Using a Metric Learning Approach for few-shot, Free-text Continuous Authentication with a Physical Keyboard.

Adam Wherrett - 33657085

A dissertation submitted in partial fulfilment of the requirements of Leeds Beckett University for the degree of BSc (Hons) Cyber Security and Digital Forensics.

April 2026

Abstract

This dissertation explores the viability of metric learning for free-text continuous authentication, specifically addressing the challenges of high enrolment data requirements and behavioural drift. Whilst most keystroke dynamics research uses small datasets, typically with 50-200 participants, this research uses the Aalto dataset with approximately 136 million keystrokes from over 168,000 participants. This scale allows for a more diverse user population to be represented in the initial training phase.

The produced two-layer LSTM model transforms raw keystroke timestamps into 128-dimensional embeddings. Results demonstrate that the model successfully learned to discriminate between users from keystroke data, achieving an ROC AUC of 0.755. However, the EER of 31.35% shows a significant performance gap against models in other literature.

This performance gap provides insights for possible future work and improvements, such as using more complex keystroke features (e.g, digraphs and trigraphs) and using semi-hard mining to produce more informative negatives. Although the metric learning approach shows potential for a KD CA system, the implemented approach still has barriers to applicability.

Student's Declaration

I understand that to use the work and ideas of others, including generative AI output, without full acknowledgement, is academic unfair practice.
I confirm that this coursework submission is all my own, original work and that all sources, summaries, paraphrases and quotes are fully referenced as required by the LBU Academic Regulations.
DECLARATION OF GENERATIVE AI USE:
I DID use Generative AI technology in the development, writing, or editing of this assignment. I have included a breakdown of what tools where used and how in Appendix E.

Acknowledgements

I am deeply grateful to my supervisor, Tom Shaw, for his invaluable guidance and for being so accommodating of my requests and queries throughout the research process. I would also like to thank Oleg Illiashenko for his additional support and oversight.

My thanks also go to my family and friends. Your constant support kept me motivated and sane during the writing of this dissertation. Thank you for always being there.

Contents

List of Figures

List of Tables

List of Listings

1 Introduction

1.1 Background

Authentication methods have grown increasingly sophisticated in recent years. The widespread adoption of multi-factor authentication, hardware security keys, and physiological biometrics - such as fingerprint and facial recognition - has strengthened digital systems. However, the attacks on these systems have evolved in parallel. Phishing, credential stuffing, session hijacking, and physical access to unlocked devices are all made easier due to a fundamental weakness in how authentication systems are designed: most systems only authenticate a session on initial sign-in. For the sake of convenience, many systems then persist that session for days, weeks, or even months without re-verifying the user's identity. Typically, there are no mechanisms to detect whether an illegitimate user is active on an authenticated session (Ali and Hassan, 2025; Stylios et al., 2022).

Continuous Authentication (CA) proposes a solution to this weakness by repeatedly verifying a user's identity throughout an active session without disturbance to the user. Rather than prompting the user to re-authenticate, CA systems passively monitor some aspect of a user's behaviour. Keystroke Dynamics (KD), the analysis of an individual's typing rhythm, has the potential to be a very practical biometric for this purpose. It uses unique patterns in the way a user types to distinguish users with no specialised hardware beyond a standard keyboard (Lu et al., 2020; Monrose and Rubin, 2000). There are two main types of KD: fixed-text, which analyses the typing of a predetermined passphrase, and free-text, which analyses the typing patterns during natural typing Activity. In the case of CA, free-text KD is more applicable because the system can operate passively without interrupting the user (Mondal and Bours, 2017).

1.2 Problem Statement

Despite all the progress in the field, there are two crucial barriers stopping adoption in real-world systems.

First, current implementations require large volumes of enrolment data. To construct a reliable user profile, existing approaches typically require hundreds of keystroke samples for each new user (Killourhy and Maxion, 2009). For KD to be applicable in a real-world environment, users expect to enrol quickly and begin using the system with minimal setup (Mahalakshmi et al., 2024).

Second, typing patterns are naturally variable. Factors such as fatigue, stress, changes in posture and switching between keyboards cause gradual drift in how a user types - this is referred to as behavioural drift (Monrose and Rubin, 2000). Systems that don't adapt to this drift are subject to being less reliable over time.

These two crucial limitations - the requirement for a large volume of enrolment data, and behavioural drift - remain unsolved in combination for free-text physical-keyboard authentication.

1.3 Aims and objectives

This project aims to investigate whether a metric learning approach can improve the practicality of free-text keystroke authentication, particularly in regard to reducing the required volumes of enrolment data and handling behavioural drift. Metric learning is an Machine Learning (ML) approach that learns to measure similarity between data points rather than classifying them by label (Bellet et al., 2015). This means that the system maps Keystroke sequences into numerical representations called embeddings, such that the sequences from the same user cluster together and sequences from different users are separated.
The objectives are:

1. To design and develop a metric learning model on free-text keystroke data.
2. To evaluate the model's performance and identify the key challenges affecting its accuracy.
3. To compare the model's performance against results from relevant research.

1.4 Research Questions

This project addresses the following research questions:

1. Can a metric learning approach achieve a competitive Equal Error Rate (EER) for free-text keystroke authentication with fewer than 50 enrolment samples per new user?
2. To what extent can a metric learning approach handle behavioural drift in keystroke patterns?
3. How does the proposed metric learning model compare to deep learning models in current research?

1.5 Significance and scope

Whilst previous studies look primarily at Keystroke-based CA and few-shot learning separately, there is a lack of research covering both areas for free-text, physical-keyboard scenarios. Existing approaches require hundreds of samples per user, which is not feasible for practical deployment (Mahalakshmi et al., 2024). If authentication systems cannot be enrolled quickly, they will not be adopted, regardless of their accuracy. This project investigates this barrier by training a KD model on a large-scale free-text dataset, investigating whether metric learning can address both few-shot enrolment and behavioural drift. By evaluating the strengths and limitations of this approach, this work contributes to understanding whether few-shot techniques can make free-text keystroke authentication more viable for real-world deployment.
This work also contributes to understanding how metric learning, extensively studied in computer vision (Koch et al., 2015; Snell et al., 2017; Vinyals et al., 2016), transfers to sequential biometric data. Results are compared against recent KD research that use alternate deep learning approaches, to place the

metric learning method within the KD research field.
This project is scoped to physical keyboards using free-text keystroke data from the Aalto ~136M dataset (Dhakal et al., 2018). It does not utilise alternate biometrics such as mouse dynamics, touch gestures or gait analysis to make the system more accurate (S. and K., 2025; Stylios et al., 2023; Stylios et al., 2022). The system is a research prototype to evaluate the feasibility of a metric learning approach, and does not address real-time deployment, inference latency, or regulatory compliance.

1.6 Structure of Dissertation

The following section outlines the structure for the remainder of the dissertation. Chapter 2 reviews the literature on KD, from foundational research to modern deep learning and few-shot studies. Chapter 3 evaluates the technologies considered during development, justifying key design decisions. Chapter 4 covers the methodology and system design, describing the architecture of the metric learning model and the training and evaluation pipelines. Chapter 5 looks at the testing and implementation of the system, including data preprocessing, model training, and evaluation procedures. Chapter 6 presents the product evaluation, analysing the model's performance against the stated aims and comparing results with existing research. Chapter 7 reflects on the project as a whole, evaluating the development process, challenges encountered, and lessons learned. Chapter 8 summarises the findings, discusses limitations, and suggests directions for future work. The bibliography and appendices follow, providing references and supplementary material.

2 Literature Review

2.1 Introduction

User authentication is a critical part of securing access to systems and services. Traditional authentication systems, such as passwords, PINs, and physical biometrics, focus on securing just the entry point into an account, leaving systems vulnerable to session hijacking attacks that occur after the user is initially authenticated (Ali and Hassan, 2025; Stylios et al., 2022). Once compromised, an attacker can potentially steal sensitive data and perform unauthorised actions without detection.
CA systems solve this by continuously monitoring a user's behaviours throughout an active session, to keep the user authenticated with minimal interruptions to the user's experience (Stylios et al., 2022). Among behavioural biometrics, KD has shown to be particularly promising due to its unintrusive nature and no requirement for specialised devices (Lu et al., 2020; Monrose and Rubin, 2000). KD refers to looking at a person's unique typing patterns, characterised by timing features such as dwell time (key press duration) and flight time (time between key releases) (Lu et al., 2020). These typing patterns are difficult to

imitate and therefore serve as reliable behavioural biometrics (Monrose and Rubin, 2000).
This Review looks at the evolution of KD research, from foundational studies through to advanced deep learning and few-shot enrolment approaches. It identifies two crucial challenges that restrict the practicality of widespread KD usage: requirement for large volumes of enrolment data, and behavioural drift over time.

2.2 Keystroke Dynamics: Foundations and Evolution

2.2.1 Early Research and Fixed-Text Approaches

The thought of using a user's unique typing pattern for authentication has a long history. Gaines et al. (1980) conducted early experiments analysing the features, typing patterns and finding consistent, measurable differences. This research established the beginning of KD research, defining the core temporal features.
Fixed-text authentication, where users type predefined pass-phrases, dominated early research until Monrose and Rubin (2000) demonstrated that patterns in free-text typing could verify a user's identity to a reasonable accuracy. Killourhy and Maxion (2009) changed the field again by conducting a systematic evaluation of 14 anomaly-detection systems using a dataset of 51 subjects, establishing a strong benchmark for future research in the field. Four years later, Teh et al. (2013) produced a comprehensive survey of past research into KD and concluded that "the advantage of [KD] is indisputable such as the ability to operate in stealth mode, low implementation cost, high user acceptance, and ease of integration to existing security systems".

2.2.2 Limitations of Traditional Approaches

Traditional approaches suffer from three fundamental limitations. First, they require large volumes of enrolment data to construct reliable user profiles. Killourhy and Maxion (2009) required each of their subjects to provide 400 keystrokes during enrolment, which is impractical and unrealistic to expect of a real user. Teh et al. (2013) also noted this as a barrier to practical adoption.
Second, traditional methods experience degraded performance over time due to behavioural drift - the natural evolution of a user's typing patterns caused by external factors like fatigue, stress and environment changes. Initial research, such as that conducted by Monrose and Rubin (2000) highlighted that typing patterns are naturally variable.
A further limitation of traditional approaches is that keystroke features have to be hand-crafted. This means they used processes that extracted fixed timing features, for example, dwell time and flight time, and performed a statistical analysis. whilst establishing their benchmark Killourhy and Maxion (2009) reported that the best performing KD systems at the time could only achieve EER around 10%, this suggests a ceiling for the capabilities of hand-crafted

features.

2.3 Deep Learning in Keystroke Dynamics

2.3.1 The shift to Neural Approaches

The introduction of deep learning into the KD field has transformed the landscape for new research. Lu et al. (2020) pioneered a system combining a Convolutional Neural Network (CNN) and Recurrent Neural Network (RNN) for free-text CA, achieving impressive results with EER ranging from 2.67% to 5.9% across two benchmark datasets. Their approach split keystroke data into fixed-length sequences and converted them into vector representations based on temporal characteristics. With this architecture, the CNN could extract n-gram sequences of a keystroke sequence, and the RNN is trained to identify a user's behavioural patterns from the n-grams. This design directly addresses the issues with hand-crafted features by transforming keystroke sequences into feature sequences.
Later work has explored alternative neural architecture. Stylios et al. (2023) compared Multi-Layer Perceptron (MLP) and Long Short-Term Memory (LSTM) networks for continuous KD and touch gestures, finding that MLP outperforms LSTM in this context. This challenged the claim from Lu et al. (2020) by suggesting that simpler feedforward networks can be accurate when combined with feature-level fusion - using multiple behavioural biometrics to curate features. More recently, Ali and Hassan (2025) demonstrated that an LSTM-CNN hybrid model is capable of "superior accuracy in identifying legitimate users" in comparison to ML algorithms tested in earlier years.

Despite the advantages, current approaches still have significant drawbacks. The most crucial of those is that current algorithms don't yet solve the intrusive requirement for large volumes of enrolment data. Gathering "a large amount of training data every time the model needs to be reconfigured" is not practical (Mahalakshmi et al., 2024).

2.4 Few-shot and Metric Learning

2.4.1 The Few-shot Challenge

For real-world applications, users expect to be able to use systems after a short enrolment process, yet conventional deep learning models are not well-suited to learning from small data samples. Mahalakshmi et al. (2024) recognised that models confronted with unfamiliar user data struggle to recognise expected behaviours accurately. In KD, this translates to the need for systems that can generalise to new users with minimal typing data.

2.4.2 Metric Learning as a Solution

Metric learning proposes a solution to the few-shot problem. Rather than learning explicit data features, which requires substantial data per class, metric learning learns the embedding space where similar inputs are pushed closer together, and dissimilar inputs are pushed apart, (Bellet et al., 2015) emphasised that metric learning techniques can "automatically learn similarities and distance functions from data" and generalise to new classes with limited samples. After the initial model is trained, new users can be enrolled by comparing their embeddings to stored references without retraining the full model.
Koch et al. (2015) suggested the idea of one-shot recognition, by proposing Siamese neural networks - using paired networks with shared weights to learn a similarity function. Later Snell et al. (2017), suggested prototypical networks which classified data by "computing distances to prototype representations of each class". Around a similar time Model-Agnostic Meta-Learning (MAML) was introduced by Finn et al. (2017), which shifted the focus from distance-based learning to an optimisation-based approach, essentially making a highly fine-tuneable model. this is valuable for CA where models must adapt quickly to each new user.

2.5 Research Gaps and Opportunities

The literature reveals there are some critical gaps that motivate present research in the field:
Future enrolment in free text KD is underexplored. so far metric learning has achieved impressive results in computer vision, however, its application to free text CA has received limited attention. considerable amounts of enrolment data are still required for KD systems (Lu et al., 2020; Mahalakshmi et al., 2024).
Behavioural drift in metric-learnt algorithms is not well understood. There is a lack of research into how resistant a metric-learnt algorithm is to natural variation in data due to behavioural drift.

2.6 Review Summary

This review has explored the keystroke dynamic research field from early empirical observations through to deep learning and few-shot learning approaches. EERs have improved drastically from approximately 10% with traditional statistical analysis to below 3% with Deep learning. The switch to free text authentication has lifted a significant practical constraint. Yet the requirement for large volumes of enrollment data, along with the complexities of handling behavioural drift, remains a barrier to real-world KD applications. The most promising solutions - metric learning for few-shot adaptation and training on datasets that naturally contain behavioural drift - have not yet been fully explored in combination for physical-keyboard-based CA.

3 Review of Technologies

3.1 Introduction

This chapter evaluates the specific tools and technologies required for the development of a metric-learnt, few-shot KD model. Each section discusses potential alternatives, presents findings from any prototypes conducted, and justifies the final selection of technologies with reference to the project's requirements: investigating whether metric learning can reduce enrolment data requirements to fewer than 50 samples per user, and training on a dataset that includes behavioural drift, so that the model learns to generalise across natural variation in typing patterns.

3.2 Programming Language

Python was selected for the development of this project. Python currently dominates the ML landscape, offering many libraries for data manipulation, model training, and evaluation. This has led to Python having extensive community support and documentation, particularly for ML use cases. This is also reflected in KD literature, where the majority of recent deep learning approaches use Python frameworks (Ali and Hassan, 2025; Lu et al., 2020; Teh et al., 2013). The main alternatives considered were R and Java. Whilst R has strong use cases for statistical and data analysis, it lacks depth in its deep learning frameworks compared to Python. It has a Keras and TensorFlow wrapper, which are limited and poorly maintained. Java also has libraries for ML such as Deeplearning4j, but overall, its ML support is less mature than that of python. Java has fewer pre-built utilities for data pre-processing and model evaluation.

3.3 Deep Learning Framework

3.3.1 Framework selection

TensorFlow (v2.21.0) with Keras (v3.13.2) was selected as the deep learning framework. Keras has been used widely within KD research: Lu et al. (2020) used it to implement their CNN-RNN free-text model, Stylios et al. (2023) used it to compare MLP and LSTM approaches for keystroke-based CA, and Ali and Hassan (2025) used it for their LSTM-CNN hybrid approach. This made Keras a natural choice for this project, as it simplifies model definition whilst retaining the ability to implement custom loss functions and training loops.
The Primary alternative considered was PyTorch. PyTorch is widely used in the greater ML landscape due to its dynamic computation graph, which allows for easier debugging and experimentation. However, Keras's sequential model definition lends itself to the proposed model architecture for this research.
For this project, TensorFlow was selected for two main factors: the simplicity of model definition with Keras, and native HDF5 (Hierarchical Data Format version 5) model serialisation through the `.keras` format.

3.3.2 Custom Components

The project required two custom components beyond the standard Keras layers: a triplet loss function and a triplet data generator. The triplet loss was implemented as a custom Keras loss, computing squared Euclidean distance between anchor, positive, and negative embeddings with a configurable margin. The `TripletDataGenerator` class extends `keras.utils.Sequence` to dynamically generate triplets at each epoch, supporting random, semi-hard, and hard negative mining strategies. This design was required because standard Keras loss functions operate on label pairs, whilst triplet loss requires three coordinated inputs.

3.4 Dataset Selection

The Aalto dataset, produced by Dhakal et al. (2018), was selected as the primary data source. It contains approximately 136 million keystrokes from 168,000 participants, making it the largest publicly available KD dataset. Each record captures key press and release timestamps for physical keyboard input during copy-typing sessions where participants copied presented sentences rather than writing their own text.

There were three main reasons why this dataset was selected. First, the data set is large enough to support training a metric learning model that can generalise across a large and diverse user population. Second, the variable content typing sessions approximate the free text CA scenario. Although the data is from transcription rather than composition, the raw timestamps the data contains can still derive keystroke features such as hold time and flight time that reflect genuine individual typing pattern regardless of who wrote the initial content. Third, with each user typing multiple sentences, the dataset provides a limited test of whether learned embeddings generalise across intra-user variation - though the sessions are short and collected in a single sitting, lacking the long-term variation that real world deployment would introduce.

No publicly available data set perfectly satisfies all requirements of this research. A truly free text dataset, capturing natural composition over extended periods with long temporal separation between sessions, does not exist at the scale required for training a metric learning model. The Aalto dataset represents the best available compromise: it provides physical keyboard typing at a scale no other dataset offers, even though users copied text rather than composing freely and sessions were not collected over extended periods.

The main alternative considered was the *CMU* KD Benchmark Data Set (Killourhy and Maxion, 2009). It has been widely used in KD research and provides a standardised benchmark. However, it contains data from only 51 subjects typing a fixed password with 400 repetitions per subject. This presents two significant limitations. First, the fixed text paradigm does not align with the variable content CA scenario. Second, the small number of subjects is insufficient for training a metric learning model that must generalise to new users. No other publicly available data set satisfies all project requirements.

3.5 Preprocessing and evaluation

The preprocessing pipeline uses Pandas and NumPy to transform keystroke timestamps into temporal features per keystroke. There are four features that could be produced from the keystroke data: hold time (key press duration), flight time (time between key release and next press), digraph latency (time between consecutive key presses), and trigraph latency (time across three key sequences). Hold time and flight time are widely used temporal features, especially in more traditional works, and formed the feature set used in the model's evaluation (Monrose and Rubin, 2000). Digraph and trigraph latencies capture longer range n-gram timing patterns, which have been shown to be more discriminative in free-text analysis and are proposed as key improvements for future work (Gunetti and Picardi, 2005). Session-level z-score normalisation is applied using scikit-learn's `StandardScaler` before windowing. A sliding window of 20 keystrokes with a stride of 5, then segments the feature sequences into a fixed-length input sequence.

The preprocessed data set is then stored in HDF5 format via the `h5py` library chosen over CSV and NumPy's `.Npy` format for its support of efficient chunk reading, which is important for dynamic triplet generation where data is loaded in batches rather than all at once. The data is split at the session level using scikit-learn's `train_test_split` with an 80:20 ratio stratified by user. Splitting on a session-level prevents data leakage by ensuring samples from the same session cannot appear in both the training and the test set, this follows the evaluation protocols emphasised by Killourhy and Maxion (2009).

Evaluation uses EER, False Acceptance Rate (FAR), False Rejection Rate (FRR), ROC AUC and d-prime, following the recommendation by Sugrim et al. (2019) to report multiple complementary metrics alongside distance distribution statistics rather than relying on a single summary measure.

3.6 Model Architecture

The model is a two-layer LSTM network producing 128-dimensional embeddings, trained with triplet loss. LSTM networks have been seen across multiple instances of KD research due to their ability to capture temporal dependencies in sequential typing data (Lu et al., 2020; Stylios et al., 2023). The trained model uses random mining, which selects negatives without considering difficulty and produces many uninformative triplets that contribute zero loss. Hard mining always selects the closest negative, providing the strongest learning signal but risking training instability. Semi-hard mining selects negatives within the margin boundary, difficult enough to be informative but not so difficult as to destabilise training, and has been shown to improve training effectiveness compared to random mining (Lu et al., 2020); this is proposed as an improvement for future works. Random mining was chosen because semi-hard mining would be computationally expensive at the scale of the chosen dataset.

3.7 Model Training

The model was trained using 128 LSTM units per layer, 0.5 dropout, a triplet margin of 1.0, random negative mining, and two features per keystroke. The configuration was deliberately simple, as the Aalto dataset is much larger than typically seen in other KD research, where typically 50-200 subjects are evaluated. In addition, training on 3.2 million samples is computationally expensive, with the project timeline allowing for only one full training run.

3.8 Review Summary

This chapter has covered the technology selected for the development of the few-shot keystroke authentication system. Python with TensorFlow and Keras provides a strong foundation for model development, with custom components for triplet-based metric learning, including a triplet loss function and a triplet data generator, supporting multiple negative mining strategies. The Aalto dataset provides the scale and transcription typing format required for this research, whilst acknowledging that no publicly available dataset perfectly matches the free-text authentication scenario. The preprocessing pipeline transforms raw keystrokes into normalised hold time and flight time feature sequences, segmented via a sliding window and stored efficiently in HDF5 format. The model architecture uses a two-layer LSTM network producing 128-dimensional embeddings, trained with triplet loss using random negative mining. A deliberately simple training configuration was adopted to confirm the fundamental pipeline functions correctly at the scale of 3.2 million samples. Evaluation is conducted using multiple metrics (EER, FAR, FRR, ROC AUC, and d-prime) to provide a robust assessment of authentication performance.

4 Methodology and Design

4.1 Introduction

This chapter presents the methodologies directing the development process and the technical design of the proposed few-shot keystroke authentication system. It explains the selection of Agile and *SDLC* frameworks, outlines key design choices, and describes the evaluation methods.

4.2 Product Development Methodology

4.2.1 Overview

The project used Agile (via GitHub projects) for iterative development and *SDLC* principles for control and reproducibility.

4.2.2 Agile Development via GitHub projects

A Kanban Agile approach emphasised continuous flow, with tasks pulled from a backlog as capacity permitted, and work-in-progress limits to prevent bottlenecks, using GitHub Projects for task tracking and prioritisation.
Agile was selected due to the project's uncertain requirements, allowing for incremental development and regular reassessment. GitHub Projects ensured transparency and progress tracking.
The work was structured into milestones as follows:

1. **M1 — Foundation & Literature Review**: Configure the development environment; conduct literature reviews on continuous authentication and few-shot learning; finalise project scope and research questions; select and acquire the keystroke dynamics dataset.
2. **M2 — Baseline Model Implementation & Evaluation**: Implement the data loading and preprocessing pipeline; build and train a baseline model; run baseline experiments and log results; draft the baseline methodology and results chapters.
3. **M3 — Metric Learning for Few-Shot Enrolment**: Select and finalise the metric learning algorithm; adapt the baseline model for metric learning with triplet loss; run training experiments and comparative evaluation; draft the metric learning methodology and results chapters.
4. **M4 — Final Write-up and Defense Preparation**: Write the introduction, discussion, and conclusion chapters; complete a full proofread and edit of the dissertation; prepare the defense presentation.

A reflection on how well this plan was followed is in Chapter 7.

4.2.3 System Development Life Cycle Integration

Agile set the pace of development, while *SDLC* principles were applied to ensure that the work was controlled, versioned, and reproducible. Specifically, the *SDLC* framework structured the following aspects of the project:

- **Requirements**: captures Github issues - defining the scope, data, architecture, and evaluation criteria for the system.
- **Design**: the system architecture, metric learning approach, and evaluation protocol was all specified before implementation began.
- **Implementation**: all development was carried out within Git with all code changes tracked.
- **Testing**: Each pipeline component (preprocessing, model training, and evaluation) was tested independently before integration
- **Evaluation**: Conducted using the metrics defined during design, ensuring results were comparable and interpretable.

The four GitHub milestones (M1–M4) structured the *SDLC* process, with issues within each milestone breaking the work into trackable sections. A reflection is provided in Chapter 7.

4.2.4 Alternatives considered

4.2.4.1 Waterfall Waterfall methodologies follow a strict, sequential process that requires a full specification before design begins, and testing only occurs after implementation is finished.
Waterfall was rejected for this project as it does not align with the project's nature. As the research investigates whether a particular approach works, requirements, such as optimal model architecture, loss function configuration, and evaluation strategy, cannot be known in advance. In addition, the project's computational overhead meant that a single oversight late in the process would leave no time for correction.

4.2.4.2 Scrum Scrum is an Agile methodology that organises work into timeboxed iterations called sprints.
Scrum was considered but rejected in favour of Kanban for two reasons. First, the project's research nature meant that tasks varied widely in complexity and uncertainty, making following sprints difficult. Kanban's continuous-flow model allowed tasks to be pulled as capacity permitted, accommodating this without the overhead of sprint estimation and planning. Second, the project required flexibility to re-prioritise tasks based on tests. Kanban enabled immediate priority changes, whereas Scrum's sprint commitments would have locked in a fixed scope for each iteration, reducing the ability to adapt to new findings.

4.3 System Overview

The proposed system comprises an offline training phase and an online authentication phase.
In the offline training phase, a neural network is trained on keystroke data that represents a large population. The objective is not to identify specific individuals, but to learn a general-purpose metric space in which samples from the same user map closer together and those from different users map farther apart. This phase requires a large volume of data and substantial computational resources, but is performed only once. After training, the embedding model is then frozen for deployment without modification.
In the online authentication phase, the frozen model enables enrollment and verification. During enrollment, a new user provides a small number of typing samples, processed by the embedding model to generate a biometric profile in the metric space. Because the model has already learned to discriminate between users generally, no retraining is required when adding a new user. During verification, typing samples are embedded and compared to the enrolled biometric profile using a distance metric. If the distance falls below a configured threshold, the user is authenticated. Otherwise, the user is flagged as an imposter. This two-phase design addresses the core research question by learning a general metric space rather than user-specific classifications. The system accommodates new users with minimal enrolment data and adapts naturally to the open-set nature of authentication (where unseen identities must be rejected

rather than classified as known users), where the set of potential impostors is unbounded and unknown at training time.

4.4 Design Approach

4.4.1 Metric Learning with Triplet Loss

Metric learning maps similar inputs closer together and separates dissimilar ones (Bellet et al., 2015).
Triplet loss trains the embedding model, where each triplet has three samples drawn from two users - an anchor and a positive from the same user, and a negative from a different user. The loss function encourages the anchor-positive distance to be smaller than the anchor-negative distance. During training, the model structures the embedding space to satisfy this constraint across the data population, producing a space that can generalise to unseen users.
The metric space also provides a mechanism for handling behavioural drift: as typing patterns evolve, the enrolled profile can be updated with newer samples without modifying the model itself.
An LSTM network serves as the embedding function because keystroke data is inherently sequential. Recurrent models have demonstrated strong performance for capturing temporal dependencies in keystroke authentication tasks (Lu et al., 2020).

4.4.2 Alternative Approaches Considered

4.4.2.1 Classification-Based Approaches Classification-based methods treat authentication as a closed-set recognition problem, training a model to predict user identity from a fixed set of classes. In keystroke dynamics, this typically involves training a neural network with a softmax output layer, where each output neuron corresponds to a known user.
Classification-based approaches were rejected for this project as adding a new user requires either retraining the entire model with additional output classes or training a separate binary classifier per user, both of which are impractical for real-time deployment. The number of output classes is fixed at training time, making it impossible to enrol new users without modifying the model architecture. Furthermore, softmax classifiers optimise for absolute class membership rather than relative similarity; they do not naturally produce a distance metric that can be thresholded for authentication decisions.

4.4.2.2 Contrastive loss contrastive loss operates on pairs rather than triplets, minimising the distance between similar pairs whilst pushing dissimilar pairs beyond a defined margin (Hadsell et al., 2006).
contrastive loss was rejected in favour of triplet loss for two reasons. First, contrastive loss requires an absolute margin for negative pairs. If an inappropriate value was selected, it can lead to collapse or poorly separated embeddings.

Contrastive loss processes pairs independently, which can lead to less efficient optimisation compared to triplet loss.

4.4.2.3 Proxy-Based Approaches Proxy-based approaches, such as proxy-NCA (Movshovitz-Attias et al., 2017), replace the pairwise odd triplet comparisons with learned proxy vectors that represent cluster centers in the embedding space.
Proxy-based approaches were considered for their computational efficiency as they avoid the $O(n^3)$ cost of triplet mining computation by comparing samples against a fixed set of proxies ($O(nk)$). However, it was rejected because they assumed a fixed, known set of classes at training time. Each proxy corresponds to a known class, and the proxy count must be specified in advance. This would not work for the authentication use case as it conflicts with the open-set requirement for authentication systems.

4.4.3 Summary of Design Decisions

The decision to use Metric Learning with a triplet loss function implemented via an LSTM embedding network was driven by two core requirements: few-shot enrolment without model retraining and open-set recognition for unrecognised users. Table 1 summarises the comparison

Table 1: Comparison of design approaches against the project's core requirements for few-shot keystroke authentication.

Approach	Few-Shot Enrolment	Open-Set Recognition	Margin Type	Verdict
Classification	✓ — requires retraining	× — fixed classes	N/A	Rejected
Contrastive Loss	✓	✓	× - absolute	Rejected
Proxy-NCA	× — fixed proxies	× — fixed proxies	N/A	Rejected
Triplet Loss	✓	✓	✓ - relative	**Selected**

4.5 Evaluation Methodology

System performance was evaluated by generating embeddings from the test data split and calculating pairwise Euclidean distances between them. These distances form genuine distance (where both samples originate from the same user) and an impostor distance (where samples originate from different users) distributions. A receiver operating characteristic (ROC) curve is then constructed by varying the distance threshold across its full range.
Several standardised metrics are used to evaluate the model's performance. The EER identifies the threshold at which the FAR equals the FRR, providing a

summary statistic that balances both error types. However, because the EER reflects the model's performance at only one operating point, it cannot capture the model's true applicability. To characterise the model's performance across all possible thresholds, the area under the receiver operating characteristic curve (ROC AUC) is also reported. Both of these metrics depend on a threshold being applied. To complement them, d-prime is also calculated. D-prime is the standardised difference between the means of the genuine and impostor distance distributions. Unlike the EER and ROC AUC, d-prime measures separability directly in the embedding space, independent of any decision threshold.
This multi-metric strategy follows the approach recommended by Sugrim et al. (2019), which states that no single metric is sufficient to characterise authentication performance.

4.5.1 Few-Shot Enrolment Evaluation

To assess performance for few-shot enrolment, a second evaluation phase was planned. This would simulate the model's online phase described in section 4.3. For this, a separate test dataset would be processed; each test user would be enrolled with n initial samples, and subsequent samples would be verified against the created biometric profile in the metric space. By varying n across a range of shot counts, the relationship between volume of enrolment data and authentication accuracy could be characterised.
This evaluation was dependent on the baseline model achieving sufficient discriminative performance. As discussed in Chapter 6, this threshold was not met, and the evaluation was deferred.

4.6 Summary

In summary, the methodology combines Agile and SDLC frameworks for project management, a two-phase system design for few-shot authentication, and evaluation using multiple performance metrics. Design choices were guided by the need for adaptability and open-set recognition.

5 Testing and Implementation

5.1 Introduction

This chapter summarises the implementation and testing of the keystroke authentication system, focusing on development, training, and evaluation. Full project source code is submitted separately. Key steps and logic are described, with code snippets referenced for illustration. Each listing starts with a comment, noting the source file path and line numbers for reference.

5.2 Data Preprocessing Implementation

5.2.1 Overview

The raw Aalto data set is distributed as a collection of tab separated value (TSV) files, each containing keystroke data for a single participant across their multiple typing sessions (Dhakal et al., 2018). The preprocessing pipeline transforms these raw keystrokes into normalised, windowed feature sequences through three stages, as illustrated in Figure 1.

The key configuration parameters in the preprocessing pipeline are summarised in Table 2.

Table 2: Preprocessing pipeline configuration parameters.

Parameter	Value	Description
Window size	20 keystrokes	Number of consecutive keystrokes per input sample
Stride	5 keystrokes	Step size between consecutive windows
Split ratio	80:20	Train/test split at session level
Chunk size	100 sessions	Number of sessions processed per write operation
Features	Hold time, flight time	Two temporal features per keystroke
Normalisation	Z-score (per session)	Zero mean, unit variance within each session
Output format	HDF5 (chunked, resizable)	Efficient storage for incremental writes

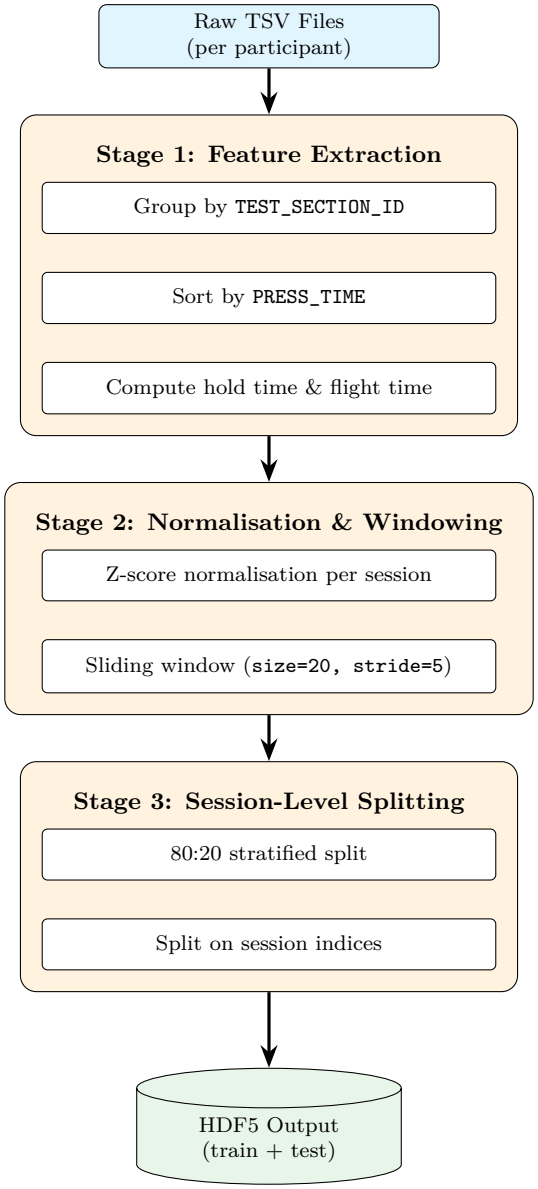

Figure 1: Three-stage preprocessing pipeline for raw keystroke data.

5.2.2 Stage 1: Feature Extraction

Each TSV file is read, and keystroke events are grouped by the `TEST_SECTION_ID` column. This allows each unique sentence transcribed by the user to be seen as a separate session. Events are sorted chronologically by press time. The events are then processed to extract hold time (release time minus press time) and flight time (next key's press time minus current key's release time) features (Monrose and Rubin, 2000). As flight time requires another keystroke to follow, the final keystroke in each session excludes flight time.

The feature extraction logic for a single session is shown in Listing 1.

```
# few-shot-keystroke-auth/code/Scripts/preprocessing/aalto/preprocessing.py
    → L105-119
session_group = session_group.sort_values(by=["PRESS_TIME"])

if len(session_group) == 0:
    continue

session_group["HOLD_TIME"] = (
    session_group["RELEASE_TIME"] - session_group["PRESS_TIME"]
)

session_group["NEXT_PRESS_TIME"] = session_group["PRESS_TIME"].shift(-1)
session_group["FLIGHT_TIME"] = (
    session_group["NEXT_PRESS_TIME"] - session_group["RELEASE_TIME"]
)
session_group = session_group.dropna(subset=["FLIGHT_TIME"])
session_features = session_group[["HOLD_TIME", "FLIGHT_TIME"]].values
```

Listing 1: Feature extraction logic.

5.2.3 Stage 2: Normalisation and Windowing

Session-level z-score normalisation is applied using scikit-learn's `StandardScaler`, giving each feature a zero mean and unit variance within a session. This normalises the difference in overall typing speed between sessions whilst preserving variation within sessions. A sliding window of 20 key strokes, with a stride of 5, splits the continuous sequence into fixed-length input samples, each labelled with its respective participant ID. Sessions shorter than the window size produced no output. The implementation of this windowing and normalisation process is shown in Listing 2.

```python
# few-shot-keystroke-auth/code/Scripts/preprocessing/aalto/preprocessing.py L26 -56
def create_sliding_windows(
    session_features: np.ndarray,
    user_label: int,
    window_size: int,
    stride: int,
) -> Tuple[List[np.ndarray], List[int]]:
    windows = []
    labels = []

    if len(session_features) < window_size:
        return windows, labels

    scaler = StandardScaler()
    if np.std(session_features, axis=0).any() == 0:
        normalised_features = session_features - np.mean(session_features, axis=0)
    else:
        normalised_features = scaler.fit_transform(session_features)

    for i in range(0, len(normalised_features) - window_size + 1, stride):
        windows.append(normalised_features[i : i + window_size])
        labels.append(user_label)

    return windows, labels
```

Listing 2: Sliding window function.

5.2.4 Stage 3: Session-Level Splitting

Each preprocessing window is split into a training and test sets at the session level, using an 80:20 ratio, stratified by user. Session-level splitting prevents data leakage by ensuring that no session has windows across both splits, following the evaluation protocols recommended by Killourhy and Maxion (2009).

5.2.5 Output Storage

The fully preprocessed output is stored in HDF5 format. The preprocessing pipeline writes data incrementally in chunks of 100 sessions to help manage memory. This produces two output files, one for training and one for testing. Figure 2 illustrates the internal structure shared by both files.

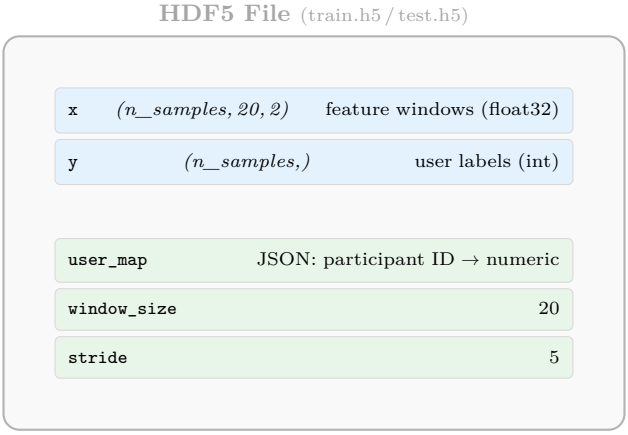

Figure 2: Internal structure of the HDF5 output files.

The complete preprocessing produced roughly 3.2 million windows from 168,608 users.

5.3 Model Implementation

5.3.1 Architecture

Figure 3 breaks down the embedding model architecture. After the preprocessing pipeline, keystroke windows have a shape of *(20, 2)* - 20 time-steps, each with two features. These windows are then processed through two LSTM layers, each with 128 units (a standard capacity for sequence-modeling). Stacking LSTMs allows the first layer to learn how low-level temporal patterns (e.g., relationship between adjacent keystrokes) and the second to build higher level representations on top of those (e.g., typing cadence over a whole window). In order for this to work, the first LSTM must expose its hidden state at every time-step so that the second receives a full sequence of processed features rather than a single summary matrix. This is achieved by setting `return_sequences=True`, which produces an output tensor of shape $(20, 128)$ — one 128-dimensional vector per time-step.

The second LSTM sets `return_sequences=False`. This means that rather than outputting a sequence of processed features, it only outputs the final hidden state. This results in a single fixed-size matrix of shape $(batch, 128)$ that summarises the whole typing window.

A dropout layer with a rate of 0.5 randomly disables half the connections during each training step, forcing the network to learn redundant representations that do not depend on any single neuron and reducing overfitting. A dense layer with a linear activation then projects the output into the final 128-dimensional embedding space. The implementation of this architecture is shown in Listing 3.

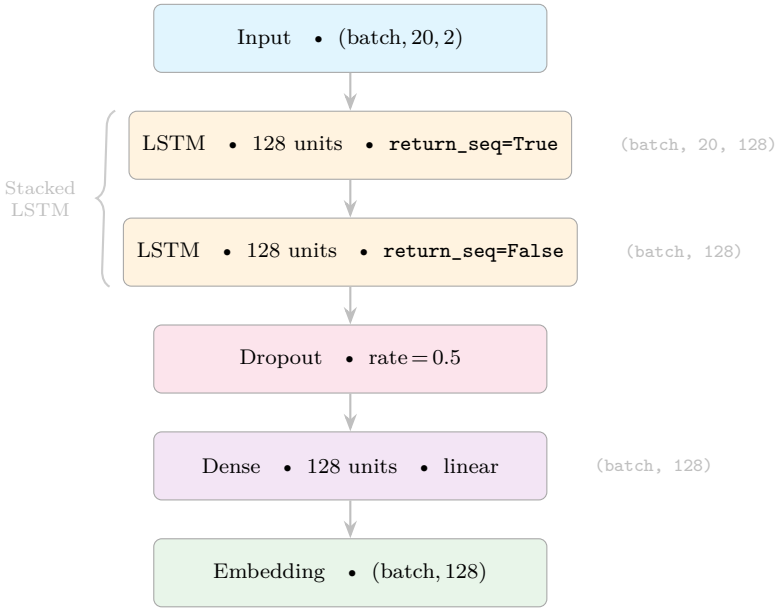

Figure 3: LSTM embedding model architecture.

```python
# few-shot-keystroke-auth/code/Scripts/train.py L235-262
def build_lstm_model(
    input_shape: tuple,
    num_classes: int,
    embedding_dim: int = 128,
    num_lstm_units: int = 128,
    dropout_rate: float = 0.5,
) -> keras.Model:
    model = keras.Sequential(
        [
            layers.Input(shape=input_shape),
            # First LSTM layer with return_sequences=True for stacking
            layers.LSTM(
                num_lstm_units,
                return_sequences=True,
                activation="tanh",
            ),
            # Second LSTM layer
            layers.LSTM(num_lstm_units, return_sequences=False, activation="
                ↪ tanh"),
            # Dropout for regularization
            layers.Dropout(dropout_rate),
            # Output embedding layer
            layers.Dense(embedding_dim, activation="linear"),
            # Cast to float32 to ensure consistent dtype
            layers.Lambda(lambda x: tf.cast(x, tf.float32)),
        ]
    )

    return model
```

Listing 3: LSTM embedding model builder.

5.3.2 Triplet Loss Function

As Keras does not provide a built-in triplet loss function, it was implemented as a custom Keras loss function. The function follows the pipeline shown in Figure 4. The model produces a matrix of embeddings shaped $(3 \times batch\_size, 128)$ - three samples per triplet (anchor, positive, negative), joined into a single tensor. The first step of the loss function is then to reshape this into $(batch\_size, 3, 128)$ so that each whole triplet is grouped. Then, the squared Euclidean distance is calculated between the anchor-positive ($d_{pos} = ||a - p||^2$) and the anchor-negative ($d_{neg} = \|a - n\|^2$). The loss for each triplet is $max(0, d\_pos - d\_neg + \alpha)$, where α is the margin (set to 1.0). This means only triplets where the negative distance is less than the positive distance plus the margin contribute to the loss. The output of the loss function is the mean loss of all triplets.

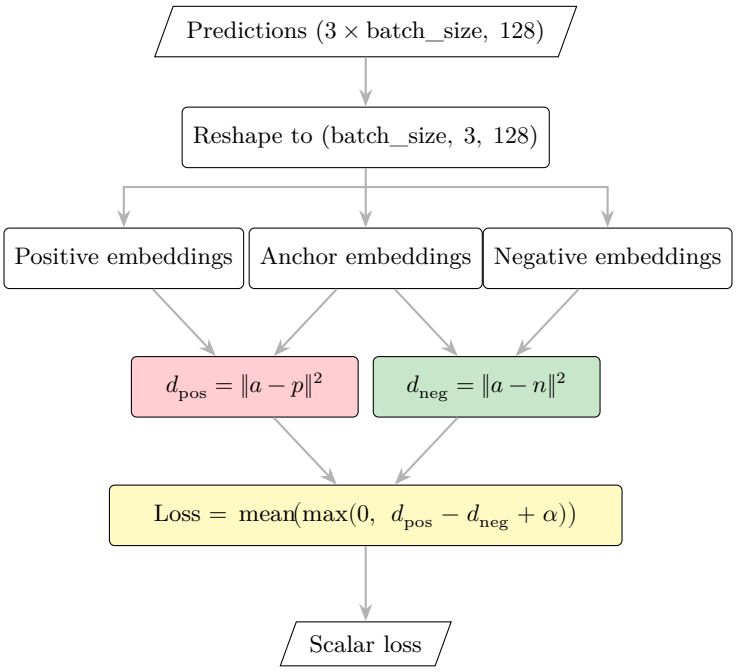

Figure 4: Triplet loss computation.

```python
# few-shot-keystroke-auth/code/Scripts/train.py L270-302
def create_triplet_loss(anchor_dim: int, margin: float = 1.0) -> Callable:
    """
    Creates a triplet loss function with online triplet mining.

    The model should output embeddings for (anchor, positive, negative) triplets.
    Input format: y_pred shape = (batch_size * 3, embedding_dim)
    where each group of 3 consecutive samples is (anchor, positive, negative).
    """

    def triplet_loss(y_true, y_pred):
        # Reshape predictions to (batch_size * 3, embedding_dim)
        y_pred = tf.cast(y_pred, tf.float32)
        batch_size = tf.shape(y_pred)[0] // 3

        # Reshape to (batch_size, 3, embedding_dim) to get triplets
        triplets = tf.reshape(y_pred, (batch_size, 3, anchor_dim))
        anchor = triplets[:, 0, :]  # (batch_size, embedding_dim)
        positive = triplets[:, 1, :]  # (batch_size, embedding_dim)
        negative = triplets[:, 2, :]  # (batch_size, embedding_dim)

        # Compute Euclidean distances
        # d(a, p) = ||a - p||_2
        pos_dist = tf.reduce_sum(tf.square(anchor - positive), axis=1)
        # d(a, n) = ||a - n||_2
        neg_dist = tf.reduce_sum(tf.square(anchor - negative), axis=1)

        # Triplet loss: max(d(a,p) - d(a,n) + margin, 0)
        # We want anchor-positive distance to be SMALLER than anchor-negative distance
        loss = tf.maximum(0.0, pos_dist - neg_dist + margin)

        return tf.reduce_mean(loss)

    return triplet_loss
```

Listing 4: Triplet loss function.

5.3.3 Triplet Data Generation

The other custom Keras component required for the training pipeline was a `TripletDataGenerator` class. By default, Keras data generators create input-label pairs for classification, but triplet loss requires dynamically assembled groups of three samples - an anchor, a positive from the same user, and a negative from a different user. The custom generator extends `keras.utils.Sequence` and constructs triplets at each epoch by maintaining an index mapping users to their samples, then randomly sampling anchor, positive, and negative indices. The generator outputs a matrix shaped $(3 \times batch\_size, 20, 2)$, where the first third of samples are anchors, the second third are positives, and the final third are negatives. The code for the custom extended class is shown in Listing 5.

```python
# few-shot-keystroke-auth/code/Scripts/train.py L156-183
    def __getitem__(self, index: int) -> tuple:
        """Get one batch of triplets."""
        start_idx = index * self.batch_size
        end_idx = min(start_idx + self.batch_size, len(self.triplet_indices))
        batch_indices = self.triplet_indices[start_idx:end_idx]

        # Build batch of triplets (anchor, positive, negative)
        batch_x = np.array(
            [
                [
                    self.features[anchor_idx],
                    self.features[positive_idx],
                    self.features[negative_idx],
                ]
                for anchor_idx, positive_idx, negative_idx in batch_indices
            ]
        )

        # Reshape to (batch_size * 3, *feature_shape)
        batch_size = len(batch_x)
        batch_x = batch_x.reshape((batch_size * 3,) + self.features.shape[1:])

        # Dummy labels (not used in triplet loss)
        batch_y = np.zeros(batch_size * 3)

        return batch_x, batch_y
```

Listing 5: Triplet data loader.

5.4 Training Process

The training configuration is summarised in Table 3. The training was completed using the Adam optimiser with an initial learning rate of 1×10^{-4}, reduced by a factor of 0.5 whenever the validation loss plateaued. A random mining strategy was used, where negatives are selected without considering difficulty. This avoids the computational overhead of alternatives such as semi-hard mining, which calculates embedding distances to help select informative negatives. The trade-off is that many randomly selected triplets will have a large distance between the positive and negative, and therefore contribute zero loss, making training less sample-efficient.

The `TripletDataGenerator` was configured to produce a maximum of 30,000 triplets per epoch from the training split. A separate generator with an identical configuration but no shuffling provided validation triplets, held out via an 80:20 stratified split within the training set. Training was capped at 100 epochs, with an early stopping mechanism set to have a patience of 50 epochs. Model checkpointing saved the best-performing model at each improvement, and a `ReduceLROnPlateau` callback halved the learning rate when validation loss plateaued, with a minimum learning rate of 1×10^{-6}.

Table 3: Training hyperparameter configuration used for the single end-to-end training run.

Hyperparameter	Value	Notes
Optimiser	Adam	Standard adaptive optimiser
Learning rate	1×10^{-4} (initial)	Reduced on plateau (factor 0.5)
Batch size	32	Triplets per gradient update
Max epochs	100	Upper bound
Early stopping patience	50 epochs	Restores best weights
Triplet margin ( )	1.0	Loss margin
Mining strategy	Random	No difficulty filtering
Max triplets per epoch	30,000	Sampled from training split
Validation split	80:20	Stratified within training set
Dropout rate	0.5	Applied after second LSTM

During training, a small number of samples were found to contain NaN values. This was caused by an oversight in the preprocessing script, which was originally uncaught due to the generated test data not perfectly representing imperfections in the real dataset. Rather than re-running the preprocessing pipeline, which would have been timely, defensive filtering was added to the training script to detect and remove affected samples before training. This workaround was acceptable given the small proportions of affected samples, although for future model training, this would have to be addressed in the pre-processing script.

5.5 Evaluation Process

The evaluation pipeline followed the authentication protocol described in Chapter 4. The trained model generates embeddings for all test samples, and pairwise Euclidean distances are calculated to form genuine distances (where both samples originate from the same user) and impostor distances (where samples originate from different users).

5.5.1 Sampling Strategy

Given the scale of the test set (over 600,000 samples from 168,608 users), computing all pairwise distances is infeasible. The evaluation uses a sampling strategy, taking up to 500,000 pairs of each type. For genuine pairs, samples are taken proportionally from each user. For impostor pairs, random cross-user pairs are selected.

5.5.2 Evaluation Metrics

The metrics outlined in Chapter 4 (EER, FAR, FRR, *ROC AUC*, and d-prime) were calculated from the sampled distance distributions. The *ROC AUC* was calculated using scikit-learn's `roc_curve`. The EER was found by sweeping 1,000 evenly spaced thresholds across the distance range and identifying the point where FAR equals FRR.

5.6 Testing

Both the preprocessing module and the training pipeline were checked through automated tests run with `pytest`. The preprocessing pipeline has 32 tests organised into seven categories: argument parsing, session-level splitting, chunked processing, error handling, mock data generation, edge cases, and sliding window logic. The training pipeline has 6 tests covering data loading from *HDF5*, model construction across multiple configurations (different LSTM units and dropout rates), triplet loss calculation with different margins (0.5, 1.0, 2.0), contrastive loss computation (not used in the actual model), a short training loop to test the training pipeline on mock data, and gradient clipping verification (`clipnorm = 1.0`).

All 38 tests pass against the current source code. Full `pytest` output is provided in the appendix (see Figure 8 and Figure 9). The evaluation pipeline was checked separately by confirming that the trained model produced consistent outputs on the test set — genuine and impostor distributions had the expected characteristics, and all five metrics were calculated without error. As this project evaluated a metric learning approach on an existing dataset rather than deploying a user-facing system, live user testing was not conducted.

5.7 Summary

This chapter has covered the implementation of the KD model. The preprocessing pipeline transforms raw Aalto keystroke data into normalised, windowed feature sequences stored in *HDF5* format with session-level splitting to prevent data leaks. The model is a two-layer LSTM producing 128-dimensional embeddings, trained with triplet loss using dynamically generated triplets. The evaluation procedure uses sampled pairwise distances and multiple metrics.

6 Product Evaluation

6.1 Introduction

This chapter evaluates the trained model against the project's stated aims and research questions, assessing the results from Chapter 5 against the objectives defined in Chapter 1.

6.2 Performance Analysis

The model achieved an EER of 31.35%, a *ROC AUC* of 0.755, a FAR at FRR = 1% of 89.05%, and a d-prime of 0.986. As the *ROC AUC* of $0.755 > 0.5$, this confirms the model learned discriminative structure (Figure 5). The mean genuine distance of 0.649 is significantly lower than the mean impostor distance of 0.997, confirming the intended directional behaviour. The complete evaluation script output, including distance distribution quartiles and model configuration, is provided in Appendix 6.

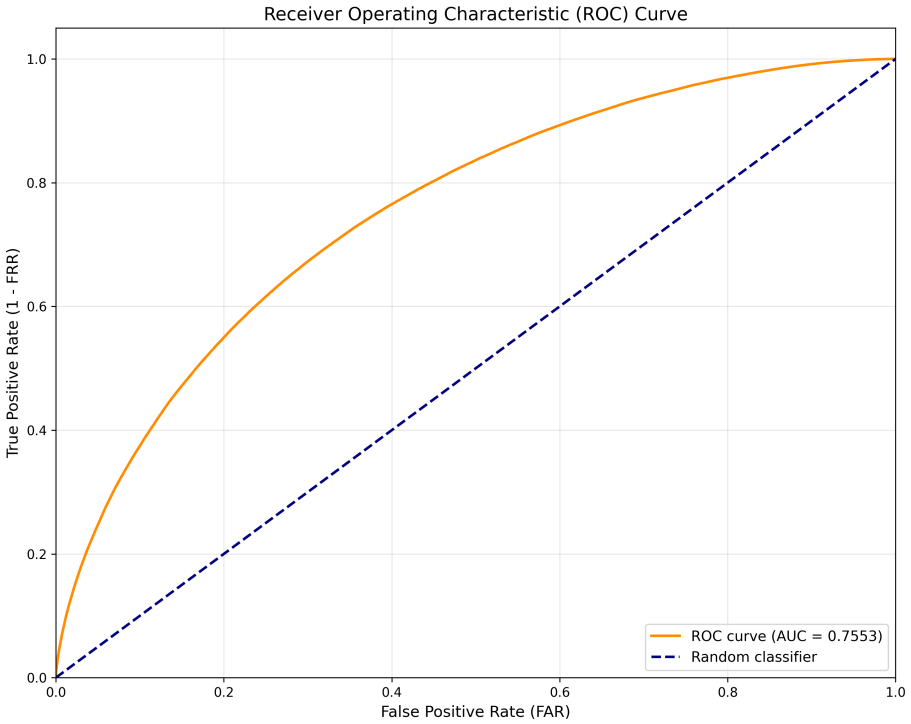

Figure 5: ROC curve for the trained embedding model.

Although the model has learned discrimination, it has its shortcomings affecting applicability. A d-prime of 0.986 indicates substantial overlap between genuine and impostor distributions (Figure 6). At the EER threshold, the system incorrectly accepts roughly one in three impostor attempts whilst rejecting one in three genuine attempts. The model can distinguish users better than chance, but not well enough for deployment.

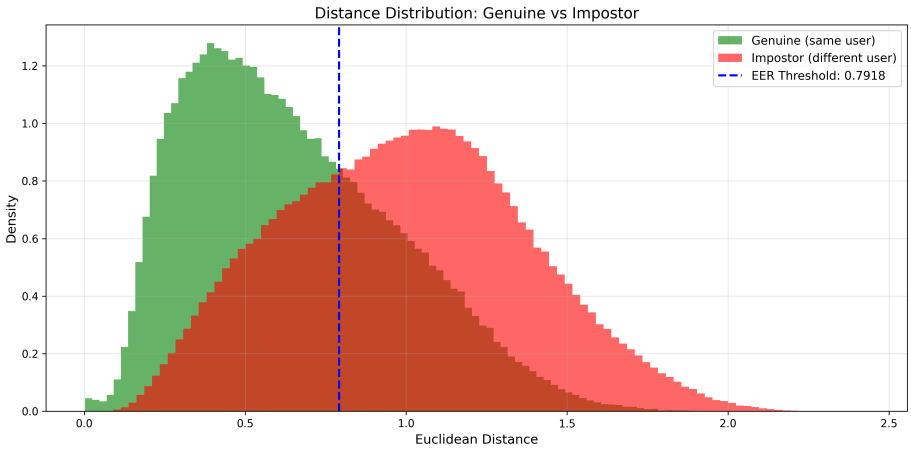

Figure 6: Distance distributions for genuine and impostor pairs.

6.3 Evaluation Against Objectives

Section 1.3 defined three objectives, assessed below:

Objective 1: Design and develop a metric learning model on free-text keystroke data. Fulfilled. The system implements a two-layer LSTM producing 128-dimensional embeddings, trained with triplet loss on the Aalto dataset (168,608 users). The full pipeline (preprocessing, triplet generation, training, and evaluation) is functional.

Objective 2: Evaluate performance and identify key challenges. Partially fulfilled. Performance was evaluated, but due to the high EER of 31.35%, the few-shot enrolment evaluation was not completed as the results would not be meaningful.

Objective 3: Compare against relevant research. Fulfilled. Section 6.4 provides detailed comparison with four benchmark studies.

6.4 Comparison with Existing Research

Killourhy and Maxion (2009) reported an EER of ~10% on 51 subjects. Lu et al. (2020) achieved 2.67–5.9% on 75 and 53 users. Ali and Hassan (2025) and Stylios et al. (2023) report similar low EERs. Across these studies, EERs below 5% are common.

The model's 31.35% EER is substantially worse than the results reported above. This could be due to many factors, including the much larger dataset used in this evaluation, but the disparity in accuracy points to clear issues with the execution and the proposed approach.

6.5 Addressing the Research Questions

6.5.1 RQ1: Can metric learning achieve a competitive EER with fewer than 50 enrolment samples?

The 31.35% EER is already too poor for a few-shot evaluation to yield meaningful results, so this was not conducted. Metric learning captures some discriminative structure ($ROC\ AUC = 0.755$), but not enough for competitive authentication.

6.5.2 RQ2: To what extent can metric learning handle behavioural drift?

The Aalto dataset was selected partly because its intra-user variation (each participant typed multiple sentences) could approximate behavioural drift during training. However, all sentences were collected in a single sitting, so this variation reflects short-term inconsistencies rather than genuine long-term drift. The test set contains the same limited variation, and the model did learn to discriminate within it ($ROC\ AUC = 0.755$), demonstrating some intra-user consistency. Whether this translates to genuine drift resistance remains unclear: the dataset does not provide a realistic temporal test, and the 31.35% EER makes it difficult to isolate drift from baseline distribution overlap. A full assessment requires both a longitudinal dataset and improved baseline performance.

6.5.3 RQ3: How does the model compare to deep learning approaches in current research?

The model significantly underperforms current approaches. The 31.35% EER is significantly higher than the 2.67–10% range in recent studies.

6.6 System Completeness

The project aimed to investigate whether metric learning could work for free-text keystroke authentication. The implemented pipeline covers the core of this research: a preprocessing pipeline that transforms raw keystrokes into normalised hold-time and flight-time feature sequences, a two-layer LSTM trained with triplet loss to produce 128-dimensional embeddings, and an evaluation pipeline measuring embedding space separability across five metrics (EER, FAR, FRR, ROC AUC, and d-prime). Table 4 summarises the status of each component.

The planned few-shot enrolment evaluation (described in 4.5.1) was not conducted. This would have simulated the online authentication, but was dependant on the baseline model achieving sufficient discriminative performance. With a 31.35% EER, the results would not be meaningful.

Table 4: Implementation status of the planned evaluation pipeline.

Component	Status
Data preprocessing pipeline	Implemented
Triplet loss training pipeline (LSTM)	Implemented
Embedding model (128-d output)	Implemented
Evaluation pipeline (5 metrics)	Implemented
Few-shot enrolment evaluation	Not conducted

6.7 Root Cause Analysis

Three interconnected implementation shortcomings produced the observed distribution overlap (see Figure 6):
Feature limitation. Training on only hold time and flight time provides insufficient discriminative signal for 168,608 users. N-gram timing features (digraph and trigraph latencies) are needed.
Random negative mining. Sampling negatives that already have a large separation from the positive is inefficient. Semi-hard mining would focus on informative triplets improving training efficiency. However would require more computational resources.
Dataset constraints. No publicly available dataset satisfies all project requirements — the Aalto dataset provides transcription typing rather than free-text composition, and sessions don't have the long-term separation needed to evaluate behavioural drift.
Together these factors help explain the observed distribution overlap, though each has an identifiable solution — suggesting the metric learning paradigm itself remains sound.

6.8 Summary

Two of three objectives were fully fulfilled, with the evaluation objective partially completed because the high EER meant a few-shot specific evaluation would have been uninformative. The product has all the core stated architecture. All three research questions are partially answered. While the 31.35% EER falls far short of deployment requirements, the *ROC AUC* of 0.755 confirms genuine discriminative signal. Overall this suggests the proposed approach may have merit but requires substantial improvement.

7 Project Evaluation

7.1 Introduction

This chapter evaluates the project development process rather than technical performance. Chapter 4 outlined a methodology combining Kanban-based Agile

with *SDLC* principles. This chapter assesses how closely the plan was followed and provides insight for managing future projects.

7.2 Project Management

Chapter 4 described a methodology combining Agile principles (via Kanban) with *SDLC* practices. A Kanban board provided task management, and four milestones (M1–M4) structured the work into phases (see Appendix E). Twenty-two issues were created during initial planning, covering everything from initial repo structuring to dissertation write-up.

Development mostly happened on a single branch rather than a feature-branch workflow. Managing branches would have enabled a better separation of features during development, but the single-branch workflow adopted was sufficient for a single developer (see Figure 10).

GitHub issues were often not updated. Early milestones were tracked adequately, but milestones and issues did not accurately follow scope changes. The Kanban board and milestones, however, did remain useful for guiding overall structure throughout (see Figure 11 and Figure 12).

The planned baseline model was never developed. Realising there was only time for one training cycle, development bypassed M2 entirely and proceeded straight to the metric learning approach planned for M3. Processing 136 million keystrokes and training on 3.2 million samples introduced computational complexity that was underestimated during planning.

Despite this, the methodology provided good value. The initial planning and structural framework kept the project progressing even as priorities shifted.

7.3 Project Successes

Despite falling short of target accuracy, the project achieved several meaningful outcomes. The developed product mostly matched the proposed design, with preprocessing, training, and evaluation pipelines capable of producing and validating a KD model. Although development methodology was not strictly followed, it provided good structure for the direction and development of the product.

7.4 Challenges Encountered

Computational constraints. Training on 3.2 million samples requires large amounts of computational resources and power. The time for preprocessing and training cycles to run was not factored into the plan, and left no time to run additional cycles to try to improve the model.

Time constraints. The project was ambitious in scope and had several unforeseen complexities that consumed significant time and left little margin for overrunning.

7.5 Future Insights

The successes and encountered challenges during this project provide good insight into how future projects should be approached and managed. First, future works would benefit from a more refined scope. This would allow for a good foundation to be built before the scope expands. Second, project plans should be updated and revised frequently with changes of scope, to help understand what is feasible and the direction of the project. Finally, future projects should factor in additional time to account for unforeseen challenges or misjudgments in the original plan.

7.6 Summary

While the project did not strictly follow its planned methodology, it provided a good structure despite scope changes. The core product matches its design, though computational and time constraints left no room for the iteration needed to improve model accuracy. The key takeaway is to start small with a solid foundation before expanding the scope and to treat the project plan as something that should evolve alongside the work itself.

8 Summary and Conclusion

8.1 Summary of Findings

This project attempted to investigate whether a metric learning approach could improve the practicality of free-text keystroke authentication, focusing on reducing enrolment data requirements and handling behavioural drift. A two-layer LSTM embedding model was trained with triplet loss on the Aalto dataset (~136 million keystrokes from 168,608 users), producing 128-dimensional embeddings evaluated across five metrics.

The model achieved an EER of 31.35%, a *ROC AUC* of 0.755, and a d-prime of 0.986. The *ROC AUC* above 0.5 confirms the model learned discriminative structure, with mean genuine distances (0.649) significantly lower than mean impostor distances (0.997). However, the high EER indicates substantial distribution overlap. This is significantly less than the 2.67–10% EER range reported in recent studies.

Three major reasons for this performance were identified: limited feature extraction using only hold and flight times (no digraphs or trigraphs); random negative sampling rather than semi-hard mining; and dataset constraints where single-sitting sessions could not reflect long-term behavioural drift.

8.2 Conclusions

Addressing the research questions, while metric learning captures discriminative structure in keystroke data, this implementation did not achieve a competitive

accuracy. The *ROC AUC* of 0.755 confirms meaningful separability, but substantial optimisation is required to match existing methods.
Regarding behavioural drift, the dataset lacked the session separation needed for a meaningful assessment. The model demonstrated some intra-user consistency, but whether this translates to drift resistance remains unanswered.
Two of three objectives were fully fulfilled: the metric learning model was designed and implemented, and performance was evaluated against relevant research. The third was partially fulfilled, as the few-shot evaluation could not be completed.

8.3 Recommendations for Alternative Approaches

In reflection, the scope was too ambitious for the available timeframe. Beginning with a smaller scope would have allowed the project to naturally grow. The computational resources required to process a large dataset was underestimated, leaving no room for extra training cycles needed to improve accuracy. Focusing this work on developing a strong baseline would have provided a strong foundation for future works to expand the scope.
External training services such as cloud-based GPU platforms should have been explored to offload computational workloads from local hardware. Access to greater compute power would have allowed additional training cycles and experimentation with more demanding configurations, potentially resulting in improved model accuracy within the available timeframe.
The project plan should have been revised as the scope evolved, ensuring unforeseen challenges were easier to manage by providing a clear direction forward.

8.4 Future Work

Future work should focus on improving the model's performance through more discriminatory features such as digraphs and trigraphs, and semi-hard triplet mining to strengthen the training pipeline. Alternative datasets with longitudinal session data are also needed to properly evaluate behavioural drift resistance.

References

Ali, T. B. and Hassan, A. (2025) Enhancing User-Centric Security Using Machine Learning Techniques for Continuous Authentication Using Keystroke Dynamics [Online]. In: *2025 International Conference on Communication Technologies (ComTech), April 2025.* pp. 1–6. Available from: <https://doi.org/10.1109/ComTech65062.2025.11034612>.

Bellet, A., Habrard, A. and Sebban, M. eds (2015) *Metric Learning.* Synthesis lectures on artificial intelligence and machine learning. San Rafael, California <1537 Fourth Street, San Rafael, CA 94901 USA>: Morgan & Claypool.

Dhakal, V., Feit, A. M., Kristensson, P. O. and Oulasvirta, A. (2018) Observations on Typing from 136 Million Keystrokes [Online]. In: *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, *April 21, 2018*. Montreal QC Canada: ACM, pp. 1–12. Available from: <https://doi.org/10.1145/3173574.3174220>.

Finn, C., Abbeel, P. and Levine, S. (2017) Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks.

Gaines, R. S., Lisowski, W., Press, S. J. and Shapiro, N. (1980) *Authentication by Keystroke Timing: Some Preliminary Results* [Online]. Available from: <ht tps://www.rand.org/pubs/reports/R2526.html> [Accessed 10 April 2026].

Gunetti, D. and Picardi, C. (2005) Keystroke Analysis of Free Text. *ACM Transactions on Information and System Security* [Online], 8 (3) August, pp. 312–347. Available from: <https://doi.org/10.1145/1085126.1085129>.

Hadsell, R., Chopra, S. and LeCun, Y. (2006) Dimensionality Reduction by Learning an Invariant Mapping [Online]. In: *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Volume 2 (CVPR'06)*, *2006*. vol. 2. New York, NY, USA: IEEE, pp. 1735–1742. Available from: <https://doi.org/10.1109/CVPR.2006.100>.

Killourhy, K. S. and Maxion, R. A. (2009) Comparing Anomaly-Detection Algorithms for Keystroke Dynamics [Online]. In: *2009 IEEE/IFIP International Conference on Dependable Systems & Networks*, *June 2009*. pp. 125–134. Available from: <https://doi.org/10.1109/DSN.2009.5270346>.

Koch, G., Zemel, R. and Salakhutdinov, R. (2015) Siamese Neural Networks for One-shot Image Recognition.

Lu, X., Zhang, S., Hui, P. and Lio, P. (2020) Continuous Authentication by Free-Text Keystroke Based on CNN and RNN. *Computers & Security* [Online], 96 September, p. 101861. Available from: <https://doi.org/10.1016/j.cose.2020.101861>.

Mahalakshmi, V., Sandhu, M., Shabaz, M., Keshta, I., Prasad, K. D. V., Kuzieva, N., Byeon, H. and Soni, M. (2024) Few-Shot Learning-Based Human Behavior Recognition Model. *Computers in Human Behavior* [Online], 151 February, p. 108038. Available from: <https://doi.org/10.1016/j.chb.2023.108038>.

Mondal, S. and Bours, P. (2017) A Study on Continuous Authentication Using a Combination of Keystroke and Mouse Biometrics. *Neurocomputing* [Online], 230 March, pp. 1–22. Available from: <https://doi.org/10.1016/j.neucom.2016.11.031>.

Monrose, F. and Rubin, A. D. (2000) Keystroke Dynamics as a Biometric for

Authentication. *Future Generation Computer Systems* [Online], 16 (4) February, pp. 351–359. Available from: <https://doi.org/10.1016/S0167-739X(99)00059-X>.

Movshovitz-Attias, Y., Toshev, A., Leung, T. K., Ioffe, S. and Singh, S. (2017) *No Fuss Distance Metric Learning Using Proxies* [Online]. Available from: <https://doi.org/10.48550/arXiv.1703.07464>.

S., A. and K., J. S. (2025) Enhancing Security and Usability with Context Aware Multi-Biometric Fusion for Continuous User Authentication. *Scientific Reports* [Online], 15 (1) August, p. 30627. Available from: <https://doi.org/10.1038/s41598-025-14833-z>.

Snell, J., Swersky, K. and Zemel, R. (2017) Prototypical Networks for Few-shot Learning.

Stylios, I., Chatzis, S., Thanou, O. and Kokolakis, S. (2023) Continuous Authentication with Feature-Level Fusion of Touch Gestures and Keystroke Dynamics to Solve Security and Usability Issues. *Computers & Security* [Online], 132 September, p. 103363. Available from: <https://doi.org/10.1016/j.cose.2023.103363>.

Stylios, I., Skalkos, A., Kokolakis, S. and Karyda, M. (2022) BioPrivacy: A Behavioral Biometrics Continuous Authentication System Based on Keystroke Dynamics and Touch Gestures. *Information & Computer Security* [Online], 30 (5) November, pp. 687–704. Available from: <https://doi.org/10.1108/ICS-12-2021-0212>.

Sugrim, S., Liu, C., McLean, M. and Lindqvist, J. (2019) Robust Performance Metrics for Authentication Systems [Online]. In: *Proceedings 2019 Network and Distributed System Security Symposium, 2019*. San Diego, CA: Internet Society. Available from: <https://doi.org/10.14722/ndss.2019.23351>.

Teh, P. S., Teoh, A. B. J. and Yue, S. (2013) A Survey of Keystroke Dynamics Biometrics. *The Scientific World Journal* [Online], 2013 November, p. 408280. Available from: <https://doi.org/10.1155/2013/408280>.

Vinyals, O., Blundell, C., Lillicrap, T., Kavukcuoglu, K. and Wierstra, D. (2016) Matching Networks for One Shot Learning [Online]. Available from: <https://www.semanticscholar.org/paper/Matching-Networks-for-One-Shot-Learning-Vinyals-Blundell/be1bb4e4aa1fcf70281b4bd24d8cd31c04864bb6> [Accessed 20 April 2026].

Appendix A. List of Abbreviations

- **CA** - Continuous Authentication
- **CNN** - Convolutional Neural Network
- **EER** - Equal Error Rate
- **FAR** - False Acceptance Rate
- **FRR** - False Rejection Rate
- **KD** - Keystroke Dynamics
- **LSTM** - Long Short-Term Memory
- **MAML** - Model-Agnostic Meta-Learning
- **ML** - Machine Learning
- **MLP** - Multi-Layer Perceptron
- **RNN** - Recurrent Neural Network
- **SDCL** - System Development Life Cycle
- **TSV** - tab separated value

Appendix B. Confirmation of Ethical Approval

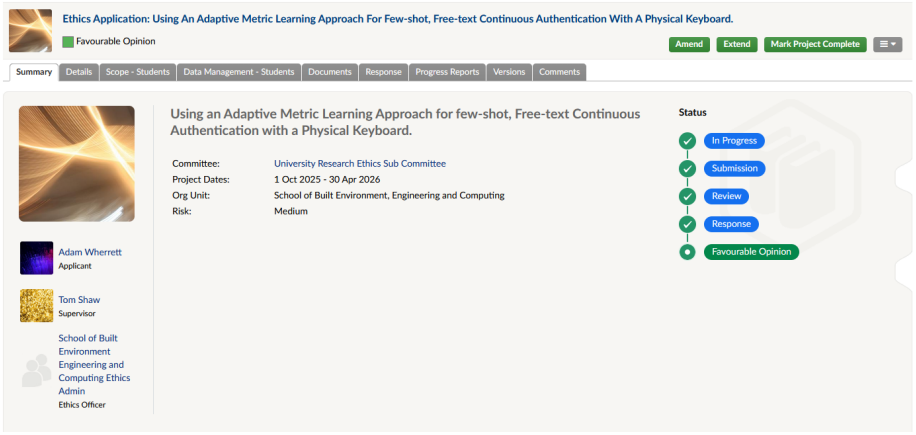

Figure 7: Confirmation of ethical approval.

Appendix C. Evidence of Testing

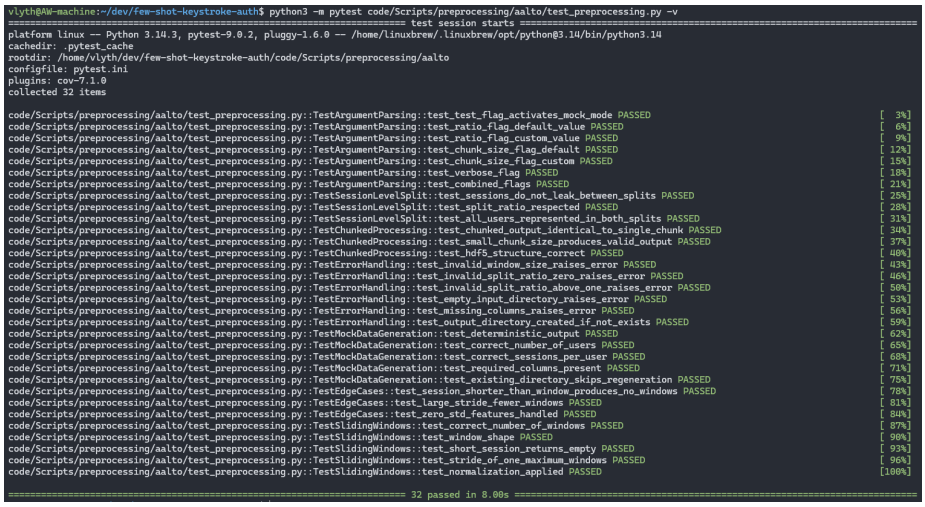

Figure 8: Preprocessing pipeline test results.

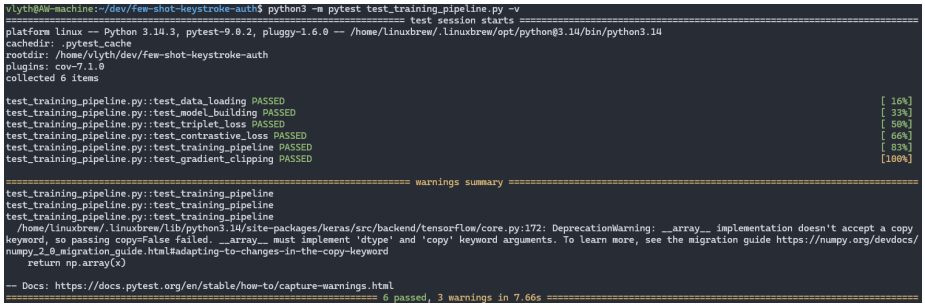

Figure 9: Training pipeline test results.

Appendix D. Full Evaluation Output

The complete output from the evaluation script, including all metrics, distance distribution statistics, and model configuration, is reproduced below.


```
{
  "timestamp": "2026-04-04T22:42:25.941793",
  "model_path": "models/keystroke_lstm_20260402_142640.keras",
  "data_path": "data/preprocessed/aalto.h5",
  "split": "test",
  "num_samples": 3197781,
  "num_users": 168608,
  "window_size": 20,
  "embedding_dim": 128,
  "metrics": {
    "eer": 0.31345393130061694,
    "eer_threshold": 0.7917629932430951,
    "far_at_eer": 0.3142,
    "frr_at_eer": 0.31270786260123384,
    "far_at_frr_01": 0.890466,
    "roc_auc": 0.7553400281868035
  },
  "distance_statistics": {
    "genuine": {
      "mean": 0.6490394473075867,
      "std": 0.32500335574150085,
      "median": 0.5987091064453125,
      "min": 0.0027180053293704987,
      "max": 2.2284114360809326,
      "q1": 0.39038433134555817,
      "q3": 0.8704227954149246
    },
    "impostor": {
      "mean": 0.9967208504676819,
      "std": 0.37811458110809326,
      "median": 0.9981082677841187,
      "min": 0.017224736511707306,
      "max": 2.4356067180633545,
      "q1": 0.7119056731462479,
      "q3": 1.2600736320018768
    },
    "separability": {
      "d_prime": 0.9861609550855385
    }
  }
}
```


Listing 6: Full evaluation script output.

Appendix E. Supporting evidence of a professional approach to product development

Figure 10: Git commit history showing development activity across the project timeline.

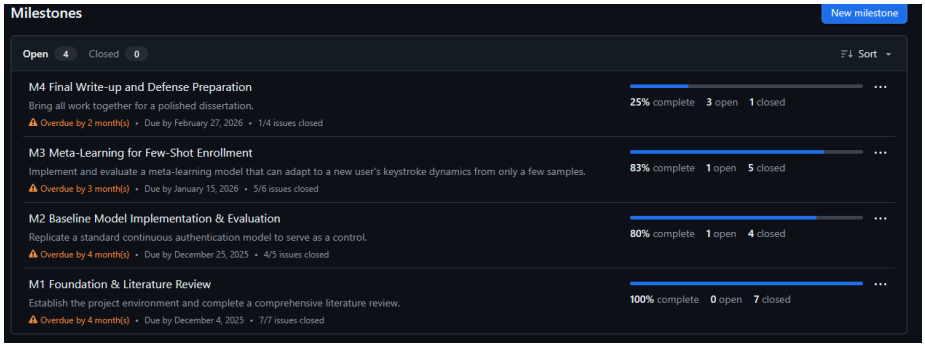

Figure 11: GitHub milestones used to organise deliverables across four phases.

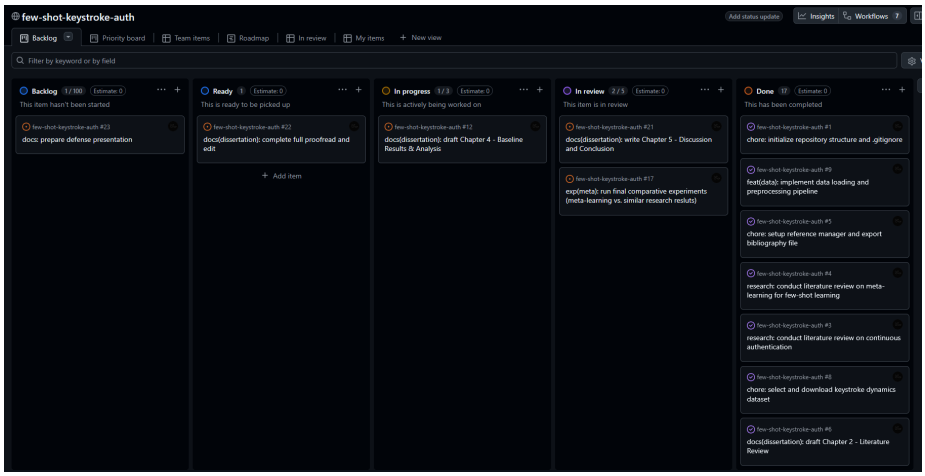

Figure 12: Kanban board used to track issue status throughout development.

Appendix F. Breakdown of Gen AI Use

Table 5: Breakdown of Gen AI Use

Tool	How it was Used	Reference
Grammarly	Grammarly was used for proofreading, to help fix grammatical and spelling error within the report	Grammarly [Online]. Available from: https://app.grammarly.com/ [Accessed 29 April 2026].
GLM 5.1	This was used to assist with LaTex Tikz diagrams syntax for figures 1-4	GLM-5.1: Towards Long-Horizon Tasks [Online]. Available from: https://z.ai/blog/glm-5.1 [Accessed 30 April 2026].

Tool	How it was Used	Reference
Qwen-3.5-35B-A3B-128K (llama-server), opencode, oh-my-openagent	Qwen was ran locally via a llama-server to assist with the development of the products code. It was used within the OpenCode harness with the Oh-My-Openagent plugin to improve agency. The decision to use a locally hosted model to assist development of the codebase was to speed up progression and reduce the impact of time constraints making an ambitious project more feasible. Furthermore, a local model was selected over a commercial api to prevent a third party vendor from receiving biometric/ keystroke data, if the model took a sample of the dataset in its context. Transcripts of the conversations are subbmitted within the codebase at docs/ai-transcripts	Qwen/Qwen3.5-35B-A3B · Hugging Face (2026) [Online]. Available from: https://huggingface.co/Qwen/Qwen3.5-35B-A3B [Accessed 30 April 2026]. Ggml-Org/Llama.Cpp (2026) [Online]. ggml. Available from: https://github.com/ggml-org/llama.cpp [Accessed 30 April 2026]. OpenCode [Online]. Available from: https://opencode.ai/ [Accessed 30 April 2026]. Kim, Y. Oh My OpenAgent — The Best Agent Harness [Online]. Oh My OpenAgent. Available from: https://ohmyopenagent.com [Accessed 30 April 2026].