1.38%
30	98.81%	1.12%
50	98.68%	1.32%

Table 7: Neural Network Size Results

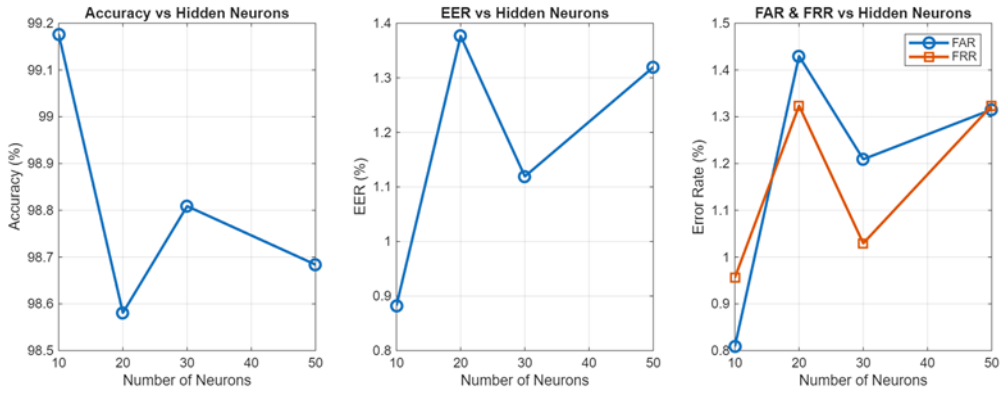


Figure 7: Effect of hidden neuron count on authentication performance

For the train-test split experiments, different ratios were tested to see how much training data is needed. Table 8 shows the results. The 80/20 split gave the best results at 99.15% accuracy with 0.87% EER. Having more training data helps the network learn patterns that work across different days.

Split Ratio	Accuracy	EER
60/40	98.74%	1.32%
70/30	98.37%	1.63%
80/20	99.15%	0.87%

Table 8: Train-Test Split Comparison

The feature reduction results showed that 70 features is a good balance. Table 4 has these results. Using 70 features kept performance at 98.48% accuracy while cutting processing requirements by 33% (Figure 8). This is important for smartwatches that don't have much processing power. Going below 70 features made the performance drop quite a bit.

Number of Features	Accuracy	EER	Cost Reduction
105 (baseline)	98.74%	1.23%	0%
70	98.48%	1.50%	33%
50	96.17%	3.83%	52%
30	96.68%	3.35%	71%

Table 9: Feature Reduction Results

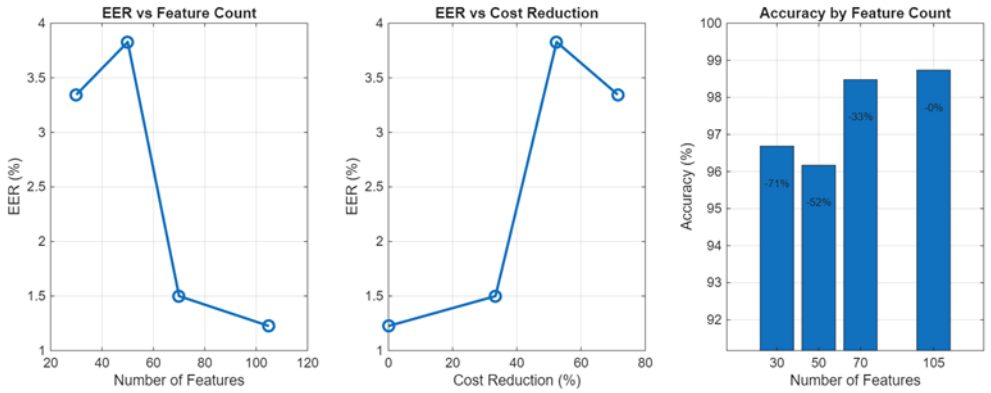


Figure 8: Trade-off between feature count and system performance

The PCA results were interesting. Table 9 shows that 70 components worked best at 98.30% (Figure 9). But using 90 components made it worse at 96.74%. This shows that keeping too many components adds noise that hurts the results instead of helping.

PCA Components	Accuracy	EER
90	96.74%	3.21%
70	98.30%	1.73%
50	97.51%	2.52%
30	97.38%	2.60%

Table 10: PCA Results

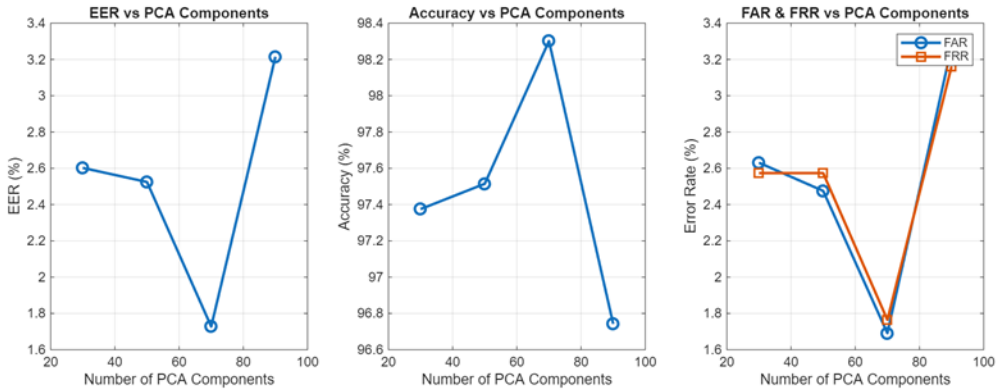


Figure 9: PCA dimensionality reduction performance analysis

For the classifier comparison, Table 11 shows the results. The neural network got 99.18% while the linear classifier only got 98.18%. This shows the neural network is worth using even though it's more complex.

Classifier	Accuracy	EER
Neural Network	99.18%	0.88%
Linear Classifier	98.18%	1.83%

Table 11: Classifier Comparison

5.3 Final System

From all these experiments, the best configuration was found to be combined sensors, 10 neurons, 80/20 split, and 70 features. This achieved 98.54% accuracy with 1.43% EER when tested on all users. Table 12 compares the baseline and optimized systems.

Configuration	Baseline	Optimized	Change
Sensors	Combined	Combined	Same
Hidden Neurons	10	30	Smaller
Train/Test Split	70/30	80/20	More training
Features	105	70	33% less
Accuracy	98.04%	98.54%	+0.50%
EER	1.94%	1.43%	-26%
FAR	1.97%	1.46%	-26%
FRR	1.91%	1.40%	-27%

Table 12: Baseline vs Optimized System

The optimized system improved accuracy by 0.50% and reduced EER by 26%. It also cut computational cost by 33% which makes it more practical for smartwatches. Three users got a perfect 100% authentication which shows the system works well. However, User 9 had lower performance at 93.09%, which shows that some people might have less distinctive walking patterns that are harder for the system to identify.

Chapter 6 - Conclusion

This project developed an acceleration-based continuous authentication system using IMU data from ten users recorded on different days. The processing pipeline involved noise filtering, segmentation, interpolation, normalization, and extraction of 105 time- and frequency-domain features. A Feedforward Multi-Layer Perceptron (FFMLP) model was trained using a cross-day setup, and performance was evaluated through accuracy, FAR, FRR, EER, and ROC-AUC metrics.

The optimized FFMLP achieved high accuracy with low error rates, proving that feature-based neural networks can effectively distinguish individual gait patterns. The system demonstrated strong cross-day consistency, supporting its potential for lightweight, continuous user verification on wearable and mobile devices.

Despite strong performance, limitations include the small dataset size and potential variability caused by user behaviour and sensor differences. Future work should focus on expanding datasets, applying multi-sensor fusion, and exploring deep learning models such as CNNs and LSTMs to enhance robustness and scalability.

References

Al-Mahadeen, E. *et al.* (2023) 'Smartphone User Identification/Authentication Using Accelerometer and Gyroscope Data', *Sustainability (Switzerland)*, 15(13). Available at: https://doi.org/10.3390/su151310456.

Kumari, N. *et al.* (2024) 'Human motion activity recognition and pattern analysis using compressed deep neural networks', *Computer Methods in Biomechanics and Biomedical Engineering: Imaging and Visualization*, 12(1). Available at: https://doi.org/10.1080/21681163.2024.2331052.

Lee, Soobin *et al.* (2022) 'Gait-Based Continuous Authentication Using a Novel Sensor Compensation Algorithm and Geometric Features Extracted from Wearable Sensors', *IEEE Access*, 10, pp. 120122–120135. Available at: https://doi.org/10.1109/ACCESS.2022.3221813.

Li, Y. *et al.* (2024) 'SNNAuth: Sensor-Based Continuous Authentication on Smartphones Using Spiking Neural Networks', *IEEE Internet of Things Journal*, 11(9), pp. 15957–15968. Available at: https://doi.org/10.1109/JIOT.2024.3349533.

Liu, J. *et al.* (2023) 'Secure User Verification and Continuous Authentication via Earphone IMU', *IEEE Transactions on Mobile Computing*, 22(11), pp. 6755–6769. Available at: https://doi.org/10.1109/TMC.2022.3193847.

Manupibul, U. *et al.* (2023) 'Integration of force and IMU sensors for developing low-cost portable gait measurement system in lower extremities', *Scientific Reports*, 13(1). Available at: https://doi.org/10.1038/s41598-023-37761-2.

Moon, J. *et al.* (2022) 'Open Set User Identification Using Gait Pattern Analysis Based on Ensemble Deep Neural Network', *IEEE Sensors Journal*, 22(17), pp. 16975–16984. Available at: https://doi.org/10.1109/JSEN.2022.3188527.

Moreno-Pérez, J.A. *et al.* (2023) 'System Based on an Inertial Measurement Unit for Accurate Flight Time Determination in Vertical Jumps', *Sensors*, 23(13). Available at: https://doi.org/10.3390/s23136022.

Schöffel, M. *et al.* (2022) 'Secure IoT in the Era of Quantum Computers—Where Are the Bottlenecks?', *Sensors*, 22(7). Available at: https://doi.org/10.3390/s22072484.

Uslu, U., Incel, Ö.D. and Alptekin, G.I. (2023) 'Evaluation of Deep Learning Models for Continuous Authentication Using Behavioral Biometrics', in *Procedia Computer Science*. Elsevier B.V., pp. 1272–1281. Available at: https://doi.org/10.1016/j.procs.2023.10.115.

Wang, Y., Zhang, X., & Hu, H. (2023). Continuous User Authentication on Multiple Smart Devices. *Information,* 14(5), 274. https://doi.org/10.3390/info14050274

Su, Y., Li, Y. and Cao, Z., 2024. Gait-Based Privacy Protection for Smart Wearable Devices. IEEE Internet of Things Journal, 11(2), pp.3497-3509. https://doi.org/10.1109/JIOT.2023.3296650

Al-Naffakh, N., Clarke, N., Li, F., Haskell-Dowland, P. et al., 2025. Real-world continuous smartwatch-based user authentication. The Computer Journal, 68(7), pp.717-733. https://doi.org/10.1093/comjnl/bxae144

Moon, J., Jung, J., Kang, E. and Choi, S.-I., 2022. Open Set User Identification Using Gait Pattern Analysis Based on Ensemble Deep Neural Network. IEEE Sensors Journal, 22(17), pp.16975-16988. https://doi.org/10.1109/JSEN.2022.3188527

Appendix

A.1 Script 1: Preprocessing (script_1_Preprocessing.m)

```
% script_1_Preprocessing.m
% Segments raw CSV files and interpolates to 150 samples

clc; clear; close all;

DATA_DIR = 'Dataset';
OUT_DIR = 'Preprocessed';
if ~exist(OUT_DIR,'dir'), mkdir(OUT_DIR); end

fs_orig = 32;
target_fs = 32;
win_sec = 5;
overlap = 0.5;
PLOT_SAMPLE = true;

files = dir(fullfile(DATA_DIR,'U*_*.csv'));
if isempty(files)
    error('No CSV files found in %s', DATA_DIR);
end

fprintf('Found %d files\n', length(files));

for f = 1:length(files)

    fname = files(f).name;
    fprintf('\n[%d/%d] %s\n', f, length(files), fname);

    % load raw data
    raw = readmatrix(fullfile(DATA_DIR, fname));

    if isempty(raw)
        warning('Empty file: %s', fname);
        continue;
    end

    % keep first 7 columns only
    if size(raw,2) < 7
```

```
    warning('File has fewer than 7 columns, skipping');
    continue;
end
raw = raw(:,1:7);

% median filter on sensor data (columns 2-7)
raw(:,2:7) = medfilt1(raw(:,2:7), 3);

% time vector
N = size(raw,1);
t_full = (0:N-1)' / fs_orig;

% segmentation
win_samples = round(win_sec * target_fs);
step_samples = max(1, round(win_samples * (1-overlap)));

starts = 1:step_samples:(N - win_samples + 1);
segments = {};

for i = 1:length(starts)
    s = starts(i);
    e = s + win_samples - 1;
    if e > N, break; end

    % get window (only sensor columns 2-7)
    window = raw(s:e, 2:7);

    % interpolate to exact length
    x_orig = (0:size(window,1)-1)';
    x_target = linspace(0, size(window,1)-1, win_samples)';

    seg_interp = zeros(win_samples, 6);

    for c = 1:6
        y = window(:,c);
        [xu, ia] = unique(x_orig);
        yu = y(ia);

        if length(xu) < 2
```

```
            seg_interp(:,c) = yu(1);
        else
            seg_interp(:,c) = interp1(xu, yu, x_target, 'linear', 'extrap');
        end
    end

    segments{end+1} = seg_interp;
end

% save
outname = fullfile(OUT_DIR, [fname(1:end-4) '_segments.mat']);
save(outname, 'segments', 'win_sec', 'target_fs', 'overlap');

fprintf('  Saved %d segments\n', length(segments));

% visualization for first file only
if PLOT_SAMPLE && f == 1
    figure('Position',[100 100 1000 600]);

    % raw snippet
    T = min(N, 4*fs_orig);
    subplot(3,1,1);
    plot(t_full(1:T), raw(1:T,2:4));
    title('Raw Accelerometer Data');
    xlabel('Time (s)'); ylabel('Acceleration');
    legend('X','Y','Z'); grid on;

    % with segment markers
    subplot(3,1,2);
    plot(t_full(1:T), raw(1:T,2:4)); hold on;
    for b = starts(starts <= T)
        xline((b-1)/fs_orig, 'k--', 'LineWidth', 0.5);
    end
    title('Segmentation Windows');
    xlabel('Time (s)'); ylabel('Acceleration');
    legend('X','Y','Z'); grid on;

    % first interpolated segment
    subplot(3,1,3);
```

```
        seg1 = segments{1};
        tx = (0:size(seg1,1)-1)/target_fs;
        plot(tx, seg1(:,1:3));
        title(sprintf('First Segment (%d samples)', size(seg1,1)));
        xlabel('Time (s)'); ylabel('Value');
        legend('X','Y','Z'); grid on;

        saveas(gcf, fullfile(OUT_DIR,'preprocessing_sample.png'));
    end
end

fprintf('\nPreprocessing done. Saved in %s\n', OUT_DIR);
```

A.2 Script 2: Feature Extraction (script_2_Feature_Extraction.m)

```
%  script_2_Feature_Extraction.m
%  Loads segments from /Preprocessed
%  Extracts time + frequency features using 6 IMU channels
%  Saves features into /Features
%  Includes REQUIRED VISUALIZATIONS:
%     (1) Example segment (X,Y,Z)
%     (2) Boxplots of features
%     (3) Mean feature values
%  ================================================================
clc; clear; close all;

%% ---------------- PATHS ----------------
PRE_DIR = fullfile(pwd, 'Preprocessed');
OUT_DIR = fullfile(pwd, 'Features');

if ~exist(OUT_DIR,'dir')
    mkdir(OUT_DIR);
end

%% ---------------- SETTINGS ----------------
mode = 'combined';      % 'time' | 'freq' | 'combined'
fs   = 32;               % sampling rate after interpolation
visualize_user = true;  % show graphs for FIRST file only
```

```
fprintf('\n=== FEATURE EXTRACTION STARTED ===\n');

%% ---------------- LOAD PREPROCESSED FILES ----------------
files = dir(fullfile(PRE_DIR, '*_segments.mat'));
if isempty(files)
    error('No preprocessed files found in %s', PRE_DIR);
end

fprintf('Found %d preprocessed files.\n\n', length(files));

%% ---------------- PROCESS EACH FILE ----------------
for f = 1:length(files)

    fname = files(f).name;
    fprintf('[%d/%d] Processing %s\n', f, length(files), fname);

    load(fullfile(PRE_DIR, fname), 'segments');

    if isempty(segments)
        warning('  No segments in %s\n', fname);
        continue;
    end

    all_features = [];

    %% ---- Extract features for each segment ----
    for s = 1:length(segments)
        seg = segments{s};

        % seg is size: win_samples × 6  (AccXYZ + GyroXYZ)
        seg_use = seg;   % 6 channels

        fv = extract_features(seg_use, fs);   % returns ~105 features
        all_features = [all_features; fv];
    end

    fprintf('  Extracted %d features per segment.\n', size(all_features,2));

    %% ---- Save features ----
```

```
    fshort = fname(1:end-13); % remove "_segments.mat"
    matname = fullfile(OUT_DIR, [fshort '_features.mat']);
    csvname = fullfile(OUT_DIR, [fshort '_features.csv']);

    save(matname, 'all_features');
    writematrix(all_features, csvname);

    fprintf('  Saved: %s\n\n', matname);

    %% ---------------- VISUALIZATION (ONLY FOR FIRST FILE) ----------------
    if visualize_user && f == 1
        fprintf('  Creating visualization...\n');

        figure('Name','Feature Visualization','Position',[50 50 1200 500]);

        % -------- (1) Boxplot of first 10 features --------
        subplot(1,2,1);
        boxplot(all_features(:,1:10));
        title('Boxplot of First 10 Features');
        xlabel('Feature Index'); ylabel('Value');
        grid on;

        % -------- (2) Mean of all features --------
        subplot(1,2,2);
        plot(mean(all_features,1),'LineWidth',1.5);
        title('Mean of All Features');
        xlabel('Feature Index'); ylabel('Mean Value');
        grid on;

        saveas(gcf, fullfile(OUT_DIR,'feature_visualization.png'));
    end

end

fprintf('\n=== FEATURE EXTRACTION COMPLETE ===\n');
fprintf('Features saved in: %s\n', OUT_DIR);
```

A.3 Feature Extraction Function (extract_features.m)

```
function fv = extract_features(seg, fs)
% FEATURE EXTRACTION FOR SMARTWATCH GAIT DATA
% Returns 105 time-domain features (appropriate for smartwatch sensors)
% Input: seg - segment matrix (150 x 6) with AccXYZ and GyroXYZ
% Output: fv - feature vector (1 x 105)
% NO TOOLBOX DEPENDENCIES - Uses manual correlation calculation

fv = [];
[N, C] = size(seg);

%% ====================================================
% TIME-DOMAIN FEATURES (96 features = 16 per channel x 6)
% ====================================================
for c = 1:C
    x = seg(:,c);

    m  = mean(x);
    sd = std(x);
    v  = var(x);
    mn = min(x);
    mx = max(x);
    rg = mx - mn;

    p25 = prctile(x,25);
    p50 = prctile(x,50);
    p75 = prctile(x,75);
    iqr_val = p75 - p25;

    if sd > 0
        skew_val = mean(((x-m)/sd).^3);
        kurt_val = mean(((x-m)/sd).^4);
    else
        skew_val = 0;
        kurt_val = 0;
    end

    rms_val = rms(x);
    energy  = sum(x.^2);
```

```
    zcr     = sum(abs(diff(sign(x)))) / (N-1);
    mad_val = mean(abs(x-m));

    fv = [fv, m, sd, v, mn, mx, rg, ...
          p25, p50, p75, iqr_val, ...
          skew_val, kurt_val, ...
          rms_val, energy, zcr, mad_val];
end

%% ====================================================
% GLOBAL GAIT FEATURES (9 features)
% ====================================================
acc  = seg(:,1:3);
gyro = seg(:,4:6);

accMag  = sqrt(sum(acc.^2,2));
gyroMag = sqrt(sum(gyro.^2,2));

SMA_acc  = sum(abs(accMag))  / N;
SMA_gyro = sum(abs(gyroMag)) / N;

SMV_acc  = mean(accMag);
SMV_gyro = mean(gyroMag);

% Manual correlation calculation (no toolbox needed)
corr_xy = manual_corr(acc(:,1), acc(:,2));
corr_xz = manual_corr(acc(:,1), acc(:,3));
corr_yz = manual_corr(acc(:,2), acc(:,3));

% Manual autocorrelation
maxLag = min(20, N-1);
ac = zeros(1, maxLag);

for k = 1:maxLag
    ac(k) = manual_corr(accMag(1:end-k), accMag(1+k:end));
end

auto_max  = max(ac);
auto_mean = mean(ac);
```

```
fv = [fv, SMA_acc, SMV_acc, SMA_gyro, SMV_gyro, ...
          corr_xy, corr_xz, corr_yz, ...
          auto_max, auto_mean];

fv = double(fv);
end

%% =====================================================
% HELPER FUNCTION: Manual Correlation
% =====================================================
function r = manual_corr(x, y)
    % Manual Pearson correlation coefficient
    % r = cov(x,y) / (std(x) * std(y))

    x = x(:);  % Make column vector
    y = y(:);

    % Remove NaN values
    valid = ~isnan(x) & ~isnan(y);
    x = x(valid);
    y = y(valid);

    if length(x) < 2
        r = 0;
        return;
    end

    % Calculate means
    mx = mean(x);
    my = mean(y);

    % Calculate deviations
    dx = x - mx;
    dy = y - my;

    % Calculate correlation
    numerator = sum(dx .* dy);
    denominator = sqrt(sum(dx.^2) * sum(dy.^2));
```

```
    if denominator == 0
        r = 0;
    else
        r = numerator / denominator;
    end
end
```

A.4 Script 3: Template Generation (script_3_Template_Generation.m)

```
% script_3_Template_Generation.m
% ----------------------------------------
% Creates templates for EACH USER:
%  - Genuine Train (FD)
%  - Impostor Train (others FD)
%  - Genuine Test (MD)
%  - Impostor Test (others MD)

clc; clear; close all;

FEATURE_DIR   = fullfile(pwd, 'Features');
TEMPLATE_DIR  = fullfile(pwd, 'Templates');

if ~exist(TEMPLATE_DIR,'dir')
    mkdir(TEMPLATE_DIR);
end

fprintf('\n=== TEMPLATE GENERATION STARTED ===\n');

% Load all extracted feature files
files = dir(fullfile(FEATURE_DIR, '*_features.mat'));
if isempty(files)
    error('No feature files found in %s', FEATURE_DIR);
end

% Identify FD and MD files
FD_files = {};
MD_files = {};
FD_users = [];
MD_users = [];
```

```
for i = 1:length(files)
    fname = files(i).name;

    % Detect user number (U1, U2, ...)
    token = regexp(fname, 'U(\d+)', 'tokens');
    if isempty(token), continue; end
    userID = str2double(token{1}{1});

    % Separate FD/MD
    if contains(fname, '_FD')
        FD_files{end+1} = fname;
        FD_users(end+1) = userID;

    elseif contains(fname, '_MD')
        MD_files{end+1} = fname;
        MD_users(end+1) = userID;
    end
end

users = unique([FD_users MD_users]);
num_users = length(users);

fprintf('Found %d users\n', num_users);
fprintf('FD files: %d | MD files: %d\n\n', length(FD_files), length(MD_files));

% ---------------------------------------------------------
% CREATE TEMPLATE FOR EACH USER
% ---------------------------------------------------------
for u = 1:num_users
    UID = users(u);
    fprintf('[%d/%d] Creating template for User %d\n', u, num_users, UID);

    train_features = [];
    train_labels   = [];
    test_features  = [];
    test_labels    = [];

    % -------------------------
```

```
% TRAINING (FD)
% -------------------------
for i = 1:length(FD_files)
    fname = FD_files{i};
    user_i = FD_users(i);

    load(fullfile(FEATURE_DIR, fname), "all_features");

    if user_i == UID
        % Genuine samples for this user
        train_features = [train_features; all_features];
        train_labels   = [train_labels; ones(size(all_features,1),1)];
    else
        % Impostor samples
        train_features = [train_features; all_features];
        train_labels   = [train_labels; zeros(size(all_features,1),1)];
    end
end

% -------------------------
% TESTING (MD)
% -------------------------
for i = 1:length(MD_files)
    fname = MD_files{i};
    user_i = MD_users(i);

    load(fullfile(FEATURE_DIR, fname), "all_features");

    if user_i == UID
        test_features = [test_features; all_features];
        test_labels   = [test_labels; ones(size(all_features,1),1)];
    else
        test_features = [test_features; all_features];
        test_labels   = [test_labels; zeros(size(all_features,1),1)];
    end
end

fprintf('  Training: %d genuine, %d impostor\n', ...
    sum(train_labels==1), sum(train_labels==0));
```

```
    fprintf('  Testing:  %d genuine, %d impostor\n', ...
        sum(test_labels==1), sum(test_labels==0));

    % -------------------------
    % SAVE TEMPLATE
    % -------------------------
    save(fullfile(TEMPLATE_DIR, sprintf('User%d_template.mat', UID)), ...
         'train_features', 'train_labels', 'test_features', 'test_labels');

    fprintf('  Saved User%d_template.mat\n\n', UID);
end

fprintf('=== TEMPLATE GENERATION COMPLETE ===\n');
fprintf('Templates saved in: %s\n\n', TEMPLATE_DIR);
```

A.5 Script 4: Classification (script_4_classification.m)

```
%  script_4_classification.m
%  One Binary Classifier Per User
%  Outputs: Accuracy, FAR, FRR, EER + Visualizations

clc; clear; close all;
rng(42);

TEMPLATE_DIR = fullfile(pwd, 'Templates');
RESULT_DIR   = fullfile(pwd, 'Results');
if ~exist(RESULT_DIR, 'dir'), mkdir(RESULT_DIR); end

fprintf("\n=== SCRIPT 4: CLASSIFICATION STARTED ===\n");

%% ----------------- Neural Network Settings ------------------
hidden_neurons  = 30;
train_algorithm = 'trainscg'; % Best default SCG
max_epochs      = 300;

%% ----------------- Load Templates ---------------------------
template_files = dir(fullfile(TEMPLATE_DIR, 'User*_template.mat'));
num_users = length(template_files);
```

```
if num_users == 0
    error("No templates found in %s", TEMPLATE_DIR);
end

fprintf("Found %d user templates.\n", num_users);

%% ------------------ Init result arrays ------------------------
ACC  = zeros(num_users,1);
PREC = zeros(num_users,1);
REC  = zeros(num_users,1);
FAR  = zeros(num_users,1);
FRR  = zeros(num_users,1);
EER  = zeros(num_users,1);
ThR  = zeros(num_users,1);

AllResults = struct();

%% ============================================================
%        TRAIN SEPARATE BINARY CLASSIFIER FOR EACH USER
% ============================================================
for u = 1:num_users

    fprintf("\n[%d/%d] User %d\n", u, num_users, u);

    load(fullfile(TEMPLATE_DIR, template_files(u).name), ...
        "train_features", "train_labels", "test_features", "test_labels");

    fprintf("Training samples: %d\n", length(train_labels));

    %% ----------- Normalization (Train Stats Only) -------------
    mu = mean(train_features);
    sd = std(train_features);

    trainN = (train_features - mu) ./ (sd + eps);
    testN  = (test_features  - mu) ./ (sd + eps);

    Xtr = trainN';  Ytr = train_labels';
    Xte = testN';   Yte = test_labels';
```

```
%% ----------- Create Neural Network -------------------------
net = feedforwardnet(hidden_neurons, train_algorithm);
net.trainParam.epochs = max_epochs;
net.trainParam.showWindow = false;
net.divideFcn = 'dividerand';
net.divideParam.trainRatio = 0.7;
net.divideParam.valRatio   = 0.15;
net.divideParam.testRatio  = 0.15;

[net,~] = train(net, Xtr, Ytr);

%% ----------- Testing -------------------------------------
out = net(Xte);

thresholds = 0:0.01:1;
FAR_curve = zeros(size(thresholds));
FRR_curve = zeros(size(thresholds));

genuine = (Yte == 1);

for t = 1:length(thresholds)
    pred = out >= thresholds(t);

    FP = sum(~genuine & pred);
    TN = sum(~genuine & ~pred);
    FN = sum(genuine & ~pred);
    TP = sum(genuine & pred);

    FAR_curve(t) = FP / (FP + TN + eps) * 100;
    FRR_curve(t) = FN / (FN + TP + eps) * 100;
end

% ---- Compute Equal Error Rate ----
[~, idx] = min(abs(FAR_curve - FRR_curve));
EER(u) = (FAR_curve(idx) + FRR_curve(idx)) / 2;
ThR(u) = thresholds(idx);

% Metrics at optimal threshold
```

```
    pred_final = out >= ThR(u);

    TP = sum(genuine & pred_final);
    TN = sum(~genuine & ~pred_final);
    FP = sum(~genuine & pred_final);
    FN = sum(genuine & ~pred_final);

    ACC(u)  = (TP + TN) / (TP + TN + FP + FN) * 100;
    PREC(u) = TP / (TP + FP + eps) * 100;
    REC(u)  = TP / (TP + FN + eps) * 100;
    FAR(u)  = FP / (FP + TN + eps) * 100;
    FRR(u)  = FN / (FN + TP + eps) * 100;

    % Save result struct
    AllResults(u).User       = u;
    AllResults(u).Accuracy   = ACC(u);
    AllResults(u).Precision  = PREC(u);
    AllResults(u).Recall     = REC(u);
    AllResults(u).FAR        = FAR(u);
    AllResults(u).FRR        = FRR(u);
    AllResults(u).EER        = EER(u);
    AllResults(u).Threshold  = ThR(u);
    AllResults(u).FAR_curve  = FAR_curve;
    AllResults(u).FRR_curve  = FRR_curve;
    AllResults(u).Thresholds = thresholds;

end

%% ============================================================
%                    SAVE RESULTS SUMMARY
% ============================================================
save(fullfile(RESULT_DIR, 'ClassificationResults.mat'), ...
     'ACC','PREC','REC','FAR','FRR','EER','ThR','AllResults');

fprintf("\n=== SUMMARY (ALL USERS) ===\n");
fprintf("Average Accuracy: %.2f%%\n", mean(ACC));
fprintf("Average EER:      %.2f%%\n", mean(EER));
fprintf("Lowest EER:       %.2f%% (User %d)\n", min(EER), find(EER==min(EER)));
fprintf("Highest EER:      %.2f%% (User %d)\n", max(EER), find(EER==max(EER)));
```

```
%% ============================================================
%                      VISUALIZATION SECTION
% ============================================================

fprintf("\n=== GENERATING VISUALIZATIONS ===\n");

%% ------------------ (1) Bar Charts --------------------------
figure('Position',[100 100 1200 700]);

subplot(2,2,1); bar(ACC);  title('Accuracy per User'); ylabel('%'); ylim([0 100]);
grid on;
subplot(2,2,2); bar(FAR);  title('FAR per User'); ylabel('%'); ylim([0 10]); grid
on;
subplot(2,2,3); bar(FRR);  title('FRR per User'); ylabel('%'); ylim([0 10]); grid
on;
subplot(2,2,4); bar(EER);  title('EER per User'); ylabel('%'); ylim([0 10]); grid
on;

saveas(gcf, fullfile(RESULT_DIR, 'PerUser_BarCharts.png'));

%% ------------------ (2) FAR/FRR Curves ----------------------
figure('Position',[100 100 1300 700]);
for u = 1:num_users
    subplot(2,5,u);
    plot(AllResults(u).Thresholds, AllResults(u).FAR_curve, 'r', 'LineWidth',1.5);
hold on;
    plot(AllResults(u).Thresholds, AllResults(u).FRR_curve, 'b', 'LineWidth',1.5);

    [~,idx] = min(abs(AllResults(u).FAR_curve - AllResults(u).FRR_curve));
    plot(AllResults(u).Thresholds(idx),              AllResults(u).FAR_curve(idx),
'ko','MarkerSize',8,'LineWidth',1.5);

    title(sprintf('User %d (EER=%.2f%%)', u, AllResults(u).EER));
    xlabel('Threshold'); ylabel('Error (%)'); grid on;
    legend('FAR','FRR','EER Point');
end
saveas(gcf, fullfile(RESULT_DIR, 'FAR_FRR_AllUsers.png'));

%% ------------------ (3) ROC Curves --------------------------
```

```
figure('Position',[100 100 1300 700]);
for u = 1:num_users
    subplot(2,5,u);
    TPR = 100 - AllResults(u).FRR_curve;
    FPR = AllResults(u).FAR_curve;

    plot(FPR, TPR, 'b', 'LineWidth',1.5); hold on;
    plot([0 100],[0 100],'r--');

    title(sprintf('User %d ROC', u));
    xlabel('FPR (%)'); ylabel('TPR (%)');
    axis([0 100 0 100]); axis square; grid on;
end
saveas(gcf, fullfile(RESULT_DIR, 'ROC_AllUsers.png'));

%% ------------------ (4) Average FAR/FRR ----------------------
avg_far = mean(cell2mat({AllResults.FAR_curve}'),1);
avg_frr = mean(cell2mat({AllResults.FRR_curve}'),1);

figure('Position',[100 100 800 600]);
plot(thresholds, avg_far,'r','LineWidth',2); hold on;
plot(thresholds, avg_frr,'b','LineWidth',2);

[~,best] = min(abs(avg_far - avg_frr));
avg_eer = (avg_far(best) + avg_frr(best))/2;

plot(thresholds(best), avg_eer,'ko','MarkerSize',10,'LineWidth',2);

title(sprintf('Average FAR/FRR (EER = %.2f%%)', avg_eer));
xlabel('Threshold'); ylabel('Error (%)'); grid on;
legend('Average FAR','Average FRR','EER Point');
saveas(gcf, fullfile(RESULT_DIR, 'Average_FAR_FRR.png'));

%% ------------------ (5) Average ROC Curve --------------------
avg_TPR = 100 - avg_frr;
avg_FPR = avg_far;

AUC = trapz(avg_FPR, avg_TPR) / 10000 * 100;
```

```
figure('Position',[100 100 800 600]);
plot(avg_FPR, avg_TPR, 'b','LineWidth',2.5); hold on;
plot([0 100],[0 100],'r--');

title(sprintf('Average ROC Curve (AUC = %.2f%%)', AUC));
xlabel('False Positive Rate (%)'); ylabel('True Positive Rate (%)');
axis([0 100 0 100]); axis square; grid on;

saveas(gcf, fullfile(RESULT_DIR, 'Average_ROC.png'));

fprintf("\n=== VISUALIZATION COMPLETE ===\n");
fprintf("All plots saved in: %s\n", RESULT_DIR);

%% ---------------- CREATE USER METRIC TABLE ---------------------
fprintf("\n=== PER-USER PERFORMANCE TABLE ===\n");

fprintf('%-8s | %-8s | %-8s | %-8s | %-8s\n', ...
        'User', 'CCR%', 'FAR%', 'FRR%', 'EER%');
fprintf('%s\n', repmat('-',1,50));

for u = 1:num_users
    fprintf('%-8d | %-8.2f | %-8.2f | %-8.2f | %-8.2f\n', ...
        u, ACC(u), FAR(u), FRR(u), EER(u));
end

% Save the table to CSV
T = table((1:num_users)', ACC, FAR, FRR, EER, ...
        'VariableNames', {'User','CCR','FAR','FRR','EER'});

writetable(T, fullfile(RESULT_DIR, 'PerUser_Metrics.csv'));

fprintf('\nSaved per-user table as PerUser_Metrics.csv\n');

fprintf("\n=== SCRIPT 4 COMPLETE ===\n");
```

A.6 Script 5: Optimization (script_5_Optimization.m)

```
% script_5_Optimization.m
% Complete optimization experiments in one script
% Tests different sensor modes and neuron counts, compares with SVM

clc; clear; close all;
rng(42);

fprintf('\n=== COMPLETE OPTIMIZATION EXPERIMENTS ===\n\n');

%% -------------------- CONFIGURATION --------------------
PREP_DIR = 'Preprocessed';
RESULT_DIR = 'Results_Optimization';
if ~exist(RESULT_DIR,'dir'), mkdir(RESULT_DIR); end

fs = 32;
sensor_modes = {'combined', 'acc_only', 'gyro_only'};
neuron_counts = [10, 20, 30, 50];

%% -------------------- LOAD PREPROCESSED FILES --------------------
fprintf('Step 1: Loading preprocessed segments...\n');

seg_files = dir(fullfile(PREP_DIR, '*_segments.mat'));
if isempty(seg_files)
    error('No preprocessed files found. Run Preprocessing.m first!');
end

% organize files by user and session
file_map = struct();
for i = 1:length(seg_files)
    fname = seg_files(i).name;

    % extract user number
    tokens = regexp(fname, 'U(\d+)', 'tokens');
    if isempty(tokens), continue; end
    user = str2double(tokens{1}{1});

    key = sprintf('U%d', user);
    if ~isfield(file_map, key)
```

```
        file_map.(key) = struct('FD','', 'MD','');
    end

    % determine session type
    if contains(fname, '_FD', 'IgnoreCase', true)
        file_map.(key).FD = fullfile(PREP_DIR, fname);
    elseif contains(fname, '_MD', 'IgnoreCase', true)
        file_map.(key).MD = fullfile(PREP_DIR, fname);
    end
end

% get valid users (have both FD and MD)
users = fieldnames(file_map);
valid_users = [];
for i = 1:length(users)
    u = file_map.(users{i});
    if ~isempty(u.FD) && ~isempty(u.MD)
        valid_users(end+1) = str2double(users{i}(2:end));
    end
end
valid_users = sort(valid_users);
num_users = length(valid_users);

fprintf('  Found %d valid users\n\n', num_users);

%% -------------------- MAIN EXPERIMENTS --------------------
results = [];
result_count = 0;

%% EXPERIMENT 1: Sensor Mode Comparison (30 neurons)
fprintf('EXPERIMENT 1: Sensor Mode Comparison\n');
fprintf('%s\n', repmat('-',1,60));

for mode_idx = 1:length(sensor_modes)
    mode = sensor_modes{mode_idx};

    fprintf('[%d/%d] Testing %s (30 neurons)... ', mode_idx, length(sensor_modes),
mode);

    % extract features and train/test for this mode
```

```
    [acc, far, frr, eer] = run_experiment(file_map, valid_users, mode, 30, 0.7,
105, fs);

    result_count = result_count + 1;
    results(result_count).name = mode;
    results(result_count).experiment = 'sensor';
    results(result_count).neurons = 30;
    results(result_count).split_ratio = 0.7;
    results(result_count).num_features = 105;
    results(result_count).accuracy = mean(acc);
    results(result_count).FAR = mean(far);
    results(result_count).FRR = mean(frr);
    results(result_count).EER = mean(eer);

    fprintf('Acc=%.2f%%, EER=%.2f%%\n', mean(acc), mean(eer));
end

%% EXPERIMENT 2: Neuron Count Comparison (combined sensors)
fprintf('\nEXPERIMENT 2: Hidden Neuron Count Comparison\n');
fprintf('%s\n', repmat('-',1,60));

for n_idx = 1:length(neuron_counts)
    num_neurons = neuron_counts(n_idx);

    fprintf('[%d/%d] Testing %d neurons... ', n_idx, length(neuron_counts),
num_neurons);

    [acc, far, frr, eer] = run_experiment(file_map, valid_users, 'combined',
num_neurons, 0.7, 105, fs);

    result_count = result_count + 1;
    results(result_count).name = sprintf('%d_neurons', num_neurons);
    results(result_count).experiment = 'neurons';
    results(result_count).neurons = num_neurons;
    results(result_count).split_ratio = 0.7;
    results(result_count).num_features = 105;
    results(result_count).accuracy = mean(acc);
    results(result_count).FAR = mean(far);
    results(result_count).FRR = mean(frr);
    results(result_count).EER = mean(eer);
```

```
    fprintf('Acc=%.2f%%, EER=%.2f%%\n', mean(acc), mean(eer));
end

%% EXPERIMENT 3: Train/Test Split Ratio
fprintf('\nEXPERIMENT 3: Train/Test Split Ratio\n');
fprintf('%s\n', repmat('-',1,60));

split_ratios = [0.6, 0.7, 0.8];
for split_idx = 1:length(split_ratios)
    ratio = split_ratios(split_idx);

    fprintf('[%d/%d] Testing %.0f/%.0f split... ', split_idx, length(split_ratios),
...
        ratio*100, (1-ratio)*100);

    [acc, far, frr, eer] = run_experiment(file_map, valid_users, 'combined', 30,
ratio, 105, fs);

    result_count = result_count + 1;
    results(result_count).name   =   sprintf('%.0f-%.0f_split',   ratio*100,   (1-
ratio)*100);
    results(result_count).experiment = 'split_ratio';
    results(result_count).neurons = 30;
    results(result_count).accuracy = mean(acc);
    results(result_count).FAR = mean(far);
    results(result_count).FRR = mean(frr);
    results(result_count).EER = mean(eer);

    fprintf('Acc=%.2f%%, EER=%.2f%%\n', mean(acc), mean(eer));
end

%% EXPERIMENT 4: Feature Reduction
fprintf('\nEXPERIMENT 4: Feature Reduction (Cost Optimization)\n');
fprintf('%s\n', repmat('-',1,60));

feature_counts = [105, 70, 50, 30];
for feat_idx = 1:length(feature_counts)
    num_feat = feature_counts(feat_idx);
```

```
    fprintf('[%d/%d] Testing %d features... ', feat_idx, length(feature_counts),
num_feat);

    [acc, far, frr, eer] = run_experiment(file_map, valid_users, 'combined', 30,
0.7, num_feat, fs);

    result_count = result_count + 1;
    results(result_count).name = sprintf('%d_features', num_feat);
    results(result_count).experiment = 'feature_reduction';
    results(result_count).neurons = 30;
    results(result_count).split_ratio = 0.7;
    results(result_count).num_features = num_feat;
    results(result_count).accuracy = mean(acc);
    results(result_count).FAR = mean(far);
    results(result_count).FRR = mean(frr);
    results(result_count).EER = mean(eer);

    cost_reduction = (105 - num_feat) / 105 * 100;
    fprintf('Acc=%.2f%%, EER=%.2f%% (Cost: -%.1f%%)\n', ...
        mean(acc), mean(eer), cost_reduction);
end

%% EXPERIMENT 5: PCA Feature Reduction
fprintf('\nEXPERIMENT 5: PCA Feature Reduction\n');
fprintf('%s\n', repmat('-',1,60));

pca_feature_counts = [90, 70, 50, 30];
for feat_idx = 1:length(pca_feature_counts)
    num_pca = pca_feature_counts(feat_idx);

    fprintf('[%d/%d]    Testing    %d    PCA    components...    ',    feat_idx,
length(pca_feature_counts), num_pca);

    [acc, far, frr, eer] = run_pca_experiment(file_map, valid_users, 'combined',
10, 0.8, num_pca, fs);

    result_count = result_count + 1;
    results(result_count).name = sprintf('%d_PCA', num_pca);
    results(result_count).experiment = 'PCA';
    results(result_count).neurons = 10;
    results(result_count).split_ratio = 0.8;
```

```
    results(result_count).num_features = num_pca;
    results(result_count).accuracy = mean(acc);
    results(result_count).FAR = mean(far);
    results(result_count).FRR = mean(frr);
    results(result_count).EER = mean(eer);

    fprintf('Acc=%.2f%%, EER=%.2f%%\n', mean(acc), mean(eer));
end

%% EXPERIMENT 6: SVM Comparison
fprintf('\nEXPERIMENT 3: SVM Comparison\n');
fprintf('%s\n', repmat('-',1,60));
fprintf('Testing SVM classifier... ');

[acc, far, frr, eer] = run_svm_experiment(file_map, valid_users, 'combined', fs);

result_count = result_count + 1;
results(result_count).name = 'SVM';
results(result_count).experiment = 'algorithm';
results(result_count).neurons = 0;
results(result_count).split_ratio = 0.7;
results(result_count).num_features = 105;
results(result_count).accuracy = mean(acc);
results(result_count).FAR = mean(far);
results(result_count).FRR = mean(frr);
results(result_count).EER = mean(eer);

fprintf('Acc=%.2f%%, EER=%.2f%%\n', mean(acc), mean(eer));

%% -------------------- SAVE RESULTS --------------------
save(fullfile(RESULT_DIR, 'optimization_results.mat'), 'results');

% print summary table
fprintf('\n=== RESULTS SUMMARY ===\n\n');
fprintf('%-20s | %-20s | %10s | %8s | %8s | %8s\n', ...
    'Configuration', 'Experiment', 'Accuracy', 'FAR', 'FRR', 'EER');
fprintf('%s\n', repmat('-',1,75));

for i = 1:length(results)
```

```
    fprintf('%-20s | %-20s | %9.2f%% | %7.2f%% | %7.2f%% | %7.2f%%\n', ...
        results(i).name, results(i).experiment, results(i).accuracy, ...
        results(i).FAR, results(i).FRR, results(i).EER);
end

% export to CSV
csvfile = fullfile(RESULT_DIR, 'optimization_summary.csv');
fid = fopen(csvfile, 'w');
fprintf(fid, 'Configuration,Experiment,Neurons,Accuracy,FAR,FRR,EER\n');
for i = 1:length(results)
    fprintf(fid, '%s,%s,%d,%.2f,%.2f,%.2f,%.2f\n', ...
        results(i).name, results(i).experiment, results(i).neurons, ...
        results(i).accuracy, results(i).FAR, results(i).FRR, results(i).EER);
end
fclose(fid);

fprintf('\nResults saved to: %s\n', RESULT_DIR);
fprintf('CSV file: %s\n', csvfile);
fprintf('\n=== ALL EXPERIMENTS COMPLETE ===\n');

%% Generate Comparison Graphs
fprintf('\nGenerating comparison graphs...\n');

% Create results directory if needed
if ~exist(RESULT_DIR,'dir'), mkdir(RESULT_DIR); end

% Graph 1: Sensor Mode Comparison
sensor_idx = find(strcmp({results.experiment}, 'sensor'));
if ~isempty(sensor_idx)
    sensor_results = results(sensor_idx);

    figure('Position',[100 100 1200 400]);

    subplot(1,3,1);
    bar([sensor_results.accuracy]);
    set(gca, 'XTickLabel', {sensor_results.name}, 'XTickLabelRotation', 45);
    title('Accuracy by Sensor Mode');
    ylabel('Accuracy (%)');
    ylim([80 100]);
```

```
    grid on;

    subplot(1,3,2);
    bar([sensor_results.EER]);
    set(gca, 'XTickLabel', {sensor_results.name}, 'XTickLabelRotation', 45);
    title('EER by Sensor Mode');
    ylabel('EER (%)');
    grid on;

    subplot(1,3,3);
    b = bar([[sensor_results.FAR]' [sensor_results.FRR]']);
    set(gca, 'XTickLabel', {sensor_results.name}, 'XTickLabelRotation', 45);
    title('FAR vs FRR by Sensor Mode');
    ylabel('Error Rate (%)');
    legend('FAR', 'FRR');
    grid on;

    saveas(gcf, fullfile(RESULT_DIR, 'Sensor_Mode_Comparison.png'));
end

% Graph 2: Neuron Count Comparison
neuron_idx = find(strcmp({results.experiment}, 'neurons'));
if ~isempty(neuron_idx)
    neuron_results = results(neuron_idx);
    neurons = [neuron_results.neurons];

    figure('Position',[100 100 1200 400]);

    subplot(1,3,1);
    plot(neurons, [neuron_results.accuracy], '-o', 'LineWidth', 2, 'MarkerSize',
8);
    title('Accuracy vs Hidden Neurons');
    xlabel('Number of Neurons');
    ylabel('Accuracy (%)');
    grid on;

    subplot(1,3,2);
    plot(neurons, [neuron_results.EER], '-o', 'LineWidth', 2, 'MarkerSize', 8);
    title('EER vs Hidden Neurons');
    xlabel('Number of Neurons');
```

```
    ylabel('EER (%)');
    grid on;

    subplot(1,3,3);
    plot(neurons, [neuron_results.FAR], '-o', 'LineWidth', 2, 'MarkerSize', 8);
    hold on;
    plot(neurons, [neuron_results.FRR], '-s', 'LineWidth', 2, 'MarkerSize', 8);
    title('FAR & FRR vs Hidden Neurons');
    xlabel('Number of Neurons');
    ylabel('Error Rate (%)');
    legend('FAR', 'FRR');
    grid on;

    saveas(gcf, fullfile(RESULT_DIR, 'Neuron_Count_Comparison.png'));
end

% Graph 3: Feature Reduction
feat_idx = find(strcmp({results.experiment}, 'feature_reduction'));
if ~isempty(feat_idx)
    feat_results = results(feat_idx);

    % Safely extract features - check if field exists
    if isfield(feat_results, 'num_features')
        feats = [feat_results.num_features];
    else
        % If field missing, use default feature counts
        feats = [105, 70, 50, 30];
        feats = feats(1:length(feat_results));  % Match length
    end

    % Make sure arrays are same size
    if length(feats) ~= length(feat_results)
        warning('Feature count mismatch, skipping feature reduction graph');
    else
        cost_reduction = (105 - feats) / 105 * 100;

        figure('Position',[100 100 1200 400]);

        subplot(1,3,1);
```

```
        plot(feats, [feat_results.EER], '-o', 'LineWidth', 2, 'MarkerSize', 8);
        title('EER vs Feature Count');
        xlabel('Number of Features');
        ylabel('EER (%)');
        grid on;

        subplot(1,3,2);
        plot(cost_reduction, [feat_results.EER], '-o', 'LineWidth', 2,
'MarkerSize', 8);
        title('EER vs Cost Reduction');
        xlabel('Cost Reduction (%)');
        ylabel('EER (%)');
        grid on;

        subplot(1,3,3);
        bar(feats, [feat_results.accuracy]);
        title('Accuracy by Feature Count');
        xlabel('Number of Features');
        ylabel('Accuracy (%)');
        ylim([min([feat_results.accuracy])-5, 100]);
        grid on;

        % Add text annotations
        for i = 1:length(feats)
            text(feats(i), feat_results(i).accuracy-1, ...
                sprintf('-%.0f%%', cost_reduction(i)), ...
                'HorizontalAlignment', 'center', 'FontSize', 8);
        end

        saveas(gcf, fullfile(RESULT_DIR, 'Feature_Reduction_Comparison.png'));
    end
end

% Graph 4: PCA Feature Reduction
pca_idx = find(strcmp({results.experiment}, 'PCA'));
if ~isempty(pca_idx)
    pca_results = results(pca_idx);
    pca_feats = [pca_results.num_features];

    figure('Position',[100 100 1200 400]);
```

```
    subplot(1,3,1);
    plot(pca_feats, [pca_results.EER], '-o', 'LineWidth', 2, 'MarkerSize', 8);
    title('EER vs PCA Components');
    xlabel('Number of PCA Components');
    ylabel('EER (%)');
    grid on;

    subplot(1,3,2);
    plot(pca_feats, [pca_results.accuracy], '-o', 'LineWidth', 2, 'MarkerSize', 8);
    title('Accuracy vs PCA Components');
    xlabel('Number of PCA Components');
    ylabel('Accuracy (%)');
    grid on;

    subplot(1,3,3);
    plot(pca_feats, [pca_results.FAR], '-o', 'LineWidth', 2, 'MarkerSize', 8);
    hold on;
    plot(pca_feats, [pca_results.FRR], '-s', 'LineWidth', 2, 'MarkerSize', 8);
    title('FAR & FRR vs PCA Components');
    xlabel('Number of PCA Components');
    ylabel('Error Rate (%)');
    legend('FAR', 'FRR');
    grid on;

    saveas(gcf, fullfile(RESULT_DIR, 'PCA_Comparison.png'));
end

fprintf('  Saved comparison graphs to: %s\n', RESULT_DIR);

%% ==================== HELPER FUNCTIONS ====================

function [acc, far, frr, eer] = run_experiment(file_map, users, sensor_mode,
num_neurons, split_ratio, num_features, fs)
    % Run neural network experiment for given configuration

    num_users = length(users);
    acc = zeros(num_users, 1);
    far = zeros(num_users, 1);
    frr = zeros(num_users, 1);
```

```
eer = zeros(num_users, 1);

for u_idx = 1:num_users
    user = users(u_idx);

    % extract features for this user
    [train_feat, train_lbl, test_feat, test_lbl] = ...
        extract_user_data(file_map, users, user, sensor_mode, fs);

    if isempty(train_feat) || isempty(test_feat)
        continue;
    end

    % NEW: Feature reduction if needed
    if num_features < size(train_feat, 2)
        train_feat = train_feat(:, 1:num_features);
        test_feat = test_feat(:, 1:num_features);
    end

    % NEW: Apply split ratio to training data
    num_train = size(train_feat, 1);
    num_use = round(num_train * split_ratio);
    idx = randperm(num_train, num_use);
    train_feat = train_feat(idx, :);
    train_lbl = train_lbl(idx);

    % normalize
    mu = mean(train_feat);
    sigma = std(train_feat) + eps;

    train_norm = (train_feat - mu) ./ sigma;
    test_norm = (test_feat - mu) ./ sigma;

    % train neural network
    net = feedforwardnet(num_neurons, 'trainscg');
    net.trainParam.epochs = 300;
    net.trainParam.showWindow = false;
    net.divideFcn = 'dividerand';
    net.divideParam.trainRatio = 0.7;
```

```
        net.divideParam.valRatio = 0.15;
        net.divideParam.testRatio = 0.15;

        net = train(net, train_norm', train_lbl');

        % test
        outputs = net(test_norm');

        % calculate metrics
        [acc(u_idx), far(u_idx), frr(u_idx), eer(u_idx)] = ...
            calculate_metrics(outputs', test_lbl);
    end
end

function [acc, far, frr, eer] = run_pca_experiment(file_map, users, sensor_mode,
num_neurons, split_ratio, num_pca, fs)
    % Run neural network experiment with PCA feature reduction

    num_users = length(users);
    acc = zeros(num_users, 1);
    far = zeros(num_users, 1);
    frr = zeros(num_users, 1);
    eer = zeros(num_users, 1);

    for u_idx = 1:num_users
        user = users(u_idx);

        [train_feat, train_lbl, test_feat, test_lbl] = ...
            extract_user_data(file_map, users, user, sensor_mode, fs);

        if isempty(train_feat) || isempty(test_feat)
            continue;
        end

        % Apply split ratio
        num_train = size(train_feat, 1);
        num_use = round(num_train * split_ratio);
        idx = randperm(num_train, num_use);
        train_feat = train_feat(idx, :);
        train_lbl = train_lbl(idx);
```

```
        % Normalize BEFORE PCA
        mu = mean(train_feat);
        sigma = std(train_feat) + eps;
        train_norm = (train_feat - mu) ./ sigma;
        test_norm = (test_feat - mu) ./ sigma;

        % Apply PCA
        [coeff, score, ~] = pca(train_norm);

        % Determine actual number of components to use
        actual_components = min(num_pca, size(coeff, 2));

        % Use only top N components
        train_pca = score(:, 1:actual_components);
        test_pca = test_norm * coeff(:, 1:actual_components);

        % Train neural network
        net = feedforwardnet(num_neurons, 'trainscg');
        net.trainParam.epochs = 300;
        net.trainParam.showWindow = false;
        net.divideFcn = 'dividerand';
        net.divideParam.trainRatio = 0.7;
        net.divideParam.valRatio = 0.15;
        net.divideParam.testRatio = 0.15;

        net = train(net, train_pca', train_lbl');
        outputs = net(test_pca');

        [acc(u_idx), far(u_idx), frr(u_idx), eer(u_idx)] = ...
            calculate_metrics(outputs', test_lbl);
    end
end

function [acc, far, frr, eer] = run_svm_experiment(file_map, users, sensor_mode,
fs)
    % Run SVM experiment

    num_users = length(users);
    acc = zeros(num_users, 1);
```

```
    far = zeros(num_users, 1);
    frr = zeros(num_users, 1);
    eer = zeros(num_users, 1);

    for u_idx = 1:num_users
        user = users(u_idx);

        % extract features
        [train_feat, train_lbl, test_feat, test_lbl] = ...
            extract_user_data(file_map, users, user, sensor_mode, fs);

        if isempty(train_feat) || isempty(test_feat)
            continue;
        end

        % normalize
        mu = mean(train_feat);
        sigma = std(train_feat) + eps;

        train_norm = (train_feat - mu) ./ sigma;
        test_norm = (test_feat - mu) ./ sigma;

        % train simple linear classifier
        X = train_norm';
        y = train_lbl';
        lambda = 0.01;
        w = (X * X' + lambda * eye(size(X,1))) \ (X * y');

        % predict
        scores = w' * test_norm';
        scores = scores';
        scores = (scores - min(scores)) / (max(scores) - min(scores) + eps);

        % calculate metrics
        [acc(u_idx), far(u_idx), frr(u_idx), eer(u_idx)] = ...
            calculate_metrics(scores, test_lbl);
    end
end
```

```
function [train_features, train_labels, test_features, test_labels] = ...
    extract_user_data(file_map, all_users, target_user, sensor_mode, fs)
    % Extract and label features for one user

    train_features = [];
    train_labels = [];
    test_features = [];
    test_labels = [];

    % process all users
    for u_idx = 1:length(all_users)
        user = all_users(u_idx);
        key = sprintf('U%d', user);

        % training data (FD)
        if ~isempty(file_map.(key).FD)
            load(file_map.(key).FD, 'segments');
            feats = extract_features_from_segments(segments, sensor_mode, fs);

            if user == target_user
                train_features = [train_features; feats];
                train_labels = [train_labels; ones(size(feats,1), 1)];
            else
                train_features = [train_features; feats];
                train_labels = [train_labels; zeros(size(feats,1), 1)];
            end
        end

        % testing data (MD)
        if ~isempty(file_map.(key).MD)
            load(file_map.(key).MD, 'segments');
            feats = extract_features_from_segments(segments, sensor_mode, fs);

            if user == target_user
                test_features = [test_features; feats];
                test_labels = [test_labels; ones(size(feats,1), 1)];
            else
                test_features = [test_features; feats];
```

```
                test_labels = [test_labels; zeros(size(feats,1), 1)];
            end
        end
    end
end


function features = extract_features_from_segments(segments, sensor_mode, fs)
    % Extract features from all segments

    features = [];

    for s = 1:length(segments)
        seg = segments{s};
        fv = extract_features_optimization(seg, fs, sensor_mode);
        features = [features; fv];
    end
end


function [acc, far, frr, eer] = calculate_metrics(outputs, labels)
    % Calculate classification metrics

    thresholds = 0:0.01:1;
    far_curve = zeros(size(thresholds));
    frr_curve = zeros(size(thresholds));

    genuine = (labels == 1);

    for t = 1:length(thresholds)
        pred = outputs >= thresholds(t);

        fp = sum(~genuine & pred);
        tn = sum(~genuine & ~pred);
        fn = sum(genuine & ~pred);
        tp = sum(genuine & pred);

        far_curve(t) = fp / (fp + tn + eps) * 100;
        frr_curve(t) = fn / (fn + tp + eps) * 100;
```

```
    end

    % find EER
    [~, eer_idx] = min(abs(far_curve - frr_curve));
    eer = (far_curve(eer_idx) + frr_curve(eer_idx)) / 2;
    optimal_threshold = thresholds(eer_idx);

    % final metrics at optimal threshold
    pred_final = outputs >= optimal_threshold;
    tp = sum(genuine & pred_final);
    tn = sum(~genuine & ~pred_final);
    fp = sum(~genuine & pred_final);
    fn = sum(genuine & ~pred_final);

    acc = (tp + tn) / (tp + tn + fp + fn) * 100;
    far = fp / (fp + tn + eps) * 100;
    frr = fn / (fn + tp + eps) * 100;
end

%% ==================== PER-USER PERFORMANCE TABLE (OPTIMIZED) ====================
fprintf('\n=== Generating Per-User Performance Table (Optimized Configuration) ===\n');

% OPTIMIZED configuration (best from experiments)
optimized_mode = 'combined';
optimized_neurons = 10;  % Best from Experiment 2
optimized_split = 0.8;   % Best from Experiment 3
optimized_features = 70; % Best cost-performance from Experiment 4

fprintf('Optimized Config: %s sensors, %d neurons, %.0f/%.0f split, %d features\n\n', ...
    optimized_mode, optimized_neurons, optimized_split*100, (1-optimized_split)*100, optimized_features);

% Arrays to store per-user results
user_ccr = zeros(num_users, 1);
user_far = zeros(num_users, 1);
user_frr = zeros(num_users, 1);
user_eer = zeros(num_users, 1);
```

```
fprintf('Processing per-user results...\n');

% Run optimized configuration for each user
for u_idx = 1:num_users
    user = valid_users(u_idx);

    fprintf('  User %d... ', user);

    % Extract features for this user
    [train_feat, train_lbl, test_feat, test_lbl] = ...
        extract_user_data(file_map, valid_users, user, optimized_mode, fs);

    if isempty(train_feat) || isempty(test_feat)
        fprintf('SKIP (no data)\n');
        continue;
    end

    % Apply feature reduction
    if optimized_features < size(train_feat, 2)
        train_feat = train_feat(:, 1:optimized_features);
        test_feat = test_feat(:, 1:optimized_features);
    end

    % Apply train/test split
    num_train = size(train_feat, 1);
    num_use = round(num_train * optimized_split);
    idx = randperm(num_train, num_use);
    train_feat = train_feat(idx, :);
    train_lbl = train_lbl(idx);

    % Normalize
    mu = mean(train_feat);
    sigma = std(train_feat) + eps;
    train_norm = (train_feat - mu) ./ sigma;
    test_norm = (test_feat - mu) ./ sigma;

    % Train neural network
    net = feedforwardnet(optimized_neurons, 'trainscg');
    net.trainParam.epochs = 300;
```

```
    net.trainParam.showWindow = false;
    net.divideFcn = 'dividerand';
    net.divideParam.trainRatio = 0.7;
    net.divideParam.valRatio = 0.15;
    net.divideParam.testRatio = 0.15;

    net = train(net, train_norm', train_lbl');

    % Test
    outputs = net(test_norm');

    % Calculate metrics
    [acc, far, frr, eer] = calculate_metrics(outputs', test_lbl);

    user_ccr(u_idx) = acc;
    user_far(u_idx) = far;
    user_frr(u_idx) = frr;
    user_eer(u_idx) = eer;

    fprintf('CCR=%.2f%%, EER=%.2f%%\n', acc, eer);
end

%% Display Per-User Performance Table
fprintf('\n=== PER-USER PERFORMANCE TABLE (OPTIMIZED) ===\n');
fprintf('User | CCR%%     | FAR%%    | FRR%%    | EER%%\n');
fprintf('-----|---------|--------|--------|--------\n');

for u_idx = 1:num_users
    user = valid_users(u_idx);
    fprintf('%-4d | %7.2f | %6.2f | %6.2f | %6.2f\n', ...
        user, user_ccr(u_idx), user_far(u_idx), user_frr(u_idx), user_eer(u_idx));
end

fprintf('-----|---------|--------|--------|--------\n');
fprintf('AVG  | %7.2f | %6.2f | %6.2f | %6.2f\n', ...
    mean(user_ccr), mean(user_far), mean(user_frr), mean(user_eer));

%% Save per-user results to CSV
per_user_csv = fullfile(RESULT_DIR, 'per_user_optimized.csv');
```

```
fid = fopen(per_user_csv, 'w');
fprintf(fid, 'User,CCR,FAR,FRR,EER\n');
for u_idx = 1:num_users
    user = valid_users(u_idx);
    fprintf(fid, '%.2f,%.2f,%.2f,%.2f,%.2f\n', ...
        user, user_ccr(u_idx), user_far(u_idx), user_frr(u_idx), user_eer(u_idx));
end
fprintf(fid, '%.2f,%.2f,%.2f,%.2f,%.2f\n', ...
    mean(user_ccr), mean(user_far), mean(user_frr), mean(user_eer));
fclose(fid);

fprintf('\nPer-user results saved to: %s\n', per_user_csv);
fprintf('\n=== PER-USER ANALYSIS COMPLETE ===\n');
```

A.7 Feature Extraction - Optimization Version (extract_features_optimization.m)

```
function fv = extract_features_optimization(seg, fs, sensor_mode)
% extract features from segment - optimization version
% seg = data segment, fs = sampling rate, sensor_mode = which sensors to use

if nargin < 3
    sensor_mode = 'combined';
end

fv = [];
[N, C] = size(seg);

% figure out which columns to use
if strcmp(sensor_mode, 'acc_only')
    use_cols = 1:3;
    has_acc = true;
    has_gyro = false;
elseif strcmp(sensor_mode, 'gyro_only')
    use_cols = 4:6;
    has_acc = false;
    has_gyro = true;
else
    use_cols = 1:6;
    has_acc = true;
```

```
    has_gyro = true;
end

% time domain features for each column
for c = use_cols
    x = seg(:,c);

    m = mean(x);
    s = std(x);
    v = var(x);
    minval = min(x);
    maxval = max(x);
    rng = maxval - minval;

    p25 = prctile(x,25);
    p50 = prctile(x,50);
    p75 = prctile(x,75);
    iqr = p75 - p25;

    if s > 0
        skew = mean(((x-m)/s).^3);
        kurt = mean(((x-m)/s).^4);
    else
        skew = 0;
        kurt = 0;
    end

    rmsval = rms(x);
    energy = sum(x.^2);
    zcr = sum(abs(diff(sign(x))))/(N-1);
    mad = mean(abs(x-m));

    fv = [fv, m, s, v, minval, maxval, rng, p25, p50, p75, iqr, skew, kurt, rmsval,
energy, zcr, mad];
end

% magnitude and correlation features
if has_acc
    acc = seg(:,1:3);
    accmag = sqrt(sum(acc.^2,2));
```

```
    sma_a = sum(abs(accmag))/N;
    smv_a = mean(accmag);

    cxy = corrcoef_manual(acc(:,1), acc(:,2));
    cxz = corrcoef_manual(acc(:,1), acc(:,3));
    cyz = corrcoef_manual(acc(:,2), acc(:,3));

    % autocorr
    lag = min(20, N-1);
    ac = zeros(1,lag);
    for k = 1:lag
        ac(k) = corrcoef_manual(accmag(1:end-k), accmag(1+k:end));
    end

    acmax = max(ac);
    acmean = mean(ac);

    fv = [fv, sma_a, smv_a, cxy, cxz, cyz, acmax, acmean];
end

if has_gyro
    gyro = seg(:,4:6);
    gyromag = sqrt(sum(gyro.^2,2));

    sma_g = sum(abs(gyromag))/N;
    smv_g = mean(gyromag);

    fv = [fv, sma_g, smv_g];
end

fv = double(fv);
end


function r = corrcoef_manual(x,y)
x = x(:);
y = y(:);
```

```
valid = ~isnan(x) & ~isnan(y);
x = x(valid);
y = y(valid);

if length(x) < 2
    r = 0;
    return;
end

mx = mean(x);
my = mean(y);

dx = x - mx;
dy = y - my;

num = sum(dx.*dy);
den = sqrt(sum(dx.^2)*sum(dy.^2));

if den == 0
    r = 0;
else
    r = num/den;
end
end
```