

UNIVERSITY OF WALES TRINITY SAINT DAVID

One-Dimensional Convolutional Neural Networks and Mouse Dynamics: Authentication with Behavioural Biometrics using Organic Mouse Data

by

Liam Kelly

A project submitted in partial fulfillment for the
degree of M.Sc. Software Engineering (Web Development)

in the

School of Applied Computing
Wales Institute of Science and Art (WISA)

Supervisor: Dr. Tim Bashford



Prifysgol Cymru
Y Drindod Dewi Sant
University of Wales
Trinity Saint David

January 2024

Declaration of Authorship

I, [Liam Kelly](#) declare that I am the sole author of this Project; that all references cited have been consulted; that I have conducted all work of which this is a record, and that the finished work lies within the prescribed word limits.

This work has not previously been accepted as part of any other degree submission.

Signed: Liam Dennis Kelly

Date: Friday 26th January 2024

Form of Consent

I [Liam Kelly](#) hereby consent that my Project, submitted in candidature for the M.Sc. Software Engineering (Web Development) degree, if successful, may be made available for inter-library loan or photocopying (subject to the law of copyright), and that the title and abstract may be made available to outside organisations.

Signed: Liam Dennis Kelly

Date: Friday 26th January 2024

UNIVERSITY OF WALES TRINITY SAINT DAVID

Abstract

Wales Institute of Science and Art (WISA)
School of Applied Computing

M.Sc. Software Engineering (Web Development)

by [Liam Kelly](#)

This research explores the feasibility of using mouse dynamics as a form of user authentication through a proof-of-concept study employing a 1D Convolutional Neural Network (1D-CNN). The main objective was to assess the effectiveness of a 1D-CNN model, named Shea, in classifying users based on their organic mouse movement data under real-world conditions. The hypothesis set forth posited that the model would achieve an overall accuracy of 95% and a user-based classification precision of 95% with a False Acceptance Rate (FAR) of 1%. Shea demonstrated remarkable efficacy, exceeding these metrics with an overall accuracy of 99.6%, precision of 99.2%, and exceptionally low FAR across users. Alongside a novel dataset, the study delivered two software artefacts: a mouse logger for data collection and the Convolutional Mouse Authentication Network (CONMAN) service for data processing and model preparation. Despite the success, the research identified limitations in data precision, particularly in timestamps and positional data, and a lack of comprehensive automated testing. The dataset, while sufficient for initial exploration, was limited in depth and diversity, highlighting the need for a more robust approach in real-world applications. The research contributes to the ongoing discourse in behavioural biometrics, showcasing the potential of 1D-CNNs in user authentication. However, it also underscores the importance of empirical methodology in model design, comprehensive data collection, and the need for broader participant diversity and larger datasets. Future studies could benefit from a more methodical approach to model parameter selection, extended data collection periods, and the inclusion of additional features and surveying of environmental variables.

Acknowledgements

My acknowledgements and thanks to Dr. Tim Bashford at the UWTSD for his guidance during this project and throughout the course, and to Thunder Squad at the DVLA.

Contents

Declaration of Authorship	i
Form of Consent	ii
Abstract	iii
Acknowledgements	iv
List of Figures	xiii
List of Tables	xv
Abbreviations	xviii
1 Introduction	2
1.1 The humble password	2
1.2 Authentication	4
1.3 Behavioural Biometrics - Mouse Dynamics	6
1.4 Deep Learning and Convolutional Neural Networks	7
2 Literature Review	9
2.1 Research in Mouse Dynamics for Authentication	9
2.2 Deep Learning in the field of Authentication with Mouse Dynamics	13
2.3 Study rationale and contribution to research	14
3 Methodology	16
3.1 Hypothesis	16
3.2 Research Design	16
3.2.1 Data Collection	17
3.2.2 Data Processing	19
3.3 Evaluation Criteria and Statistical Analysis	26
3.3.1 Methodology Equations: Feature Computation	26
3.3.2 Methodology Equations: Model Performance Metrics	28
3.4 Participant Considerations	31

4	Design & Implementation	32
4.1	Procedure for specification requirements, user stories and design documentation	32
4.1.1	Mouse Logger & The CONMAN Service	32
4.1.2	1D-CNN Model Design	36
4.2	Materials & Tools	37
4.2.1	Hardware	37
4.2.2	Python	37
4.2.3	Development Environment	38
4.2.4	PyInstaller	39
4.2.5	TensorFlow and Keras API	39
5	Results	42
5.1	Description of the finished working product	42
5.1.1	Mouse Logger and The CONMAN Service	42
5.1.2	Model Architecture	43
5.2	Model Results	44
5.3	Best Model (Shea) vs. Unseen Dataset	49
6	Discussion	55
6.1	Evaluation of the proof-of-concept	55
6.2	Evaluation of study deliverables	56
6.2.1	Mouse logger	56
6.2.2	The CONMAN Service	57
6.2.3	The 1D-CNN	58
6.2.4	Dataset	58
6.2.5	Research Outcomes	60
6.2.6	Reflections on this study	61
7	Conclusion	63
7.1	Suggestions for Further Work	63
7.1.1	Dataset Expansion, Diversification and Feature Inclusion	63
7.1.2	Controlled Environment vs. Organic Data	64
7.1.3	Enhancement of the CONMAN Service	64
7.1.4	Empirical Hyperparameter and Model Variable Definition	64
7.1.5	User Surveys	64
7.1.6	Real-time Deployment and Testing with 'Bad-Actors'	64
7.1.7	Balancing Accessibility and Security	65
7.1.8	Longitudinal studies	65
7.2	Conclusion	65
A	CONMAN Service: Software Documentation	67
A.1	Introduction	67
A.1.1	Purpose	67
A.2	User Guide	67
A.2.1	Configuring the service	67

A.2.2	Running the Service	69
A.2.3	Adding additional data	69
A.3	System Detail	70
A.3.1	main.py	70
A.3.1.1	Workflow	70
A.3.1.2	Components Used	71
A.3.1.3	Entry Point	71
A.3.2	runner.ExperimentRunner.py	72
A.3.2.1	Attributes	72
A.3.2.2	Methods	72
A.3.2.3	Workflow	72
A.3.3	Config.py	74
A.3.3.1	Data Configuration	74
A.3.3.2	Model Configuration	74
A.3.3.3	Sequence Creation Configuration	74
A.3.3.4	Experiment Configuration	74
A.3.4	config.data_directories.json	76
A.3.4.1	Raw Data Directory Configuration	76
A.3.4.2	User Label Mapping Configuration	76
A.3.5	controllers.Controller.py	77
A.3.5.1	Abstract Methods	77
A.3.5.2	Role and Implementation	77
A.3.6	controllers.DataController.py	78
A.3.6.1	Attributes	78
A.3.6.2	Methods	78
A.3.6.3	Workflow	78
A.3.7	controllers.ExperimentController.py	80
A.3.7.1	Attributes	80
A.3.7.2	Methods	80
A.3.7.3	Workflow	80
A.3.8	controllers.ResultsController.py	81
A.3.8.1	Attributes	81
A.3.8.2	Methods	81
A.3.8.3	Workflow	81
A.3.9	controllers.data_utils.DataLoader.py	83
A.3.9.1	Static Methods	83
A.3.9.2	Workflow	83
A.3.10	controllers.data_utils.DataProcessor.py	84
A.3.10.1	Attributes	84
A.3.10.2	Methods	84
A.3.10.3	Workflow	84
A.3.11	controllers.experiment_utils.DataNormaliser.py	86
A.3.11.1	Attributes	86
A.3.11.2	Methods	86
A.3.11.3	Workflow	86
A.3.12	controllers.experiment_utils.SequenceCreator.py	87
A.3.12.1	Attributes	87

A.3.12.2	Methods	87
A.3.12.3	Workflow	87
A.3.13	controllers.experiment_utils.DatasetCompiler.py	88
A.3.13.1	Attributes	88
A.3.13.2	Methods	88
A.3.13.3	Workflow	88
A.3.14	controllers.results_utils.MetricCalculator.py	89
A.3.14.1	Attributes	89
A.3.14.2	Methods	89
A.3.14.3	Workflow	89
A.3.15	controllers.results_utils.OutputHandler.py	90
A.3.15.1	Methods	90
A.3.15.2	Workflow	90
A.3.16	controllers.results_utils.Plotter.py	91
A.3.16.1	Attributes	91
A.3.16.2	Methods	91
A.3.16.3	Workflow	91
A.3.17	ModelCreator and create_model Function	92
A.3.17.1	create_model Function	92
A.3.17.2	ModelCreator Attributes	92
A.3.17.3	ModelCreator Methods	92
A.3.17.4	Workflow	93
A.4	Unit Testing	93
A.5	Development Guide	93
A.6	Source Code with Documentation	97
A.6.0.1	main.py	97
A.6.0.2	SystemUtils.py	99
A.6.0.3	Config.py	101
A.6.0.4	ModelCreator.py	102
A.6.0.5	ExperimentRunner.py	105
A.6.0.6	Controller.py	108
A.6.0.7	DataController.py	109
A.6.0.8	ExperimentController.py	111
A.6.0.9	ResultsController.py	115
A.6.0.10	DataLoader.py	118
A.6.0.11	DataProcessor.py	120
A.6.0.12	DataNormaliser.py	124
A.6.0.13	DatasetCompiler.py	127
A.6.0.14	SequenceCreator.py	129
A.6.0.15	OutputHandler.py	131
A.6.0.16	MetricCalculator.py	133
A.6.0.17	Plotter.py	136
A.6.0.18	test_compute_features.py	139
A.6.0.19	test_create_model.py	140
A.6.0.20	test_data_splitter.py	142
A.6.0.21	test_preprocess_data.py	144
A.6.0.22	test_sequence_creator.py	145

B	Project Briefing and Informed Consent Confirmation	147
B.1	Introduction	147
B.2	Purpose of the Study	147
B.3	What Will Be Collected	147
B.4	Anonymity and Data Privacy	148
B.5	Voluntary Participation	148
B.6	Review of Source Code	148
B.7	Contact Information	149
C	Software Requirements Specification	150
C.1	Introduction	150
C.1.1	Scope	150
C.2	Functional requirements	151
C.2.1	Mouse-logger	151
C.2.1.1	Movement tracking	151
C.2.1.2	Click recording	151
C.2.1.3	Time recording	151
C.2.1.4	Data storage	151
C.2.1.5	Exiting programme	152
C.2.2	CONMAN	152
C.2.2.1	Model Architecture	152
C.2.2.2	Model Training and Validation	152
C.2.2.3	Data Processing	153
C.3	Non-functional requirements	153
C.3.1	Mouse-logger	153
C.3.1.1	Software compatibility	153
C.3.1.2	Hardware compatibility	153
C.3.1.3	Discreet operation	154
C.3.1.4	Security	154
C.3.1.5	Performance efficiency	154
C.3.1.6	User interface	154
C.3.2	CONMAN	155
C.3.2.1	Data Management	155
C.3.2.2	Usability and Accessibility	155
C.3.2.3	Reliability	156
C.3.2.4	Compatibility and Integration	156
C.3.2.5	Performance and Efficiency	156
C.3.2.6	Security and Privacy	156
C.3.2.7	Data Storage and Management	156
C.4	Design constraints	157
C.4.1	Technical constraints	157
C.4.2	Regulatory constraints	157
C.4.3	Budgetary constraints	157
D	Epics and User Stories	158
D.1	Epics: mouse logger	158
D.1.1	Create mouselogging software	158

D.1.1.1	Associated User Stories to the epic	158
D.1.2	Deploy mouse logger and collect data	160
D.1.2.1	Associated User Stories to the epic	160
D.2	Epics: The CONMAN Service	160
D.2.1	Create 1DCNN model within the Service	160
D.2.1.1	Associated User Stories	161
D.2.1.2	Spikes	161
D.2.2	Create a data processing module	161
D.2.2.1	Associated User Stories	162
D.2.2.2	Spikes	162
D.2.3	Create a module which directs the flow of an experiment run	162
D.2.3.1	Associated User Stories	163
D.2.3.2	Spikes	163
D.2.4	Create a module which produces outflow results	164
D.2.4.1	Associated User Stories	164
D.2.4.2	Spikes	164
E	Software Design Documentation	165
E.1	Introduction	165
E.1.1	Scope	165
E.2	Mouse logger	165
E.2.1	Architectural Design	165
E.2.2	Decomposition Description	166
E.2.2.1	FileManager	166
E.2.2.2	EventHandler	167
E.3	The CONMAN Service	168
E.3.1	Architectural Design	168
E.3.2	Decomposition Description	169
E.3.2.1	Model Creator	169
E.3.2.2	Associated Controller Classes	170
E.3.2.3	data_utils.DataLoader	172
E.3.2.4	data_utils.Data Processor	173
E.3.2.5	data_utils.DataNormaliser	174
E.3.2.6	SequenceCreator	175
E.3.2.7	data_utils.Dataset Compiler	175
E.3.2.8	results_utils.Metric Calculator	176
E.3.2.9	results_utils.OutputHandler	177
E.3.2.10	ResultsUtils.Plotter	177
E.4	Human Interface Design	178
E.4.1	Overview	178
E.4.2	Mouse Logger Interface User Interaction	178
E.4.2.1	Icon Click to Start	178
E.4.2.2	Terminal Feedback	178
E.4.2.3	Termination Command	179
E.4.3	Design Principles for Mouse Logger	179
E.4.4	The CONMAN Service User Interaction	179
E.4.4.1	Initiating the Service	179

E.4.4.2	Error, Logging, and Output	179
E.4.4.3	Parameter Adjustment and Expansion	179
E.4.5	Design Principles for The CONMAN Service	180
F	Mouse logger: Software Documentation	181
F.1	Introduction	181
F.1.1	Purpose	181
F.2	User Guide	181
F.2.1	Packaging the service	181
F.2.2	Distribution and Deployment	182
F.2.3	Running the mouse-logger	182
F.3	System Detail	183
F.3.1	MouseLogger.py	183
F.3.1.1	Workflow	183
F.3.1.2	Components Used	184
F.3.1.3	Entry Point	184
F.3.2	FileManager.py	184
F.3.2.1	Attributes and Constants	184
F.3.2.2	Methods	184
F.3.2.3	Usage	185
F.3.3	EventHandler.py	185
F.3.3.1	Attributes	185
F.3.3.2	Methods	185
F.3.3.3	Usage	186
F.4	Testing and Benchmark	186
F.5	Development Guide	187
F.6	Source Code with Documentation	189
F.6.0.1	MouseLogger.py	189
F.6.0.2	FileManager.py	191
F.6.0.3	EventHandler.py	194
F.6.0.4	Benchmark.py	196
F.6.0.5	test_event_handler.py	200
F.6.0.6	test_file_manager.py	202
F.6.0.7	test_mouse_logger.py	204
G	User Hardware	206
H	Proposed User Survey	207
H.1	[Proposal] User Hardware Setup Survey	207
I	CONMAN Service Raw Data Output	210
I.1	Overall Training/Testing set of Performance Metrics for each iteration of the CONMAN service	211
I.2	User-centric Training/Testing set of Performance Metrics for each iteration of the CONMAN service	212
I.3	User-centric Training/Testing set of Performance Metrics - statistical summary for each iteration of the CONMAN service	213
I.4	Overall Model 22 (Shea) performance vs. Unseen dataset	213

I.5	User-centric Model 22 (Shea) performance vs. Unseen dataset	213
I.6	Confusion Matrices: Model 22 (Shea) performance vs. Unseen dataset . .	214
 Bibliography		 215

List of Figures

1.1	Single factor authentication through a username and password	2
1.2	An illustration of the protective measure behind 2FA/MFA is if a user's password is compromised the probability of second or third authentication medium being compromised as well is less likely, acting as a safe-guard to access.	3
1.3	Author generated graphic of typical mouse behaviour: Mouse-movement: planar movement across the two-axes of freedom, Drag-and-Drop: movement beginning with a click down of a mouse button and ends with the button release, Point-and-Click: Mouse Movement followed by a click or double-click, Mouse silence: a period of no movement.	6
1.4	An illustrated example from Saturn Cloud of a CNN in the example of computer vision. Layers of the neural network detect values of pixels, if the threshold mark is passed this will stimulate neurons to the next layer which, in this example are used to identify patterns. This leads to the correct identification of the number from a handwritten input.	8
3.1	Author-generated graphic summarising the number of user-captured mouse events. User D was removed from this study due to its small sample size in comparison the other three users. Down-sampling was opted for in order to provide a balanced but simplified classification pipeline of the data.	20
3.2	An example of a confusion matrix from a model iteration. Model 6, User A.	29
4.1	Author-generated initial design of mouse-logger logic	34
4.2	Author-generated initial design of complete system	36
5.1	The final architecture of the 1D-CNN model	44
5.2	An example of an Training vs Validation plot for both accuracy (top) and loss (bottom) - Model 1, epochs = 18.	45
5.3	An example of an Training vs Validation plot for both accuracy (top) and loss (bottom) - Model 4, epochs = 17.	46
5.4	An example of an Training vs Validation plot for both accuracy (top) and loss (bottom) - Model 22, epochs = 28.	47
5.5	A box-and-whisker plot of the overall model performance by metrics (excluding AoC OvR, (n=25) (Green triangle = Mean), (Orange line = Median)	48
5.6	The RoC AuC OvR plot for Model 22 against the unseen dataset.	50
5.7	The Confusion Matrix from the Shea model vs Unseen data - User A contributing 9733 sequences to the unseen set.	51

5.8	The Confusion Matrix from the Shea model vs Unseen data - User B contributed 6598 sequences to the unseen set.	52
5.9	The Confusion Matrix from the Shea model vs Unseen data - User C contributed 1629 sequences to the unseen set.	53
6.1	A histogram showing the frequency of mouse movement capture along the positional x-axis for all three users.	59
A.1	How class directories are routed to the service, this formatting should be followed along with the incremental increase of any additional class labels	69
A.2	Order of operations when main() is initiated	71
A.3	Order of operations in Experiment Runner	73
A.4	Order of operations managed by DataController.load()	79
E.1	The overall design of the logical flow of the mouse logger system.	166
E.2	The proposed high-level system design of the service.	169
E.3	The flow of data through the controllers and their respective utility classes when managing data.	172
E.4	The relation of the Data Processor and its associated classes	174
E.5	Activation icon for the mouse logger	178
F.1	Activation icon for the mouse logger	183
I.1	A copy of the raw overall data outputs from the CONMAN service, models 1 to 25.	211
I.2	A copy of the raw user-centric data outputs from the CONMAN service, models 1 to 25.	212
I.3	A copy of the raw user-centric data outputs statistical summary from the CONMAN service, models 1 to 25.	213
I.4	A copy of the raw overall data outputs from the CONMAN service, model 22 vs Unseen dataset	213
I.5	A copy of the raw user-centric data outputs from the CONMAN service, model 22 vs Unseen dataset	213
I.6	A copy of the confusion matrix data outputs from the CONMAN service, model 22 vs Unseen dataset	214

List of Tables

2.1	Performance metrics and their definitions	10
2.2	Summary of research in the investigation of CNNs as authentication binary classifiers for mouse dynamic data, assessed on the Balabit dataset (Chong <i>et al.</i> 2020)	13
2.3	Summary of research in the investigation of 1D-CNNs as authentication classifiers for mouse dynamic data	14
2.4	Study Objective Deliverables	15
3.1	Performance metrics and their definitions	17
3.2	Summary of data captured and rationale.	18
3.3	A sample of a mouse activity log	19
5.1	Average and Peak CPU and RAM usage expressed as a percentage of the total availability at different intensities, specifications of these tests can be seen in the mouse logger documentation.	43
5.2	Overall Testing Performance across Models - rounded to 3sf.	47
5.3	Mean Testing Performance across Users	48
5.4	The Shea model vs. Unseen Data Performance Metrics	49
5.5	The Shea model vs Unseen Users Performance Metrics	50
5.6	Summary of research in the investigation of 1D-CNNs as authentication classifiers for mouse dynamic data, compared to the findings of this paper	54
6.1	Total Number of Mouse Clicks for Each User by Button	59
E.1	Associated Controllers	171
F.1	Average and Peak CPU and RAM usage expressed as a percentage of the total availability at different intensities	186
G.1	Summary of each user’s hardware set-up	206

List of Code Inserts

3.1	The preprocess_data method in the data_utils.DataProcessor class	21
3.2	The data_splitter method in the data_utils.DataProcessor class	22
3.3	The compute_features method implementing the feature computation equations in the data_utils.DataProcessor class	24
3.4	The shuffle_sequences_labels method sequence and label shuffling uniformly	25
3.5	The called method compute_individual_metrics	30
3.6	The additional calculation methods called by compute_features	31
4.1	The create model method implementing Keras layer definitions	40
4.2	The config.py file which acted as a global variable configuration center . .	41
A.1	The Config.py where variables for the model and service can be adjusted .	68
A.2	CONMAN Source Code: Main.py	97
A.3	CONMAN Source Code: SystemUtils.py	99
A.4	CONMAN Source Code: Config.py	101
A.5	CONMAN Source Code: ModelCreator.py	102
A.6	CONMAN Source Code: ExperimentRunner.py	105
A.7	CONMAN Source Code: Controller.py	108
A.8	CONMAN Source Code: DataController.py	109
A.9	CONMAN Source Code: ExperimentController.py	111
A.10	CONMAN Source Code: ResultsController.py	115
A.11	CONMAN Source Code: DataLoader.py	118
A.12	CONMAN Source Code: DataProcessor.py	120
A.13	CONMAN Source Code: DataNormaliser.py	124
A.14	CONMAN Source Code: DatasetCompiler.py	127
A.15	CONMAN Source Code: SequenceCreator.py	129
A.16	CONMAN Source Code: OutputHandler.py	131
A.17	CONMAN Source Code: MetricCalculator.py	133
A.18	CONMAN Source Code: Plotter.py	136
A.19	CONMAN Source Code: test_compute_features.py	139
A.20	CONMAN Source Code: test_create_model.py	140
A.21	CONMAN Source Code: test_data_splitter.py	142
A.22	CONMAN Source Code: test_process_data.py	144
A.23	CONMAN Source Code: test_sequence_creator.py	145
F.1	Mouse logger Source Code: MouseLogger.py	189
F.2	Mouse logger Source Code: FileManager.py	191
F.3	Mouse logger Source Code: EventHandler.py	194
F.4	Mouse logger Source Code: Benchmark.py	196
F.5	Mouse logger Source Code: test_event_handler.py	200

F.6	Mouse logger Source Code: <code>test_file_manager.py</code>	202
F.7	Mouse logger Source Code: <code>test_mouse_logger.py</code>	204

Abbreviations

1D-CNN	One-Dimensional Convolutional Neural Network
2D-CNN	Two-Dimensional Convolutional Neural Network
2FA	Two Factor Authentication
3D	Three Dimensional
ANN	Artificial Neural Network
CNN	Convolutional Neural Networks
CONMAN	Convolutional Mouse Authentication Network
CSV	Comma Separated Values
EER	Equal Error Rate
FAR	False Acceptance Ratio
FP	False Positive
FN	False Negative
FRR	False Rejection Ratio
GUI	Graphical User Interface
IBA	Inheritance Based Authentication
KBA	Knowledge Based Authentication
LSTM	Long Short Term Memory
MFA	Multi-Factor Authentication
OS	Operating System
OTP	One Time Passcode
PBS	Possession Based Authentication
ROC-AUC	Receiver Operating Characteristic Area Under the Curve
SDD	Software Design Documentation
SMS	Short Messaging Service
SRS	Software Requirements Specification

SVM	Support Vector Machine
TP	True Positive
TPR	True Positive Rate
TN	True Negative
TNR	True Negative Rate

Dedicated to Tyler Curtis and her infinite patience with me.

UNIVERSITY OF WALES TRINITY SAINT DAVID

Abstract

Wales Institute of Science and Art (WISA)
School of Applied Computing

M.Sc. Software Engineering (Web Development)

by [Liam Kelly](#)

This research explores the feasibility of using mouse dynamics as a form of user authentication through a proof-of-concept study employing a 1D Convolutional Neural Network (1D-CNN). The main objective was to assess the effectiveness of a 1D-CNN model, named Shea, in classifying users based on their organic mouse movement data under real-world conditions. The hypothesis set forth posited that the model would achieve an overall accuracy of 95% and a user-based classification precision of 95% with a False Acceptance Rate (FAR) of 1%. Shea demonstrated remarkable efficacy, exceeding these metrics with an overall accuracy of 99.6%, precision of 99.2%, and exceptionally low FAR across users. Alongside a novel dataset, the study delivered two software artefacts: a mouse logger for data collection and the Convolutional Mouse Authentication Network (CONMAN) service for data processing and model preparation. Despite the success, the research identified limitations in data precision, particularly in timestamps and positional data, and a lack of comprehensive automated testing. The dataset, while sufficient for initial exploration, was limited in depth and diversity, highlighting the need for a more robust approach in real-world applications. The research contributes to the ongoing discourse in behavioural biometrics, showcasing the potential of 1D-CNNs in user authentication. However, it also underscores the importance of empirical methodology in model design, comprehensive data collection, and the need for broader participant diversity and larger datasets. Future studies could benefit from a more methodical approach to model parameter selection, extended data collection periods, and the inclusion of additional features and surveying of environmental variables.

Chapter 1

Introduction

1.1 The humble password

The username and password (*see figure 1.1 [1]*) were a cost-effective choice for security when computing was in its infancy in the 1960s and remains the most widely-used form of authentication to this day [2]. During the 1960s however, users were limited and computing resources were few. The username and password were initially devised only to service a time-sharing operating system at the Massachusetts Institute of Technology [1]. As technology progressed however, the computing end-user population began to reach beyond the walls of universities - eventually into the realms of business, public service and homes around the world. This expansion however, gave way to an increased vulnerability for security exploitation.

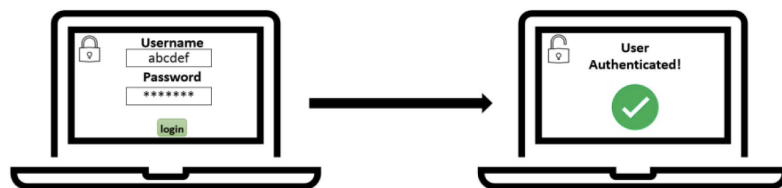


FIGURE 1.1: Single factor authentication through a username and password

With over six decades passing since the creation of the password, it is now estimated that the average user could hold as many as 130 online accounts [3], with users estimated to have a limited pool of only 16 unique identities, usernames and passwords [4]. This clearly shows a shortfall from the guidance of bodies such as The National Cyber Security Centre, which recommend unique passwords for each account [5, 6]. Despite this well-documented password guidance [7], Google found in a survey of 3000 Americans, that 65% of users reused passwords across at least one other or more sites [7].

Similarly, it was found that pushes for more complex passwords led to users reusing said complex passwords, or variations of it, across multiple platforms [7]. The infamous 87 gigabyte data breach in 2019 known as *Collection #1* compromised 1,160,253,228 unique combinations of email addresses and passwords [8], demonstrating the fragility of poor password hygiene. The growing sophistication of attackers has expanded alongside software implementations such as brute-force and keylogging attacks, to social engineering techniques such as phishing which have been reported to be at an all time high at the close of 2022 - with almost five million attacks logged [9]. Not only this, but in the industrial space, insider threats give rise to a unique vulnerability where unattended machines can be commandeered for identity theft attacks where a bad actor accesses a machine if the true user has left it unlocked (known as tailgating [10]).

This all highlights the vulnerability surrounding password authentication. With the growing number of accounts we have to manage as individuals, constructing secure, unique, yet memorable passwords for each account is neither a convenient nor a reasonable task to expect from end-users anymore. This *arms race* between authentication and cybercriminals has led to other, more complex and integrated authentication techniques which rely on more than one factor to reliably validate user identity. Combining these methods is known as two-factor (2FA), or in the case of multiple mediums - multi-factor authentication (MFA) [11](*Fig.1.2* [1]).

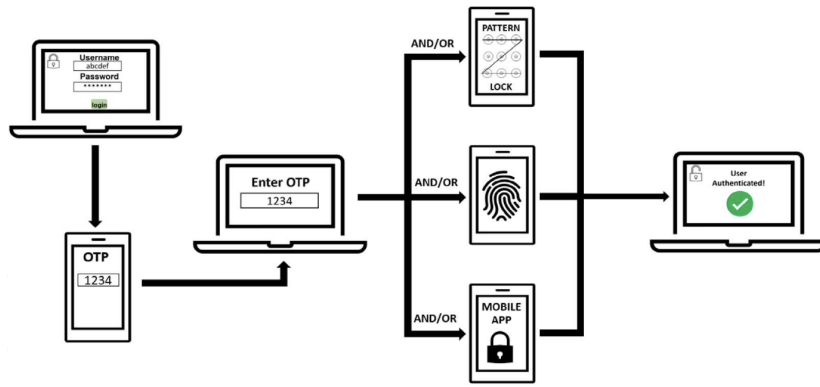


FIGURE 1.2: An illustration of the protective measure behind 2FA/MFA is if a user's password is compromised the probability of second or third authentication medium being compromised as well is less likely, acting as a safe-guard to access.

The opening sections of this report and subsequent literature review seek to briefly outline how behavioural biometrics and deep learning models such as 1-Dimensional Convolutional Neural Networks (1D-CNN) could offer the cybersecurity discipline an opportunity in keeping user data secure.

1.2 Authentication

The National Institute of Standards and Technology defines authentication as:

Verifying the identity of a user, process, or device, often as a prerequisite to allowing access to a system's resources. [11]

Broadly speaking, authentication technology can be divided into three modalities; knowledge-based authentication (KBA) (*something you know*), inheritance-based authentication (IBA) (*something you are*) and possession-based authentication (PBA) (*something you have*) [1, 2].

IBAs rely on physical or behavioural biometrics of the user as a form of identification. IBA has a unique advantage over KBA and PBA in that its the mode of authentication that should, in theory, always be in possession of the user and unique to them [1]. KBAs like passwords including Password Managers and secret phrases all share the same vulnerability where compromise of the knowledge can lead to the compromise of security. Whereas, PBAs have the vulnerability of being misplaced, stolen or damaged. PBAs are now more widely available through the ubiquitous presence of smartphones, often with an accompanying application or even the reception of a one-time passcode (OTP) via SMS or some such similar service [1]. PBA is now often used in combination with a KBA in 2FA/MFA to increase security. However, although viewed favourably for their usability, security and acceptance compared to traditional passwords [12], PBAs still have the vulnerability of needing physical possession of the security artefact for service access. Biometrics, ideally speaking, cannot be stolen nor do they have an information-artefact that can be compromised, but are ever-present with the user as a part of their anatomy or their innate behaviour.

Studies have taken place on a variety of biometric authenticators. Most commonplace commercially are fingerprint scanners, facial recognition and voice authentication [1]. These methods have been spoofed previously by images of a user's face [13] or copies of a fingerprint from a surface the user has touched in the past [14]. However, this technology has improved. Fingerprint scanners now often requiring conductivity or 'live' detection from an oximeter [14], and facial recognition now having a combination of infrared light and depth analysis [15]. The *arms race* of authentication however, has led to these being spoofable through 3D-printed masks [13] and finger replicas [14] with alterations to circumnavigate these additional technologies [2]. This has steered researchers to explore more advanced biometrics which would be harder to replicate such as palm vein mapping [16, 17], acoustic ear canal mapping [18] and electrocardiogram patterning [14]. Although yielding quite highly in terms of accuracy in research, these

are regarded as impractical in most settings due to their need for specialised additional hardware. This also brings in the ethical debate of personal biometric data storage, whereby data which could identify an individual through their biological features could be compromised. Unlike a password, once this happens it is irreplaceable and potentially compromised indefinitely. Biometric data leaks have already occurred [\[19\]](#) and perhaps demonstrate a vulnerability concern for this technology going forward.

1.3 Behavioural Biometrics - Mouse Dynamics

Behavioural Biometrics provide a solution to these challenges by leveraging individuals' intrinsic behaviours, such as gait [20, 21], keystroke dynamics [22], smartphone usage [23], and mouse or trackpad interactions (mouse dynamics) [24], among others [25]. Specifically, mouse dynamic authentication unobtrusively gathers data using standard equipment integral to all graphical user interface (GUI) enabled operating systems (OS). Virtually every user, barring those with specialised hardware for accessibility, operates with either a trackpad or mouse. Mouse dynamics are less invasive compared to traditional biological biometrics, offering enhanced data privacy [25]. The biometric data collected, while unique, is not identifiable in the same way as a fingerprint, for example. Current literature [26] documents various *signatures* in mouse dynamics (*see figure 1.3*), more recently including nuanced data such as speed, acceleration, jerk, jitter, angle, and velocity [27, 28]. This complex *signature* provides a potential authenticator with a readily available and data-rich biometric. However, with processing large swathes of data, bears an additional problem - processing that data.

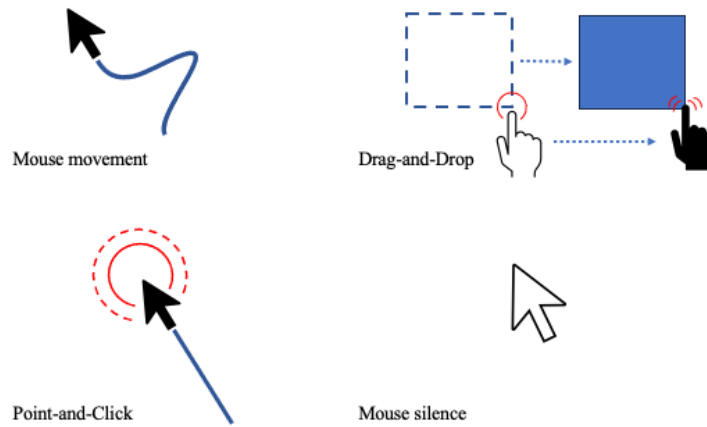


FIGURE 1.3: Author generated graphic of typical mouse behaviour: Mouse-movement: planar movement across the two-axes of freedom, Drag-and-Drop: movement beginning with a click down of a mouse button and ends with the button release, Point-and-Click: Mouse Movement followed by a click or double-click, Mouse silence: a period of no movement.

1.4 Deep Learning and Convolutional Neural Networks

Deep Learning models, particularly multilayered Neural Networks, effectively address the challenge of processing extensive mouse dynamics data for authentication and identifying unique user behaviours. These models, adept at learning from large data volumes [29], offer an efficient alternative to manual feature computation required by traditional neural networks.

Deep Learning models have proven their efficacy across various domains through a *training* process that adjusts internal parameters or *weights* based on input data and corresponding target outputs, thus enabling accurate new data classification [30]. This ability, referred to as *generalisability*, results from the model's internal feedback mechanism and weight adjustment process.

Convolutional Neural Networks (CNNs), particularly effective for grid-like data, have become prominent in areas such as computer vision tasks [31], speech processing [32], and biometric recognition [33]. Their efficiency originates from parameter sharing and hierarchical feature learning, facilitated by convolutional and pooling layers that capture spatial patterns.

One-dimensional CNNs (1D-CNNs) show considerable promise for continuous authentication via mouse dynamics, efficiently representing temporal sequences of timestamped coordinates $((x, y \text{ coordinates}), \text{time})$ [34]. Studies [35, 36] have reported high accuracy rates, underscoring their suitability for such applications. 1D-CNNs are inherently designed for time-series data, efficiently detecting and learning temporal patterns through convolutional layers that apply filters across the data's temporal dimension [30]. Moreover, Kiranyaz *et al.* highlights how 1D-CNNs offer an alternative more lightweight model compared to their two-dimensional counterparts [37]:

...an image with $N \times N$ dimensions convolve with $K \times K$ kernel will have a computational complexity $O(N^2 K^2)$ while in the corresponding 1D convolution (with the same dimensions, N and K) this is $O(NK)$. This means that under equivalent conditions (same configuration, network and hyper parameters) the computational complexity of a 1D CNN is significantly lower than the 2D CNN. [37]

1D-CNNs' convolutional layers excel in identifying local dependencies and invariant features, crucial in learning individual user interactions within mouse dynamics [30]. Additionally, 1D-CNNs are more computationally efficient than 2D-CNNs for one-dimensional

data, requiring fewer parameters and resources, thus facilitating faster training and suitability for real-time applications [37].

Figure 1.4 [38], illustrates how CNNs analyse input data values, process them through various layers, and identify increasingly complex features. This hierarchical approach enables accurate feature or object recognition in the input data.

In summary, deep learning, particularly through CNNs, has revolutionised multiple fields by enabling models to automatically learn data representations. The use of 1D-CNNs in mouse dynamics research is a promising development for creating accurate and efficient authentication systems [28], effectively addressing the challenges of manual feature computation in time-sensitive security contexts. Moreover, the benefits of data privacy, unobtrusive data collection and accessibility in regards to mouse dynamics previously discussed, combined with the processing capability of 1D-CNNs, outlines a field worth further exploration. This research report will seek to validate a proof-of-concept for the use of mouse dynamics as a potential form of user authentication under organic conditions (routine computing activities including but not limited to; internet surfing, shopping, playing videogame, word processing etc.) utilising a 1D-CNN model as a multi-label classifier.

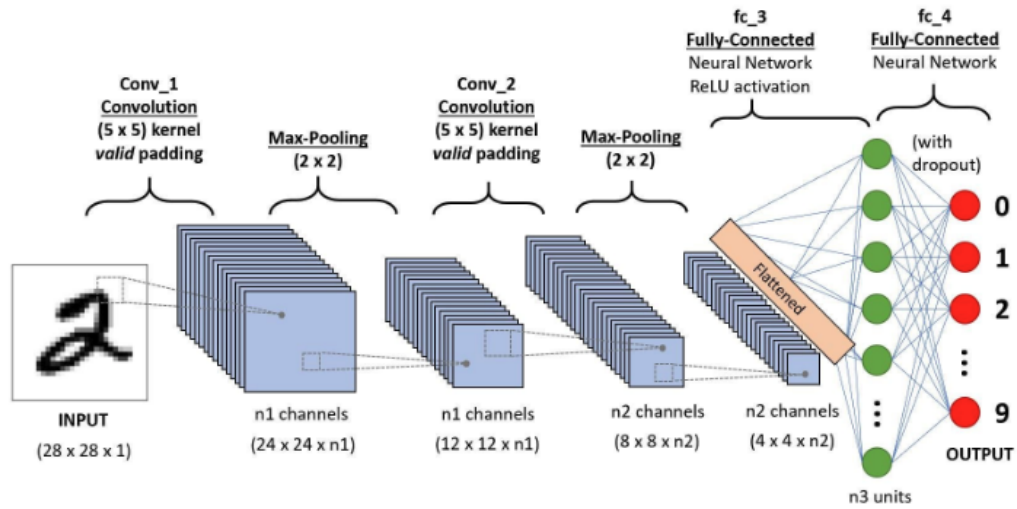


FIGURE 1.4: An illustrated example from Saturn Cloud of a CNN in the example of computer vision. Layers of the neural network detect values of pixels, if the threshold mark is passed this will stimulate neurons to the next layer which, in this example are used to identify patterns. This leads to the correct identification of the number from a handwritten input.

Chapter 2

Literature Review

2.1 Research in Mouse Dynamics for Authentication

Verifying a user by their mouse operation has been a field of research for over two decades now [24]. Everitt and McOwan in 2003 [24] investigated the authentication efficacy of several users ($n=41$) who *wrote* password signatures with their mouse. They evaluated the success of their system in regards to False Acceptance Rate (FAR) and False Rejection Rate (FRR). Yielding an FAR of 8.5% and FRR of 0.2% (achieved with 5 attempts at the signature). Despite these impressive results, variance in user signature was found even on genuine attempts but the signature served as a compelling safe-guard when used in combination with a password. However, when imposters were shown the signature to simulate it for access the FAR still yielded 4.4%.

$$FAR = \frac{FalsePositive}{FalsePositive + TrueNegative}$$

$$FRR = \frac{FalseNegative}{FalseNegative + TruePositive}$$

Everitt and McOwan's findings nonetheless showed promise for further research in using a mouse for authentication. Falling short arguably, due to its lack of practical advantage over username and password access and that the FAR of users increased almost three-fold when trialled on random words compared to words chosen by the user [24], a method employed to overcome compromised signatures. Research in this field moved on from signatures to then analysing mouse dynamics as a behavioural biometric for authentication.

Equal Error Rate (EER) is commonly seen in the literature to evaluate the performance of binary classification systems. It provides the measure of the intersection where the FAR is equal to that of the FRR. As it is threshold-independent, EER provides information relating to the trade-off without the need to tune decision thresholds. It's application however, is less common in multi-class classification. As a result, across the literature many other metrics are used to illustrate the performance of a model (*see table 2.1*).

Metric	Purpose
Accuracy	Measures the proportion of true results (both true positives and true negatives) among the total number of cases examined.
Precision	Assesses the proportion of true positive identifications among all positive identifications made by the model.
F1 Score	Provides a balance between Precision and Recall, useful when seeking a balance between recognising true positives and minimising false positives.
ROC-AUC	Represents a model's ability to distinguish between classes, with a higher score indicating better distinction capabilities.
TPR (Recall)	The proportion of correctly predicted positive observations to all observations in the actual positive class.
TNR (Specificity)	The proportion of correctly predicted negative observations to all observations in the actual negative class.
FAR	The proportion of incorrect positive predictions to all actual negative observations.
FRR (Miss rate)	The proportion of incorrect negative predictions to all actual positive observations.

TABLE 2.1: Performance metrics and their definitions

Gamboa and Fred [39] looked into mouse movement analysis utilising machine learning. A sequential forward selection algorithm which categorised the behaviour of a user, based on feature extraction from their mouse *strokes* (the set of points between two mouse clicks) when completing a memory game. *Features* are the input variables that are fed into a machine learning model, they are data attributes that the system uses to learn patterns and make predictions [40]. 63 features were extracted including velocity, acceleration, jerk and jitter and were tuned for each of the 50 users of the data set. Gamboa and Fred demonstrated that the more mouse strokes collected in user testing data the greater the improvement to the EER (1 stroke, 50 strokes and 100 strokes yielded an EER of 48.9%, 2% and 0.7% respectively) [39]. However, this posed a similar problem to Everitt and McOwan where the authentication technique was not only static,

the enrollment was highly specialised and would require a large amount of data input from users to calibrate.

Pusara and Brodley investigated continuous authentication (what they described as re-authentication) through mouse movements [41] across 18 users utilising decision trees and personalised parameterisation. Data was collected from users on a Windows OS system running *internet explorer* until 10,000 unique cursor locations were collected. Despite yielding an impressive FAR and FRR of 1.75% and 0.43% respectively [41]. The study has been criticised for its lack of standardisation and a public data set for peer-review [42]. Not only this but feature extraction was customised to each user leading to significant research overhead [41].

Ahmed & Traore investigated the introduction of a neural network to analyse user mouse dynamics [26], specifically a feed-forward multilayer-perceptron network. Low-level mouse events were captured with a detector unit to formulate patterned actions split into five features of mouse behaviour (movement speed, movement direction, action type, travelled distance and elapsed time). Ahmed & Traore's detection unit converted raw data of mouse action categories to build a *profile* of a user and use this as a model reference point to classify users [26]. Ahmed & Traore collected mouse dynamic data from 22 users over a period of nine weeks, averaging 12 hours and 55 minutes of input per user. With this model, Ahmed & Traore were able to demonstrate an average FAR and FRR of 2.46%. In their paper, Ahmed & Traore highlighted mouse dynamics potential in the field of continuous (dynamic) authentication:

Mouse biometrics may also be used for dynamic authentication or identity confirmation in cases where the actions of an active user raise some suspicions. Intrusion detection is the most suitable of the possible applications of this biometric. The passive and continuous user data acquisition and monitoring capabilities of mouse dynamics biometrics make it appropriate for the efficient detection of masqueraders. [26]

Despite their research's effectiveness in regards to FAR and FRR, Ahmed & Traore highlighted the limitations of their bespoke detector were factors such as screen resolution, mouse design and workstation speed all of which impeded the model [26]. The detector, also requiring a significant overhead to compute features and extract data.

Shen *et al.* [42] captured *organic* (collected during users routine computing activities) user's data over 30 sessions of 30 minutes. With each user ($n = 28$) contributing around 90,000 actions. Shen *et al.* demonstrated best results with a one-class Support Vector Machine (SVM) detector and managed to achieve a FAR and FRR of 0.37% and 1.12%

after 3000 mouse operations, equating to around 30 minutes of use [42]. Shen admitted that ideally authentication would need to happen within 10 minutes to be viable in preventing serious harm from a intrusive attack. At the 10-minute mark a FAR of 2.75% and FRR of 3.39% was achieved. Shen *et al.* [42] also highlighted a mouse dynamic authentication system shortfall for real-world applications.

It should be also noted that, in most previous research, samples from both the impostors and the legitimate user are required for training the classification or detection model. This is not realistic in practice since it is impossible to collect enough data to represent all of the imposters. [42]

Nevertheless, in scenarios where potential attackers are pre-identified, such as in an industrial context, a multi-class authentication system could effectively classify users on machines they are unauthorised to access, for example in the instance of a tailgaiting attack. Moreover, the resources required for feature extraction, is a common limitation of early mouse dynamic research. Deep Learning however, has the ability to automate the learning of feature sets and expands the capability of classification on a greater amount of user data [30].

2.2 Deep Learning in the field of Authentication with Mouse Dynamics

Chong *et al.* (2018) [28] conducted a study on mouse dynamics using deep learning, specifically investigating 1D-CNNs and 2D-CNNs, with an SVM serving as a control model. The study reported Receiver Operating Characteristic Area Under the Curve (ROC-AUC) scores of 0.90, 0.96, and 0.87 for the 1D-CNN, 2D-CNN, and SVM, respectively. This research was carried out using two publicly available datasets: the Balabit dataset [43] and the TWOS dataset [44]. Furthermore, Chong *et al.* explored the use of multi-label joint classifiers, which were found to reduce training time while effectively yielding multiple binary classifiers within a single multi-label model [28].

In their subsequent study, Chong *et al.* [36] expanded their research to include transfer learning applied to a 2D-CNN model, a 1D-CNN, a long short-term memory (LSTM) model, and various combination models such as CNN-LSTM. While these models were trained as binary classifiers, the study also included evaluations of session-based and sequence-based learning approaches against the Balabit and TWOS datasets. Session-based training, trained models on sessions as a whole, whereas sequence-based training, trained models on multiple sequences of 130 data events in length. The study found that while 2D-CNNs learning from visual inputs were the top performers in three scenarios, the 1D-CNN emerged as the best classifier in the fourth scenario. The results of this study are summarised in table 2.2, which is adapted from the original research [36].

TABLE 2.2: Summary of research in the investigation of CNNs as authentication binary classifiers for mouse dynamic data, assessed on the Balabit dataset (Chong *et al.* 2020)

Test Session	Neural Network	Mean ROC_AUC
Legal (session-based)	1D-CNN	0.87
Legal (session-based)	2D-CNN	0.99
Legal (sequenced-based)	1D-CNN	0.90
Legal (sequenced-based)	2D-CNN	0.96
Legal + Illegal (session-based)	1D-CNN	0.84
Legal + Illegal (session-based)	2D-CNN	0.91
Legal + Illegal (sequence-based)	1D-CNN	0.90
Legal + Illegal (sequence-based)	2D-CNN	0.78

Results showed that 1D-CNNs were particularly successful in classifying *genuine* users and *imposters* in sequence-based sessions in the Balabit dataset, although it is worth mentioning that 2D-CNNs consistently performed well through all scenarios and both datasets.

Siddiqui *et al.* followed up their previous research [45], gathering mouse dynamic data from an expanded data set of 40 users over a 20 minute period while playing *Minecraft* and the inclusion of deep learning neural network models [27]. The focus of this subsequent study was to compare machine learning algorithms such as SVM and k-nearest neighbour algorithm (KNN) against three deep learning models (1D-CNN, 2D-CNN and an ANN). The research also assessed the performance of binary and multi-class classifiers. The deep learning models were shown to demonstrate a 92% accuracy (ANN as a multi-label classifier) and 85% (1D-CNN as a binary classifier) whereas the SVM achieved 60% [27]. This echoes the findings of Chong *et al.* regarding the potential power of deep learning models in respect to mouse dynamics as an authentication medium.

The summary of the studies outlined, specifically to the field of deep learning and their results with 1D-CNN models is shown in table 2.3.

TABLE 2.3: Summary of research in the investigation of 1D-CNNs as authentication classifiers for mouse dynamic data

Paper	Neural Network	Accuracy	Mean ROC_AUC	Dataset
Siddiqui <i>et al.</i> [27]	1D-CNN	85.73%	-	Novel
Chong <i>et al.</i> [36]	1D-CNN	-	0.90	Balabit [43]
Chong <i>et al.</i> [36]	1D-CNN	-	0.85	TWOS [44]
Chong <i>et al.</i> [28]	1D-CNN	-	0.90	Balabit [43]
Chong <i>et al.</i> [28]	1D-CNN	-	0.77	TWOS [44]

2.3 Study rationale and contribution to research

This research aims to advance the understanding of deep learning models, particularly 1D-CNNs, in the context of user authentication through mouse dynamics. It introduces a novel dataset capturing *organic* mouse usage outside controlled laboratory settings, reflecting a diversity of user system setups. The study explores the efficacy of a 1D-CNN in multi-label classification under organic conditions, assessing its ability to classify users during everyday activities of their choice. This in essence reflects the diverse application of user's systems in a wider context in order to assess if such a classifier is capable of distinguishing classes through a high variability of input data. This approach deviates from previous research in controlled lab environments or restricted to standardised tasks [26, 27, 39, 41]. While honouring research previously which has used *organic* collections [42].

The project serves as a proof-of-concept for onward application of mouse dynamics as a potential dynamic authenticator, which would have an application in intrusion

detection. This research also seeks to contribute to the increased need in diversity of publicly available mouse dynamics data.

Drawing inspiration from Shen *et al.*'s design of *organic* capture [42], this study seeks to apply this to a sequence-based learning 1D-CNN model as demonstrated in Chong *et al.* [36] whilst building upon the multi-label and binary classifiers used in the literature [27, 28, 36]. This study aims to minimise feature extraction and train models indiscriminately across various user activities, terminal specifications, and environments. The objective deliverables for this project are outlined in table 2.4.

TABLE 2.4: Study Objective Deliverables

Deliverables	Description
A single-file executable Mouse Logger	A functional software script to collate user mouse dynamic data, deployable across both Windows and Mac OS.
An experimental software service	A functional software service that can process user's raw data and train and evaluate a 1D-CNN model to classify mouse dynamic data to its attributed user.
A 1D-CNN model	A saved best performance model of a 1D-CNN validated against unseen data from this research's dataset
Experimental Write-up and associated Software Documentation	Documentation presenting the results of the software artefacts operation, capabilities and design.
Public Data Set (with user permission)	A dataset on mouse dynamics made publicly available to contribute to future research in the field.
Research Outcomes	Contributions to the body of knowledge in the field of authentication with mouse dynamics and deep learning models.

Chapter 3

Methodology

3.1 Hypothesis

The hypothesis for this research is as follows:

Employing a 1D-CNN Deep Learning model to classify organic mouse dynamic data as a behavioural biometric, will bolster security by correctly classifying users generalised data with an overall accuracy of 95% and user-based classification with a precision of 95% and FAR of 1%.

Other data metrics collated such as F1-Score, Receiver operating characteristic - area under the curve (ROC-AUC), TNR, TPR, and FRR will also feature in the subsequent Results and Discussion to assess the model's performance and provide a more descriptive picture. However, due to the nature of authentication where FAR is more detrimental than FRR and the nature of the multi-label model being evaluated on a balanced dataset [46] - it has been assessed that accuracy of the overall model and precision and FAR for user-specific measures will be the primary indicators for the acceptance of this hypothesis.

3.2 Research Design

A mixed-group of participants ($n=7$) were recruited. Across the duration of the study however, three participants withdrew, leaving the final study sample size at ($n=4$). A minimum commitment of four weeks was requested from each participant in order to gather sufficient mouse dynamic data from the cohort which included both users of Mac and Windows OS.

The aim of the project was oriented towards implementing a comprehensive mouse dynamics analysis framework. Initially, a mouse logger script was developed and deployed to gather raw mouse data. Subsequently, this data was aggregated and processed by a dedicated service, which would partition the data into training, testing, validation, and unseen datasets, extract features and organise the data into sequences, each comprising 100 mouse events. Utilising the first three datasets, a multi-label classification 1D-CNN was trained and its preliminary performance evaluated through a series of 25 iterative experiments. The most promising model from these iterations was then subjected to further evaluation against the unseen dataset. This step was crucial for assessing the model’s capability to generalise across a broader spectrum of user data. The outcome of this evaluation included user-specific confusion matrices and a set of both overall and class-specific performance metrics (see table 3.1) for detailed analysis.

Metric	Purpose
Accuracy	Measures the proportion of true results (both true positives and true negatives) among the total number of cases examined.
Precision	Assesses the proportion of true positive identifications among all positive identifications made by the model.
F1 Score	Provides a balance between Precision and Recall, useful when seeking a balance between recognising true positives and minimising false positives.
ROC-AUC	Represents a model’s ability to distinguish between classes, with a higher score indicating better distinction capabilities.
TPR (Recall)	The proportion of correctly predicted positive observations to all observations in the actual positive class.
TNR (Specificity)	The proportion of correctly predicted negative observations to all observations in the actual negative class.
FAR	The proportion of incorrect positive predictions to all actual negative observations.
FRR (Miss rate)	The proportion of incorrect negative predictions to all actual positive observations.

TABLE 3.1: Performance metrics and their definitions

3.2.1 Data Collection

The study’s first phase entailed data collection from users’ organic mouse activity on their personal computer. In order to capture this organic data, participants were encouraged to use their machine as they normally would and for the knowledge of their

participation in the study not to consciously influence their behaviour - users were not given assigned tasks or asked to complete sessions of a set length. This allowed, the capture of organic data as close as possible to genuine user interaction with their machine.

Upon briefing and acceptance of participation (*discussed in further detail in Participant Considerations*), data collection began. As outlined in table 3.2, six data points were captured from users at a maximum rate of 1 event per 40 millisecond interval. In addition, buttons pressed and released were logged with all events having an accompanying timestamp (*see table 3.3*), intermittently saved to comma-separated values (CSV) files stored on the user's local machine.

Data captured	Rationale
Timestamp	• Captured (yyyy-MM-dd HH:mm:ss.SSS) for the purpose of organising data chronologically to ensure subsequent sequence formatting and user behavior was aligned. • To allow for the calculation of features such as speed and acceleration.
Mouse Event	• Captured whether the event logged was a Mouse Movement or Mouse Click event.
Button	• Captured what button (if any) were involved in the event. Consisted of left, middle, right, x1, and x2.
Action	• Captured the action associated with the event and button. Consisted of button.pressed and button.released.
Position X	• Captured the x-axis position of the cursor on the user's display at the point of the event. Used to calculate distance between subsequent movements, which in turn led to calculations of additional features such as Jitter and Angle.
Position Y	• Captured the y-axis position of the cursor on the user's display at the point of the event. Used to calculate distance between subsequent movements, which in turn led to calculations of additional features such as Jitter and Angle.

TABLE 3.2: Summary of data captured and rationale.

Timestamp	Event	Button	Action	X	Y
2023-10-11 21:50:20.718	Mouse Movement	<null>	<null>	856	553
2023-10-11 21:50:20.784	Mouse Click	Button.left	Pressed	857	553
2023-10-11 21:50:20.858	Mouse Click	Button.left	Released	855	553
2023-10-11 21:50:20.926	Mouse Movement	<null>	<null>	855	554
2023-10-11 21:50:21.002	Mouse Movement	<null>	<null>	856	552

TABLE 3.3: A sample of a mouse activity log

In total, 16 defined features were extracted from the users mouse dynamic data. Nine of which (is_mouse_movement, is_mouse_click, is_button_left, is_button_right, is_button_middle, is_button_x1, is_button_x2, is_button_pressed, is_button_released) were binary indicators, two were the positional coordinates (Position_X and Position_Y). The remaining five features (Distance, Speed, Acceleration, Angle and Jitter) were calculated during data processing.

3.2.2 Data Processing

Due to variations in user data available (*see figure 3.1*), User D's data set was dropped from the sample. The raw data was then sourced into a data directory within the Convolutional Mouse Authentication Network (CONMAN) service. Data was loaded in preparation for manipulation. Values were preprocessed and encoded (*see code insert 3.1*). The datasets of each user were then split (*see code insert 3.2*) into four data dictionaries; training (350,000 mouse events), validation (100,000 mouse events), testing (100,000 mouse events) and unseen (>100,000 mouse events, user variable) segments. Features were then computed for each users now split datasets (see Methodology Equations: Feature Computation) (*see code insert 3.3*).

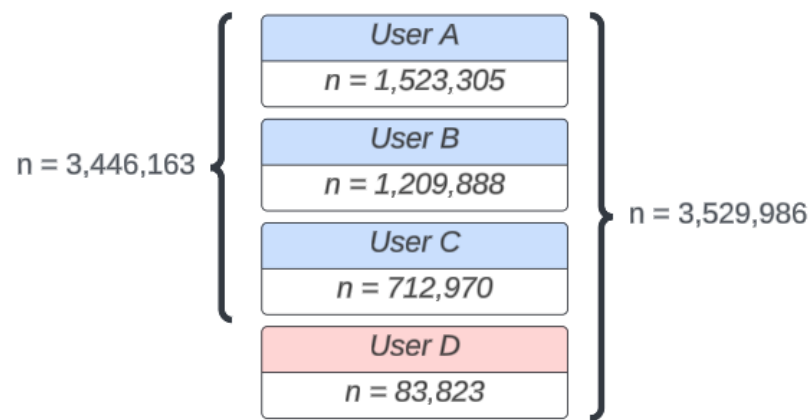


FIGURE 3.1: Author-generated graphic summarising the number of user-captured mouse events. User D was removed from this study due to its small sample size in comparison the other three users. Down-sampling was opted for in order to provide a balanced but simplified classification pipeline of the data.

```

1  def preprocess_data(self, data):
2      """
3      Preprocesses the raw data by converting timestamps and encoding
4      categorical columns.
5
6      Args:
7          data (pd.DataFrame): The raw data to preprocess.
8
9      Returns:
10         pd.DataFrame: The preprocessed data.
11     """
12     data = data.copy()
13     data['Timestamp'] = pd.to_datetime(data['Timestamp'], format='%Y-%m
14     -%d %H:%M:%S.%f')
15     data['Time_Diff'] = data['Timestamp'].diff().dt.total_seconds().
16     fillna(0)
17
18     event_encoded = pd.get_dummies(data['Event'], prefix='Event')
19     button_encoded = pd.get_dummies(data['Button'], prefix='Button')
20     action_encoded = pd.get_dummies(data['Action'], prefix='Action')
21
22     all_buttons = ['Button_Button.left', 'Button_Button.middle', '
23     Button_Button.right', 'Button_Button.x1',
24     'Button_Button.x2']
25     # Not all buttons feature in all datasets, so these are
26     standardised across users.
27     for button_type in all_buttons:
28         if button_type not in button_encoded.columns:
29             button_encoded[button_type] = 0
30
31     data = pd.concat([data, event_encoded, button_encoded,
32     action_encoded], axis=1)
33     return data

```

CODE INSERT 3.1: The preprocess_data method in the data_utils.DataProcessor class

```

1  def data_splitter(self, data):
2      """
3          Splits the data into training, validation, testing, and assigns the
4          remaining data to the unseen dataset
5          based on a flag.
6
7          Args:
8              data (pd.DataFrame): The preprocessed data to split.
9              use_unbalance_unseen_dataset (bool): Flag indicating whether to
10             use all remaining data for the unseen dataset.
11
12             Returns:
13                 Tuple: Data split into training, validation, testing, and
14                 unseen datasets.
15
16             The method determines the size of the training, validation, and
17             testing datasets based on predefined
18             sizes in the configuration (config.py). If
19             use_unbalance_unseen_dataset is True, all remaining data is assigned
20             to the unseen dataset. Otherwise, the unseen dataset is created
21             with a predefined size, similar to the other datasets.
22             """
23             data = data.sort_values(by=['Timestamp'])
24
25             # Determines size of the splits based on the config.py values for
26             sequence_length and sizes.
27             train_size = (config.train_size * config.sequence_length)
28             val_size = (config.valid_size * config.sequence_length)
29             test_size = (config.test_size * config.sequence_length)
30
31             # Split data into training, validation, and testing datasets.
32             train_data = data.iloc[:train_size]
33             val_data = data.iloc[train_size:train_size + val_size]
34             test_data = data.iloc[train_size + val_size:train_size + val_size +
35             test_size]
36
37             if self.use_unbalance_unseen_dataset:
38                 # Assign all remaining data to the unseen dataset.
39                 unseen_data = data.iloc[train_size + val_size + test_size:]
40             else:
41                 # Allocate a predefined size to the unseen dataset.
42                 unseen_size = (config.unseen_size * config.sequence_length)
43                 unseen_data = data.iloc[train_size + val_size + test_size:
44                 train_size + val_size + test_size + unseen_size]
45
46             return train_data, val_data, test_data, unseen_data

```

CODE INSERT 3.2: The `data_splitter` method in the `data_utils.DataProcessor` class

```

1  def compute_features(self, data):
2      """
3      Computes additional features based on the preprocessed data.
4
5      Args:
6          data (pd.DataFrame): The data to compute features on.
7
8      Returns:
9          pd.DataFrame: The data with computed features.
10     """
11     data['Distance'] = np.sqrt(data['Position_X'].diff().pow(2) + data[
12     'Position_Y'].diff().pow(2))
13     data['Speed'] = data['Distance'] / data['Time_Diff']
14     data['Acceleration'] = data['Speed'].diff() / data['Time_Diff']
15     data['Angle'] = np.arctan2(data['Position_Y'].diff(), data[
16     'Position_X'].diff()) * 180 / np.pi
17     data['Jitter'] = data['Distance'].diff().abs()
18
19     data.replace([np.inf, -np.inf], np.nan, inplace=True)
20     data.fillna(0, inplace=True)
21     return data

```

CODE INSERT 3.3: The `compute_features` method implementing the feature computation equations in the `data_utils.DataProcessor` class

This partitioning and feature computation was done prior to scaling in order to minimise the risk of data leaking and to prevent data overlap. Min-max scaling [47] of the training dataset was then executed, with the scaling being applied to the remaining three datasets of each user. This ensured scaled data of each user being applied to their respective datasets. Once the processes above were completed for all users, the four data segments (training, testing, validation and unseen) were compiled with their counterparts from other users, with the conversion of sequence data to float32 formatting and label data to categorical formatting to be received by the model. The training set underwent shuffling prior to this conversion (*see code insert 3.4*) alongside corresponding class labels with concerted efforts to ensure pairing. Shuffling preserved the time-series integrity within each sequence but altered the sequence order. Shuffling was implemented as a precaution to prevent the model from learning the sequence order, thereby ensuring that it focuses on learning the features within each sequence.

```

1  def shuffle_sequences_labels(self, sequences, labels):
2      """
3      Shuffles the given sequences and their corresponding labels in
4      unison.
5
6      Args:
7          sequences (np.array): Array of data sequences.
8          labels (np.array): Array of labels corresponding to the
9          sequences.
10
11      Returns:
12          Tuple: Shuffled sequences and labels.
13
14      Ensures that the sequences and labels are shuffled in the same
15      order to maintain correct pairing.
16      """
17      assert len(sequences) == len(labels)
18      indices = np.arange(len(sequences))
19      np.random.shuffle(indices)
20      shuffled_sequences = sequences[indices]
21      shuffled_labels = labels[indices]
22      return shuffled_sequences, shuffled_labels

```

CODE INSERT 3.4: The `shuffle_sequences_labels` method sequence and label shuffling uniformly

This data was then received by the `ModelCreator` class along with configuration data (see code insert 4.2) to instantiate a model, utilising the training and validation sets for training with subsequent testing data for initial evaluation. To ensure reliability of the results, this process was repeated 25 times, wherein a fresh model was trained against the data pool. When models were trained they were then saved by the service. At the end of the 25 iterations the best saved model was extracted, evaluated by the lowest validation loss and was then used to evaluate against the unseen data set verifying its ability to generalise to unseen data.

3.3 Evaluation Criteria and Statistical Analysis

When analysing the results of the data, performance metrics were calculated by the CONMAN's `ResultsController` class and its associated `results_utils` module and output to CSV and the console. The classes utilised the `sklearn.metrics` package [48] for determining the accuracy score, precision score, recall score, F1 score, ROC-AUC score and confusion matrices. The choice of these metrics (*see table 3.1*) allowed for a descriptive image of the model's performance to be outlined. These metrics were selected due their current representation in the field of behavioural biometrics in order to act as a point of comparative evaluation to current research, as outlined previously in the literature review.

3.3.1 Methodology Equations: Feature Computation

As mentioned, feature computation was executed by the CONMAN service, with the `DataProcessor` class and `compute_features` function (*see code insert 3.3*), utilising the `numpy` [49] and `pandas` [50] Python packages (all of which is discussed further in the CONMAN Service documentation, *see appendix A*). These features upon calculation were appended to the dataset for each mouse event to prepare the data for model training.

$$Distance_i = \sqrt{(Position_{X_i} - Position_{X_{i-1}})^2 + (Position_{Y_i} - Position_{Y_{i-1}})^2}$$

Distance: the computation of the Euclidean distance between consecutive mouse positions, quantifying the amount of movement the mouse covered over time.

$$Speed_i = \frac{Distance_i}{Time_{Diff_i}}$$

Speed: the distance travelled over the time difference, representing the interval between consecutive mouse positions.

$$Acceleration_i = \frac{Speed_i - Speed_{i-1}}{Time_{Diff_i}}$$

Acceleration: the change in speed divided by the time difference, computing the rate of the change of speed.

$$Angle_i = \arctan 2(Position_{Y_i} - Position_{Y_{i-1}}, Position_{X_i} - Position_{X_{i-1}}) \times \frac{180}{\pi}$$

Angle: the direction of the mouse path relative to the horizontal axis, articulated the trajectory of consecutive mouse movements.

$$Jitter_i = |Distance_i - Distance_{i-1}|$$

Jitter: the absolute change in the distance between points, indicating the variability of positioning in a movements distance.

These equations standardised features across the classes as part of the processing set-up, all being appended to the data frame. This enabled the service to take advantage of extra descriptive features all the while being able to remove a resource overhead in explicit feature composition whereby the defining of mouse movements, drag and drops, point-and-clicks etc. are no longer needed, a benefit of deep learning networks outlined by Hu *et al.* [51].

3.3.2 Methodology Equations: Model Performance Metrics

Metric scores for model performance overall were computed through the use of the **sklearn** package with the relevant imports from the **metrics** module within the CON-MAN service. The metrics captured for model performance are those previously shown in table 3.1. The equations for each of which are as follows:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

$$\text{Precision} = \frac{TP}{TP + FP}$$

$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

$$\text{ROC AUC} = \int_{x=0}^1 \text{TPR}(x) dx$$

$$\text{TPR} = \frac{TP}{TP + FN}$$

$$\text{TNR} = \frac{TN}{TN + FP}$$

$$\text{FAR} = \frac{FP}{FP + TN}$$

$$\text{FRR} = \frac{FN}{FN + TP}$$

For calculating user-specific data, TNR, TPR, FAR, FRR were calculated using the variables obtained from each users confusion matrix (*see figure 3.2*) and a Area under the Curve - One versus Rest (AuC OvR) was performed for each user using the **sklearn.metrics** module. This effectively splits the multi-label classifier into one binary classification problem per class [48]. TNR, TPR, FAR and FRR were implemented in the code through the **compute_individual_metrics** method (*see code insert 3.5*) and its associated methods (*see code insert 3.6*) within the **Metric_Calculator** class when called by the **Results_Controller**.

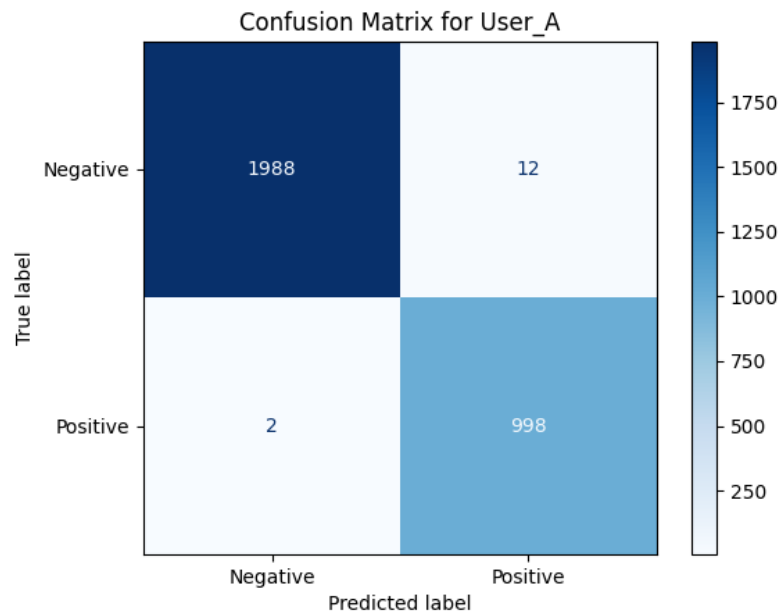


FIGURE 3.2: An example of a confusion matrix from a model iteration. Model 6, User A.

```

1  def compute_individual_metrics(self, y_true, y_pred, y_pred_probs):
2      """
3      Calculates performance metrics for each class, including confusion
4      matrices.
5
6      Parameters:
7      y_true (array): True labels of the test dataset.
8      y_pred (array): Predicted labels by the model.
9      y_pred_probs (array): Prediction probabilities by the model.
10
11     Returns:
12     Tuple: A dictionary of metrics for each class and a dictionary of
13     confusion matrices for each class.
14     """
15
16     metrics_per_class = {}
17     confusion_matrices = {}
18
19     # Calculate metrics and confusion matrix for each class.
20     for i in range(self.n_classes):
21         user_label = list(self.user_labels.keys())[i]
22         current_test_labels = (y_true == i).astype(int)
23
24         cm = confusion_matrix(current_test_labels, y_pred == i)
25         confusion_matrices[user_label] = cm
26
27         user_metrics = {
28             "Precision": precision_score(current_test_labels, y_pred ==
29 i, zero_division=0),
30             "F1": f1_score(current_test_labels, y_pred == i),
31             "ROCAUC.OvR": roc_auc_score(current_test_labels,
32 y_pred_probs[:, i]),
33             "TPR": self.true_positive_rate(cm),
34             "TNR": self.true_negative_rate(cm),
35             "FAR": self.false_alarm_rate(cm),
36             "FRR": self.false_rejection_rate(cm)
37         }
38         metrics_per_class[user_label] = user_metrics
39
40     return metrics_per_class, confusion_matrices

```

CODE INSERT 3.5: The called method compute_individual_metrics

```

1  # Additional methods to calculate specific rates from confusion
   matrices.
2  def true_positive_rate(self, confusion_matrix):
3      TP, FN = confusion_matrix[1, 1], confusion_matrix[1, 0]
4      return TP / (TP + FN) if TP + FN > 0 else 0
5
6  def true_negative_rate(self, confusion_matrix):
7      TN, FP = confusion_matrix[0, 0], confusion_matrix[0, 1]
8      return TN / (TN + FP) if TN + FP > 0 else 0
9
10 def false_alarm_rate(self, confusion_matrix):
11     FP, TN = confusion_matrix[0, 1], confusion_matrix[0, 0]
12     return FP / (FP + TN) if FP + TN > 0 else 0
13
14 def false_rejection_rate(self, confusion_matrix):
15     FN, TP = confusion_matrix[1, 0], confusion_matrix[1, 1]
16     return FN / (FN + TP) if FN + TP > 0 else 0

```

CODE INSERT 3.6: The additional calculation methods called by compute_features

3.4 Participant Considerations

As mentioned, the research project required extensive data collection of mouse dynamic data while users operated their personal computers. Mouse-logging, which is pivotal in accessibility, cybersecurity, and user experience research, is also utilised in malware. With this in mind, the software was developed with transparency about its purpose and operational mechanisms.

Participants were briefed regarding the mouse-logging programme and the intention of the study, along with the opportunity to ask questions and to review the source code for the sake of complete transparency. This was all done in the aid of gaining informed consent from users. Participants were also informed that, should they wish, their taking part in the study could be withdrawn at any time along with all the data gathered to be safely discarded. A briefing document (*see appendix B*) was provided to all participating users to read and sign.

Given the invasive potential of mouse logger software, data collection adhered strictly to the necessary scope of the research. The data collected contained no identifiable information but it was reviewed for sanitation and stored securely. Detail regarding the data captured during this phase along with rationale is outlined in table 3.2.

Chapter 4

Design & Implementation

4.1 Procedure for specification requirements, user stories and design documentation

4.1.1 Mouse Logger & The CONMAN Service

A Software Requirements Specification (SRS) document (*see appendix C*) was developed first as a formal requirements framework for the project inspired from a standardised template, outlined in IEEE 830-1998 [52]. This SRS document provided a detailed and structured description of both software artefacts intended functionalities, performance, interface, and design constraints.

As this project was a sole endeavour, the SRS served as a single point of truth for the requirements and subsequent implementation of the project. Furthermore, in my comprehensive role encompassing development, design, and product ownership, the necessity for a versatile and effective development approach was key. As a result, principles of Extreme Programming (XP), an Agile software development methodology, were employed. XP's focus on iterative development, constant feedback (albeit internal), and flexibility proved to be highly compatible with the project's requirements, especially with the development of the CONMAN service.

Following the drafting of the SRS, rough epics (*see appendix D*) were developed in order to structure the project into modular software objectives. Serving as the sole researcher with multiple responsibilities, the structuring of initial epics facilitated the management of this expansive and multifaceted project. Drafting of initial user stories and spikes [53], derived from these epics (*see appendix D*), functioned as modular goals that aligned with the SRS. This framework enabled the seamless transition between the roles of end-user,

tester, product owner, software engineer, machine learning engineer, and data analyst [53] — each critical to the project’s completion. This breakdown enhanced the clarity and deliverability of each requirement within the epics, aligning closely with the project’s scope.

Following this, the Software Design Documentation (SDD) was drafted (*see appendix E*). This too inspired by formal documentation [54]. It systematically described the software’s architecture and design, providing a comprehensive overview of the system’s structure, components, and their interactions. In industry, the use of formal documentation methods in the SDD ensures that the design principles and architectural choices are communicated effectively, enabling developers to translate the requirements into a coherent and efficient software architecture [54]. It also serves as a critical tool for future maintenance and scalability efforts, allowing new developers to understand the system’s design efficiently. Although as mentioned, this project was a sole-endeavour these benefits of a comprehensive SDD were limited to myself only, but were nonetheless helpful throughout the project’s duration. The SRS and SDD, essentially mapped to the designed components and overall software interplay. Further allowing the ‘overlay’ of the initial user stories to satisfy requirement with a design blueprint.

This modularised design approach naturally lent itself to the adoption of an Object-Oriented Programming (OOP) paradigm [55, 56] and was a cornerstone of the development methodology. This approach facilitated code maintainability and adaptability, which was essential given the project’s scale. Moreover, my familiarity with OOP and the awareness of its benefits in allowing for the reuse of code components and its ability to simplify the creation of diverse functionalities without duplicating effort, not only conserved valuable development time but also streamlined the overall project timeline, aligning with the strategic focus on time efficiency and resource management.

Informed by the specified requirements, tools, user stories, and the adoption of an Object-Oriented Programming (OOP) approach, an initial high-level design of the mouse logger was developed (*see figure 4.1*). This logic mapping provided a clear blueprint of the functional requirements and their interplay within the broader context of the script. Utilising this schematic as a guide, the SDD was expanded to partition the logic flows into segmented classes with associated methods including detailed descriptions of each component. Leading to further refinement of user stories. Unit testing and benchmarking then steered the implementation of the components, in order to ascertain the accuracy of the software and to assess the impact of the logger on end-user’s systems. Additionally, the SDD comprehensively listed all necessary libraries as development progressed, affirming their relevance and roles in fulfilling the project’s objectives.

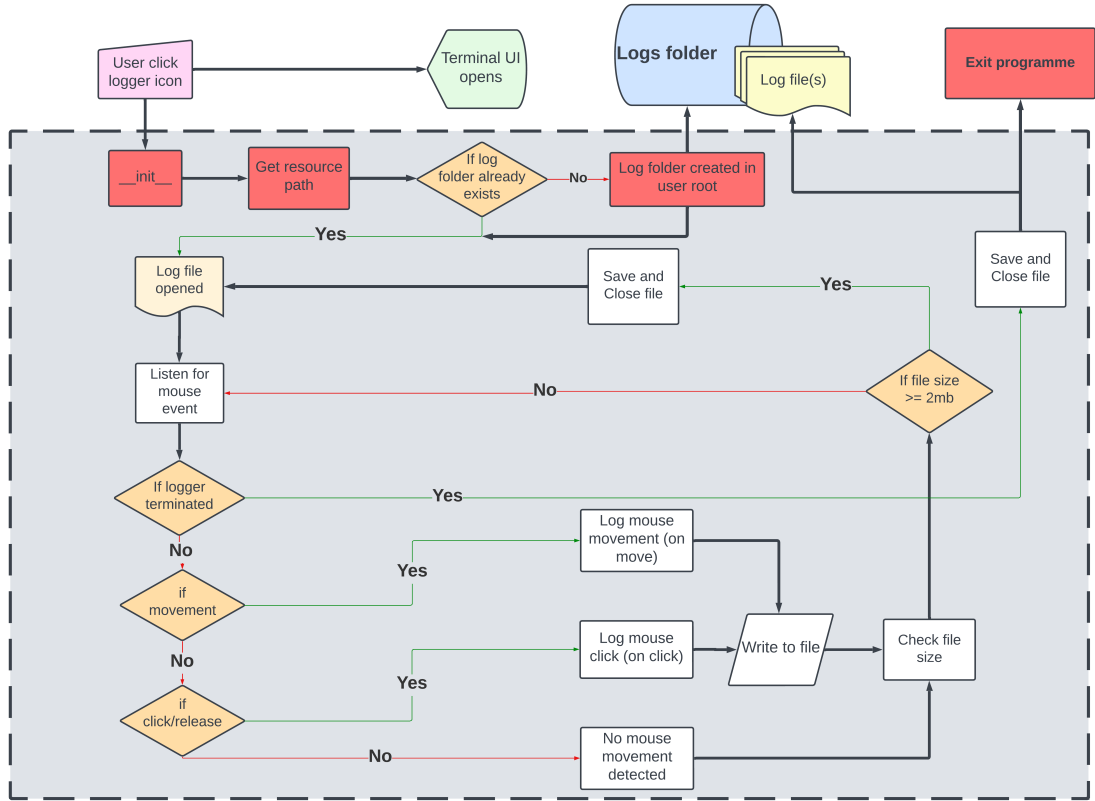


FIGURE 4.1: Author-generated initial design of mouse-logger logic

Similar to the approach taken with the mouse logger, the partitioning of requirements for the CONMAN system, with overlaid user stories, was divided into functional components. This identified that the service would require four principal operational aspects:

1. Pre-processing data from its raw format into a split and featured set.
2. Creating sequenced data, scaling data, shuffling and compiling the data pool.
3. Creating, training, evaluating, saving and loading models.
4. Processing and calculation of statistical metrics post model performance.

The preliminary concept design of this system was drafted (see Figure 4.2), with these requirements managed by the `DataController`, `ExperimentController`, `ModelCreator` and the `ResultsController`. This approach led to the detailed design of specific components and associated utility classes, with each `Controller` class having dedicated utility packages: `data_utils`, `experiment_utils`, `results_utils`. The `ModelCreator` class was created to handle processes related to the 1D-CNN, such as model creation, storage, and loading. Initially, the `main` class directed the application's flow, but this was eventually superseded by the `ExperimentRunner` class.

The development of the CONMAN service involved a series of iterative prototypes and manual testing. Given the service's focus on a novel field of research for me, the development process presented more complex and unpredictable challenges compared to the mouse logger. This complexity often led to unforeseen issues, requiring extensive research, problem-solving efforts, and considerable time dedicated to additional exploratory spikes.

The presence of these 'unknown unknowns' led to a departure from a strictly methodical approach. Consequently, the development of the CONMAN service frequently relied on experimental and exploratory techniques to address emerging challenges. This situation led to unit testing being relegated to a secondary consideration, often viewed as an afterthought due to time constraints. In hindsight, this prioritisation proved to be detrimental to the development timeline, as it reduced the efficiency and predictability of the software development process.

The adoption of an exploratory development strategy, where solutions were iteratively tested and refined based on their effectiveness, became a key aspect of the project. While this approach was less structured, it was essential for navigating the complexities and uncertainties associated with integrating machine learning into the service. It facilitated flexibility and rapid adaptation in response to the unique challenges encountered during the development of the CONMAN service. However, the retrospective recognition of the importance of consistent unit testing highlights the need for a more balanced approach in future projects, integrating both exploratory techniques and rigorous testing methodologies to enhance development efficiency and software reliability.

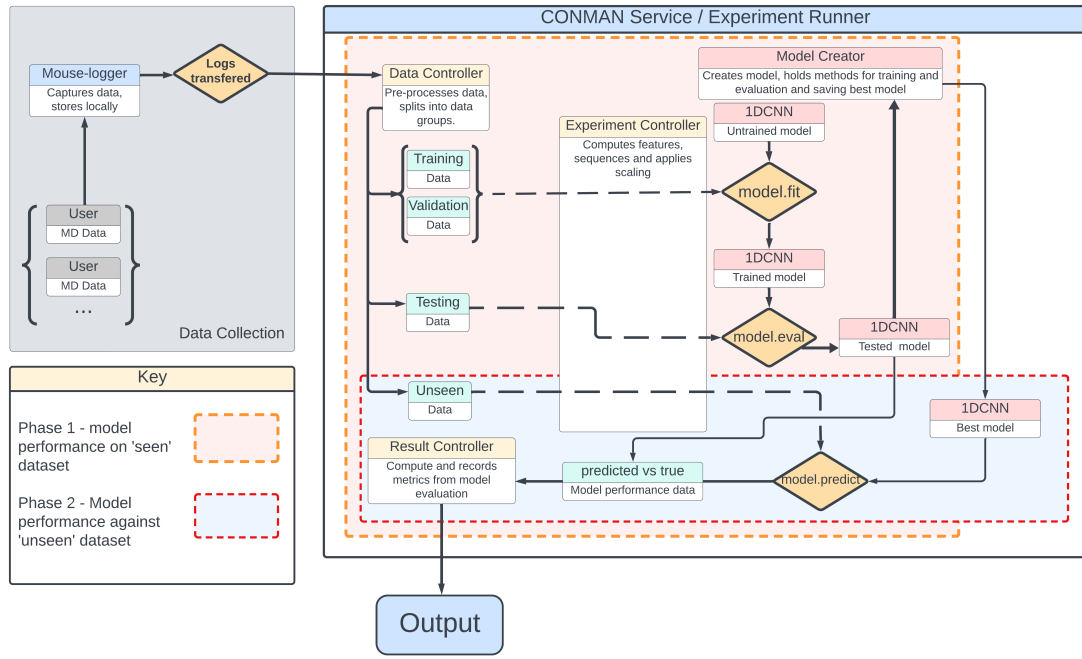


FIGURE 4.2: Author-generated initial design of complete system

The use of formal methods reinforced consistency and gave clarity to the project's goals. Also significantly contributing to the overall quality and reliability of the development process. These practices were essential in mitigating risks, preventing scope creep, and ensuring that the final software product aligned closely with the study objectives. Although not employed uniformly or consistently they were essential in the design and development process of the software artefacts.

4.1.2 1D-CNN Model Design

The initial design of the model was inspired by Siddiqui *et al.* [27]. The model input would comprise of multiple linear stacks of layers, each represented as two-dimensional arrays [sequence length, number of features]. Specifically, each layer contains 100 chronological mouse events, forming a sequence with 16 features at each event, with the output being one of three classes. The choice of a Sequential model was driven by its single tensor input and single tensor output architecture [57]. Sequential models are also known for their standardised architecture and user-friendly interface, making them a suitable choice for those new to machine learning [40, 57]. As a multi-label model, a softmax activation function was employed for the output layer to generate a probability distribution across the three classes. This design allows the model to train on and identify a user's intrinsic mouse dynamic features within a 100-event window.

The decision to use a sequence length of 100 events was informed by the literature review [36] and was considered a round number appropriate for partitioning data into dictionaries during down-sampling. The rest of the model’s architecture, including the number of convolutional layers, number of filters, kernel size, hyperparameters, choice of global max pooling versus max pooling layer(s), and the configuration of the Dense layer, was largely determined heuristically. This approach, lacking a rigorous empirical methodology, will be further discussed and addressed in the subsequent discussion.

4.2 Materials & Tools

4.2.1 Hardware

The development of this project was undertaken on an Apple MacBook Pro, 2GHz Quad-Core Intel Core i5, Intel Iris Plus Graphics 1536 MB, 16 GB 3733 MHz LPDDR4X.

4.2.2 Python

As mentioned, for this investigation two artefacts of software were developed. The first, a single-file executable Python programme, mouse logger (*documentation available in appendix F*) and the second being the Python project which included the training and testing of the 1D-CNN model and it’s associated scripts to process data in its various forms, known as the CONMAN service (*see appendix A for service documentation*).

The mouse logger deliverable and associated benchmark tests were a single project developed in Python. Initially, considerations were made to develop the mouse logger in Java due to familiarity with the language and the availability of the **Abstract Window Toolkit** package which allows for the logging of mouse events, captured by the **MouseEvent** class, an extension of **InputEvent** [58]. This package and associated classes are commonly used in Java in the development of GUIs [58], however they are restricted to what is known as the component geometry. This limits the software’s capability in recording global mouse event activity. This can be overcome by bypassing the abstraction of the Java Virtual Machine, but would require bespoke solutions across different OS, leading to a more verbose software solution.

As a result of this, Python was deemed more appropriate because of its more *direct* accessibility to a system’s OS [59]. Simple library modules such as `os`, `sys` [59], [60] and `subprocess` [61] as well as third-party libraries such as `pynput` [62] simplify file manipulation and python’s libraries bare the complexity of differences between OS. This

presented a unified interface to developing a software solution that was targeted to Mac and Windows OS [59]. This allowing a more straightforward development process. As mentioned, the `pynput` library was used. `Pynput` allows a programmer to control and monitor input devices, of which there are two supported, with their respective sub-packages; `pynput.mouse` and `pynput.keyboard` [62]. Not only does this library provide management of the backend for the platform it is running on [62], but it would have allowed for a simplified expansion into software that could key-log as well, demonstrated by Da Silva Beserra *et al.* [63] as a potential avenue for future research.

The data pipeline involved the comprehensive processing of large volumes of raw data sourced from CSV files. This process encompassed transforming the data into an organised data frame structure compatible with several stages of data processing and manipulation (feature computation, data splitting, sequence creation). A key aspect of the pipeline was the simultaneous management of user labels and data, for preparation for tensor operations required by the 1D-CNN model. And finally, assessing the model's performance and presenting the results in both CSV format and graphically. Python proved highly compatible with a variety of data structures which allowed for comprehensive data management - including data dictionaries, tuples, and data frames, and facilitated tensor operations through libraries such as `NumPy` [49]. Additionally, `Matplotlib` [64] offered robust capabilities for data visualisation. Python's extensive range of data structures was instrumental in maintaining data integrity and ensuring the pipeline's functionality.

The choice of Python (v3.9.6), bolstered by its extensive library ecosystem and inherent cross-platform compatibility, aligns well with the prevalent trends in Machine Learning development, with researchers already cited using Python as a capable mouse-logging scaffold [45]. In addition to these capabilities, Python's synergy with Object-Oriented Programming (OOP) principles further justified its selection for developing the CON-MAN service. Finally, Python was found to be an accessible language in relation to machine learning frameworks, compatible with both TensorFlow and PyTorch.

This strategic choice ensured coherence across both segments of the software, facilitating a unified development approach.

4.2.3 Development Environment

To leverage Python's full potential in the context described above, JetBrains's Integrated Development Environment (IDE), PyCharm, was employed. PyCharm's robust features and Python-centric environment offered an integrated solution for coding, debugging,

and testing, further enhancing the development efficiency of both the mouse logger and the CONMAN service.

4.2.4 PyInstaller

The mouse logger, when completed, was packaged through PyInstaller(v.5.13.2), an open-source utility programme which grants the deployment of Python projects as a single-file executable [65]. PyInstaller copies the script for executing, the required packages and modules, as well as the Python interpreter, collating them all as a single file. This permitted the distribution of the mouse logger to participant's systems without them needing the understanding of, or the software required, to run a Python script.

4.2.5 TensorFlow and Keras API

In this study, TensorFlow (v2.15.0), an open-source machine learning library [66], was employed alongside its high-level API, Keras (v.2.15.0) [67]. This pair, simplified the design and deployment of the 1D-CNN. As shown in code insert 4.1, layer definition is accessible, as are the possible tweaking of the hyperparameters in the model, which were further streamlined through system configuration variables in the `Config.py` file (see code insert 4.2).

The project faced a constraint in terms of time, which influenced the choice of TensorFlow as the foundational framework for model development. Keras, in particular, facilitated a more streamlined construction process of the neural network, aligning with the project's timeline.

PyTorch, another framework highly regarded within the academic realm, presented a more complex learning pathway and was found to be preferred for research-intensive applications. However, considering that the project's initial ambitions were to integrate the machine learning model into a deployable application, TensorFlow's production-oriented ecosystem was deemed more suitable. The user-friendly nature of TensorFlow with Keras combined with the option for a fluid transition from model development to application deployment resulted in it being selected as the library of choice for this project.

```
1 def create_model(sequence_length, no_of_features, no_of_users):
2     """
3     Creates and compiles a 1D Convolutional Neural Network (CNN) model for
4     user authentication using mouse dynamics.
5
6     Args:
7         sequence_length (int): Length of the input sequences.
8         no_of_features (int): Number of features in the input data.
9         no_of_users (int): Number of users (classes) for the model to
10        classify.
11
12    Returns:
13        Sequential: A compiled Keras CNN model.
14    """
15    model = Sequential()
16    model.add(Conv1D(filters=32, kernel_size=3, input_shape=(
17        sequence_length, no_of_features)))
18    model.add(Activation('relu'))
19    model.add(tf.keras.layers.GlobalMaxPooling1D())
20
21    model.add(Dense(60, activation='relu', kernel_regularizer=l1_l2(l1=
22        config.l1, l2=config.l2)))
23    model.add(tf.keras.layers.Dropout(config.dropout))
24    model.add(Dense(no_of_users, activation='softmax'))
25    model.compile(optimizer='adam', loss='categorical_crossentropy',
26                  metrics=['accuracy'])
27    return model
```

CODE INSERT 4.1: The create model method implementing Keras layer definitions

```
1 """ Configuration for running the experiment environment """
2 user_data = 'config_data_directories.json'
3
4 """ Model Config """
5 model_store = "ModelHub/model_store"
6 no_of_users = 3
7 sequence_length = 100
8 no_of_features = 16
9 batch_size = 34
10 max_epochs = 100
11 early_stopping_patience = 5
12 minimum_improvement = 0.01
13 # Regularisation
14 l1 = 0.001
15 l2 = 0.001
16 dropout = 0.05
17
18
19 """ Number of sequences to be created from mouse events """
20 train_size = 3500
21 valid_size = 1000
22 test_size = 1000
23 unseen_size = 1500
24
25 """ This boolean flag can be set to False to accept 1500 unseen_size
26     sequences rather than all (True) """
27 use_unbalanced_unseen_data = True
28
29 """ Number of iterations per experimental run """
30 REPETITIONS = 25
```

CODE INSERT 4.2: The config.py file which acted as a global variable configuration center

Chapter 5

Results

5.1 Description of the finished working product

5.1.1 Mouse Logger and The CONMAN Service

Both software artefacts were successfully completed, meeting their specifications as stipulated in the SRS (*see appendix C*). The mouse logger was effectively deployed across both Mac and Windows OS. Feedback gathered from end-users was positive with no issues or errors raised - underlining the script's effective and user-friendly requirement. Although, feedback from users did highlight that each user on one occasion or more forgot to initiate the service when using their machine. Notably, users also stated that the mouse logger demonstrated no discernible interference with machine performance, effectively capturing continuous mouse dynamic data as outlined earlier in the methodology with minimal disruption. Benchmark testing was undertaken before deployment and were found to be satisfactory (*see table 5.1*). During the data collection process however, a minor anomaly was observed: 13 sessions from the aggregate user data pool yielded no data. While this did not appear to obstruct the overall data capture process, these *empty* sessions were identified as outliers and were subsequently excluded from the final results computation. Data collection yielded 146 csv files totalling 169.42 MB in size across four users amassing 3,529,986 captured mouse events. With User D being dropped from results analysis, the final total was 3,446,163 captured mouse events over 130 files totalling 165.45 MB in size. The CONMAN Service was successfully executed across the datasets to yield data for model training and evaluation and service output was as specified in the methodology regarding overall model performance and user-centric specifics which were successfully delivered.

Intensity	Avg. CPU	Peak CPU	Avg. RAM	Peak RAM
Low	0.63%	3.90%	0.21%	0.23%
Medium	0.89%	3.60%	0.23%	0.23%
High	1.26%	3.70%	0.19%	0.23%

TABLE 5.1: Average and Peak CPU and RAM usage expressed as a percentage of the total availability at different intensities, specifications of these tests can be seen in the mouse logger documentation.

5.1.2 Model Architecture

The final model architecture can be seen in figure 5.1. TensorFlow’s ‘Adam’ optimiser was implemented using its default setting on the free parameters, with a base learning rate of 0.001 a β_1 of 0.9, β_2 of 0.999 and an $\epsilon = 1 \times 10^{-7}$ [68]. Categorical cross entropy was employed as the loss function for the multi-class classification models, with batch size = 34. Dropout probability was 0.05, with L1 and L2 deregularisers both equalling 0.001, batch normalisation was not implemented in the final structure. An early-stopping callback was implemented, assessing validation loss to its minimum value, with a patience of 5 epochs and a minimum δ improvement of 0.01. A model-checkpoint callback was also used in order to save best weights within model performances, again scoped to the minimum validation loss of the model. Each model iteration was run for a maximum of 100 epochs, the final range used by all 25 models was between 18 and 29. The best model running 28 epochs. These callbacks allowed for model’s to be adequately fitted without risk of over-training occurring, thereby preserving computing resources.

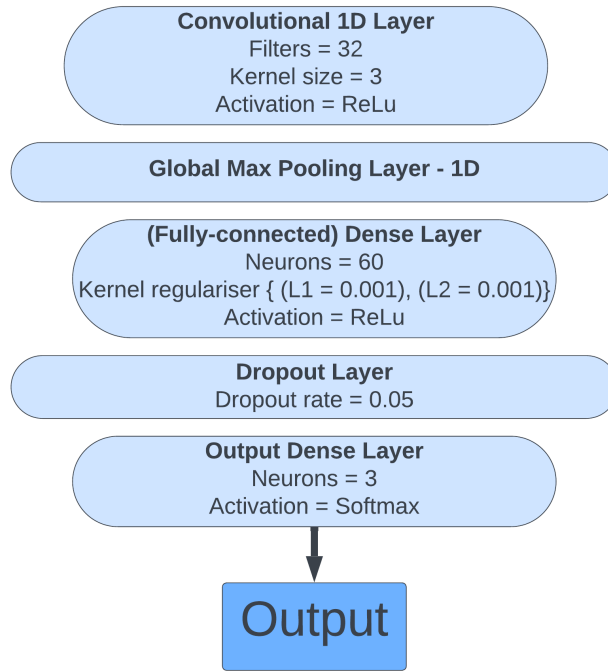


FIGURE 5.1: The final architecture of the 1D-CNN model

5.2 Model Results

Model training performance was assessed through training vs. validation accuracy and loss. There were little variations in the learning curves of these models (*see figures 5.2, 5.3, 5.4*), with models generally fitting well with little evidence of under or over-fitting. The validation accuracy was consistently higher than the training accuracy and validation loss less than training loss. This is believed to be due to the implementation of regularisation techniques which are present at the time of training but not validation or testing [69]. These techniques allow for better generalisation and prevent the model from over-fitting to the training dataset [40].

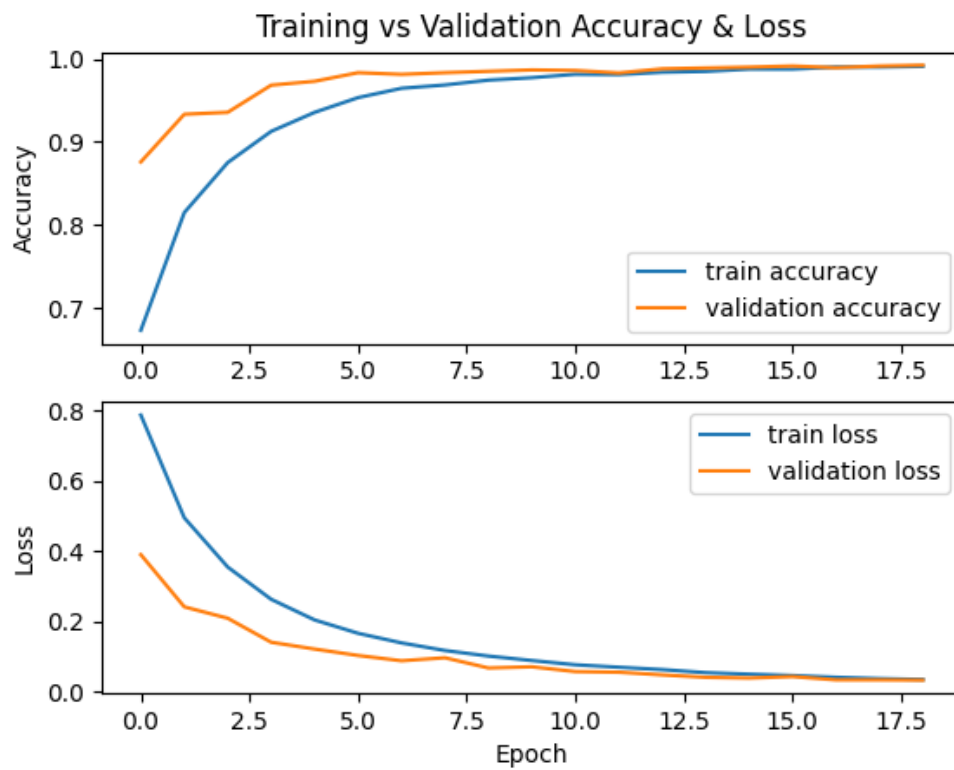


FIGURE 5.2: An example of an Training vs Validation plot for both accuracy (top) and loss (bottom) - Model 1, epochs = 18.

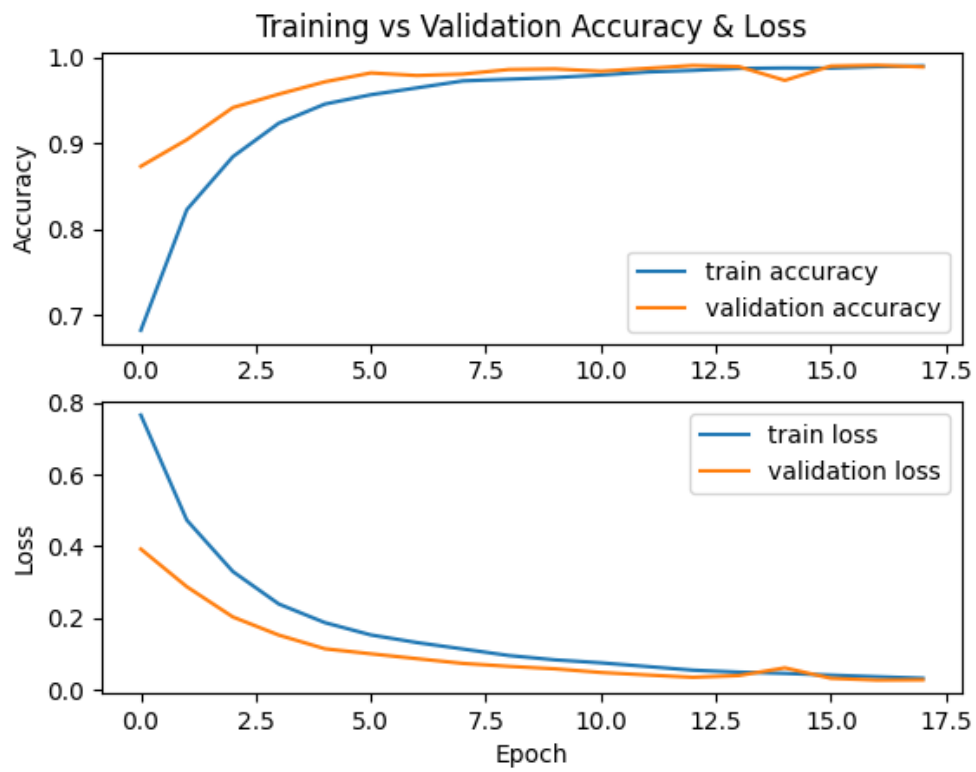


FIGURE 5.3: An example of an Training vs Validation plot for both accuracy (top) and loss (bottom) - Model 4, epochs = 17.

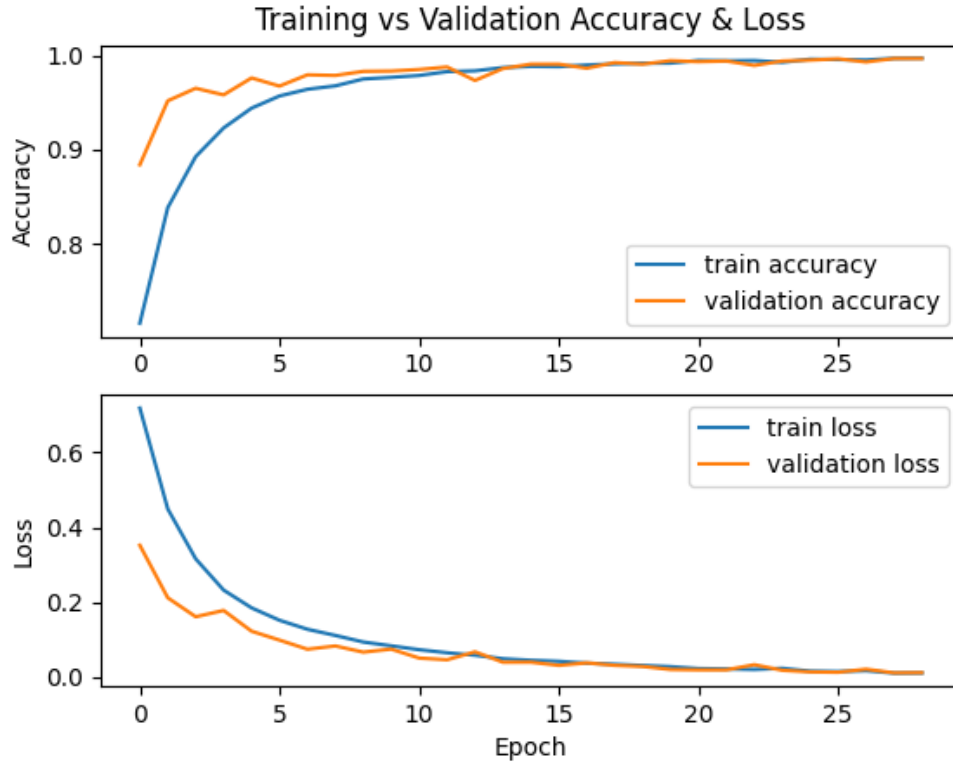


FIGURE 5.4: An example of an Training vs Validation plot for both accuracy (top) and loss (bottom) - Model 22, epochs = 28.

The average testing performance of all models can be seen in table 5.2.

TABLE 5.2: Overall Testing Performance across Models - rounded to 3sf.

Stat	Accuracy	Precision Macro	Recall Macro	F1 Macro	AuC OvR
MEAN	0.997	0.997	0.997	0.997	1.000
STD	0.001	0.001	0.001	0.001	2.89 E-08
MEDIAN	0.997	0.997	0.997	0.997	1.000
MIN	0.995	0.995	0.995	0.995	1.000
MAX	0.999	0.999	0.999	0.999	1.000

This table and the box and whisker plot (*see figure 5.5*) demonstrates that models yielded high accuracy, precision, recall and F1-Score, demonstrating a very tight distribution around the mean values, indicating a high level of consistency in model performance across different iterations, with all models returned metrics surpassing 0.99. The median and mean are exact when considered to 3 significant figures (0.997), for each metric demonstrating a balanced distribution of results. The variability of quartile one is slightly larger across metrics but not enough to warrant a skewed distribution. The datasets for testing are balanced so little variation would be expected in precision

and recall. Overall it demonstrates the models performed very well, however, note it is acknowledged that these models appear to have performed *exceptionally* well, which may be due to limitation in the dataset as whole, later discussed.

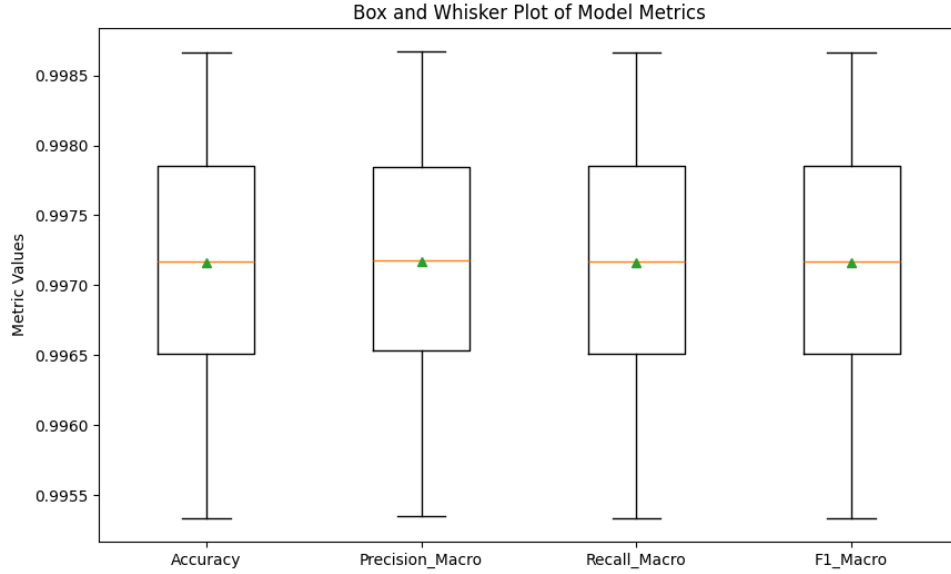


FIGURE 5.5: A box-and-whisker plot of the overall model performance by metrics (excluding AoC OvR, (n=25) (Green triangle = Mean), (Orange line = Median)

Looking at the results from table 5.3 it shows that during testing, models were adeptly able to classify all users with 99% precision, User B being the label which was classified with the highest precision.

TABLE 5.3: Mean Testing Performance across Users

User	Precision	F1	AUC	TPR	TNR	FAR	FRR
A	0.994	0.996	0.999	0.998	0.997	0.003	0.002
B	0.999	0.999	0.999	0.999	0.999	0.001	0.001
C	0.998	0.996	0.999	0.995	0.999	0.001	0.005

5.3 Best Model (Shea) vs. Unseen Dataset

As mentioned, the unseen dataset was to act as a generalisability validation for the best performing model across the iterations. As this dataset was compiled from each user it also gives an indication on the performance at intrusion detection. The best performing was model iteration 22 (now on referred to as 'Shea'). Shea expressed excellent results when classifying the unseen dataset compiled from the three user's data (*see table 5.4*).

TABLE 5.4: The Shea model vs. Unseen Data Performance Metrics

Accuracy	Precision_Macro	Recall_Macro	F1_Macro	ROC_AUC_OvR
0.996	0.992	0.994	0.993	0.999

Overall model performance was exceptional, with every metric exceeding a result of 0.99. The AUC-OvR achieving 0.999 with almost perfect classification between labels (*see figure 5.6*). When looking on the user-centric level (*see table 5.5*), it is clear to see that Shea performed best when classifying user B, with a TPR and TNR achieving 99% and FAR and FRR at 0.001%. Much like the results observed in training data, Shea also expressed more difficulty (although only slight) with discerning between users A and C in a few instances, with an average FAR of 0.003% observed across these two labels. For a more detailed view of the Shea model's performance in regards to user classification, the confusion matrices of users A, B and C can be seen in figures 5.7, 5.8 and 5.9, respectively.

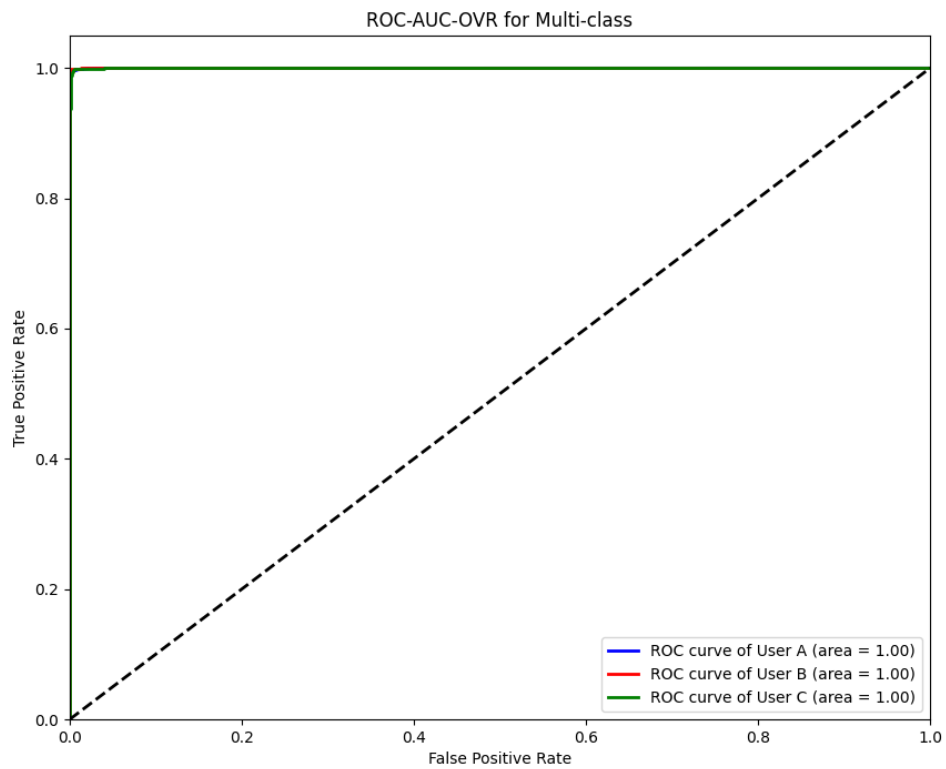


FIGURE 5.6: The RoC AuC OvR plot for Model 22 against the unseen dataset.

TABLE 5.5: The Shea model vs Unseen Users Performance Metrics

User	Precision	F1	ROC_AUC	TPR	TNR	FAR	FRR
A	0.997	0.997	0.999	0.996	0.996	0.004	0.004
B	0.999	0.999	0.999	0.998	0.999	0.001	0.001
C	0.981	0.985	0.999	0.988	0.998	0.002	0.002

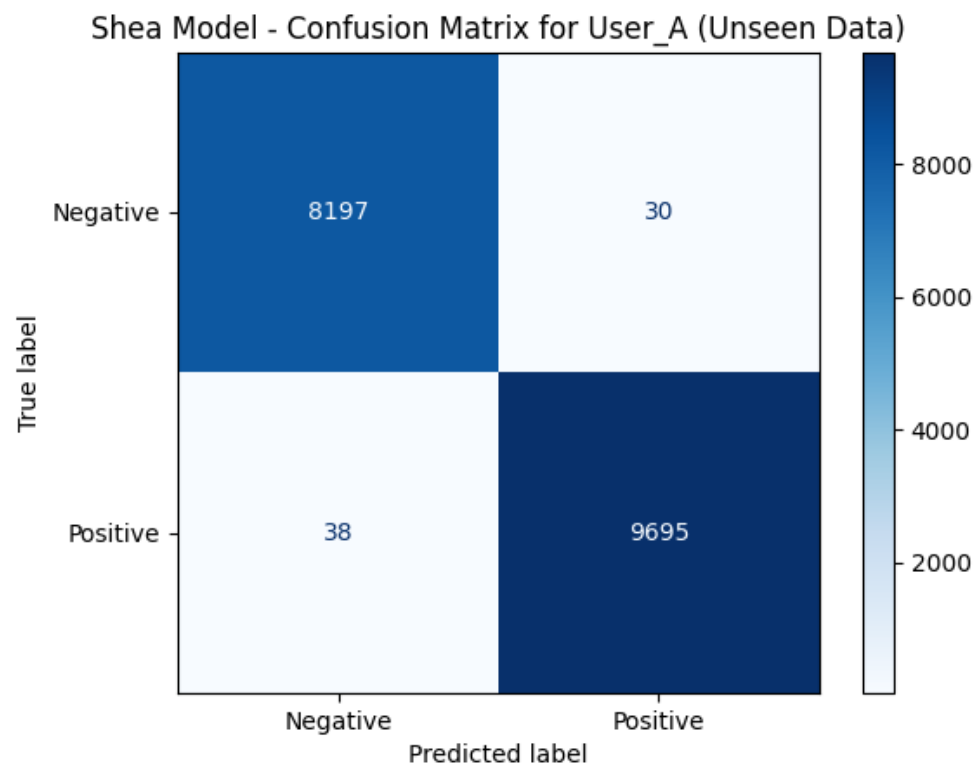


FIGURE 5.7: The Confusion Matrix from the Shea model vs Unseen data - User A contributing 9733 sequences to the unseen set.

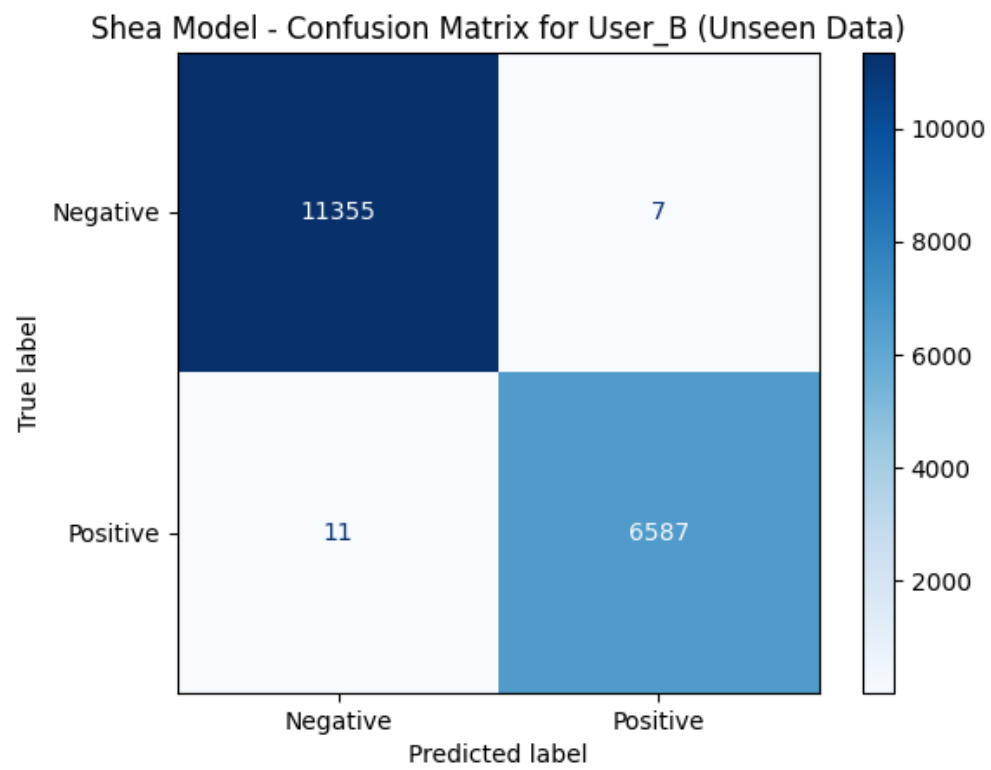


FIGURE 5.8: The Confusion Matrix from the Shea model vs Unseen data - User B contributed 6598 sequences to the unseen set.

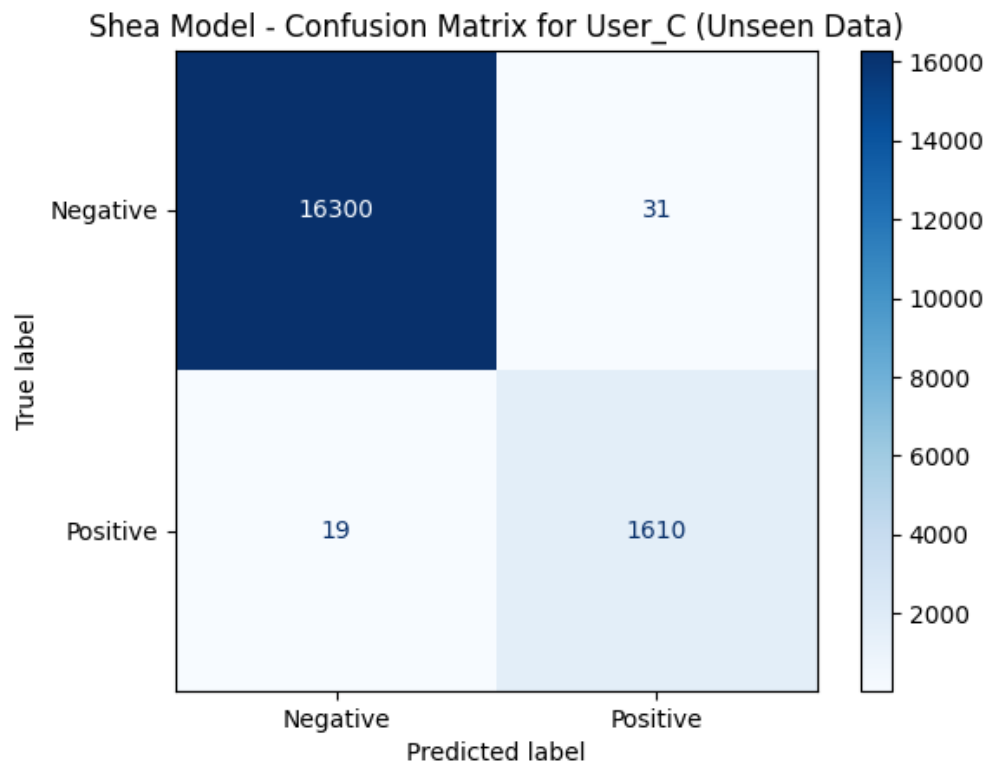


FIGURE 5.9: The Confusion Matrix from the Shea model vs Unseen data - User C contributed 1629 sequences to the unseen set.

Table 5.6 highlights the findings of this study compared to those that were listed earlier in table 2.3. This demonstrates the capacity of 1D-CNNs as multi-label classifiers in the field of authentication and mouse dynamic behavioural biometrics, successfully achieving higher accuracy and ROC-AUC scores compared to all three papers.

However, these results should be interpreted with caution, with the acknowledgement that further study should be undertaken to warrant these findings. There are factors which limit the application of this research, namely, the dataset, lack of solidified control instance and lack of comparison with other established datasets. All of which will be explored in the discussion of this project.

TABLE 5.6: Summary of research in the investigation of 1D-CNNs as authentication classifiers for mouse dynamic data, compared to the findings of this paper

Paper	Neural Network	Accuracy	Mean ROC_AUC	Dataset
Siddiqui <i>et al.</i> [27]	1D-CNN	85.73%	-	Novel
Chong <i>et al.</i> [36]	1D-CNN	-	0.90	Balabit [43]
Chong <i>et al.</i> [36]	1D-CNN	-	0.85	TWOS [44]
Chong <i>et al.</i> [28]	1D-CNN	-	0.90	Balabit [43]
Chong <i>et al.</i> [28]	1D-CNN	-	0.77	TWOS [44]
This research	1D-CNN	99.6%	0.99	Novel

Chapter 6

Discussion

6.1 Evaluation of the proof-of-concept

The main objective of this study was to validate a proof-of-concept for the use of mouse dynamics as a potential form of user authentication under organic conditions utilising a 1D-CNN model as a multi-label classifier. In order to assess whether this was achieved or not the hypothesis for this research was stated as follows:

Employing a 1D-CNN Deep Learning model to classify organic mouse dynamic data as a behavioural biometric, will bolster security by correctly classifying users generalised data with an overall accuracy of 95% and user-based classification with a precision of 95% and FAR of 1%.

The model demonstrated the capacity to classify users based on mouse dynamic data with considerable accuracy. The Shea model was able to discern between users on an unseen dataset with an overall accuracy of 99.6%. Considering FAR, the model performed well in this regard, with FAR achieved across users (A,B,C) in the unseen validation at 0.004%, 0.001% and 0.002% respectively. Looking at the balance of this value as well as precision macro achieving 99.2% and the ROC-AUC for each user exceeding 0.99 it can be conclusively stated that the hypothesis of this study was fulfilled and therefore should be accepted, however alongside the caveat that this is in the instance of this particular model design performing on *this* research's dataset.

6.2 Evaluation of study deliverables

As stated in table 2.4 in the methodology, the goals of this research were to contribute several deliverables to the field of mouse dynamics as a behavioural biometric. The two software artefacts were delivered successfully. As well as a saved 1D-CNN model (*Shea*) which has the best performance against the accompanying dataset of *this* study.

6.2.1 Mouse logger

Largely performing well, the mouse logger successfully collected over three million unique mouse events across four users with no known issues, excluding the anomalous files previously mentioned. However, it is worth noting that the mouse logger lacked in its precision within two key indices; Timestamps and Positional (x, y) data. In time-series research these values are of course of the upmost importance.

During initial prototyping, under high-intensity manual testing and subsequent benchmarking there was a noticeable discrepancy with the time logged in the more precise formatting timestamp (yyyy-mm-dd HH:mm:ss:SSSSSS). The order that events were logged were not ordered chronologically in time but the positional reference indicated a discrepancy. This was put down as an limitation of the file writer not being able to process and write events in quick succession accurately. As a result, the decision was made to trim timestamp data to the less precise but more manageable (yyyy-mm-dd HH:mm:ss:SSS) format. Although the sequence of events were preserved it would have had a knock-on effect in feature computation (*see code insert 3.3*), where the time difference between two events was a key variable in the computation of the acceleration and speed features. This oversight may have been overcome with the use of Unix epoch time. This not only preserves computing resource with the removal of conversion to a human-readable format (not necessary for this research) but also removes edge cases such as time zone differences which timestamps can be susceptible to. Although all users were in the same time zone throughout the study, issues could have arisen through other factors such as daylight savings. It may be for these reasons why Unix epoch time is used in other mouse dynamic research [27, 42–44].

The positional references to cursor location also lack precision, where coordinates were rounded to a whole figure. This removed decimal placements of positional data, and with it, removed specificity of loci capture. This would have reduced the models ability to pick up, perhaps, more characteristic 'micro-movements' or features in user mouse behaviour. From the results achieved in this study, it appears to have not been a significant factor, but if the sample size or class number were greater this could be a

invaluable data. Given that location data are fundamental variables in the computation of five features directly (distance, speed, acceleration, angle, jitter), as well as being two features in their own right (x , y) and featuring in each data row regardless of mouse event, this was a huge oversight from a design perspective.

Lastly, although mouse clicks of buttons of all types were assessed (left, right, middle, x1, x2) in this study, use of the scroll-wheel was not. Scroll interaction has been captured as a valuable mouse dynamic in other studies [28] so this was an oversight of the design implementation. Not only this, but edge cases such as those on Apple trackpads or 'magic mouse', where gestures can be used to manipulate the user's display were also not accounted for. Although movement of the mouse in this instances would have been. As already mentioned, there were instances of users forgetting to initiate the mouse-logger at the start of sessions. This led, undoubtedly, to further data loss across the set.

6.2.2 The CONMAN Service

The CONMAN service adeptly processes raw user data CSV files, preparing them for the 1D-CNN model. As detailed in the SDD (*see appendix E*), it is designed to accommodate additional users seamlessly via the `Config.py` file. The application of OOP principles facilitates the integration of new features or enhanced functionality while preserving the existing code's modularity. The feature computation is comprehensive, deriving 16 features from the user data, though there is scope for enhancement. Furthermore, in order to mitigate the risk of data leakage but to preserve the time-series data within sequences, data splitting was handled manually. There are utilities widely used such as `sklearn.model_selection.train_test_split` [70], but this can compromise the inherent order and temporal dependencies of time-series data. As a result, manual splitting of this data was employed in order to respect the chronological order of observations. However, this exposed potential vulnerability with failing to evenly distribution data [40] which was not addressed.

Although the modularity of the service is beneficial for expansion, the service itself is perhaps too verbose. It manages a lot of processes and data structures, turning raw data into model ready data and generating output. This could have been partitioned into separate services altogether and would allow more granular manual review of the datasets at different stages of the pipeline. Currently, there is also a lack of thorough automated testing, and unit testing of the service is limited at best. The absence of this comprehensive testing framework is a significant vulnerability of the service and was not addressed due to poor time management.

6.2.3 The 1D-CNN

The 1D-CNN model demonstrated high efficacy in processing user data, employing a straightforward layer architecture augmented with callbacks and hyperparameters to optimise performance. The selection of hyperparameters, including dropout, L1, L2, and batch normalisation, was primarily heuristic, guided by training loss outputs. This approach refined the parameters based on a foundational model structure inspired by existing literature [35, 45, 71].

Sequence length determination followed a similar heuristic method, taking into account the findings of Chong *et al.* [36], which suggest a notable impact of sequence length on ROC-AUC, with a preference for longer sequences. However, this report acknowledges a lack of statistical validation for its choice of sequence length.

Lastly, the 'black box' nature of deep learning models presents a challenge in identifying the specific features and inter-sequence patterns that the model identified. This opacity makes it difficult to precisely understand how the model discerns and utilises key characteristics of the input data for authentication.

6.2.4 Dataset

If additional permission is granted by users after the completion of this project, this dataset will be made public to contribute to the field of mouse dynamic research. However, there are limitations of this data to be considered in terms of its application in this project and its structure as a whole.

The balancing of class sizes through down-sampling, while facilitating streamlined training and evaluation, led to the exclusion of substantial segments of data for training. Specifically, considerable volumes of mouse events from User A and User B, numbering 823,305 and 509,888 respectively, were not represented into the training process. This represents a significant limitation of the study, as the data employed was not fully representative of the comprehensive datasets acquired from the participants. Coupled with the relatively small sample size ($n = 3$), the Shea model's results should be assessed with an awareness of these constraints.

This appropriately brings *organic* data collection into consideration. Unlike other datasets, such as the Balabit and TWOS datasets, which were limited by their specific usage environments, this research's approach benefits from a broader context. However, this broadness raises the question of whether the Shea model's strong performance in this study is due to it learning features from environmental variables, such as the number

of displays or the variety of buttons on different mouse designs, rather than focusing on the users' behavioral patterns. The data in table 6.1 shows that three mouse buttons were exclusively used by User A, for example. Furthermore, examining the distribution of data across the x-axis (*see figure 6.1*) reveals that, while the range of 0 - 1500 encompasses most user data, variations in display setups and user hardware (*see table G.1*) result in substantial data from Users A and B falling outside this range. This demonstrates that the model may have learned these exclusive features to classify users in some instances. While such distinctions are beneficial for classification results, it has a negative impact on the onward application in an authentication context. The absence of a robust control makes it difficult to ascertain whether the model could accurately detect an intrusion, such as User B on User A's system, and vice versa.

TABLE 6.1: Total Number of Mouse Clicks for Each User by Button

User	Button.left	Button.middle	Button.right	Button.x1	Button.x2
User_A	64900	11745	1846	1032	48
User_B	67587	-	2532	-	-
User_C	47087	-	896	-	-

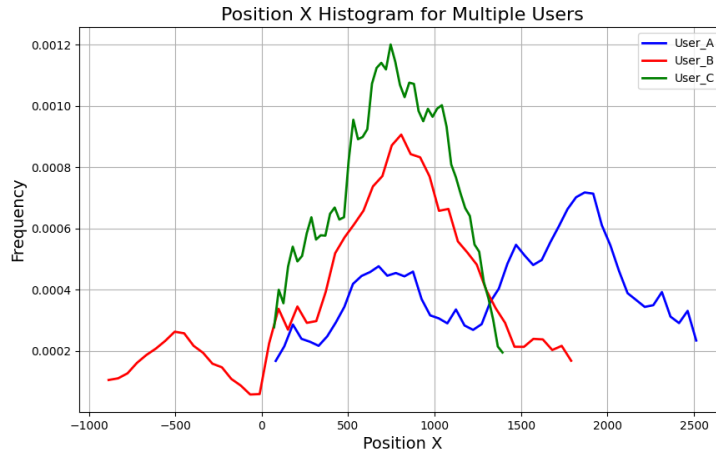


FIGURE 6.1: A histogram showing the frequency of mouse movement capture along the positional x-axis for all three users.

As mentioned, Chong *et al.* [36] conducted an evaluation using the Balabit and TWOS datasets. The Balabit dataset [43] comprises mouse dynamics data from 10 users performing unspecified administrative tasks, with unauthorised users included in the test set, this mirrors real-world authentication scenarios. The TWOS dataset [44], collected during a gamified competition in a simulated corporate environment, offers a more nuanced setup. It involved 24 participants, some assigned as bad actors, tasked with sabotaging team efforts or accessing other teams' machines. This dataset not only represents a broader range of user classes but also incorporates imposter data, enhancing

its suitability for training deep learning models to identify actual malicious activities. The inclusion of 'unauthorised actors' in these datasets allows for a more robust assessment of a model's ability to distinguish between legitimate users and imposters. This aspect was partially addressed in the Shea model through the use of an unseen dataset, which it performed well in identifying users with a high accuracy. Although data was simply concatenated in this instance and given the environmental variation in set-ups outlined - does not adequately emulate an attempt at intrusion. Furthermore, this research project's dataset lacked in breadth and depth of data with only 3 classes with an average of 1,148,721 mouse events captured per class, compared to that of the Balabit dataset with 10 classes with an average of 2,489,427 mouse events per class [43]. In summary, with this dataset lacking an adequate control, alongside the high variability of user system set-ups, spectrum of use cases and a lack of representation of the wider population, the net of *organic* data-capture was perhaps cast too wide in this research.

6.2.5 Research Outcomes

This research has showcased the capability of a multi-label 1D-CNN in classifying three users based on sequences of 100 mouse events, validated by an unseen dataset drawn from users organic data. While the model effectively distinguished between users in this context, its application to a wider authentication system, particularly for detecting live attacks, requires further exploration.

The dataset used in this research, though adequate for initial exploration, lacks the depth and variety necessary for comprehensive authentication systems. Critical elements in real-world applications, such as processing live data and adjusting to behavioural variability factors were not within the scope of this research. For a viable application, a broader authentication context is needed, real-time adaptation of a user's behavioural *signature* and effective handling of live attack scenarios are essential. In reaffirming the limited literature on 1D-CNNs as multi-label classifiers in the context of mouse dynamic authentication [28], this research contributes to the ongoing discourse in this field. Despite these limitations, the study highlights the potential of 1D-CNNs as tools in behavioural biometrics and user authentication. Potentially, paving the way for future research to investigate these models in more complex and nuanced environments, deepening our understanding of their applicability and effectiveness in real-world authentication scenarios.

6.2.6 Reflections on this study

If conducted again, this research would seek to incorporate the limitations highlighted above. The mouse logger would seek to capture more precise data over a longer period of time. The issues surrounding the consistency of data capture could be remedied in future by including the mouse logger initiation on user's start up scripts, along with precision of data capture being improved through comparative implementations of the use of Unix epoch time. More steadfast manual testing could be implemented and a reduction in the maximum frequency of data capture (40ms) would be reviewed.

The CONMAN service could explore the changes of feature computation on an empirical basis, with other literatures indicating further potential improvements; studies such as those by Siddiqui *et al.* [27] which incorporated more granular features like velocity, angular velocity, and mouse trajectory length. The use of feature computation and the necessity of this step outright could also be compared to a data pipeline which does not implement hard-coded feature computation at all, as explored in Hu *et al.* [51]. An expansion to the service to process and train the model on other widely used datasets in the literature would also be investigated - giving a direct comparison of the Shea model's performance against other well-cited and explored datasets such as Balabit and TWOS.

As highlighted in the deliverable design, there is also a lack of thorough automated testing, and unit testing of the service is limited at best. The absence of this comprehensive testing framework is a significant vulnerability of the service and was not addressed in this study - potentially leading to significant oversight, calling the validity of the results in this research into question.

In regard to model design, a more methodical approach, incorporating experimental usage and statistical analysis, could have been employed in this study. This strategy would involve systematically evaluating various model parameters, such as the number of convolutional layers, kernel sizes, filter counts, and hyperparameters, based on empirical evidence rather than the heuristic approach used in this research. By conducting a series of controlled experiments, it would be possible to quantitatively assess the impact of these parameters on the model's training performance [40]. The outcome would enable the selection of the most effective model design for the final implementation. Furthermore, exploring alternative loss functions, optimisers, or advanced architectures like multi-headed convolutional networks, as suggested in [72] and [36], represents another potential area of improvement. While the model demonstrated satisfactory performance with the current dataset, considering scenarios with an increasing number of classes may necessitate further refinements to ensure scalability and continued effectiveness. Sequence length again, like model parameters, could have been considered on an

evidence-driven basis. Moreover, the simplistic model architecture may not be robust enough on more diverse datasets, as a result, alternative training methodologies, such as curriculum learning as introduced by Bengio *et al.* [73], or transfer learning investigated by Antal *et al.* [72] and Chong *et al.* [36] should have also been explored.

The length of data collection time would have been increased and more concerted effort into recruiting a larger, more diverse participant pool. In the event of unbalanced datasets, alternative dataset augmentation approaches, such as random up-sampling and the Synthetic Minority Over-sampling Technique (SMOTE) [40], would be investigated. As highlighted, down-sampling was a simplified solution to a complex problem. In this research, my first project in machine learning, I opted for down-sampling for its simplicity, despite the resultant under representation of data from two classes.

In regards to participants, this study lacked a survey structure in order to garner relevant environmental variables from users, a survey such as the one seen in appendix H, could have been implemented to gather more tangible data from users. A control would also be implemented to assess the model's capability in detecting users on a standardised system and hardware, as well as perhaps attempting to assess intrusion detection of users on others machines. As a result, this would lead to greater insights into whether a model was capable of determining users by their intrinsic features rather than simply their machine environment.

Chapter 7

Conclusion

7.1 Suggestions for Further Work

The study has identified several key areas for future research to enhance the understanding and effectiveness of mouse dynamics as a behavioural biometric using a 1D-CNN model.

7.1.1 Dataset Expansion, Diversification and Feature Inclusion

A major focus for further research should be the expansion and diversification of the dataset. This involves increasing the number of participants, ensuring the dataset is diverse and representative of the wider population in terms of gender, race, age, educational background, career field, and computer literacy. Such a dataset would provide a more accurate reflection of the general population, enhancing the model's applicability and robustness. The length of commitment by users to the study of four weeks was too brief and could have been done better with expanding. A small sample of user data was only needed to manually test the service, this could have been acquired from an initial collection followed by a later data collection period of three months. Additionally, capturing a wider range of mouse features, including gestures and scroll wheel usage, would provide a more comprehensive dataset. Improvements in the precision of timestamping and positional data capture are also recommended to enhance the quality of the dataset.

7.1.2 Controlled Environment vs. Organic Data

Establishing a control group within the expanded dataset is essential. I would propose that users that partake in research such as this are also assigned a session to complete standardised tasks on a standardised hardware set-up. This would open the field of exploration, giving tangible comparison to whether a model can learn a user's intrinsic *organic* behavioural biometric and use this to identify that user within the confines of a controlled environment.

7.1.3 Enhancement of the CONMAN Service

Expanding the CONMAN service to process other publicly available datasets, such as those mentioned in the discussion, would allow for a direct comparison of this model's architecture with others in the field. This expansion would enable the model to be quantifiably assessed against current literature results. In cases of unbalanced datasets, focusing on additional metrics like the F1-Score could offer a more suitable performance metric, or the use of up-sampling techniques such as SMOTE could be more appropriate.

7.1.4 Empirical Hyperparameter and Model Variable Definition

A methodical approach to defining hyperparameters and model variables, backed by empirical evidence, is essential. This systematic approach would clarify the model's architecture and its positioning relative to alternative models in the field.

7.1.5 User Surveys

User surveys were not employed in this project, even a simple quantitative survey outlining the user's typical hardware, design of pointing device and typical use case for their machine could have gone a long way in illustrating the environmental features the model may have picked up.

7.1.6 Real-time Deployment and Testing with 'Bad-Actors'

Exploring the deployment of the CONMAN service in a live system to respond to planned 'bad-actors' in a real-time environment would be a significant step forward. Integration of the model into datasets that include 'unauthorised actors', similar to those found in the Balabit or TWOS datasets, would provide a realistic assessment of its performance in actual security scenarios.

7.1.7 Balancing Accessibility and Security

A key area for future research in the field of authentication systems is the exploration of the balance between accessibility and security [74, 75]. This project primarily concentrated on the mechanisms of a potential authentication system, leaving the intricate trade-off between user convenience and system security largely unaddressed. Future investigations should focus on how to design authentication systems that are not only robust and secure but also user-friendly and easily adoptable. Excessive restrictions in the name of security can deter users, compromising the overall effectiveness of the system [75]. In the context of deploying authentication systems in real-world scenarios, it becomes crucial to understand and cater to user preferences and limitations. Further work should include empirical studies on user experiences, aiming to develop a harmonious blend of accessibility and security that can lead to higher user satisfaction and broader acceptance of the system.

7.1.8 Longitudinal studies

To the researcher's knowledge, as of yet no literature has committed to a long-term study in the variability of user's behavioural biometric activity for mouse dynamics in response to stress, hardware setup, and the evolution of a user's mouse dynamics over time - termed 'behavioural variability' by Shen *et al.* [42]. If systems are to use mouse dynamic data as an authentication system this would require additional and concerted efforts in dataset expansion across the discipline.

7.2 Conclusion

This research introduces a novel dataset and further explores user authentication in mouse dynamics, employing a multi-label classifier 1D-CNN model with high accuracy in an organic context. The study successfully validates the initial hypothesis by demonstrating the accuracy, precision and FAR results from the Shea model.

However, it is important to acknowledge the limitations and challenges encountered. The European standards for commercial biometric systems require a False Acceptance Rate (FAR) of less than 0.001% and a False Rejection Rate (FRR) of less than 1% [76]. Currently, iris scanning is the only biometric modality meeting these stringent standards [76], highlighting the significant challenges in developing an accurate biometric authenticator. While this study meets these criteria and the hypothesis, its scope and

depth, with a sample size of only three and limited sample depth compared to larger datasets like the Balabit set, restrict its applicability beyond academic research.

Moreover, this research's methodology faces limitations regarding comparability with existing studies in the field. While it addresses a gap in the literature on 1D-CNN multi-label classifiers for mouse dynamics, authentication is more commonly approached as a binary classification problem. The multi-label approach of this study might find greater relevance in scenarios such as tailgating attacks within shared industrial systems but requires expansion into authentication scenarios in this context. The applicability of this method in home environments, where *organic* mouse behaviour is more common, is not be as pressing. Additionally, a notable shortcoming of this research is the absence of a control group. This lack of a control set limits the ability of this research to assess the model's precision in capturing the unique behavioral characteristics of users in an authentication scenario.

In conclusion, while this study provides a solid proof-of-concept for the use of multi-class 1D-CNNs in identifying users through organic data, its practical application is constrained without further research and development. Future work should focus on expanding the dataset size, depth, and exploring real-world applications to enhance the viability of this authentication method.

Appendix A

CONMAN Service: Software Documentation

A.1 Introduction

This is the software documentation for the Convolutional Mouse Authentication Network (CONMAN) service. This documentation will outline the purposes the service as well as the service's user guide, system detail, test facility and further development guide.

A.1.1 Purpose

The CONMAN service was created as a software artefact for a final research project as part of the completion of the MSc Software Engineering (Web development) course at the University of Wales, Trinity Saint David (UWTSD). The CONMAN service was designed to accept raw mouselog session files in the form of CSV files, process them to prepare for use in a One-Dimensional Convolutional Neural Network (1D-CNN). The service is also design to output the results of model performance in training, testing and against an unseen dataset.

A.2 User Guide

A.2.1 Configuring the service

The high-level parameters for the service can be found in the `Config.py` file in the project root [A.1](#). Here, the user data directory path, model configuration, data splitting

for sequences, number of model repetitions and the use of the full unbalanced dataset (for unseen data) can be adjusted here. The default settings as seen in the source code were the settings used for the data collected in this research project.

```

1 """ Configuration for running the experiment environment """
2 USER_DATA = 'config_data_directories.json'
3
4 """ Model Config """
5 MODEL_STORE = "model.hub/model_store"
6 NO_OF_USERS = 3
7 SEQUENCE_LENGTH = 100
8 NO_OF_FEATURES = 16
9 BATCH_SIZE = 34
10 MAX_EPOCHS = 100
11 EARLY_STOPPING_PATIENCE = 5
12 MINIMUM_IMPROVEMENT = 0.01
13 # Regularisation
14 L1 = 0.001
15 L2 = 0.001
16 DROPOUT = 0.05
17
18
19 """ Number of sequences to be created from mouse events """
20 TRAIN_SIZE = 3500
21 VALID_SIZE = 1000
22 TEST_SIZE = 1000
23 UNSEEN_SIZE = 1500
24
25 """ This boolean flag can be set to False to accept 1500 unseen_size
    sequences rather than all (True) """
26 USE_UNBALANCED_UNSEEN_DATA = True
27
28 """ Number of iterations per experimental run """
29 REPETITIONS = 25

```

CODE INSERT A.1: The Config.py where variables for the model and service can be adjusted

```
{
  "raw_data_directory": {
    "User_A": "data_directories/raw_data/User_A_Directory",
    "User_B": "data_directories/raw_data/User_B_Directory",
    "User_C": "data_directories/raw_data/User_C_Directory"
  },
  "user_labels": {
    "User_A": 0,
    "User_B": 1,
    "User_C": 2
  }
}
```

FIGURE A.1: How class directories are routed to the service, this formatting should be followed along with the incremental increase of any additional class labels

A.2.2 Running the Service

The service has been deployed for the duration of this project in the IDE, PyCharm. Running the `main()` method in `main.py` will run the default service. User data, additional classes and data repositories can be added to expand the service (*see figure A.1*). If the current structure is followed as stipulated in the `Config.py` and `config_data_directories.json` then additional user classes can be added to the service with ease.

A.2.3 Adding additional data

Data should be added to the service following the following file convention:

`data_directories/raw_data/User_[I]_directory/mouselog_session[j].csv`, where 'I' is an incremental alphabetical uppercase character and 'j' is the session number. The service is able to pick up CSV files even in the event of missing session numbers but this naming convention should be followed. If using the CONMAN service concurrently with the associated mouse logger (*appendix F*), the mouselog session file convention will be this by default.

A.3 System Detail

This section outlines the classes and methods within the service, their workflow, components and function. Initially outlined are the classes which direct the flow of the software alongside configuration files, followed by the functional classes which contain the bulk of the software's functionality.

A.3.1 `main.py`

The main function serves as the entry point for the CONMAN Service (*see figure A.2*), calling the `ExperimentRunner`. It orchestrates a sequence of steps for running training experiments and evaluating models. This function is responsible for handling data configurations, loading datasets, validating data integrity, executing training processes, and evaluating models on unseen datasets.

A.3.1.1 Workflow

1. **Run Number Retrieval:** Retrieve the current run number using `SystemUtils.get_run_number()`.
2. **Data Configuration Loading:** Load the user data configuration from `Config.py` using `SystemUtils.load_config(Config.USER_DATA)`.
3. **Experiment Runner Initialisation:** Initialise an instance of `ExperimentRunner` with the retrieved run number and data configuration.
4. **Data Loading:** Load the training, validation, testing, and unseen datasets using `ExperimentRunner.load_data()`.
5. **Data Integrity Check:** Checks the integrity and consistency of the loaded sequences using `SystemUtils.check_sequences()`.
6. **Training Experiment Execution:** Run the training experiment using the training and validation datasets, and also pass the testing datasets for model evaluation, with `ExperimentRunner.run_training_experiment()`.
7. **Model Evaluation:** Evaluate the best performing model on the unseen dataset using `ExperimentRunner.evaluate_best_model()`.
8. **Completion Message:** Print a completion message, indicating the end of the run and displaying the run number.

A.3.1.2 Components Used

- **SystemUtils:** Provides utility functions for operations such as retrieving the current run number and loading configurations. It is also used for checking the integrity and consistency of data sequences.
- **Config:** Contains user data configurations, which are loaded and used in setting up the data environment for the experiments.
- **ExperimentRunner:** Central component responsible for initialising the experiment with run number and data configuration, loading data, running the training experiment, and evaluating the best performing model.

A.3.1.3 Entry Point

The script is intended to be run as a standalone program, with `main()` being the entry point. This function orchestrates the entire process, from retrieving the run number and loading data configurations to executing the training experiment and evaluating the model on unseen data. It concludes with a message indicating the completion of the run.

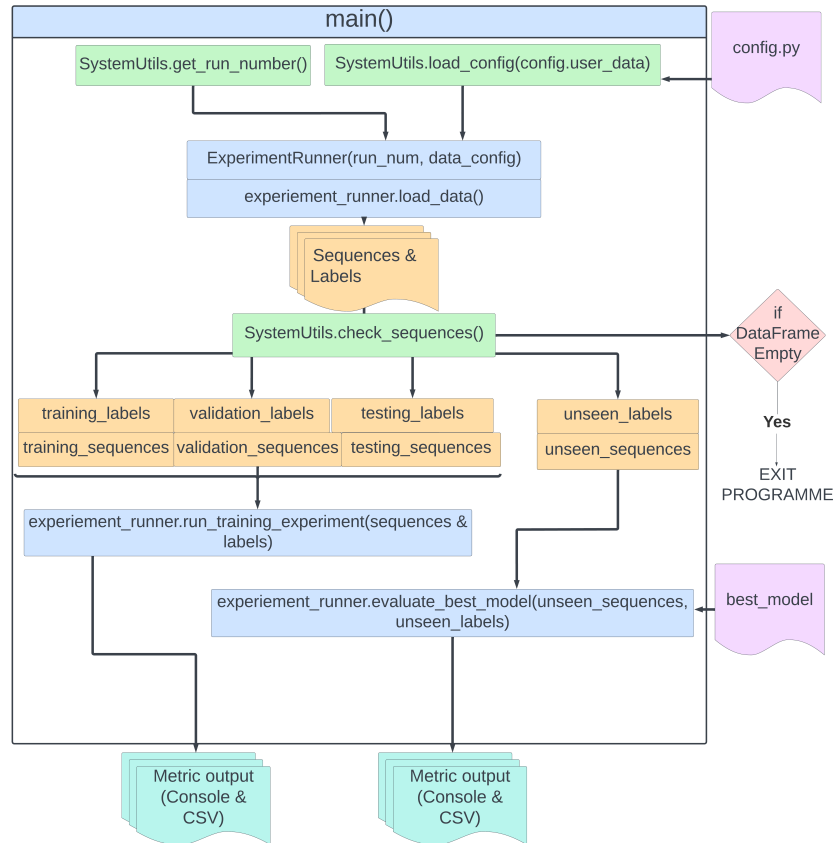


FIGURE A.2: Order of operations when `main()` is initiated

A.3.2 runner.ExperimentRunner.py

The `ExperimentRunner` class is responsible for orchestrating the execution of model training and evaluation experiments. It manages the overall process from data loading to model training and final evaluation on unseen data.

A.3.2.1 Attributes

- **run_num (int)**: The current run number of the experiment.
- **data_config (dict)**: Configuration settings for data processing.
- **user_labels (list)**: List of user labels for classification.
- **data_controller (DataController)**: Controller class instance for managing data loading and preprocessing.
- **best_val_loss (float)**: Best validation loss achieved during training iterations.
- **best_model_iteration (int)**: Iteration number of the model with the best validation loss.
- **accuracies, precisions, roc_aucs (list)**: Lists to store performance metrics for each iteration.

A.3.2.2 Methods

- **load_data()**: Loads and preprocesses the data for the experiment.
- **run_training_experiment(training_sequences, training_labels, validation_sequences, validation_labels, test_sequences, test_labels)**: Runs the training experiment for multiple iterations, recording the best validation loss and the corresponding model iteration.
- **evaluate_best_model(unseen_sequences, unseen_labels)**: Evaluates the best performing model on unseen data.

A.3.2.3 Workflow

1. **Initialisation**: Initialise the runner with the run number, data configuration, and data controller.
2. **Data Loading**: Load and preprocess the data using `DataController` and `ExperimentController`.

3. **Training Experiment Execution:** For each iteration, creates and trains a new model instance using `ModelCreator`, evaluates it on the test dataset, and stores performance metrics.
4. **Best Model Selection:** Records the iteration with the best validation loss.
5. **Unseen Data Evaluation:** Evaluate the best performing model on unseen data using `ResultsController`.

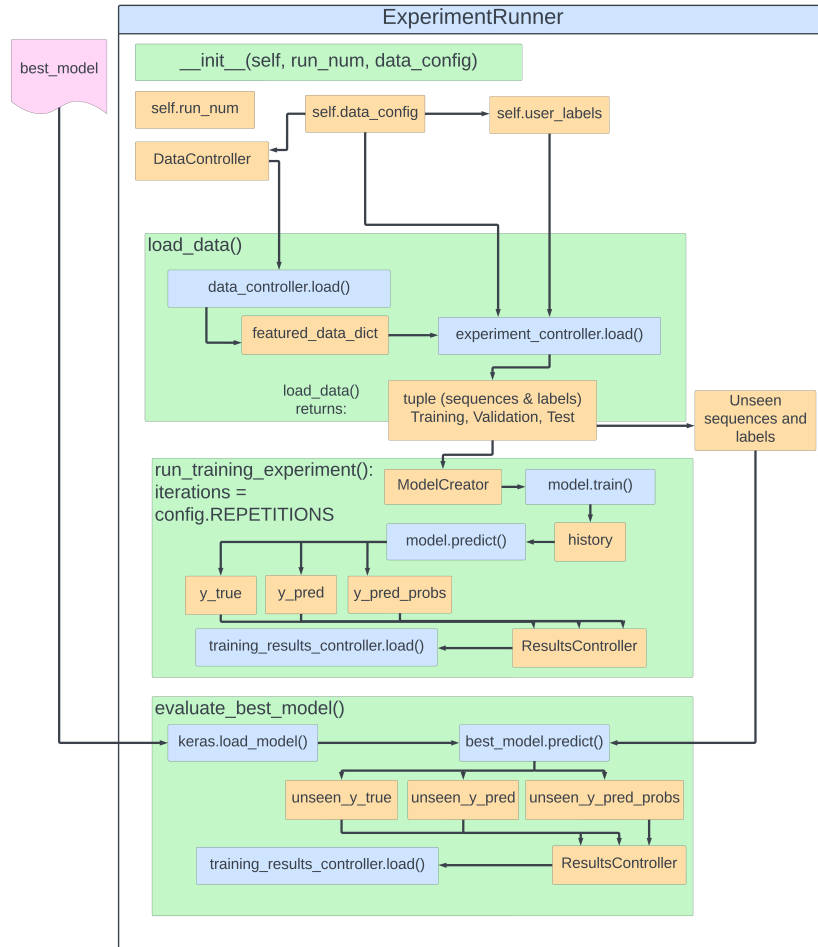


FIGURE A.3: Order of operations in Experiment Runner

A.3.3 Config.py

This section outlines the `Config.py` file used for setting up and running the experiment environment. These settings define key parameters for data handling, model configuration, and experiment execution.

A.3.3.1 Data Configuration

- **USER_DATA:** Specifies the path to the JSON file containing data directories for different classes. Example: `'config_data_directories.json'`.

A.3.3.2 Model Configuration

- **MODEL_STORE:** Directory where trained models are stored. Example: `"ModelHub/model_store"`.
- **NO_OF_USERS:** The number of users (classes) in the dataset. Example: 3.
- **SEQUENCE_LENGTH:** The length of each sequence to be created from the data. Example: 100.
- **NO_OF_FEATURES:** The number of features in the dataset. Example: 16.

A.3.3.3 Sequence Creation Configuration

- **TRAIN_SIZE:** The number of sequences to be created for the training dataset. Example: 3500.
- **VALID_SIZE:** The number of sequences to be created for the validation dataset. Example: 1000.
- **TEST_SIZE:** The number of sequences to be created for the testing dataset. Example: 1000.
- **UNSEEN_SIZE:** The number of sequences to be created for the unseen dataset. Example: 1500.

A.3.3.4 Experiment Configuration

- **use_unbalanced_unseen_data:** A boolean flag that can be set to control the size of the unseen dataset. If set to `True`, all remaining data is used as the unseen dataset; if `False`, the unseen dataset is limited to `UNSEEN_SIZE` number of sequences.

- **REPETITIONS:** The number of iterations per experimental run. Example: 25.

A.3.4 `config_data_directories.json`

The `config_data_directories.json` file specifies the configuration for raw data directories and user label mappings used in the experiment environment. This configuration file plays a crucial role in setting up the data handling structure for the experiment.

A.3.4.1 Raw Data Directory Configuration

This section of the configuration file defines the directory paths for raw data associated with each user. It maps user identifiers to their respective data directories.

- **User_X**: Specifies the directory path for User X's raw data. Example: `"data_directories/raw_data/User_X.Directory"`.
- **User_B**: Specifies the directory path for User B's raw data. Example: `"data_directories/raw_data/User_B.Directory"`.
- **User_C**: Specifies the directory path for User C's raw data. Example: `"data_directories/raw_data/User_C.Directory"`.

A.3.4.2 User Label Mapping Configuration

This section of the configuration file defines the mapping of user identifiers to integer labels used in the machine learning models.

- **User_A**: Maps User A to an integer label. Example: 0.
- **User_B**: Maps User B to an integer label. Example: 1.
- **User_C**: Maps User C to an integer label. Example: 2.

This mapping and associated labelling, ensure the correct directory is loaded for each user and their associated labels are contained in pairing when passed to the data dictionary data structure.

A.3.5 `controllers.Controller.py`

The `Controller` class is an abstract base class that serves as a foundational template for other controller classes within the software architecture. It enforces a consistent interface across different types of controllers by implementing the `load()` method in all subclasses.

A.3.5.1 Abstract Methods

- `load()`: An abstract method that must be overridden by subclasses. It defines the logic for loading or initialising resources or processes specific to the controller. This method is essential for ensuring that each subclass of `Controller` adheres to a standard approach for resource and process management.

A.3.5.2 Role and Implementation

The `Controller` class does not provide a concrete implementation of the `load` method. Instead, it requires that all derived classes implement this method, thereby ensuring that each controller has its unique way of handling resource or process initialisation. This design promotes modularity and consistency within the software's architectural framework.

A.3.6 `controllers.DataController.py`

The `DataController` class, extending `Controller`, is responsible for managing the data loading, processing, and preparation for experiments. It manages the workflow of handling raw data, transforming it into a format suitable for training, validation, testing, and unseen datasets.

A.3.6.1 Attributes

- **`user_directories` (dict):** Dictionary mapping user labels to their data directories.
- **`user_labels` (dict):** Mapping of user labels to integers.
- **`sequence_length` (int):** Length of each data sequence.

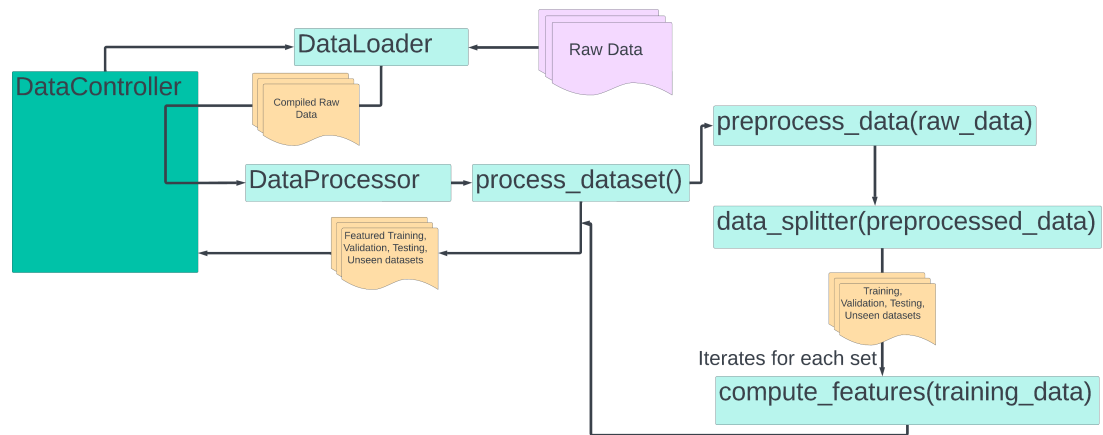
A.3.6.2 Methods

- **`load()`:** Processes and prepares the data for the experiment. This method involves loading raw data, processing it for each user, and splitting it into training, validation, testing, and unseen datasets. The processed data is then saved and returned as a dictionary.

A.3.6.3 Workflow

1. **Initialisation:** Initialise the controller with the raw data directory, user labels, and sequence length.
2. **Data Processing:** For each user, load the raw data using `DataLoader`, process it with `DataProcessor`, and split it into different datasets.
3. **Data Preparation:** Save the processed and featured data for each dataset type (training, validation, testing, unseen).
4. **Data Organisation:** Compile the featured data into a dictionary, mapping each user to their respective datasets.

Mapping of the data flow within the `DataController` and associated classes within the `data_utils` package can be seen in figure [A.4](#)

FIGURE A.4: Order of operations managed by `DataController.load()`

A.3.7 `controllers.ExperimentController.py`

The `ExperimentController` class, extending `Controller`, is responsible for managing the experiment's data preparation and sequence creation processes. It handles the compilation of datasets for training, validation, testing, and unseen data partitions, managing these datasets alongside their respective labels using a data dictionary.

A.3.7.1 Attributes

- **`train_size`, `valid_size`, `test_size`, `unseen_size` (`int`):** Sizes of the respective data partitions.
- **`user_labels` (`dict`):** Mapping of user labels to integers.
- **`sequence_length` (`int`):** Length of each data sequence.
- **`featured_data_dict` (`dict`):** Dictionary containing featured data for each user.
- **`dataset_compiler` (`DatasetCompiler`):** Helper object for compiling datasets.

A.3.7.2 Methods

- **`load()`:** Processes and compiles training, validation, testing, and unseen datasets. If an error occurs during processing, it logs the error and returns empty datasets.
- **`shuffle_sequences_labels(sequences, labels)`:** Shuffles sequences and their corresponding labels, ensuring that sequences and labels are shuffled in the same order to maintain correct pairing.

A.3.7.3 Workflow

1. **Initialisation:** Initialise the controller with data partition sizes, user labels, sequence length, and featured data.
2. **Data Processing:** For each user label, normalise and process the data, then create sequences for each data type (training, validation, testing, unseen).
3. **Error Handling:** Catch and log any errors that occur during data processing.
4. **Sequence and Label Shuffling:** Shuffle only the training sequences and their corresponding labels.
5. **Dataset Compilation:** Compile the shuffled and processed data into datasets for training, validation, testing, and unseen data.

A.3.8 `controllers.ResultsController.py`

The `ResultsController` class, extending `Controller`, is responsible for calculating, outputting, and plotting the results of a model's performance. It handles the computation of performance metrics, generation of outputs, and visual representation of results.

A.3.8.1 Attributes

- **`dataset_label` (`str`):** Label identifying the dataset.
- **`user_labels` (`dict`):** Mapping of user labels to integers.
- **`n_classes` (`int`):** Number of classes (users).
- **`y_true` (`np.array`):** True labels for the dataset.
- **`y_pred` (`np.array`):** Predicted labels from the model.
- **`y_pred_probs` (`np.array`):** Predicted probabilities from the model.
- **`history` (`History object`):** Training history of the model.
- **`run_num` (`int`):** Run number of the experiment.
- **`iteration` (`int`):** Iteration number of the experiment.
- **`metric_calculator` (`MetricCalculator`):** Object for computing performance metrics.
- **`output_handler` (`OutputHandler`):** Object for handling output of results.
- **`plotter` (`Plotter`):** Object for plotting results.

A.3.8.2 Methods

- **`load()`:** Computes, outputs, and plots the model's performance metrics. This includes calculating overall and per-class metrics, outputting results to the console and CSV files, and plotting training history and ROC-AUC-OVR curves.

A.3.8.3 Workflow

1. **Initialisation:** Initialise the controller with dataset information, performance metrics, and experiment details.

2. **Performance Metrics Calculation:** Calculate overall and individual class metrics using `MetricCalculator`.
3. **Results Output:** Output the computed metrics to the console and as CSV files using `OutputHandler`.
4. **Results Plotting:** Plot the training history and ROC-AUC-OVR curves using `Plotter`.

A.3.9 `controllers.data_utils.DataLoader.py`

The `DataLoader` class is a utility class used for loading data from CSV files in a given directory. It aggregates the data from all files into a single `DataFrame`, ensuring that the files are read in chronological order based on the session number. This is completed for each class in the raw data.

A.3.9.1 Static Methods

- `load_data(directory)`: Loads data from all CSV files in a specified directory and aggregates it into a single `DataFrame`. The method sorts files chronologically based on session numbers extracted from file names. It returns an aggregated `DataFrame`, or an empty `DataFrame` in case of an error.

A.3.9.2 Workflow

1. **Session Number Extraction:** Use a regular expression to extract the session number from the filenames of CSV files in the directory.
2. **File Sorting:** Sort all CSV files in the directory based on the extracted session number.
3. **Data Reading:** Read each CSV file into a `DataFrame` and concatenate all `DataFrames` into a single `DataFrame`.
4. **Error Handling:** Log any errors encountered during data loading and return an empty `DataFrame` if an error occurs.

A.3.10 `controllers.data_utils.DataProcessor.py`

The `DataProcessor` class is designed for processing and preparing mouse dynamics data for machine learning. It handles the preprocessing of raw data, computation of additional features, and splitting the data into training, validation, testing, and unseen sets.

A.3.10.1 Attributes

- **`raw_data (pd.DataFrame)`**: The raw mouse dynamics data.
- **`user (str)`**: Identifier for the user whose data is being processed.
- **`file_name (str)`**: Filename for storing processed data.

A.3.10.2 Methods

- **`process_dataset()`**: Orchestrates the processing of the raw data through preprocessing, splitting, and feature computation steps and returns the processed datasets.
- **`preprocess_data(data)`**: Preprocesses the raw data by converting timestamps and encoding categorical columns.
- **`compute_features(data)`**: Computes additional features from the preprocessed data.
- **`data_splitter(data)`**: Splits the data into training, validation, testing, and unseen datasets.

A.3.10.3 Workflow

1. **Initialisation**: Initialise the processor with raw data, user identifier, and the flag for unbalanced unseen dataset.
2. **Dataset Processing**: Orchestrates preprocessing and feature computation to each dataset and return the processed datasets.
3. **Data Preprocessing**: Convert timestamps, encode categorical columns, and perform initial data preprocessing.
4. **Feature Computation**: Compute additional features such as distance, speed, acceleration, angle, and jitter.

5. **Data Splitting:** Split the data into training, validation, testing, and unseen datasets.

A.3.11 `controllers.experiment_utils.DataNormaliser.py`

The `DataNormaliser` class is responsible for normalising and scaling mouse dynamics data using the `MinMaxScaler` from Scikit-Learn. It offers functionality to fit the scaler on the training data and normalise various datasets, including training, validation, testing, and unseen data.

A.3.11.1 Attributes

- **scaler (`MinMaxScaler`):** An instance of `MinMaxScaler` for data normalisation.

A.3.11.2 Methods

- **`fit_scaler(data)`:** Fits the scaler to specific columns of the provided `DataFrame`.
- **`normalise_data(data)`:** Normalises the provided data using the fitted scaler.
- **`normalise_and_save_data(training_data, validation_data, testing_data, unseen_data)`:** Orchestrates the aforementioned methods leading to the normalisation of the training, validation, testing, and unseen datasets - returning them as a dictionary.

A.3.11.3 Workflow

1. **Initialisation:** Initialise the `DataNormaliser` class with an instance of `MinMaxScaler`.
2. **Scaler Fitting:** Fit the scaler to specific columns of the training data, including `Position_X`, `Position_Y`, `Distance`, `Speed`, `Acceleration`, `Angle`, and `Jitter`.
3. **Data Normalisation:** Normalise each dataset (training, validation, testing, and unseen) using the fitted scaler.
4. **Data Organisation:** Store the normalised datasets in a dictionary and drop irrelevant columns.

A.3.12 `controllers.experiment_utils.SequenceCreator.py`

The `SequenceCreator` class is designed for creating sequences of a specified length from time-series data. It generates sequences and their corresponding target labels from the provided dataset, making them compatible for model reception.

A.3.12.1 Attributes

- **`sequence_length (int)`:** The length of each sequence to be created.

A.3.12.2 Methods

- **`create_sequences(data, user_label, num_sequences, generate_max_sequences)`:** Generates non-overlapping sequences and their corresponding labels from the data, allowing for the option to produce the maximum possible number of sequences from the dataset.

A.3.12.3 Workflow

1. **Initialisation:** Initialise the `SequenceCreator` with the specified sequence length.
2. **Sequence Generation:** Create sequences by dividing the time-series data into consecutive, non-overlapping data points of the specified length.
3. **Label Association:** Associate each sequence with the corresponding target label (user identifier).
4. **Sequence Count Control:** Optionally generate the maximum possible number of sequences or a specified number of sequences from the unseen dataset.

A.3.13 `controllers.experiment_utils.DatasetCompiler.py`

The `DatasetCompiler` class is designed for compiling and formatting datasets for use in the 1D-CNN model. It is responsible for converting sequence data to a float32 format and transforming label data into a categorical format.

A.3.13.1 Attributes

- `user_labels (dict)`: Mapping of user identifiers to integer labels.

A.3.13.2 Methods

- `compile_datasets(train_sequences, train_labels, valid_sequences, valid_labels, test_sequences, test_labels, unseen_sequences, unseen_labels)`: Compiles and formats the training, validation, testing, and unseen datasets, converting sequence data to float32 and label data to categorical format for model reception.

A.3.13.3 Workflow

1. **Initialisation:** Initialise the `DatasetCompiler` with a dictionary mapping user identifiers to integer labels.
2. **Dataset Compilation:** Check if any dataset is empty and log an error if so. Process each dataset (training, validation, testing, and unseen) by converting sequence data to float32 and label data to categorical format.
3. **Data Formatting:** Convert sequence data of each dataset to the appropriate numerical format and label data to a categorical format for classification tasks.
4. **Dataset Return:** Return the compiled and formatted datasets as a tuple.

A.3.14 `controllers.results_utils.MetricCalculator.py`

The `MetricCalculator` class is responsible for calculating various metrics to evaluate the performance of machine learning models. It computes both overall and per-class performance metrics, providing insights into the model's effectiveness and accuracy in predictions.

A.3.14.1 Attributes

- **`user_labels` (dict):** A dictionary mapping user identifiers to labels.
- **`n_classes` (int):** The number of classes in the dataset.

A.3.14.2 Methods

- **`compute_overall_metrics(y_true, y_pred, y_pred_probs)`:** Computes overall metrics for classification results, including accuracy, precision, recall, F1 score, and ROC AUC.
- **`compute_individual_metrics(y_true, y_pred, y_pred_probs)`:** Calculates performance metrics for each class, including confusion matrices, precision, F1 score, ROC AUC, TPR, TNR, FAR, and FRR.
- **Additional methods for specific rates:** `true_positive_rate`, `true_negative_rate`, `false_alarm_rate`, `false_rejection_rate`.

A.3.14.3 Workflow

1. **Initialisation:** Initialise the `MetricCalculator` with the number of classes and user label mapping.
2. **Overall Metrics Computation:** Compute overall performance metrics such as accuracy, precision, recall, F1 score, and ROC AUC using the true and predicted labels.
3. **Individual Metrics Computation:** Calculate performance metrics for each class, including confusion matrices and specific rates such as TPR, TNR, FAR, and FRR.
4. **Metrics Formatting:** Organise overall and per-class metrics into dictionaries for easy interpretation and analysis.

A.3.15 `controllers.results_utils.OutputHandler.py`

The `OutputHandler` class is designed for handling the output of machine learning model performance metrics. It provides output model performance metrics, including overall and per-class metrics, to the console and to a CSV file. This facilitates both immediate review and longer-term storage of results.

A.3.15.1 Methods

- `output_results_to_console(dataset_label, overall_metrics, metrics_per_class, confusion_matrices)`: Outputs metrics to the console, displaying overall model performance and individual class performance metrics, including confusion matrices.
- `output_results_to_csv(dataset_label, overall_metrics, metrics_per_class, confusion_matrices, run, iteration)`: Outputs metrics to a CSV file, creating separate files for overall metrics and per-class metrics, including confusion matrices. Saves the files in a structured directory format based on experiment run and iteration.

A.3.15.2 Workflow

1. **Results Output to Console:** Display overall model performance metrics and individual class metrics, including confusion matrices, on the console for immediate review.
2. **Results Output to CSV:** Write overall and per-class performance metrics to CSV files, including the extraction and formatting of confusion matrices. Save the CSV files in a structured directory format for organised storage and future reference.
3. **Directory Creation:** Create necessary directories for saving CSV files, structured based on experiment run and iteration number.

A.3.16 `controllers.results_utils.Plotter.py`

The `Plotter` class is designed for visualising various aspects of machine learning model performance. It provides functionalities to plot the training history and Receiver Operating Characteristic (ROC) Area Under Curve (AUC) specifically for multi-class classifications.

A.3.16.1 Attributes

- **`user_labels` (dict):** A dictionary mapping user identifiers to labels.
- **`n_classes` (int):** The number of classes in the dataset.
- **`y_true` (array):** True labels of the test dataset.
- **`y_pred_probs` (array):** Prediction probabilities by the model.
- **`history` (History object):** Training history of the model.

A.3.16.2 Methods

- **`plot_history(history, run, iteration)`:** Plots the training and validation accuracy and loss over epochs, and saves the plot to a file.
- **`plot_roc_auc_ovr(y_true, y_pred_probs, run, iteration)`:** Plots the ROC AUC curve for multi-class classification using the true labels and prediction probabilities. Saves the plot to a file.

A.3.16.3 Workflow

1. **Training History Plotting:** Plot training and validation accuracy and loss over epochs using the provided history object. Save the plot to a file in a structured directory.
2. **ROC AUC Plotting:** Plot the ROC curve for multi-class classification based on the number of classes. Calculate and display the AUC for each class, and save the plot to a file.
3. **Directory Creation:** Create necessary directories for saving the plots, structured based on experiment run and iteration number.

A.3.17 ModelCreator and create_model Function

The `ModelCreator` class is responsible for creating, training, and evaluating a machine learning model for user authentication using mouse dynamics. The class uses the `create_model` function to construct the model.

A.3.17.1 create_model Function

- **Functionality:** Creates and compiles a 1D Convolutional Neural Network (CNN) model for user authentication.
- **Parameters:** `sequence_length`, `no_of_features`, `no_of_users`.
- **Return Value:** A compiled Keras CNN model.

A.3.17.2 ModelCreator Attributes

- **training_sequences, training_labels:** Training data sequences and labels.
- **validation_sequences, validation_labels:** Validation data sequences and labels.
- **test_sequences, test_labels:** Test data sequences and labels.
- **no_of_users, sequence_length, no_of_features:** Number of user classes, length of input sequences, and number of features in the input data.
- **model_store:** Path to store the trained model.
- **model (Sequential):** The machine learning model.
- **early_stopping_callback (EarlyStopping):** Callback for early stopping.
- **model_checkpoint_callback (ModelCheckpoint):** Callback for model checkpointing.

A.3.17.3 ModelCreator Methods

- **train():** Trains the CNN model using training and validation datasets.
- **predict(sequences, labels):** Generates predictions for the provided sequences using the trained model.

A.3.17.4 Workflow

1. **Model Initialisation:** Initialises the `ModelCreator` with training, validation, and test datasets, model parameters, and callbacks.
2. **Model Training:** Train the CNN model using the training and validation datasets, implementing callbacks for early stopping and model checkpointing.
3. **Model Prediction:** Generate predictions using the trained model on given sequences and compare them with the true labels.

A.4 Unit Testing

Unit tests can be run by following the relevant sections in the `README.md` within the root directory, albeit limited and incomplete.

A.5 Development Guide

The requirements to run the service are listed in the software's '*Requirements.txt*', and shown below.

- `absl-py` ~ 2.0.0
- `astunparse` ~ 1.6.3
- `cachetools` ~ 5.3.2
- `certifi` ~ 2023.11.17
- `charset-normalizer` ~ 3.3.2
- `contourpy` ~ 1.2.0
- `cycler` ~ 0.12.1
- `flatbuffers` ~ 23.5.26
- `fonttools` ~ 4.45.1
- `gast` ~ 0.5.4
- `google-auth` ~ 2.23.4
- `google-auth-oauthlib` ~ 1.1.0
- `google-pasta` ~ 0.2.0
- `grpcio` ~ 1.59.3
- `h5py` ~ 3.10.0
- `idna` ~ 3.5
- `importlib-metadata` ~ 6.8.0
- `importlib-resources` ~ 6.1.1
- `joblib` ~ 1.3.2
- `keras` ~ 2.15.0
- `kiwisolver` ~ 1.4.5
- `libclang` ~ 16.0.6
- `Markdown` ~ 3.5.1
- `MarkupSafe` ~ 2.1.3
- `matplotlib` ~ 3.8.2
- `ml-dtypes` ~ 0.2.0

- numpy ~ 1.26.2
- oauthlib ~ 3.2.2
- opt-einsum ~ 3.3.0
- packaging ~ 23.2
- pandas ~ 2.1.3
- Pillow ~ 10.1.0
- protobuf ~ 4.23.4
- pyasn1 ~ 0.5.1
- pyasn1-modules ~ 0.3.0
- pyparsing ~ 3.1.1
- python-dateutil ~ 2.8.2
- pytz ~ 2023.3.post1
- requests ~ 2.31.0
- requests-oauthlib ~ 1.3.1
- rsa ~ 4.9
- scikit-learn ~ 1.3.2
- scipy ~ 1.11.4
- six ~ 1.16.0
- tensorboard ~ 2.15.1
- tensorboard-data-server ~ 0.7.2
- tensorflow ~ 2.15.0
- tensorflow-estimator ~ 2.15.0
- tensorflow-io-gcs-filesystem ~ 0.34.0
- termcolor ~ 2.3.0
- threadpoolctl ~ 3.2.0
- typing_extensions ~ 4.8.0

- `tzdata` ~ 2023.3
- `urllib3` ~ 2.1.0
- `Werkzeug` ~ 3.0.1
- `wrapt` ~ 1.14.1
- `zipp` ~ 3.17.0

Should a developer wish to expand on this software these requirements are to be noted. This project was developed in the PyCharm IDE and Git was used for version control, no branch name conventions are specified, but going forward would be advisable. The PEP 8 [77] coding style standard for Python was followed for the duration of development, with Object-Oriented Programming principles used as a guide.

Docstring conventions as outlined PEP 257 [78] were adhered to in this project in order to adequately document Python classes, methods, variables and modules.

A.6 Source Code with Documentation

A.6.0.1 main.py

```

1 import SystemUtils
2 import Config
3 from runner.ExperimentRunner import ExperimentRunner
4
5
6 def main():
7     """
8     Main method to run the CONMAN Service using the ExperimentRunner.
9
10    This method performs the following tasks:
11    1. Retrieves the current run number.
12    2. Loads the user data configuration from Config.py.
13    3. Initialises an instance of ExperimentRunner with the run number and
14       data configuration.
15    4. Loads the data training, validation, testing, and unseen datasets.
16    5. Checks the integrity and consistency of the loaded sequences.
17    6. Runs the training experiment using the training and validation data.
18    7. Evaluates the best performing model on the unseen dataset.
19    8. Prints a completion message with the run number.
20
21    Returns:
22    None
23    """
24    run_num = SystemUtils.get_run_number()
25    data_config = SystemUtils.load_config(Config.USER_DATA)
26
27    experiment_runner = ExperimentRunner(run_num, data_config)
28    (training_sequences, training_labels, validation_sequences,
29     validation_labels, test_sequences, test_labels,
30     unseen_sequences, unseen_labels) = experiment_runner.load_data()
31
32    SystemUtils.check_sequences([training_sequences, validation_sequences,
33                                test_sequences, unseen_sequences])
34
35    experiment_runner.run_training_experiment(training_sequences,
36                                              training_labels, validation_sequences, validation_labels,
37                                              test_sequences, test_labels)
38    experiment_runner.evaluate_best_model(unseen_sequences, unseen_labels)
39
40    print(f"Run {run_num} complete!")

```

```
40 if __name__ == "__main__":  
41     main()
```

CODE INSERT A.2: CONMAN Source Code: Main.py

A.6.0.2 SystemUtils.py

```
1 import json
2 import sys
3
4
5 def load_config(config_path):
6     """
7     Load configuration data from a given file path.
8
9     Args:
10         config_path (str): The path to the configuration file.
11
12     Returns:
13         dict: Configuration data loaded from the file.
14     """
15     with open(config_path, 'r') as file:
16         return json.load(file)
17
18
19 def get_run_number():
20     """
21     Prompt the user to enter a run number for the model experiment.
22
23     This function continuously prompts the user until a valid positive
24     integer is entered.
25     If an invalid input is entered, it prompts again with an error message.
26
27     Returns:
28         int: The run number entered by the user.
29     """
30     while True:
31         try:
32             run_number = int(input("Please enter the run number of the
33 model experiment: "))
34             if run_number < 1:
35                 print("Please try again, using a positive integer.")
36             else:
37                 print(f"Run {run_number} started")
38                 return run_number
39         except ValueError:
40             print("Invalid input. Please enter a valid integer.")
41
42
43 def check_sequences(sequences):
44     """
45     Checks if any of the provided datasets are empty and exits the program
46     if they are.
```

```
44
45     Args:
46         sequences (list): A list of datasets to be checked.
47
48     Returns:
49         None
50
51     If any dataset in the list is empty, prints an error message and exits
52     the program.
53     """
54     if len(sequences) == 0:
55         print("One or more datasets are empty. Cannot proceed.")
56         sys.exit(0)
57     return
```

CODE INSERT A.3: CONMAN Source Code: SystemUtils.py

A.6.0.3 Config.py

```

1 """ Configuration for running the experiment environment """
2 USER_DATA = 'config_data_directories.json'
3
4 """ Model Config """
5 MODELSTORE = "model_hub/model_store"
6 NO_OF_USERS = 3
7 SEQUENCELENGTH = 100
8 NO_OF_FEATURES = 16
9 BATCH_SIZE = 34
10 MAX_EPOCHS = 100
11 EARLY_STOPPING_PATIENCE = 5
12 MINIMUM_IMPROVEMENT = 0.01
13 # Regularisation
14 L1 = 0.001
15 L2 = 0.001
16 DROPOUT = 0.05
17
18
19 """ Number of sequences to be created from mouse events """
20 TRAIN_SIZE = 3500
21 VALID_SIZE = 1000
22 TEST_SIZE = 1000
23 UNSEEN_SIZE = 1500
24
25 """ This boolean flag can be set to False to accept 1500 unseen_size
    sequences rather than all (True) """
26 USE_UNBALANCED_UNSEEN_DATA = True
27
28 """ Number of iterations per experimental run """
29 REPETITIONS = 25

```

CODE INSERT A.4: CONMAN Source Code: Config.py

A.6.0.4 ModelCreator.py

```

1 import numpy as np
2 import tensorflow as tf
3 from keras.layers import Conv1D, Dense
4 from keras.models import Sequential
5 from keras.src.layers import Activation
6 from keras.src.regularizers import l1_l2
7 from tensorflow.python.keras.callbacks import EarlyStopping,
    ModelCheckpoint
8
9 import Config
10
11
12 def create_model(sequence_length, no_of_features, no_of_users):
13     """
14     Creates and compiles a 1D Convolutional Neural Network (CNN) model for
15     user authentication using mouse dynamics.
16
17     Args:
18         sequence_length (int): Length of the input sequences.
19         no_of_features (int): Number of features in the input data.
20         no_of_users (int): Number of users (classes) for the model to
21         classify.
22
23     Returns:
24         Sequential: A compiled Keras CNN model.
25
26     Sources:
27         https://machinelearningmastery.com/cnn-models-for-human-activity-
28         recognition-time-series-classification/
29     """
30     model = Sequential()
31     model.add(Conv1D(filters=32, kernel_size=3, input_shape=(
32         sequence_length, no_of_features)))
33     model.add(Activation('relu'))
34     model.add(tf.keras.layers.GlobalMaxPooling1D())
35
36     model.add(Dense(60, activation='relu', kernel_regularizer=l1_l2(l1=
37         Config.L1, l2=Config.L2)))
38     model.add(tf.keras.layers.Dropout(Config.DROPOUT))
39     model.add(Dense(no_of_users, activation='softmax'))
40     model.compile(optimizer='adam', loss='categorical_crossentropy',
41         metrics=['accuracy'])
42     return model
43
44 class ModelCreator:

```

```

41 """
42 A class responsible for creating, training, and evaluating a machine
43 learning model for user authentication.
44
45 Attributes:
46     training_sequences (np.array): Training data sequences.
47     training_labels (np.array): Training data labels.
48     validation_sequences (np.array): Validation data sequences.
49     validation_labels (np.array): Validation data labels.
50     test_sequences (np.array): Test data sequences.
51     test_labels (np.array): Test data labels.
52     no_of_users (int): Number of user classes.
53     sequence_length (int): Length of the input sequences.
54     no_of_features (int): Number of features in the input data.
55     model_store (str): Path to store the trained model.
56     model (Sequential): The machine learning model.
57     early_stopping_callback (EarlyStopping): Callback for early
58     stopping.
59     model_checkpoint_callback (ModelCheckpoint): Callback for model
60     checkpointing.
61
62 Methods:
63     train(): Trains the model using the provided data.
64     predict(sequences, labels): Makes predictions using the trained
65     model.
66
67 Sources:
68 https://machinelearningmastery.com/cnn-models-for-human-activity-
69 recognition-time-series-classification/
70 """
71
72 def __init__(self, training_sequences, training_labels,
73 validation_sequences, validation_labels,
74 test_sequences, test_labels, no_of_users, sequence_length,
75 no_of_features, model_store, run_num,
76 iteration):
77     self.training_sequences = training_sequences
78     self.training_labels = training_labels
79     self.validation_sequences = validation_sequences
80     self.validation_labels = validation_labels
81     self.test_sequences = test_sequences
82     self.test_labels = test_labels
83     self.no_of_users = no_of_users
84     self.sequence_length = sequence_length
85     self.no_of_features = no_of_features
86     self.model_store = model_store
87     self.model = create_model(self.sequence_length, self.no_of_features
88 , self.no_of_users)

```

```

81         self.early_stopping_callback = EarlyStopping(monitor="val_loss",
82             min_delta=Config.MINIMUMIMPROVEMENT,
83             patience=Config.
EARLY_STOPPING_PATIENCE,
84             restore_best_weights=
True)
85         self.model_checkpoint_callback = ModelCheckpoint(
86             f"model_hub/model_store/Run-{run_num}/model-{iteration}.hdf5",
87             save_best_only=True,
88             verbose=0,
89             mode="min",
90             monitor="val_loss"
91         )
92     def train(self):
93         """
94         Trains the CNN model using training and validation datasets.
95
96         Returns:
97             History object containing recorded training process information
98         """
99         print(
100             f"Training data shape: {self.training_sequences.shape},
Training labels shape: {self.training_labels.shape}")
101         history = self.model.fit(self.training_sequences, self.
training_labels, batch_size=Config.BATCH_SIZE,
102             epochs=Config.MAX_EPOCHS, callbacks=[
103             self.model_checkpoint_callback, self.early_stopping_callback],
104             validation_data=(self.validation_sequences
, self.validation_labels))
105         return history
106
107     def predict(self):
108         """
109         Generates predictions for the provided sequences using the trained
110         model.
111
112         Returns:
113             Tuple: Prediction probabilities, predicted labels, and true
114             labels.
115         """
116         y_pred_probs = self.model.predict(self.test_sequences)
117         y_pred = np.argmax(y_pred_probs, axis=1)
118         y_true = np.argmax(self.test_labels, axis=1)
119         return y_pred_probs, y_pred, y_true

```

CODE INSERT A.5: CONMAN Source Code: ModelCreator.py

A.6.0.5 ExperimentRunner.py

```

1 import numpy as np
2 import tensorflow as tf
3
4 import Config
5 from model_hub.ModelCreator import ModelCreator
6 from controllers.DataController import DataController
7 from controllers.ResultsController import ResultsController
8 from controllers.ExperimentController import ExperimentController
9
10
11 class ExperimentRunner:
12     """
13     A class responsible for orchestrating the execution of model training
14     and evaluation experiments.
15
16     Attributes:
17         run_num (int): The current run number of the experiment.
18         data_config (dict): Configuration settings for data processing.
19         user_labels (list): List of user labels for classification.
20         data_controller (DataController): Controller for managing data
21         loading and preprocessing.
22         best_val_loss (float): Best validation loss achieved during
23         training iterations.
24         best_model_iteration (int): Iteration number of the model with the
25         best validation loss.
26         accuracies, precisions, roc_aucs (list): Lists to store performance
27         metrics for each iteration.
28
29     Methods:
30         load_data(): Loads and preprocesses the data for the experiment.
31         run_training_experiment(): Runs the training experiment for
32         multiple iterations.
33         evaluate_best_model(): Evaluates the best performing model on
34         unseen data.
35     """
36     def __init__(self, run_num, data_config):
37         self.run_num = run_num
38         self.data_config = data_config
39         self.user_labels = data_config['user_labels']
40         self.data_controller = DataController(data_config['
41 raw_data_directory'], self.user_labels,
42                                             Config.SEQUENCELENGTH)
43
44         self.best_val_loss = float('inf')
45         self.best_model_iteration = 0
46         self.accuracies = []
47         self.precisions = []

```

```

39         self.roc_aucs = []
40
41     def load_data(self):
42         """
43         Loads and preprocesses the data for the experiment.
44
45         Returns:
46             Tuple: Training, validation, testing, and unseen data sequences
47             and labels.
48         """
49         featured_data_dict = self.data_controller.load()
50
51         experiment_controller = ExperimentController(Config.TRAIN_SIZE,
52             Config.VALID_SIZE, Config.TEST_SIZE,
53             Config.UNSEEN_SIZE,
54             self.user_labels, Config.SEQUENCELENGTH,
55             featured_data_dict)
56
57         return experiment_controller.load()
58
59     def run_training_experiment(self, training_sequences, training_labels,
60         validation_sequences, validation_labels,
61         test_sequences, test_labels):
62         """
63         Runs the training experiment for a specified number of iterations.
64
65         Args:
66             training_sequences, validation_sequences, test_sequences (np.
67             array): Data sequences for training, validation, and testing.
68             training_labels, validation_labels, test_labels (np.array):
69             Labels for the data sequences.
70
71             Each iteration involves training a model and evaluating its
72             performance on the test dataset.
73             The iteration with the best validation loss is recorded.
74         """
75         for i in range(Config.REPETITIONS):
76             print(f"Running Experiment Iteration: {i + 1}")
77
78             model = ModelCreator(training_sequences, training_labels,
79                 validation_sequences, validation_labels,
80                 test_sequences, test_labels, Config.
81                 NO_OF_USERS, Config.SEQUENCELENGTH,
82                 Config.NO_OF_FEATURES, Config.MODELSTORE,
83                 self.run_num, iteration=i + 1)
84             history = model.train()
85
86             y_pred_probs, y_pred, y_true = model.predict()

```

```

76         training_results_controller = ResultsController("Testing",
77         y_pred_probs, y_pred, y_true, self.user_labels,
78         self.run_num, i
79         , history)
80         training_results_controller.load()
81
82         current_val_loss = min(history.history['val_loss'])
83         if current_val_loss < self.best_val_loss:
84             self.best_val_loss = current_val_loss
85             self.best_model_iteration = i + 1
86
87     def evaluate_best_model(self, unseen_sequences, unseen_labels):
88         """
89         Evaluates the best performing model from the training experiment on
90         unseen data.
91
92         Args:
93             unseen_sequences (np.array): Unseen data sequences for
94             evaluation.
95             unseen_labels (np.array): Labels for the unseen data sequences.
96
97         """
98         best_model_filepath = f"{Config.MODELSTORE}/Run-{self.run_num}/
99         model-{self.best_model_iteration}.hdf5"
100         best_model = tf.keras.models.load_model(best_model_filepath)
101         unseen_y_pred_probs = best_model.predict(unseen_sequences)
102         unseen_y_pred = np.argmax(unseen_y_pred_probs, axis=1)
103         unseen_y_true = np.argmax(unseen_labels, axis=1)
104         unseen_results_controller = ResultsController(f"Best Model ({self.
105         best_model_iteration}) vs. Unseen",
106         unseen_y_pred_probs,
107         unseen_y_pred, unseen_y_true,
108         self.user_labels,
109         self.run_num, Config.REPETITIONS + 1, history=None)
110         unseen_results_controller.load()

```

CODE INSERT A.6: CONMAN Source Code: ExperimentRunner.py

A.6.0.6 Controller.py

```

1 from abc import ABC, abstractmethod
2
3
4 class Controller(ABC):
5     """
6     An abstract base class for controllers.
7
8     This class serves as a foundational template for other controller
9     classes
10    within the software architecture. It enforces the implementation of the
11    'load' method in all subclasses, ensuring a consistent interface across
12    different types of controllers.
13
14    Methods:
15        load: An abstract method that needs to be overridden by subclasses.
16              It defines the logic for loading or initialising resources
17              or processes specific to the controller.
18    """
19    @abstractmethod
20    def load(self):
21        """
22        Abstract method to load or initialise resources or processes.
23
24        This method must be implemented by all subclasses of the Controller
25        class.
26        """
27        pass

```

CODE INSERT A.7: CONMAN Source Code: Controller.py

A.6.0.7 DataController.py

```

1 import Config
2 from controllers.Controller import Controller
3 from controllers.data_utils.DataLoader import DataLoader
4 from controllers.data_utils.DataProcessor import DataProcessor
5
6
7 class DataController(Controller):
8     """
9     A class responsible for controlling the data loading, processing, and
10    preparation for the experiment.
11
12    Attributes:
13        user_directories (dict): Dictionary mapping user labels to their
14        data directories.
15        user_labels (dict): Mapping of user labels to integers.
16        sequence_length (int): Length of each data sequence.
17
18    Methods:
19        load(): Processes and prepares the data for the experiment.
20    """
21
22    def __init__(self, raw_data_directory, user_labels, sequence_length):
23        """
24        Initialises the DataController with the raw data directory, user
25        labels, and sequence length.
26
27        Args:
28            raw_data_directory (dict): Dictionary mapping user labels to
29            their data directories.
30            user_labels (dict): Mapping of user labels to integers.
31            sequence_length (int): Length of each data sequence.
32        """
33        self.user_directories = raw_data_directory
34        self.user_labels = user_labels
35        self.sequence_length = sequence_length
36
37    def load(self):
38        """
39        Loads, processes, and prepares data for training, validation,
40        testing, and unseen datasets.
41
42        Returns:
43            dict: Dictionary containing processed and featured data for all
44            users.
45        """

```

```

40     For each user , loads raw data , processes it , and splits it into
training , validation , testing , and unseen datasets .
41     Each dataset is then saved and added to the returned dictionary .
42     """
43     featured_data_dict = {}
44
45     # Process and save data for each user
46     for user , directory in self.user_directories.items():
47         print(f"Pre-processing and featurizing data for {user}")
48         data_loader = DataLoader
49
50         # Load data and calculate the total number of events
51         loaded_data = data_loader.load_data(directory)
52         total_events = len(loaded_data)
53         print(f"Total number of events for {user}: {total_events}")
54
55         data_processor = DataProcessor(loaded_data , user , Config.
USE_UNBALANCED_UNSEEN_DATA)
56
57         # Process data and get the split datasets
58         featured_train_data , featured_val_data , featured_test_data ,
featured_unseen_data = data_processor.process_dataset()
59
60         # Save the processed and featured data for each dataset
61         datasets = {
62             'train': featured_train_data ,
63             'val': featured_val_data ,
64             'test': featured_test_data ,
65             'unseen': featured_unseen_data
66         }
67
68         # Add the featured data to the dictionary
69         featured_data_dict[user] = datasets
70
71         print("All data split , processed and featured , preparing to create
sequences...")
72
73         # Return the dictionary containing featured data for all users
74         return featured_data_dict

```

CODE INSERT A.8: CONMAN Source Code: DataController.py

A.6.0.8 ExperimentController.py

```

1 import logging
2 import numpy as np
3
4 import Config
5 from controllers.Controller import Controller
6 from controllers.experiment_utils.DataNormaliser import DataNormaliser
7 from controllers.experiment_utils.DatasetCompiler import DatasetCompiler
8 from controllers.experiment_utils.SequenceCreator import SequenceCreator
9
10
11 class ExperimentController(Controller):
12     """
13     A class responsible for managing the experiment's data preparation and
14     sequence creation processes.
15
16     Attributes:
17         train_size, valid_size, test_size, unseen_size (int): Sizes of the
18         respective data partitions.
19         user_labels (dict): Mapping of user labels to integers.
20         sequence_length (int): Length of each data sequence.
21         featured_data_dict (dict): Dictionary containing featured data for
22         each user.
23         dataset_compiler (DatasetCompiler): Helper object for compiling
24         datasets.
25
26     Methods:
27         load(): Processes and compiles training, validation, testing, and
28         unseen datasets.
29         shuffle_sequences_labels(sequences, labels): Shuffles sequences and
30         their corresponding labels.
31     """
32
33     def __init__(self, train_size, valid_size, test_size, unseen_size,
34                 user_labels, sequence_length,
35                 featured_data_dict):
36         self.train_size = train_size
37         self.valid_size = valid_size
38         self.test_size = test_size
39         self.unseen_size = unseen_size
40         self.user_labels = user_labels
41         self.sequence_length = sequence_length
42         self.featured_data_dict = featured_data_dict
43         self.dataset_compiler = DatasetCompiler(user_labels)
44
45     def load(self):
46         """

```

```

40     Loads, normalises, creates sequences, and compiles datasets for
    training, validation, testing, and unseen data.
41
42     Returns:
43         Tuple: Compiled training, validation, testing, and unseen
    sequences and labels.
44
45     Processes the featured data for each user, normalises it, creates
    sequences, and compiles them into datasets.
46     If an error occurs during processing, it logs the error and returns
    empty datasets.
47     """
48     all_train_sequences, all_train_labels = [], []
49     all_valid_sequences, all_valid_labels = [], []
50     all_test_sequences, all_test_labels = [], []
51     all_unseen_sequences, all_unseen_labels = [], []
52
53     for user_label in self.user_labels:
54         try:
55             user_data = self.featured_data_dict[user_label]
56             train_data = user_data['train']
57             val_data = user_data['val']
58             test_data = user_data['test']
59             unseen_data = user_data['unseen']
60
61             data_normaliser = DataNormaliser()
62             scaled_data_dict = data_normaliser.normalise_and_save_data(
    train_data, val_data, test_data,
63
    unseen_data)
64             sequence_creator = SequenceCreator(self.sequence_length)
65
66             user_label_int = self.user_labels[user_label]
67             train_sequences, train_labels = sequence_creator.
    create_sequences(scaled_data_dict['training'],
68
    user_label_int,
69
    self.train_size)
70             validation_sequences, valid_labels = sequence_creator.
    create_sequences(scaled_data_dict['validation'],
71
    user_label_int,
72
    self.valid_size)
73             test_sequences, test_labels = sequence_creator.
    create_sequences(scaled_data_dict['testing'],

```

```

74         user_label_int ,
75
76         self.test_size)
77         unseen_sequences , unseen_labels = sequence_creator .
78         create_sequences(scaled_data_dict[ 'unseen' ],
79
80         user_label_int , self.unseen_size ,
81
82         Config.USE_UNBALANCED_UNSEEN_DATA)
83
84         print(f"{user_label}: Train sequences: {len(train_sequences
85         )}, Train labels: {len(train_labels)}")
86         print(
87             f"{user_label}: Validation sequences: {len(
88             validation_sequences)}, Validation labels: {len(valid_labels)}")
89         print(f"{user_label}: Test sequences: {len(test_sequences)
90         }, Test labels: {len(test_labels)}")
91         print(f"{user_label}: Unseen sequences: {len(
92         unseen_sequences)}, Unseen labels: {len(unseen_labels)}")
93
94         all_train_sequences.append(train_sequences)
95         all_train_labels.append(train_labels)
96         all_valid_sequences.append(validation_sequences)
97         all_valid_labels.append(valid_labels)
98         all_test_sequences.append(test_sequences)
99         all_test_labels.append(test_labels)
100        all_unseen_sequences.append(unseen_sequences)
101        all_unseen_labels.append(unseen_labels)
102
103        except Exception as e:
104            logging.error(f"Error processing data for {user_label}: {e}
105        ")
106
107        return [], [], [], [], [], [], [], []
108
109        # Shuffle only training sequences with their labels
110        concatenated_train_sequences = np.concatenate(all_train_sequences)
111        concatenated_train_labels = np.concatenate(all_train_labels)
112
113        concatenated_valid_sequences = np.concatenate(all_valid_sequences)
114        concatenated_valid_labels = np.concatenate(all_valid_labels)
115
116        shuffled_train_sequences , shuffled_train_labels = self .
117        shuffle_sequences_labels(concatenated_train_sequences ,
118
119
120        concatenated_train_labels)
121        shuffled_valid_sequences , shuffled_valid_labels = self .
122        shuffle_sequences_labels(concatenated_valid_sequences ,

```

```

108         concatenated_valid_labels)
109
110     # Compile the datasets from the processed data
111     compiled_datasets = self.dataset_compiler.compile_datasets(
112         shuffled_train_sequences, shuffled_train_labels,
113         shuffled_valid_sequences, shuffled_valid_labels,
114         np.concatenate(all_test_sequences), np.concatenate(
all_test_labels),
115         np.concatenate(all_unseen_sequences), np.concatenate(
all_unseen_labels)
116     )
117     if compiled_datasets is None:
118         return [], [], [], [], [], [], [], []
119
120     return compiled_datasets
121
122 def shuffle_sequences_labels(self, sequences, labels):
123     """
124     Shuffles the given sequences and their corresponding labels in
unison.
125
126     Args:
127         sequences (np.array): Array of data sequences.
128         labels (np.array): Array of labels corresponding to the
sequences.
129
130     Returns:
131         Tuple: Shuffled sequences and labels.
132
133     Ensures that the sequences and labels are shuffled in the same
order to maintain correct pairing.
134     """
135     assert len(sequences) == len(labels)
136     indices = np.arange(len(sequences))
137     np.random.shuffle(indices)
138     shuffled_sequences = sequences[indices]
139     shuffled_labels = labels[indices]
140     return shuffled_sequences, shuffled_labels

```

CODE INSERT A.9: CONMAN Source Code: ExperimentController.py

A.6.0.9 ResultsController.py

```

1 from controllers.Controller import Controller
2 from controllers.results_utils.MetricCalculator import MetricCalculator
3 from controllers.results_utils.OutputHandler import OutputHandler
4 from controllers.results_utils.Plotter import Plotter
5
6
7 class ResultsController(Controller):
8     """
9     A class responsible for calculating, outputting, and plotting the
10    results of the model's performance.
11
12    Attributes:
13        dataset_label (str): Label identifying the dataset.
14        user_labels (dict): Mapping of user labels to integers.
15        n_classes (int): Number of classes (users).
16        y_true (np.array): True labels for the dataset.
17        y_pred (np.array): Predicted labels from the model.
18        y_pred_probs (np.array): Predicted probabilities from the model.
19        history (History object): Training history of the model.
20        run_num (int): Scenario number of the experiment.
21        iteration (int): Iteration number of the experiment.
22        metric_calculator (MetricCalculator): Object for computing
23    performance metrics.
24        output_handler (OutputHandler): Object for handling output of
25    results.
26        plotter (Plotter): Object for plotting results.
27
28    Methods:
29        load(): Computes, outputs, and plots the model's performance
30    metrics.
31    """
32    def __init__(self, dataset_label, y_pred_probs, y_pred, y_true,
33    user_labels, run_num, iteration, history):
34        """
35        Initialises the ResultsController with the provided dataset
36    information and performance metrics.
37
38    Args:
39        dataset_label (str): Label identifying the dataset.
40        y_pred_probs (np.array): Predicted probabilities from the model
41    .
42        y_pred (np.array): Predicted labels from the model.
43        y_true (np.array): True labels for the dataset.
44        user_labels (dict): Mapping of user labels to integers.
45        run_num (int): Scenario number of the experiment.
46        iteration (int): Iteration number of the experiment.

```

```

40         history (History object): Training history of the model.
41     """
42     self.dataset_label = dataset_label
43     self.user_labels = user_labels
44     self.n_classes = len(self.user_labels)
45     self.y_true = y_true
46     self.y_pred = y_pred
47     self.y_pred_probs = y_pred_probs
48     self.history = history
49     self.run_num = run_num
50     self.iteration = iteration
51     self.metric_calculator = MetricCalculator(self.n_classes, self.
user_labels)
52     self.output_handler = OutputHandler()
53     self.plotter = Plotter(self.n_classes, self.user_labels, self.
y_true, self.y_pred_probs, self.history)
54
55     def load(self):
56         """
57         Computes and outputs the results of the model's performance.
58
59         Calculates overall and per-class metrics, outputs results to
console and CSV,
60         and plots training history and ROC-AUC-OVR curves.
61         """
62         # Compute overall and individual metrics.
63         overall_metrics = self.metric_calculator.compute_overall_metrics(
self.y_true, self.y_pred, self.y_pred_probs)
64         metrics_per_class, confusion_matrices = self.metric_calculator.
compute_individual_metrics(self.y_true,
65
66                             self.y_pred,
67
68                             self.y_pred_probs)
69
70         # Output the results to console and CSV file.
71         self.output_handler.output_results_to_console(self.dataset_label,
overall_metrics, metrics_per_class, confusion_matrices)
72         self.output_handler.output_results_to_csv(self.dataset_label,
overall_metrics, metrics_per_class, confusion_matrices, self.run_num,
self.iteration)
73
74         # Plot learning history
75         self.plotter.plot_history(self.history, self.run_num, self.
iteration)
76
77         # Plot ROC-AUC-OVR curves

```

```
76         self.plotter.plot_roc_auc_ovr(self.y_true, self.y_pred_probs, self.  
run_num, self.iteration)
```

CODE INSERT A.10: CONMAN Source Code: ResultsController.py

A.6.0.10 DataLoader.py

```

1 import os
2 import pandas as pd
3 import re
4 import logging
5
6 class DataLoader:
7     @staticmethod
8     def load_data(directory):
9         """
10         Loads data from all CSV files in a given directory and aggregates
11         it into a single DataFrame,
12         ensuring files are read in chronological order based on session
13         number.
14
15         Parameters:
16         directory (str): The path to the directory containing the CSV files
17         .
18
19         Returns:
20         pd.DataFrame: A DataFrame containing the aggregated data from all
21         CSV files in the directory.
22         Returns an empty DataFrame if an error occurs.
23         """
24         try:
25             # Regular expression to extract the session number from the
26             # filename
27             # https://stackoverflow.com/questions/47189714/reading-multiple
28             # -csv-files-from-a-folder-with-r-using-regex
29             session_pattern = re.compile(r"mouselog_session_(\d+)\.csv")
30
31             # List all CSV files and sort them based on the session number
32             all_files = [os.path.join(directory, f) for f in os.listdir(
33             directory)
34                         if os.path.isfile(os.path.join(directory, f)) and
35                         f.endswith('.csv')]
36             all_files.sort(key=lambda x: int(session_pattern.search(x).
37             group(1)) if session_pattern.search(x) else 0)
38
39             # Read and concatenate all files
40             data_list = [pd.read_csv(file) for file in all_files]
41             concatenated_data = pd.concat(data_list, ignore_index=True)
42
43             # Print the length of the extracted data
44             print(f"Total number of data points loaded: {len(
45             concatenated_data)}")
46             return concatenated_data

```

```
37         except Exception as e:  
38             logging.error(f"Error loading data from {directory}: {e}")  
39         return pd.DataFrame()
```

CODE INSERT A.11: CONMAN Source Code: DataLoader.py

A.6.0.11 DataProcessor.py

```

1 import pandas as pd
2 import numpy as np
3 import Config
4
5
6 class DataProcessor:
7     """
8     A class for processing and preparing mouse dynamics data for machine
9     learning.
10
11     Handles preprocessing of raw data, computation of additional features,
12     splitting the data
13     into training, validation, testing, and unseen sets, and returning the
14     processed data.
15
16     Attributes:
17         raw_data (pd.DataFrame): The raw mouse dynamics data.
18         user (str): Identifier for the user whose data is being processed.
19         file_name (str): Filename for storing processed data.
20
21     Methods:
22         preprocess_data(data): Preprocesses the raw data.
23         compute_features(data): Computes additional features from the data.
24         data_splitter(data): Splits the data into training, validation,
25         testing, and unseen datasets.
26         process_dataset(): Processes the raw data and returns the processed
27         datasets.
28     """
29
30     def __init__(self, data, user, use_unbalanced_unseen_dataset):
31         self.raw_data = data
32         self.user = user
33         self.file_name = f'_data_{self.user}'
34         self.use_unbalance_unseen_dataset = use_unbalanced_unseen_dataset
35
36     def preprocess_data(self, data):
37         """
38         Preprocesses the raw data by converting timestamps and encoding
39         categorical columns.
40
41         Args:
42             data (pd.DataFrame): The raw data to preprocess.
43
44         Returns:
45             pd.DataFrame: The preprocessed data.
46         """

```

```

41     data = data.copy()
42     data['Timestamp'] = pd.to_datetime(data['Timestamp'], format='%Y-%m
-%d %H:%M%S.%f')
43     data['Time_Diff'] = data['Timestamp'].diff().dt.total_seconds().
fillna(0)
44
45     event_encoded = pd.get_dummies(data['Event'], prefix='Event')
46     button_encoded = pd.get_dummies(data['Button'], prefix='Button')
47     action_encoded = pd.get_dummies(data['Action'], prefix='Action')
48
49     all_buttons = ['Button_Button.left', 'Button_Button.middle', '
Button_Button.right', 'Button_Button.x1',
50                   'Button_Button.x2']
51     # Not all buttons feature in all datasets, so these are
standardised across users.
52     for button_type in all_buttons:
53         if button_type not in button_encoded.columns:
54             button_encoded[button_type] = 0
55
56     data = pd.concat([data, event_encoded, button_encoded,
action_encoded], axis=1)
57     return data
58
59 def compute_features(self, data):
60     """
61     Computes additional features based on the preprocessed data.
62
63     Args:
64         data (pd.DataFrame): The data to compute features on.
65
66     Returns:
67         pd.DataFrame: The data with computed features.
68     """
69     data['Distance'] = np.sqrt(data['Position_X'].diff().pow(2) + data[
'Position_Y'].diff().pow(2))
70     data['Speed'] = data['Distance'] / data['Time_Diff']
71     data['Acceleration'] = data['Speed'].diff() / data['Time_Diff']
72     data['Angle'] = np.arctan2(data['Position_Y'].diff(), data[
'Position_X'].diff()) * 180 / np.pi
73     data['Jitter'] = data['Distance'].diff().abs()
74
75     data.replace([np.inf, -np.inf], np.nan, inplace=True)
76     data.fillna(0, inplace=True)
77     return data
78
79 def data_splitter(self, data):
80     """

```

```

81         Splits the data into training , validation , testing , and assigns the
           remaining data to the unseen dataset
82         based on a flag.
83
84         Args:
85             data (pd.DataFrame): The preprocessed data to split.
86             use_unbalance_unseen_dataset (bool): Flag indicating whether to
           use all remaining data for the unseen dataset.
87
88         Returns:
89             Tuple: Data split into training , validation , testing , and
           unseen datasets.
90
91         The method determines the size of the training , validation , and
           testing datasets based on predefined
92         sizes in the configuration (Config.py). If
           use_unbalance_unseen_dataset is True, all remaining data is assigned
93         to the unseen dataset. Otherwise, the unseen dataset is created
           with a predefined size , similar to the other datasets.
94         """
95         data = data.sort_values(by=['Timestamp'])
96
97         # Determines size of the splits based on the Config.py values for
           sequence_length and sizes.
98         train_size = (Config.TRAIN_SIZE * Config.SEQUENCELENGTH)
99         val_size = (Config.VALID_SIZE * Config.SEQUENCELENGTH)
100        test_size = (Config.TEST_SIZE * Config.SEQUENCELENGTH)
101
102        # Split data into training , validation , and testing datasets.
103        train_data = data.iloc[:train_size]
104        val_data = data.iloc[train_size:train_size + val_size]
105        test_data = data.iloc[train_size + val_size:train_size + val_size +
           test_size]
106
107        if self.use_unbalance_unseen_dataset:
108            # Assign all remaining data to the unseen dataset.
109            unseen_data = data.iloc[train_size + val_size + test_size:]
110        else:
111            # Allocate a predefined size to the unseen dataset.
112            unseen_size = (Config.UNSEEN_SIZE * Config.SEQUENCELENGTH)
113            unseen_data = data.iloc[train_size + val_size + test_size:
           train_size + val_size + test_size + unseen_size]
114
115        return train_data , val_data , test_data , unseen_data
116
117    def process_dataset(self):
118        """

```

```
119         Processes the raw data through preprocessing , splitting , and
        feature computation steps .
120
121     Returns :
122         Tuple: Processed training , validation , testing , and unseen data
        .
123     """
124     # Preprocess the data
125     preprocessed_data = self.preprocess_data(self.raw_data)
126
127     # Split the preprocessed data
128     train_data , val_data , test_data , unseen_data = self.data_splitter(
        preprocessed_data)
129
130     # Compute features for each dataset
131     featured_train_data = self.compute_features(train_data)
132     featured_val_data = self.compute_features(val_data)
133     featured_test_data = self.compute_features(test_data)
134     featured_unseen_data = self.compute_features(unseen_data)
135
136     # Return the datasets
137     return featured_train_data , featured_val_data , featured_test_data ,
        featured_unseen_data
```

CODE INSERT A.12: CONMAN Source Code: DataProcessor.py

A.6.0.12 DataNormaliser.py

```

1 from sklearn.preprocessing import MinMaxScaler
2
3
4 class DataNormaliser:
5     """
6     A class responsible for normalising and scaling mouse dynamics data
7     using MinMaxScaler.
8
9     Provides functionality to fit the scaler on the training data,
10    normalise various datasets,
11    and return the normalised data.
12
13    Attributes:
14        scaler (MinMaxScaler): An instance of MinMaxScaler for data
15        normalisation.
16
17    Methods:
18        fit_scaler(data): Fits the scaler to specific columns of the data.
19        normalise_data(data): Normalises the provided data using the fitted
20        scaler.
21        normalise_and_save_data(): Normalises training, validation, testing
22        , and unseen datasets.
23    """
24    def __init__(self):
25        """
26        Initialises the DataNormaliser class and creates an instance of
27        MinMaxScaler.
28        """
29        self.scaler = MinMaxScaler()
30
31    def fit_scaler(self, data):
32        """
33        Fits the MinMaxScaler to specific columns of the provided DataFrame
34        .
35
36        Args:
37            data (pd.DataFrame): DataFrame containing the data for fitting
38            the scaler.
39
40            Columns fitted include Position_X, Position_Y, Distance, Speed,
41            Acceleration, Angle, and Jitter.
42        """
43        self.scaler.fit(data[['Position_X', 'Position_Y', 'Distance', '
44        Speed', 'Acceleration', 'Angle', 'Jitter']])
45
46    def normalise_data(self, data):

```

```

37         """
38         Normalises the provided data using the fitted MinMaxScaler.
39
40         Args:
41             data (pd.DataFrame): DataFrame to be normalised.
42
43         Returns:
44             pd.DataFrame: Normalised DataFrame.
45         """
46         scaled_data = data.copy()
47         columns_to_scale = ['Position-X', 'Position-Y', 'Distance', 'Speed',
48                             'Acceleration', 'Angle', 'Jitter']
49         scaled_data[columns_to_scale] = self.scaler.transform(scaled_data[
50             columns_to_scale])
51         return scaled_data
52
53     def normalise_and_save_data(self, training_data, validation_data,
54                                testing_data, unseen_data):
55         """
56         Normalises training, validation, testing, and unseen datasets using
57         the fitted scaler.
58
59         Args:
60             training_data, validation_data, testing_data, unseen_data (pd.
61             DataFrame): Datasets to be normalised.
62
63         Returns:
64             dict: Dictionary containing normalised datasets.
65
66         Normalises each dataset and drops irrelevant columns before
67         returning them in a dictionary.
68         """
69         # Fit the scaler on the training data
70         self.fit_scaler(training_data)
71
72         # Normalise the training data
73         scaled_training_data = self.normalise_data(training_data)
74
75         # Normalise the other datasets using the same fitted scaler
76         scaled_validation_data = self.normalise_data(validation_data)
77         scaled_testing_data = self.normalise_data(testing_data)
78         scaled_unseen_data = self.normalise_data(unseen_data)
79
80         # Create a dictionary to map data types to their respective
81         normalised DataFrames
82         scaled_data_dict = {
83             'training': scaled_training_data,
84             'validation': scaled_validation_data,

```

```
78         'testing': scaled_testing_data ,
79         'unseen': scaled_unseen_data
80     }
81
82     # Iterate through the normalised data and save each
83     for data_type, scaled_data in scaled_data_dict.items():
84         # Dropping irrelevant columns
85         columns_to_drop = [ 'Timestamp', 'Event', 'Button', 'Action', '
Time.Diff' ]
86         scaled_data.drop(columns=columns_to_drop, inplace=True, errors=
'ignore')
87
88     return scaled_data_dict
```

CODE INSERT A.13: CONMAN Source Code: DataNormaliser.py

A.6.0.13 DatasetCompiler.py

```
1 import logging
2
3 import numpy as np
4 from tensorflow.keras.utils import to_categorical
5
6
7 class DatasetCompiler:
8     """
9     A class responsible for compiling and formatting datasets for use in
10     machine learning models.
11
12     It handles the conversion of sequence data to a suitable numerical
13     format and transforms label
14     data into a categorical format for classification tasks.
15
16     Attributes:
17         user_labels (dict): Mapping of user identifiers to integer labels.
18
19     Methods:
20         compile_datasets(): Compiles and formats training, validation,
21         testing, and unseen datasets.
22     """
23
24     def __init__(self, user_labels):
25         """
26         Initialises the DatasetCompiler with the provided user labels
27         mapping.
28
29         Args:
30             user_labels (dict): A dictionary mapping user identifiers to
31             integer labels.
32         """
33         self.user_labels = user_labels
34
35     def compile_datasets(self, train_sequences, train_labels,
36                         valid_sequences, valid_labels, test_sequences,
37                         test_labels, unseen_sequences, unseen_labels):
38         """
39         Compiles and formats the training, validation, testing, and unseen
40         datasets.
41
42         Args:
43             train_sequences, valid_sequences, test_sequences,
44             unseen_sequences (np.array): Sequences for each dataset.
45             train_labels, valid_labels, test_labels, unseen_labels (np.
46             array): Labels for each dataset.
```

```

38
39     Returns:
40         Tuple: Compiled and formatted training, validation, testing,
         and unseen datasets.
41
42         The method checks if any dataset is empty and logs an error if so.
         It converts sequence data to float32
43         and label data to categorical format, and then returns the
         processed datasets.
44         """
45         if train_sequences.size == 0 or valid_sequences.size == 0 or
         test_sequences.size == 0 or unseen_sequences.size == 0:
46             logging.error("One or more of the datasets are empty.")
47             return None
48
49         # Convert sequence data to float32.
50         train_sequences = train_sequences.astype(np.float32)
51         valid_sequences = valid_sequences.astype(np.float32)
52         test_sequences = test_sequences.astype(np.float32)
53         unseen_sequences = unseen_sequences.astype(np.float32)
54
55         # Convert label data to categorical format.
56         # https://stackoverflow.com/questions/54399752/i-am-trying-to-to-
         # convert-my-variables-to-categorical-using-to-categorical-metho
57         train_labels = to_categorical(train_labels, num_classes=len(self.
         user_labels))
58         valid_labels = to_categorical(valid_labels, num_classes=len(self.
         user_labels))
59         test_labels = to_categorical(test_labels, num_classes=len(self.
         user_labels))
60         unseen_labels = to_categorical(unseen_labels, num_classes=len(self.
         user_labels))
61
62         return (train_sequences, train_labels, valid_sequences,
         valid_labels, test_sequences, test_labels,
63                 unseen_sequences, unseen_labels)

```

CODE INSERT A.14: CONMAN Source Code: DatasetCompiler.py

A.6.0.14 SequenceCreator.py

```

1 import numpy as np
2
3
4 class SequenceCreator:
5     """
6     A class for creating sequences of a specified length from time-series
7     data.
8
9     This class is responsible for generating sequences and their
10    corresponding target labels
11    from the provided dataset, suitable for training machine learning
12    models.
13
14    Attributes:
15        sequence_length (int): The length of each sequence to be created.
16
17    Methods:
18        create_sequences(): Generates sequences and their corresponding
19        labels from the data.
20    """
21
22    def __init__(self, sequence_length):
23        """
24        Initialises the SequenceCreator with the specified sequence length.
25
26        Args:
27            sequence_length (int): The length of sequences to be created
28            from the data.
29        """
30        self.sequence_length = sequence_length
31
32    def create_sequences(self, data, user_label, num_sequences,
33        generate_max_sequences=False):
34        """
35        Creates non-overlapping sequences and their corresponding target
36        labels from the provided data.
37
38        Args:
39            data (pd.DataFrame): The DataFrame containing time-series data.
40            user_label (int): The label associated with the sequences (user
41            identifier).
42            num_sequences (int): The number of sequences to generate.
43            generate_max_sequences (bool, optional): Flag to generate the
44            maximum possible sequences from the data.
45
46            Overrides the
47            num_sequences parameter if set to True.

```

```

37
38     Returns:
39         Tuple: A tuple containing an array of sequences and an array of
40               their corresponding target labels.
41
42         Each sequence consists of consecutive, non-overlapping data points
43         of a specified length.
44         If generate_max_sequences is True, the method generates the maximum
45         possible number of sequences.
46         Otherwise, if num_sequences is specified and non-zero, it generates
47         the specified number of sequences.
48         If num_sequences is zero or not specified, it generates the maximum
49         possible number of sequences.
50         The start index of each sequence is incremented by the sequence
51         length, ensuring non-overlapping sequences.
52         """
53
54         sequences = []
55         targets = []
56         data_length = len(data)
57         max_sequences = data_length // self.sequence_length
58
59         if generate_max_sequences:
60             num_sequences = max_sequences
61         else:
62             num_sequences = min(num_sequences, max_sequences)
63
64         for i in range(num_sequences):
65             # Increment by sequence length for non-overlapping sequences
66             # https://stackoverflow.com/questions/60142153/python-3-slice-a
67             -rhyme
68             start_index = i * self.sequence_length
69             end_index = start_index + self.sequence_length
70             seq = data.iloc[start_index:end_index].values
71             sequences.append(seq)
72             targets.append(user_label)
73
74         return np.array(sequences), np.array(targets)

```

CODE INSERT A.15: CONMAN Source Code: SequenceCreator.py

A.6.0.15 OutputHandler.py

```

1 import csv
2 import os
3
4
5 class OutputHandler:
6     """
7     A class for handling the output of machine learning model performance
8     metrics.
9
10    This class provides functionality to output model performance metrics,
11    including overall and per-class metrics,
12    to the console and to a CSV file, facilitating both immediate review
13    and longer-term storage of results.
14
15    Methods:
16        output_results_to_console: Outputs metrics to the console.
17        output_results_to_csv: Outputs metrics to a CSV file.
18    """
19
20    def output_results_to_csv(self, dataset_label, overall_metrics,
21                             metrics_per_class, confusion_matrices, run,
22                             iteration):
23
24        # Create directories for saving results
25        base_dir = f'Experiment-Results/Run-{run}/Iteration-{iteration}'
26        os.makedirs(base_dir, exist_ok=True)
27
28        # Output Overall Metrics to CSV
29        overall_filename = os.path.join(base_dir, f'{dataset_label}
30        _Overall-Metrics.csv')
31        with open(overall_filename, 'w', newline='') as file:
32            writer = csv.writer(file)
33            # Write header with metric names
34            writer.writerow(list(overall_metrics.keys()))
35            # Write row with metric values
36            writer.writerow(list(overall_metrics.values()))
37
38        # Output Per-User Metrics and Confusion Matrices
39        user_filename = os.path.join(base_dir, f'{dataset_label}
40        _User-Metrics.csv')
41        with open(user_filename, 'w', newline='') as file:
42            # Prepare headers based on available metrics and confusion
43            matrix
44            headers = ['User'] + list(next(iter(metrics_per_class.values()))
45            ).keys()) + \

```

```

38         ['True Negative', 'False Positive', 'False Negative',
39         'True Positive']
40         writer = csv.DictWriter(file, fieldnames=headers)
41         writer.writeheader()
42
43         # Write metrics for each user
44         for user, metrics in metrics_per_class.items():
45             # Combine metrics with confusion matrix for the user
46             cm = confusion_matrices[user]
47             row_data = {
48                 'User': user,
49                 **metrics, # Unpack metrics into the row
50                 'True Negative': cm[0, 0],
51                 'False Positive': cm[0, 1],
52                 'False Negative': cm[1, 0],
53                 'True Positive': cm[1, 1]
54             }
55             writer.writerow(row_data)
56
57     def output_results_to_console(self, dataset_label, overall_metrics,
58     metrics_per_class, confusion_matrices):
59         """
60         Outputs the model performance metrics to the console.
61
62         Parameters:
63         overall_metrics (dict): Dictionary containing overall performance
64         metrics.
65         metrics_per_class (dict): Dictionary containing performance metrics
66         for each class.
67         confusion_matrices (dict): Dictionary containing confusion matrices
68         for each class.
69         """
70         print(f"Overall {dataset_label} Model Performance:")
71         for metric_name, metric_value in overall_metrics.items():
72             print(f"{metric_name}: {metric_value:.3f}")
73
74         print(f"\nIndividual {dataset_label} Class Performance:")
75         for user, metrics in metrics_per_class.items():
76             print(f"\nMetrics for {user}:")
77             for metric_name, metric_value in metrics.items():
78                 print(f"    {metric_name}: {metric_value:.3f}")
79             print(f"Confusion Matrix for {user}:\n{confusion_matrices[user]}")

```

CODE INSERT A.16: CONMAN Source Code: OutputHandler.py

A.6.0.16 MetricCalculator.py

```

1 from sklearn.metrics import accuracy_score, precision_score, recall_score,
  f1_score, roc_auc_score, confusion_matrix
2 from tensorflow.keras.utils import to_categorical
3
4
5 class MetricCalculator:
6     """
7     A class for calculating various metrics to evaluate the performance of
8     machine learning models.
9
10    This class computes both overall and per-class performance metrics,
11    providing insights
12    into the model's effectiveness and accuracy in predictions.
13
14    Attributes:
15        user_labels (dict): A dictionary mapping user identifiers to labels
16        .
17        n_classes (int): The number of classes in the dataset.
18    """
19
20    def __init__(self, n_classes, user_labels):
21        self.user_labels = user_labels
22        self.n_classes = n_classes
23
24    def compute_overall_metrics(self, y_true, y_pred, y_pred_probs):
25        """
26        Computes overall metrics for classification results.
27
28        Parameters:
29            y_true (array): True labels.
30            y_pred (array): Predicted labels.
31            y_pred_probs (array): Predicted probabilities for each class.
32
33        Returns:
34            dict: A dictionary containing overall performance metrics.
35        """
36        overall_metrics = {
37            'Accuracy': accuracy_score(y_true, y_pred),
38            'Precision_Macro': precision_score(y_true, y_pred, average='
macro'),
39            'Recall_Macro': recall_score(y_true, y_pred, average='macro'),
40            'F1_Macro': f1_score(y_true, y_pred, average='macro')
41        }
42
43        # Convert y_true to one-hot encoding for multi-class ROC AUC

```

```

41         y_true_binarized = to_categorical(y_true, num_classes=self.
n_classes)
42         overall_metrics['ROC-AUC-OvR'] = roc_auc_score(y_true_binarized,
y_pred_probs, multi_class='ovr')
43
44         return overall_metrics
45
46     def compute_individual_metrics(self, y_true, y_pred, y_pred_probs):
47         """
48         Calculates performance metrics for each class, including confusion
matrices.
49
50         Parameters:
51         y_true (array): True labels of the test dataset.
52         y_pred (array): Predicted labels by the model.
53         y_pred_probs (array): Prediction probabilities by the model.
54
55         Returns:
56         Tuple: A dictionary of metrics for each class and a dictionary of
confusion matrices for each class.
57         """
58         metrics_per_class = {}
59         confusion_matrices = {}
60
61         # Calculate metrics and confusion matrix for each class.
62         for i in range(self.n_classes):
63             user_label = list(self.user_labels.keys())[i]
64             current_test_labels = (y_true == i).astype(int)
65
66             cm = confusion_matrix(current_test_labels, y_pred == i)
67             confusion_matrices[user_label] = cm
68
69             user_metrics = {
70                 "Precision": precision_score(current_test_labels, y_pred ==
i, zero_division=0),
71                 "F1": f1_score(current_test_labels, y_pred == i),
72                 "ROC-AUC-OvR": roc_auc_score(current_test_labels,
y_pred_probs[:, i]),
73                 "TPR": self.true_positive_rate(cm),
74                 "TNR": self.true_negative_rate(cm),
75                 "FAR": self.false_alarm_rate(cm),
76                 "FRR": self.false_rejection_rate(cm)
77             }
78             metrics_per_class[user_label] = user_metrics
79
80         return metrics_per_class, confusion_matrices
81

```

```
82  # Additional methods to calculate specific rates from confusion
    matrices.
83  def true_positive_rate(self, confusion_matrix):
84      TP, FN = confusion_matrix[1, 1], confusion_matrix[1, 0]
85      return TP / (TP + FN) if TP + FN > 0 else 0
86
87  def true_negative_rate(self, confusion_matrix):
88      TN, FP = confusion_matrix[0, 0], confusion_matrix[0, 1]
89      return TN / (TN + FP) if TN + FP > 0 else 0
90
91  def false_alarm_rate(self, confusion_matrix):
92      FP, TN = confusion_matrix[0, 1], confusion_matrix[0, 0]
93      return FP / (FP + TN) if FP + TN > 0 else 0
94
95  def false_rejection_rate(self, confusion_matrix):
96      FN, TP = confusion_matrix[1, 0], confusion_matrix[1, 1]
97      return FN / (FN + TP) if FN + TP > 0 else 0
```

CODE INSERT A.17: CONMAN Source Code: MetricCalculator.py

A.6.0.17 Plotter.py

```

1 import matplotlib.pyplot as plt
2 import os
3 from itertools import cycle
4 from sklearn.metrics import roc_curve, auc
5 from sklearn.preprocessing import label_binarize
6
7
8 class Plotter:
9     """
10     The Plotter class is designed for visualising the performance of models
11     ,
12     providing methods for plotting training history and
13     Receiver Operating Characteristic (ROC) curves for multi-class
14     classification.
15
16     Attributes:
17         user_labels (dict): A dictionary mapping user identifiers to labels
18         .
19         n_classes (int): The number of classes in the dataset.
20         y_true (array): True labels of the dataset.
21         y_pred_probs (array): Predicted probabilities for each class.
22         history (object): Training history object containing accuracy and
23         loss values per epoch.
24
25     Sources:
26     https://machinelearningmastery.com/plotting-the-training-and-validation-loss-curves-for-the-transformer-model/
27     """
28     def __init__(self, n_classes, user_labels, y_true, y_pred_probs,
29         history):
30         self.user_labels = user_labels
31         self.n_classes = n_classes
32         self.y_true = y_true
33         self.y_pred_probs = y_pred_probs
34         self.history = history
35
36     def plot_history(self, history, run, iteration):
37         if history is None:
38             return
39
40         fig, axs = plt.subplots(2)
41         axs[0].plot(history.history["accuracy"], label="train accuracy")
42         axs[0].plot(history.history["val_accuracy"], label="validation
43 accuracy")
44         axs[0].set_ylabel("Accuracy")
45         axs[0].legend(loc="lower right")

```

```

40     axs[0].set_title("Training vs Validation Accuracy & Loss")
41
42     axs[1].plot(history.history["loss"], label="train loss")
43     axs[1].plot(history.history["val_loss"], label="validation loss")
44     axs[1].set_ylabel("Loss")
45     axs[1].set_xlabel("Epoch")
46     axs[1].legend(loc="upper right")
47
48     # Save the plot to a file
49     base_dir = f'Experiment-Results/Run-{run}/Iteration-{iteration}'
50     os.makedirs(base_dir, exist_ok=True)
51     plot_path = os.path.join(base_dir, f'
training-vs-validation-acc-loss-plot-iteration-{iteration}.png')
52     plt.savefig(plot_path)
53     plt.close()
54
55     def plot_roc_auc_ovr(self, y_true, y_pred_probs, run, iteration):
56         """
57         Plots the Receiver Operating Characteristic (ROC) curves for each
58         class, using One-vs-Rest (OvR) approach.
59
60         Args:
61             y_true (array): True labels of the dataset.
62             y_pred_probs (array): Predicted probabilities for each class.
63             run (int): The current run number.
64             iteration (int): The current iteration number.
65         """
66         user_labels = ['A', 'B', 'C']
67         y_true_binarized = label_binarize(y_true, classes=range(self.
n_classes))
68         fpr, tpr, roc_auc = {}, {}, {}
69         colors = cycle(['blue', 'red', 'green'])
70
71         plt.figure(figsize=(10, 8))
72         for i, (color, label) in enumerate(zip(colors, user_labels)):
73             fpr[i], tpr[i], _ = roc_curve(y_true_binarized[:, i],
y_pred_probs[:, i])
74             roc_auc[i] = auc(fpr[i], tpr[i])
75             plt.plot(fpr[i], tpr[i], color=color, lw=2,
label='ROC curve of User {0} (area = {1:0.2f})'.format
(label, roc_auc[i]))
76
77         plt.plot([0, 1], [0, 1], 'k--', lw=2)
78         plt.xlim([0.0, 1.0])
79         plt.ylim([0.0, 1.05])
80         plt.xlabel('False Positive Rate')
81         plt.ylabel('True Positive Rate')
82         plt.title('ROC-AUC-OVR for Multi-class')

```

```
83     plt.legend(loc="lower right")
84
85     # Save the ROC plot to a file
86     base_dir = f'Experiment_Results/Run-{run}/Iteration-{iteration}'
87     os.makedirs(base_dir, exist_ok=True)
88     plot_path = os.path.join(base_dir, f'roc_auc_ovr_iteration{
iteration}.png')
89     plt.savefig(plot_path)
90     plt.close()
```

CODE INSERT A.18: CONMAN Source Code: Plotter.py

A.6.0.18 test_compute_features.py

```

1 import unittest
2 import pandas as pd
3
4 from controllers.data_utils.DataProcessor import DataProcessor
5
6 class TestComputeFeatures(unittest.TestCase):
7
8     @classmethod
9     def setUpClass(cls):
10         cls.test_data = pd.read_csv('test_suite/test_utils/
test_mouselog_session.csv')
11         cls.user = "test_user"
12         cls.use_unbalanced_unseen_dataset = False
13
14     def test_compute_features(self):
15         # Create an instance of the class containing the compute_features
method
16         data_processor = DataProcessor(self.test_data, self.user, self.
use_unbalanced_unseen_dataset)
17         processed_data = data_processor.preprocess_data(self.test_data)
18         self.assertEqual(processed_data.shape[1], self.test_data.shape[1] +
19                             10)
20
21         # Compute features on the test data
22         processed_data_capture = processed_data.copy()
23         computed_data = data_processor.compute_features(processed_data)
24
25         # Check if the expected columns are present
26         for column in ['Distance', 'Speed', 'Acceleration', 'Angle', '
Jitter']:
27             self.assertIn(column, computed_data.columns)
28
29         # Check for NaN values
30         self.assertFalse(computed_data.isnull().values.any())
31
32         # Check data integrity (number of rows and columns remains the same
)
33         self.assertEqual(computed_data.shape[0], self.test_data.shape[0])
34         self.assertEqual(computed_data.shape[1], self.test_data.shape[1] +
15)
35         self.assertEqual(computed_data.shape[1], processed_data_capture.
shape[1] + 5)

```

CODE INSERT A.19: CONMAN Source Code: test_compute_features.py

A.6.0.19 test_create_model.py

```

1 import unittest
2 from keras.models import Sequential
3 from model_hub.ModelCreator import create_model
4
5 sequence_length = 15
6 no_of_features = 20
7 no_of_users = 30
8 dropout_rate = 0.5
9
10 class TestCreateModel(unittest.TestCase):
11
12     def test_model_creation(self):
13         # Define parameters for the model
14         sequence_length = 100
15         no_of_features = 10
16         no_of_users = 3
17
18         # Create the model
19         model = create_model(sequence_length, no_of_features, no_of_users)
20
21         # Check if the model is an instance of Sequential
22         self.assertIsInstance(model, Sequential)
23
24         # Check if the model has the correct number of layers
25         self.assertEqual(len(model.layers), 6)
26
27         # Check if the first layer is a Conv1D layer with the correct
28         # configuration
29         conv1d_layer = model.layers[0]
30         self.assertEqual(conv1d_layer.__class__.__name__, 'Conv1D')
31         self.assertEqual(conv1d_layer.filters, 32)
32         self.assertEqual(conv1d_layer.kernel_size, (3,))
33         self.assertEqual(conv1d_layer.input_shape, (None, sequence_length,
34             no_of_features))
35
36         # Check if the last layer is a Dense layer with the correct
37         # configuration
38         output_layer = model.layers[-1]
39         self.assertEqual(output_layer.__class__.__name__, 'Dense')
40         self.assertEqual(output_layer.units, no_of_users)
41         self.assertEqual(output_layer.activation.__name__, 'softmax')
42
43         # Check the compilation settings of the model
44         self.assertEqual(model.optimizer.__class__.__name__, 'Adam')
45         self.assertEqual(model.loss, 'categorical_crossentropy')

```

```
44 if __name__ == '__main__':  
45     unittest.main()
```

CODE INSERT A.20: CONMAN Source Code: test_create_model.py

A.6.0.20 test_data_splitter.py

```

1 import unittest
2 import pandas as pd
3 import Config
4 from controllers.data_utils.DataProcessor import DataProcessor # Replace
   with the actual module and class names
5
6
7 class TestDataSplitter(unittest.TestCase):
8     def setUp(self):
9         # Sample DataFrame
10        self.data = pd.DataFrame({
11            'Timestamp': pd.date_range(start='1/1/2021', periods=1500000,
12            freq='T'),
13            'Feature1': range(1500000),
14            'Feature2': range(1500000)
15        })
16
17        self.instance = DataProcessor(self.data, "A", True)
18
19    def test_correct_split_sizes(self):
20        train_data, val_data, test_data, unseen_data = self.instance.
21        data_splitter(self.data)
22
23        self.assertEqual(len(train_data), Config.TRAIN_SIZE * Config.
24        SEQUENCELENGTH)
25        self.assertEqual(len(val_data), Config.VALID_SIZE * Config.
26        SEQUENCELENGTH)
27        self.assertEqual(len(test_data), Config.TEST_SIZE * Config.
28        SEQUENCELENGTH)
29        # Check unseen data size based on the flag
30        if Config.USEUNBALANCEDUNSEEN_DATA:
31            expected_unseen_size = len(self.data) - (len(train_data) + len(
32            val_data) + len(test_data))
33        else:
34            expected_unseen_size = Config.UNSEEN_SIZE * Config.
35            SEQUENCELENGTH
36        self.assertEqual(len(unseen_data), expected_unseen_size)
37
38    def test_data_ordering(self):
39        -, -, -, - = self.instance.data_splitter(self.data)
40        sorted_data = self.data.sort_values(by=['Timestamp'])
41        pd.testing.assert_frame_equal(self.data, sorted_data)
42
43    def test_unbalanced_unseen_data_flag(self):
44        # Set the flag to True and False and test the sizes of the unseen
45        dataset

```

```
38         self.instance.use_unbalance_unseen_dataset = True
39         -, -, -, unseen_data_true = self.instance.data_splitter(self.data)
40
41         self.instance.use_unbalance_unseen_dataset = False
42         -, -, -, unseen_data_false = self.instance.data_splitter(self.data)
43
44         self.assertNotEqual(len(unseen_data_true), len(unseen_data_false))
45
46
47 if __name__ == '__main__':
48     unittest.main()
```

CODE INSERT A.21: CONMAN Source Code: test_data_splitter.py

A.6.0.21 test_preprocess_data.py

```

1 import unittest
2 import pandas as pd
3
4 from controllers.data_utils.DataProcessor import DataProcessor
5
6 class TestPreprocessData(unittest.TestCase):
7
8     @classmethod
9     def setUpClass(cls):
10         cls.test_data = pd.read_csv('test_suite/test_utils/
test_mouselog_session.csv')
11         cls.user = "test_user"
12         cls.use_unbalanced_unseen_dataset = False
13
14     def test_preprocess_data(self):
15         # Create an instance of the class containing the compute_features
method
16         data_processor = DataProcessor(self.test_data, self.user, self.
use_unbalanced_unseen_dataset)
17         processed_data = data_processor.preprocess_data(self.test_data)
18         self.assertEqual(processed_data.shape[1], self.test_data.shape[1] +
19                             10)
20
21         for column in ['Time_Diff', 'Event_Mouse Click', 'Event_Mouse
Movement', 'Button_Button.left',
22                         'Button_Button.middle', 'Button_Button.right', '
Button_Button.x1', 'Button_Button.x2',
23                         'Action_Pressed', 'Action_Released']:
24             self.assertIn(column, processed_data.columns)

```

CODE INSERT A.22: CONMAN Source Code: test_preprocess_data.py

A.6.0.22 test_sequence_creator.py

```

1 import unittest
2 import pandas as pd
3 import numpy as np
4 from controllers.experiment_utils.SequenceCreator import SequenceCreator
5
6 class TestSequenceCreator(unittest.TestCase):
7     def setUp(self):
8         # Define sequence length
9         self.sequence_length = 10
10
11        # Create sample data
12        self.sample_data = pd.DataFrame({
13            'Feature1': np.random.rand(100),
14            'Feature2': np.random.rand(100)
15        })
16
17        self.sequence_creator = SequenceCreator(self.sequence_length)
18
19        def test_sequence_generation(self):
20            sequences, _ = self.sequence_creator.create_sequences(self.
21            sample_data, user_label=1, num_sequences=5)
22            self.assertEqual(sequences.shape, (5, self.sequence_length, 2)) #
23            5 sequences, each of length 10, with 2 features
24
25            def test_correct_number_of_sequences(self):
26                sequences, _ = self.sequence_creator.create_sequences(self.
27                sample_data, user_label=1, num_sequences=3)
28                self.assertEqual(len(sequences), 3)
29
30                sequences, _ = self.sequence_creator.create_sequences(self.
31                sample_data, user_label=1, num_sequences=20)
32                self.assertLessEqual(len(sequences), 20)
33
34                sequences, _ = self.sequence_creator.create_sequences(self.
35                sample_data, user_label=1, num_sequences=0, generate_max_sequences=True
36                )
37                self.assertEqual(len(sequences), len(self.sample_data) // self.
38                sequence_length)
39
40            def test_non_overlapping_sequences(self):
41                sequences, _ = self.sequence_creator.create_sequences(self.
42                sample_data, user_label=1, num_sequences=5)
43                for i in range(len(sequences) - 1):
44                    self.assertFalse(np.array_equal(sequences[i], sequences[i+1]))
45
46            def test_target_labels(self):

```

```
39         _, targets = self.sequence_creator.create_sequences(self.  
40         sample_data, user_label=1, num_sequences=5)  
41         self.assertTrue((targets == 1).all())  
42 if __name__ == '__main__':  
43     unittest.main()
```

CODE INSERT A.23: CONMAN Source Code: test_sequence_creator.py

Appendix B

Project Briefing and Informed Consent Confirmation

B.1 Introduction

Thank you for considering participation in my study on mouse dynamics. This document provides essential information about the study, ensuring you are fully informed before deciding to participate.

B.2 Purpose of the Study

The goal of this study is to collect and analyse mouse dynamic data, such as cursor position, mouse movements, mouse clicks, and timestamps, to train a 1D Convolutional Neural Network (1D-CNN) model. The model aims to classify users based on their unique mouse interaction patterns. This research is to explore the opportunity deep learning can provide to enhance user authentication methods to improve security systems.

B.3 What Will Be Collected

During the study, the software will collect the following mouse dynamic data:

- Cursor position coordinates
- Timestamps of mouse movements

- Mouse button clicks and releases

Please note: No personal or sensitive data will be captured. The study focuses solely on mouse movement and interaction data, no information regarding applications used, websites visited or keyboard interactions will be collected during this research.

B.4 Anonymity and Data Privacy

- All data collected will be completely anonymised. Participant identities will not be recorded or associated with the collected data.
- The data will be used exclusively for research purposes.
- The data will be shared and may be reviewed by project coordinators at the time of marking my project as part of my MSc Software Engineering evaluation at the University of Wales, Trinity Saint David.
- Your data will not be shared with third parties without **additional** consent, separate to this form.
- Data storage and handling will comply with data protection regulations to ensure confidentiality and security.

B.5 Voluntary Participation

Your participation in this study is entirely voluntary. You have the right to withdraw from the study at any time, no questions asked. With your withdrawal your data if in my possession will be destroyed securely. If the data is in your possession at time of withdrawal I will be happy to assist you in the safe discarding of the data and the removal of the programme from your personal machine should you need it.

B.6 Review of Source Code

Should you wish to review the source code of this project it is available here, or via email upon request:

<https://github.com/Lez-Dev/MouseLogger>

If you have any questions related to the software at any time during or after your participation, please feel free to contact me.

B.7 Contact Information

Liam Kelly

Email: 2222134@student.uwtsd.ac.uk or liamk0157@gmail.com

Phone: 07774557614

Consent

If you decide to participate in this study, please confirm your consent by signing below.

I, _____, hereby confirm that I have read and understood the information provided in this informed consent briefing document and agree to participate in the mouse dynamics study.

Signature: _____

Date: _____

Appendix C

Software Requirements Specification

C.1 Introduction

This Software Requirements Specification (SRS) provides a detailed description of the software deliverable requirements and considerations for the research project during the implementation of a **mouse logger** and **Convolutional Mouse Authentication Network** (CONMAN) service. This document will explain the purpose, features and requirements of the system.

C.1.1 Scope

Two software artefacts will be produced - the first a mouse-logger will be implemented on participants local machines, recording mouse dynamic data over the course of user *sessions*. The second artefact, the CONMAN service - containing a one-dimensional convolutional neural network (1DCNN), and associated controllers to process and load data input to the model as well as analysing resulting output from the model. These artefacts serve as two arms of the research study in order to answer the hypothesis of the associated research.

Employing a 1D-CNN Deep Learning model to classify organic mouse dynamic data as a behavioural biometric, will bolster security by correctly classifying users generalised data with an accuracy of 95% and FAR of 1% or less.

C.2 Functional requirements

C.2.1 Mouse-logger

C.2.1.1 Movement tracking

1. The software shall accurately record the mouse movements (X,Y coordinates) of the user's cursor throughout each initiated session.
2. The software shall record the coordinate position of the mouse with precise time referencing and event indicators (e.g., mouse movement, mouse click).
3. The software shall optimise resource usage by not recording data when the mouse is stationary for an extended period.

C.2.1.2 Click recording

1. The software shall record all mouse click actions (pressed and released) during each session.
2. The software shall capture the coordinate position of the mouse at the time of clicks with timestamps and event indicators (e.g., button clicked).
3. The software shall differentiate and record various types of mouse buttons.

C.2.1.3 Time recording

1. The software shall affix a precise time to each mouse event for chronological data storage and subsequent feature computation.
2. The software shall record regular mouse movement positions that balances detail with data volume.

C.2.1.4 Data storage

1. The software shall store data in a readily accessible location on the user's local machine.
2. The software shall store data in universally compatible formats, facilitating ease of data transfer and subsequent processing.
3. The software shall segment data storage sessions for manageability.

4. The software shall automatically initiate a new session upon reaching the file threshold within a user-initiated session, ensuring continuous data capture.
5. The software shall focus on recording only essential mouse dynamics, with additional features processed during subsequent data analysis.

C.2.1.5 Exiting programme

1. The software shall have simple exit function to halt sessions at the user's discretion.

C.2.2 CONMAN

C.2.2.1 Model Architecture

1. The service shall implement a 1D Convolutional Neural Network (1DCNN) with associated layers for processing input data pertinent to mouse dynamics.
2. The model shall be compiled with optimisers, aiming to optimise for data relevant to mouse dynamic network performance.
3. The model shall be compiled with necessary regularisation assets to reduce overfitting.
4. The model shall implement the option to tweak hyperparameters to optimise outputs.
5. The model shall implement the option to apply callbacks in order to save best models and conserve computing resources when running.

C.2.2.2 Model Training and Validation

1. The service shall support model training with robust datasets, ensuring comprehensive learning of mouse dynamics.
2. The service shall incorporate validation mechanisms during training to monitor model performance and prevent overfitting.
3. The service shall support periodic retraining of the model to integrate new data and maintain accuracy.
4. The service shall support the saving of a 'best model' by model performance.
5. The service shall support the validation of the 'best model' against an unseen dataset to assess generalisability.

C.2.2.3 Data Processing

1. The service shall manage the loading of data from specified user directories.
2. The service shall preprocess loaded data to ensure format consistency and integrity for processing, including parsing timestamps, encoding categorical data, and handling missing values.
3. The service shall be equipped to handle and log exceptions gracefully during the data loading process.
4. The service shall compute additional features from raw data to enhance subsequent model learning.
5. The service shall apply scaling to relevant features for normalisation, ensuring data scale uniformity across different users.
6. The service shall support saving the normalised data in structured formats for straightforward retrieval and use in model training and testing.
7. The service shall segment the processed data into training, validation, testing, and unseen datasets, saving these into separate files for each user, ensuring proper organisation and accessibility.

C.3 Non-functional requirements

C.3.1 Mouse-logger

C.3.1.1 Software compatibility

1. The software shall offer compatibility across both Mac OS and Windows OS, ensuring a broad user base.

C.3.1.2 Hardware compatibility

1. The software shall operate seamlessly with various cursor control devices: mouse and/or trackpads.
2. The software shall capture mouse clicks from devices with non-standard button setups (e.g., additional buttons for specific functions).
3. The software shall capture positional coordinates on setups ranging from single to multiple display configurations.

C.3.1.3 Discreet operation

1. The software shall operate unobtrusively, ensuring no hindrance to the functionality of the user's mouse.
2. The software shall be optimised to minimise its impact on the user's system performance.
3. The software shall capture global mouse movements discreetly without altering or affecting the user's display or interaction experience.

C.3.1.4 Security

1. The software shall be designed with privacy in mind, ensuring no capture of personal or sensitive information from the user.
2. The software shall operate entirely offline, with data transfer facilitated via direct file sharing between the user and researcher.

C.3.1.5 Performance efficiency

1. The software shall undergo benchmark testing under various conditions to verify its robustness before deployment.
2. The software shall having minimal impact on CPU and memory resources.

C.3.1.6 User interface

1. The software shall be designed for ease of use, requiring no prior programming knowledge nor additional software for operation.
2. The software shall be straightforward to install, requiring minimal user interaction and guidance through a clear setup wizard.
3. The software shall initiate through the execution of a single file.
4. The software shall offer a simple method to terminate its operation, such as a keyboard shortcut or a user-friendly interface with a stop button.
5. The software shall provide a complete un-installation option, ensuring no residual files or settings remain on the user's system.
6. The software shall feature a simple user interface, requiring minimal interaction and providing clear options for starting and stopping the logging process.

7. The software shall provide explicit notifications or messaging to inform the user of its operational status.

C.3.2 CONMAN

C.3.2.1 Data Management

1. The service shall process the output from the 1DCNN, interpreting the results in the context of mouse dynamics analysis.
2. The service shall be developed in conjunction with Git version control.
3. The service shall offer mechanisms for result visualisation and reporting, providing insights into the data patterns recognised by the model.
4. The service shall facilitate the extraction and storage of key findings from the model's output.
5. The system shall manage the execution of different model experiments, varying parameters and configurations.
6. The system shall provide a framework for systematic testing of model variations, recording performance metrics for each experiment.
7. The service shall allow easy modification of model parameters for experimental purposes.
8. The service shall process data for individual users, ensuring that each user's data is appropriately managed and sequences are created.
9. The service shall generate fixed-length sequences from the loaded data for each user.
10. The service shall ensure that datasets are appropriately preprocessed, including type casting and categorical encoding of labels.

C.3.2.2 Usability and Accessibility

1. The service shall offer an intuitive interface for user interaction, requiring minimal technical knowledge for basic operations.
2. The service shall be provided with clear documentation on the usage of different components.

C.3.2.3 Reliability

1. The service shall employ shuffling of the training set to increase inter-sequence learning of user patterns.
2. The service shall be pre-packaged with the default settings and raw datasets used to achieve the experimental results of the project.

C.3.2.4 Compatibility and Integration

1. The service shall be compatible with standard operating systems (Windows, Mac OS), implemented through the integrated development environment, PyCharm.

C.3.2.5 Performance and Efficiency

1. The service shall be optimised for efficient processing, ensuring minimal delay in data analysis and result generation.
2. The service shall be unit tested to ensure valid data analysis and result generation.
3. The service shall provide feedback and logging capabilities during data processing to track progress and identify potential issues.

C.3.2.6 Security and Privacy

1. The service shall adhere to data privacy principles, ensuring that user data is sanitised.

C.3.2.7 Data Storage and Management

1. The service shall provide efficient data storage solutions, effectively handling large datasets.
2. The service shall be developed alongside Git version control.

C.4 Design constraints

C.4.1 Technical constraints

1. The development of these artefacts shall be undertaken by a sole developer. Given that neural networks represent a novel area of exploration, there is an anticipation of a steep learning curve, which may influence the development timeline.
2. The service shall utilise third-party libraries and Application Programming Interfaces (APIs), such as TensorFlow and Keras, to underpin the machine learning functionalities. This reliance is aimed at harnessing established computational intelligence and deep learning protocols.
3. The computational intensity required for model training and execution is expected to be substantial. The available processing capabilities, constrained by the limitations of the developer's personal computing resources and the impending project submission deadline, may impact the performance and complexity of the developed model.

C.4.2 Regulatory constraints

1. The service shall adhere to the General Data Protection Regulation (GDPR), ensuring the privacy and security of individuals' behavioural biometrics which are classified as personal data. Specific attention shall be given to the lawful processing of this data, implementing principles of data protection and confidentiality.
2. The project shall incorporate mechanisms to obtain explicit consent from users for the collection and processing of their biometric data. It shall also provide users with clear information on how their data is used, stored, and protected, along with the ability to access, rectify, and erase their data upon request.
3. The software development process shall respect intellectual property rights, avoiding third-party code or content without appropriate authorisation.

C.4.3 Budgetary constraints

1. There is no budget for this project.

Appendix D

Epics and User Stories

D.1 Epics: mouse logger

D.1.1 Create mouselogging software

Engineer a comprehensive mouse logger single file executable for efficient and accurate mouse data tracking.

D.1.1.1 Associated User Stories to the epic

- As a user, I want the mouselogger to automatically create a designated folder for logs if it doesn't already exist, so that I don't have to manually set up storage for log files.
- As a researcher, I want the mouselogger to create a new log file for each session, with a unique incrementing session number, so that data from different sessions do not overwrite each other.
- As a researcher, I want the mouselogger to record the timestamp, type of button clicked, and action (pressed/released) along with the cursor's x and y coordinates whenever a mouse click occurs, so that I have detailed interaction data for analysis.
- As a researcher, I want the mouselogger to record mouse events with a precise timestamp structure, so that data can be detailed and organised chronologically.
- As a researcher, I want the mouselogger to record the cursor's x and y coordinates along with the timestamp every 40 milliseconds during movement, to capture a continuous stream of data without unnecessary redundancy.

- As a researcher, I want the mouselogger to handle unexpected errors gracefully, providing an error message and exiting without crashing, to maintain system stability.
- As a researcher, I want all mouse events to be logged in a consistent CSV format with specific fieldnames, so that the data is structured and easy to analyse later.
- As a user, I want the ability to exit the mouselogger gracefully using a keyboard interrupt, so that it stops recording without data loss or corruption.
- As a user, I want the mouselogger to monitor the size of the current log file and switch to a new file if it exceeds 2MB, ensuring that no single log file becomes too large.
- As a user, I want to have working benchmark tests to ensure that the mouselogger can perform in different resource environments once deployed.
- As a user, I want to ensure that the mouselogger can locate and utilise resources relative to the executable path, so that it can run reliably regardless of the environment.
- As a user, I want the mouselogger script to be packaged into a single executable file, so that I can deploy it easily without worrying about dependencies or environment setup.
- As a user, I want the mouselogger executable to be compatible with different operating systems, so that I can use the same software across various platforms without significant modification.
- As a user, I want the executable to correctly access and utilise external resources or files when run, so that it maintains all functionality present in the script version.
- As a user, I want the executable to run efficiently and have a small footprint, so that it does not significantly impact system performance during operation.
- As a user with limited technical skills, I want to be able to run the mouselogger executable with a simple double-click, without using command-line instructions.
- As a user, I want the executable to handle errors gracefully and provide clear messages if issues occur, so that I can use it reliably without requiring technical support.
- As a user with limited technical skills, I want to be able to stop the mouselogger running simply and correctly through a basic command line instruction.

- As a user I want the data to be stored in an accessible place and in a format that is easy for me to export in order to streamline the data transfer process.

D.1.2 Deploy mouse logger and collect data

Execute deployment of the mouse logger software across user systems and establish a protocol for data aggregation.

D.1.2.1 Associated User Stories to the epic

- As a user, I want the collated data to be easily accessible for analysis, while still adhering to privacy standards and consent agreements.
- As a participant, I want the ability to withdraw from the mouselogger study at any point, without any negative repercussions and with the assurance that all my collected data will be removed promptly, ensuring that I retain control over my participation and data.
- As a user I want the data transfer to be simple and direct to ensure data security.
- As a participant, I want to receive clear information about what data will be collected and how it will be used, so that I can provide informed consent for my involvement in the study.
- As a participant, I want assurance that my data will be kept secure and my privacy will be respected throughout the study, so that I can trust the research team with my information.
- As a participant with limited technical experience, I want the option to receive personalised assistance during the installation process, so that I can ensure the software is set up correctly on my machine.
- As a participant, I want the briefing document to clearly explain what data will be collected, how it will be stored, and who will have access to it, so that I am assured of my privacy and the security of my data.

D.2 Epics: The CONMAN Service

D.2.1 Create 1DCNN model within the Service

Architect a 1D-CNN model for the interpretation of time-series mouse movement data.

D.2.1.1 Associated User Stories

- As a user, I want to easily train a machine learning model with different datasets so that I can evaluate its performance across various user scenarios.
- As a user, I want the model to automatically save its best state during training so that I can always have access to the most effective version of the model.
- As a product owner, I want the model training process to stop automatically when no significant improvement is observed to save computational resources and time.
- As a user, I want to receive clear information about the training process, including data shapes and training/validation results, to monitor the model's progress and performance.
- As an user, I want the system to accurately predict user classes based on input sequences so that the authentication process is reliable.
- As a user, I want to evaluate the model's performance using standard classification metrics so that I can compare its effectiveness with other models.
- As a user, I want the ability to load pre-trained models for further evaluation on different datasets.
- As a user, I want to implement callbacks and model hyperparameters to increase the performance of my models.

D.2.1.2 Spikes

- Spike to determine the optimal architecture for the 1D-CNN model.
- Spike to explore the purpose and use-case of hyperparameters.
- Spike to learn about callbacks their configurations and purpose in training.
- Spike to investigate data preprocessing requirements for optimal model input.
- Spike to learn about loss functions, underfitting and overfitting.
- Spike to determine the best strategy for handling imbalanced datasets.

D.2.2 Create a data processing module

Develop a robust data processing module to handle preprocessing, normalisation, and feature extraction of mouse dynamics data.

D.2.2.1 Associated User Stories

- As a user, I want to easily load and process raw data from different users to ensure the data is ready for model training and evaluation.
- As a analyst, I want the data to be split into training, validation, testing, and unseen datasets to facilitate comprehensive model evaluation.
- As a user, I want to aggregate data from multiple CSV files into a single DataFrame to streamline the data analysis process.
- As a project manager, I need the data loading process to be robust and handle potential errors gracefully, ensuring data integrity.
- As a user, I want confirmation of the total number of data points loaded to verify successful data aggregation.
- As a machine learning engineer, I need the raw data to be preprocessed, including timestamp conversion and categorical encoding, for effective model training.
- As an analyst, I want additional features to be computed from the raw data to enrich the dataset and improve model performance.

D.2.2.2 Spikes

- Spike to investigate data splitting in machine learning.
- Spike to explore data formatting in 1D-CNNs
- Spike to enhance the robustness of the data preprocessing workflow/pipeline - how am I going to manage it?
- Impact of different data splitting strategies on model training and evaluation.
- Error handling, how best to debug errors.

D.2.3 Create a module which directs the flow of an experiment run

Construct a module to manage and direct the workflow of experimental runs, ensuring process integrity and reproducibility.

D.2.3.1 Associated User Stories

- As a user I want to implement adequate and correct scaling and correct application to reduce the risk of data leakage.
- As a user, I want to ensure data normalisation and sequence creation for effective model input preparation.
- As a user, I want the training data sequences and their corresponding labels to be shuffled to ensure randomness and prevent model bias.
- As a machine learning engineer, I want to scale the feature values to a uniform range to improve model training and performance.
- As a data analyst, I need the normalised data to be returned in a format suitable for direct use in machine learning models.
- As a research scientist, I want to compile and format datasets for classification tasks with label data in a categorical format.
- As a project manager, I require the datasets to be checked for completeness and integrity before model training.
- As a machine learning engineer, I need to generate fixed-length sequences from time-series data for consistent model input
- As a system administrator, I want the flexibility to generate the maximum possible number of sequences from the data or a specified number.

D.2.3.2 Spikes

- Spike to optimise the performance of data normalisation and sequence creation. What order do these need to happen in? What are the consequences of these if not done properly? How do I know if its wrong?
- Investigate feature sets for normalisation
- Spike to investigate size of sequence length.
- Spike to explore advanced sequence generation for time-series data.
- Spike to evaluate the best approach for handling edge cases in sequence creation.
- Spike to investigate seeding for reliability, is it needed in this study?

D.2.4 Create a module which produces outflow results

Develop a module dedicated to generating, analysing, and reporting experimental outcomes.

D.2.4.1 Associated User Stories

- As a system integrator, I require the results controller to handle multiple runs and iterations of experiments for comparative analysis.
- As a analyst I want data of the training, testing and unseen output into manageable CSV formatting for analysis in excel.
- As a user, I want to visualise the training history and ROC-AUC-OVR curves to assess the model's learning process and discriminative ability.
- As a machine learning engineer, I want to compute and review overall and per-class performance metrics to evaluate the model's effectiveness.

D.2.4.2 Spikes

- Spike to investigate the results pipeline - what is needed, what are my intended outputs?
- Spike to enhance visualisation techniques for model evaluation.
- What is validation loss, why is it important?
- How do I interpret learning curves?
- Spike to evaluate methods for comparative analysis of different runs and iterations.
- What is roc-auc, why is this important?

Appendix E

Software Design Documentation

E.1 Introduction

This Software Design Documentation (SDD) provides a detailed description of the software deliverable design and implementation for the research project of a **mouse logger** and **Convolutional Mouse Authentication Network** (CONMAN) service. This document will explain the architectural design, component function and software interaction of the systems.

E.1.1 Scope

Two software artefacts will be produced - the first a mouse logger will be implemented on participants local machines, recording mouse dynamic data over the course of user *sessions*. The second artefact, the CONMAN service - containing a one-dimensional convolutional neural network (1DCNN), and associated controllers to process and load data input to the model as well as analysing resulting output from the model. These artefacts serve as two arms of the research study in order to answer the hypothesis of this study as seen in Chapter 3.

E.2 Mouse logger

E.2.1 Architectural Design

The mouse logger system is designed as a modular application to efficiently log mouse events, these being movements and clicks. The system is structured using Object-Oriented Programming (OOP) principles, with separate classes for file management

(FileManager), event handling (EventHandler), and the main application logic (MouseListener). These subsystems collaborate to form a cohesive service that monitors user interaction with their mouse/trackpad and records the data for research analysis. The mapping of the system's logic can be seen in figure E.1.

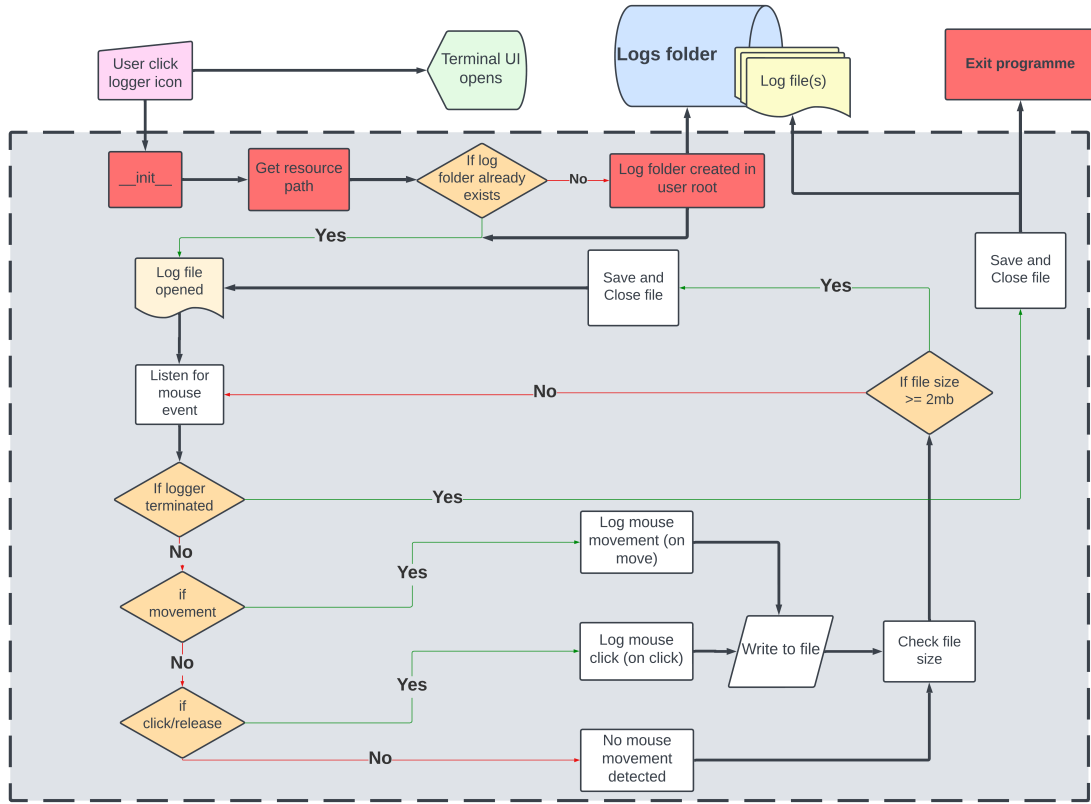


FIGURE E.1: The overall design of the logical flow of the mouse logger system.

E.2.2 Decomposition Description

E.2.2.1 FileManager

Responsibility: Responsible for handling file operations and resource path resolution.

Components:

- **create_log_folder():** This component will ensure the creation and use of a dedicated directory for storing log files. It will encapsulate the logic required to create and confirm the existence of this directory.
- **get_log_file_path():** This component will be tasked with generating unique file paths for new log sessions, preventing overwriting of existing files and ensuring that each logging session has its own file.

- **open_log_file()**: This component will open a new log file in CSV form, ready for writing.
- **close_log_file()**: This component will close a current log file.
- **get_file_size()**: Will retrieve the size of current working file in order to check for file size limit.
- **file_switch()**: Will check file size, if it has exceeded the threshold, the component will call to close the current file and open a new one to continue logging.
- **write_event()**: Will write rows of data to file, will call to check file size to see if writing can continue.

Imports:

- **os**: This module will provide a portable way of using operating system dependent functionality [79].
- **csv**: The csv module implements classes to read and write tabular data in CSV format [80].
- **sys**: This module provides access to some objects used or maintained by the interpreter and to functions that interact strongly with the interpreter [60]. By assigning other file objects (or objects that behave like files) to these, it is possible to redirect all of the interpreter's I/O.

E.2.2.2 Eventhandler

Responsibility: To manage the flow of process when a mouse event occurs. It will call methods to write events in the FileManager passing it the data parameters. It will interact with the Listener from pynput.mouse in order to capture mouse events.

Components:

- **on_click()**: Handles mouse click events.
- **on_move()**: Handles mouse movement events.
- **start_listening()**: Starts the mouse event listening process by calling the pynput.Listener.

Imports:

- `pynput.mouse.Listener`: A listener for mouse events. The package `pynput.mouse` contains classes for controlling and monitoring the mouse. A mouse listener is a `threading.Thread`, and all callbacks will be invoked from the thread [62].
- `time`: This module provides utility in manipulating time values [81].
- `datetime`: Enables the capturing of date time record [82].

E.3 The CONMAN Service

E.3.1 Architectural Design

The CONMAN system is designed as a modular application with a number of features.

- Initialise the 1D-CNN model and save said model upon completion of fitting.
- Process raw mouse dynamic data in preparation to be received by the model.
- Structure data in readable sequences of data.
- Provide a read-out of model performance, be this in response to training, validation and testing data as well as assessment against unseen data.

The system is structured using Object-Oriented Programming (OOP) principles, with Controller classes for data management and the output of results. These Controller classes will call associated utility classes to provide the functionality of each Controller. The other arm of this system will be a Model Class which will be responsible for the creation, saving and loading of a model. These subsystems collaborate to form a cohesive service that will provide model performance when assessed against user mouse dynamics. this will all be orchestrated through an Experiment Runner class and initiated from the Main class. The high-level architecture of this service can be seen in figure E.2.

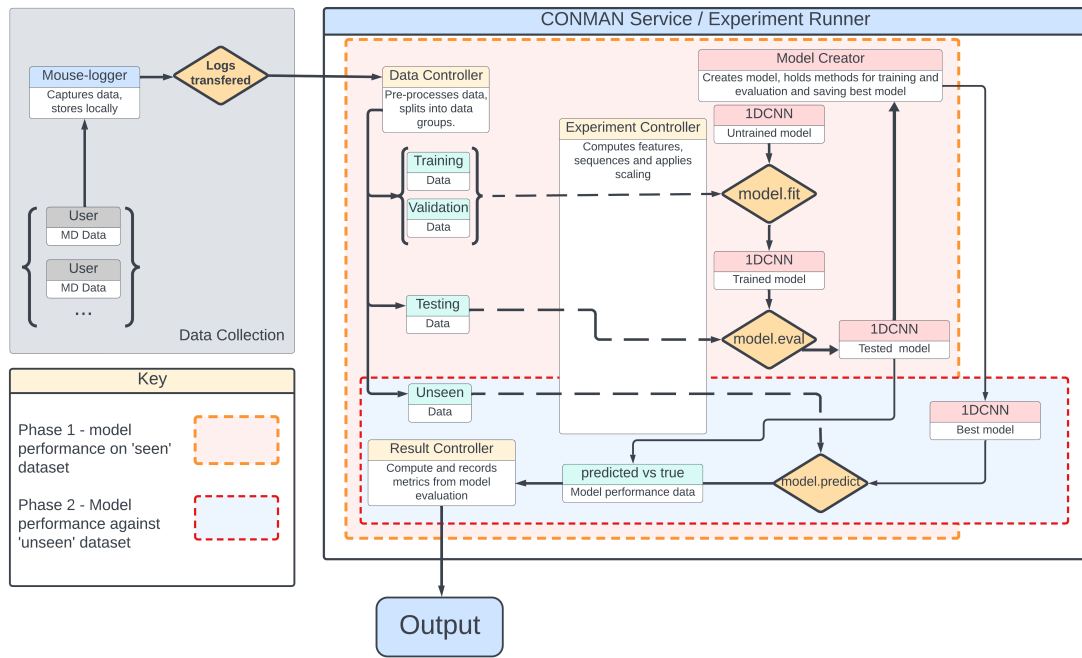


FIGURE E.2: The proposed high-level system design of the service.

E.3.2 Decomposition Description

E.3.2.1 Model Creator

Responsibility: The Model Creator is responsible for building, training, evaluating, and predicting using a deep learning model specifically tailored for sequence data processing. This class will utilise the Keras API to define the structure of a 1D Convolutional Neural Network (1D-CNN) model, inspired by the structure outlined by research [27], [83], [71] with parameter adjustments for optimal performance.

Components:

- **create_model():** Constructs the CNN model using 1D convolutional layers, ideal for sequenced data. Includes dense layers for final decision-making and softmax activation for multi-class classification.
- **train():** Will train the model on provided training sequences and labels, using specified batch size and number of epochs. Incorporates validation data for performance evaluation during training. Encapsulates model.fit method from TensorFlow.
- **predict(sequences, labels):** Will make predictions on provided sequences using the trained model and returns prediction probabilities, predicted classes, and true classes for further analysis. Encapsulates model.evaluate method from TensorFlow.

Imports:

- **import numpy as np:** For handling large, multi-dimensional arrays and matrices in Python [49].
- **import tensorflow as tf:** For building and training neural network models [66].
- **from keras.layers import Conv1D, Dense:** For 1D convolutional layers and fully connected layers in the model [67].
- **from keras.models import Sequential:** For constructing a linear stack of layers in the CNN model [57].
- **from keras.src.layers import Activation:** For applying activation functions to the model layers [84].
- **from keras.src.regularizers import l1_l2:** For applying L1 and L2 regularisation to reduce overfitting [69].
- **from tensorflow.python.keras.callbacks import EarlyStopping, ModelCheckpoint:** For implementing callbacks like early stopping and model checkpointing during training [85].

Additional Notes:

- The class will explore additional methods for training the model with early stopping and checkpointing features.
- The CNN model created by this class is particularly optimised for sequence data processing, making it well-suited for applications like user authentication using mouse dynamics.

E.3.2.2 Associated Controller Classes

Responsibility: a group of Controller classes extending a Controller abstract class and will be responsible for the data manipulation prior during and after model evaluation. Their responsibilities will cover the pre-processing of data, the management of data during model fitting and evaluation and then subsequent statistical analysis of the results. The responsibilities of each class are summarised in table E.1 with their relation to one another shown in figure E.3.

TABLE E.1: Associated Controllers

Controller class	Responsibility
Data Controller	Responsible for orchestrating and manipulating data input in order to be utilised and processed by the model. This included; preprocessing, normalising, computing advanced features from raw data and splitting processed data into a testing/training/validation collection and an unseen dataset for subsequent analysis.
Experiment Controller	Responsible for creating sequences and loading the training experiment prior to the model running. This included; creating sequences, shuffling data samples and compiling final datasets.
Results Controller	Responsible for manipulating data output from the model, in the event of testing and validation against the unseen dataset. This includes; computing metrics of the model, writing outputs to file/console and computing further analysis such as AoC OvR, TPR, TNR, FAR, FRR as well as standard metrics ie: accuracy, precision and F1 score. The controllers will share a load() method extended from the parent class and will call utility classes for the more intricate processes outlined above.

- **import pandas as pd:** For data manipulation and analysis. Pandas provide data structures like DataFrame, which are used to work with structured data [50].
- **import logging:** For logging errors that might occur during the data loading process [86].

E.3.2.4 data_utils.Data Processor

Responsibility: The DataProcessor class is tasked with processing and preparing mouse dynamics data for machine learning reception. It encompasses the preprocessing of raw data, computation of additional features, and splitting the data into training, validation, testing, and unseen sets. The Data Processor and its associated classes and their relation to the Data Controller can be seen in figure E.4.

Components:

- **preprocess_data(data):** Will preprocesses the raw data, including converting timestamps and encoding categorical columns, preparing the data for feature computation.
- **compute_features(data):** Computes additional features from the preprocessed data, such as distance, speed, acceleration, angle, and jitter.
- **data_splitter(data):** Will split the preprocessed data into distinct sets for training, validation, testing, and unseen datasets.
- **process_dataset():** Orchestrates the complete processing pipeline, including data preprocessing, splitting, and feature computation, returning the processed datasets.

Imports:

- **import pandas as pd:** For handling and manipulating structured data in DataFrame format.
- **import numpy as np:** For numerical operations on arrays and matrices, aiding in feature computation.

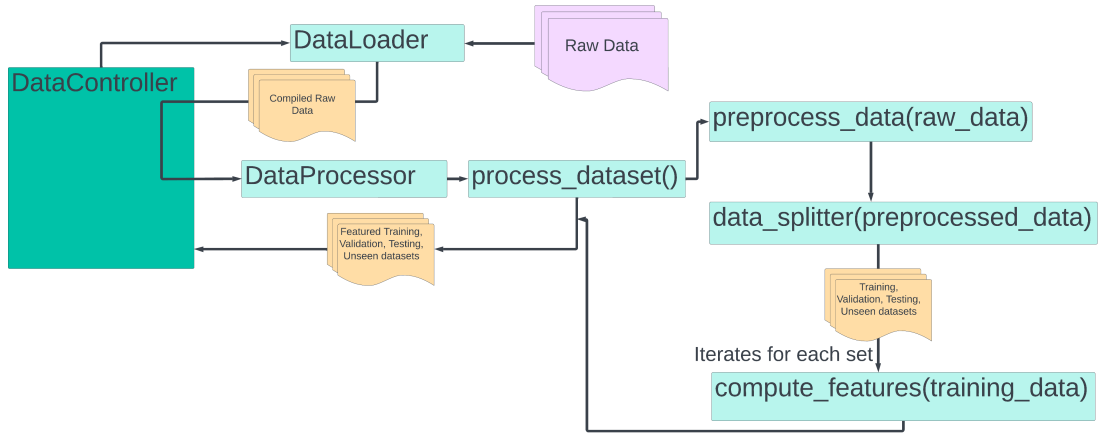


FIGURE E.4: The relation of the Data Processor and its associated classes

E.3.2.5 data_utils.DataNormaliser

Responsibility: The DataNormaliser class will be tasked with normalising and scaling mouse dynamics data using the MinMaxScaler from scikit-learn. It ensures that various datasets are normalised uniformly.

Components:

- **fit_scaler(data):** Will fit the MinMaxScaler to specific columns of the provided DataFrame, preparing it for data normalisation. The columns fitted include features like Position_X, Position_Y, Distance, Speed, Acceleration, Angle, and Jitter.
- **normalise_data(data):** Normalises the provided data using the previously fitted scaler. This method transforms the data into a scale where all features contribute equally to the final model.
- **normalise_and_save_data(training_data, validation_data, testing_data, unseen_data):** Applies normalisation to training, validation, testing, and unseen datasets. Each dataset is normalised and returned as part of a dictionary for easy access and use in model training and evaluation.

Imports:

- **from sklearn.preprocessing import MinMaxScaler:** Imports the MinMaxScaler, a tool for scaling and normalising input features. MinMaxScaler scales each feature to a given range (0 to 1), ensuring that no single feature dominates the model due to its scale [47].

Additional Notes:

- The normalisation process is crucial and vulnerable to data leakage which will have a negative impact on the model - exercise caution.

E.3.2.6 SequenceCreator

Responsibility: The SequenceCreator class is responsible for creating sequences of a specified length from time-series data. It generates sequences and corresponding target labels from the provided dataset, making them suitable for training machine learning models.

Components:

- **create_sequences(data, user_label, num_sequences):** This method creates non-overlapping sequences from time-series data, each of a specified length, and associates them with corresponding target labels.

Imports:

- **import numpy as np:** For numerical operations and handling arrays. NumPy is used here to store and manage the sequences and labels in an efficient, array-based format [49].

E.3.2.7 data_utils.Dataset Compiler

Responsibility: The DatasetCompiler class is tasked with compiling and formatting datasets. It specialises in converting sequence data to a suitable numerical format and transforming label data into a categorical format - this formatting is necessary for reception by the model's tensor operators.

Components:

- **compile_datasets(train_sequences, train_labels, valid_sequences, valid_labels, test_sequences, test_labels, unseen_sequences, unseen_labels):** This method compiles and formats datasets for training, validation, testing, and unseen data. It ensures that the sequence data is in a numerical format (float32) and that label data is transformed into a categorical format.

Imports:

- **import logging:** For logging errors that may occur during dataset compilation, such as encountering empty datasets [86].
- **import numpy as np:** For handling numerical operations and array manipulations. NumPy is used to convert sequence data into a float32 format [49].
- **from tensorflow.keras.utils import to_categorical:** For converting label data into a categorical format, particularly important for multi-class classification tasks [87].

E.3.2.8 results_utils.Metric Calculator

Responsibility: The Metric Calculator class is responsible for calculating various metrics to evaluate the performance of models. It computes both overall and per-class performance metrics, offering insights into the model's effectiveness and accuracy in predictions.

Components:

- **compute_overall_metrics(y_true, y_pred, y_pred_probs):** Computes overall performance metrics for classification results, including accuracy, precision, recall, and F1 score. It also calculates ROC AUC for multi-class scenarios using one-hot encoded true labels (for one vs rest).
- **compute_individual_metrics(y_true, y_pred, y_pred_probs):** Calculates performance metrics for each class, including precision, F1 score, ROC AUC, true positive rate (TPR), true negative rate (TNR), false alarm rate (FAR), and false rejection rate (FRR). Also computes confusion matrices for each class.
- Additional methods for calculating specific rates from confusion matrices:
`true_positive_rate`, `true_negative_rate`, `false_alarm_rate`, `false_rejection_rate`.

Imports:

- **from sklearn.metrics:** Imports various functions for computing accuracy, precision, recall, F1 score, ROC AUC score, and confusion matrix. These metrics are essential for evaluating model performance [48].
- **from tensorflow.keras.utils import to_categorical:** For converting label arrays to a one-hot encoded format [87].
- **import numpy as np:** For handling numerical operations and array manipulations in Python [49].

E.3.2.9 results_utils.OutputHandler

Responsibility: The OutputHandler class is tasked with managing the output of machine learning model performance metrics. This class facilitates the display of both overall and per-class metrics to the console and their storage in CSV files, thereby providing immediate review and durable record-keeping of results.

Methods:

- **output_results_to_console(dataset_label, overall_metrics, metrics_per_class, confusion_matrices) :**
This method will output the model performance metrics to the console, including overall metrics for the entire dataset.

Imports:

- **import csv:** Allows the writing of output data to CSV files, allowing the structured storage of model performance metrics [80].
- **import os:** Facilitates directory and file management for storing output files [79].

E.3.2.10 ResultsUtils.Plotter

Responsibility: The Plotter class is responsible for data visualisation. It provides functionalities for plotting training history and Receiver Operating Characteristic (ROC) Area Under Curve (AUC) for multi-class classification scenarios.

Methods:

- **plot_history(history, run, iteration):** Plots the training and validation accuracy and loss over the course of epochs. It saves the plot to a file in a structured directory for analysis and review.
- **plot_roc_auc_ovr(y_true, y_pred_probs, run, iteration):** Plots the ROC AUC curve for multi-class classification. The method calculates and displays the AUC for each class and saves the plot to a file for further analysis.

Imports:

- **import matplotlib.pyplot as plt:** For creating and customising plots and charts. Matplotlib is a widely used library for visualisations in Python [64].
- **import os:** For handling file and directory paths, particularly for saving plots [59].

- **from itertools import cycle:** For cycling through a list of colors when plotting multiple lines or curves.
- **from sklearn.metrics import roc_curve, auc:** For calculating the ROC curve and AUC, which are important metrics in evaluating classification models, especially in multi-class scenarios [48].
- **from sklearn.preprocessing import label_binarize:** For converting label arrays to a one-hot encoded format, essential for ROC AUC calculation in multi-class classification.

E.4 Human Interface Design

E.4.1 Overview

This section of the Software Design Document provides an overview of the Human Interface Design for two key components of the software system: the Mouse Logger and the CONMAN service. The Mouse Logger is designed with a user-friendly interface accessible to general users. In contrast, the CONMAN service can be executed in an IDE or through a technical command-line interface aimed at a coding-literate audience, including the developer and project reviewers.

E.4.2 Mouse Logger Interface User Interaction

E.4.2.1 Icon Click to Start

The Mouse Logger will be initiated by clicking an identifiable icon, either on the desktop or within the system files (*see figure E.5*). This action is straightforward and requires minimal computer proficiency, catering to users with diverse experience levels.

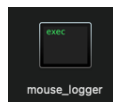


FIGURE E.5: Activation icon for the mouse logger

E.4.2.2 Terminal Feedback

Upon activation, a terminal window will open, providing immediate and clear visual feedback that the mouse logger is operational. The window displays essential information, focusing on clarity and avoiding overwhelming the user with excessive details.

E.4.2.3 Termination Command

Users can terminate the logging process at any time using a simple keyboard shortcut (Control + C). This widely recognised command ensures familiarity and ease of use for most users.

E.4.3 Design Principles for Mouse Logger

- **Simplicity and Accessibility:** The interface will be minimalistic, emphasising ease of use and accessibility. It avoids complicated navigation, settings, or configurations to cater to a broad user base.
- **Feedback and Reassurance:** The terminal window will act as a feedback mechanism, reassuring users of the software's functionality and operational status. The terminal window will also indicate where on the system log files are being saved for further peace-of-mind.

E.4.4 The CONMAN Service User Interaction

E.4.4.1 Initiating the Service

The CONMAN service will be developed in the IDE PyCharm, offering a simple initiation process through the 'Run' icon. Additionally, the service can be initiated via the Command Line Interface (CLI), providing flexibility for different user preferences.

E.4.4.2 Error, Logging, and Output

To ensure clarity on the service's status, various steps, from data pre-processing to model training output and error messaging, the console output will succinctly detail the position of the programme in the IDE's 'run' window. Specific error capturing and messaging are implemented to facilitate straightforward troubleshooting.

E.4.4.3 Parameter Adjustment and Expansion

The service's design allows for a nested project structure with clear file distinctions. Key parameters, such as model configurations and sequence lengths, are configurable from a centralised file (`Config.py`). User data is organised in the `data.directories` folder, with separate subdirectories for each anonymised user

(e.g., `data_directories/raw_data/User_A_Directory`). This structure supports seamless integration of new users and automatic adaptation of the service to changes in user data.

E.4.5 Design Principles for The CONMAN Service

- **Efficiency and Control:** The CLI offers a direct and efficient interaction mode, granting advanced users greater control and flexibility.
- **Simplicity in Complexity:** While underlying processes are complex, user interaction remains straightforward, with limited input options to reduce potential user errors.
- **Clear Output Management:** Systematic output of data into CSV files ensures organised data management and facilitates accessibility for analysis.

Conclusion: The Human Interface Design for both the Mouse Logger and CONMAN has been carefully designed to meet the needs of their respective user bases. The mouse logger focuses on accessibility and ease of use, while CONMAN emphasises efficiency and direct control for technically adept users. Both interfaces are grounded in principles of simplicity and clarity, ensuring an optimal user experience.

Appendix F

Mouse logger: Software Documentation

F.1 Introduction

This documentation contains the information necessary for understanding the architecture and execution of the mouse logger software utilised in the project attached. This documentation will go through the purpose, user guide and system detail of the Mouselogger programme.

F.1.1 Purpose

The purpose of the Mouselogger programme is to gather mouse dynamic data from users during their operation of the machine it is installed upon. The Mouselogger collects *organic* data from the moment of activation and collects continuously unless interrupted by a system halt be that user determined or a failure of the system. The information is collated locally in the user's storage for it to be transferred to the researcher in order to conduct experiments on model training within the project's CONMAN service.

F.2 User Guide

F.2.1 Packaging the service

The mouse logger is an executable project developed and deployable within the Integrated Development Environment (IDE) PyCharm, and is compatible with similar

software environments. Developers may execute the Mouse-Logger from this IDE for testing purposes, ensuring proper execution of the code. For deployment to participant users, in order to enhance accessibility, the logger is packaged using PyInstaller [65]. To ensure the correct installation of PyInstaller, the following command line instructions can be entered from the project's root directory

```
pip install -U pyinstaller
```

After which the following command packages the logger into a one file executable.

```
pyinstaller --onefile mouse_logger.py
```

The packaged software is then available for distribution in the dist directory of the project.

It is worth noting that line 18 in mouse_logger.py is accepted by MacOS but will require minor editing for Windows OS, from the file path "Documents" to "My Documents" in order to have an equivalent outcome.

```
1 self.log_folder = os.path.join(os.path.expanduser("~"), "Documents", "  
    Mouse_logger_logs")
```

F.2.2 Distribution and Deployment

Distribution was managed via Google Drive, issue was found with email distribution as many providers do not allow the sending or receiving of files of this kind. Deploying code to participant users was found to have some challenges. For example, Windows and Mac OS have default protections in place for files of this type due to their association with malware. In addition, for similar reasons, anti-virus software may prohibit users from downloading and running single file executables, especially those that write and save files to the system. On the OS level, this may require the user to grant permissions exclusively for the file or to execute commands as an administrator. Similar processes will be required to grant permission through the settings of anti-viral applications, although none were encountered in this project.

F.2.3 Running the mouse-logger

Once distributed among participants, an icon shortcut will be created for user access (see figure F.1).

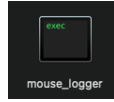


FIGURE F.1: Activation icon for the mouse logger

Clicking or running the programme in the command-line will run with a terminal window opening (if not already) with a message indicating that logging has begun, along with the filepath for saved logs. Closing of the terminal window will end the programme, however, to end the programme gracefully ensuring all files are saved appropriately the user can end the application by pressing **control + c** in the terminal window. The window will print **Gracefully exiting...**, with files being saved to the newly created file path in the user's storage - by default this is in the **Documents** directory.

F.3 System Detail

F.3.1 MouseLogger.py

The `MouseLogger` class serves as the central mechanism for logging mouse events in a system. It handles the creation of log files and capturing mouse interactions such as clicks and movements.

F.3.1.1 Workflow

1. **Initialisation:** Instantiate the `MouseLogger` class, specifying the folder for storing log files. The `FileManager` and `EventHandler` classes are also initialised.
2. **Log Folder Creation:** The `FileManager` creates the specified folder for storing the mouse event logs.
3. **Opening Log File:** `FileManager` opens a new log file to record the mouse events.
4. **Event Handling:** `EventHandler` starts listening for mouse events (clicks and movements) and records them in the log file through `FileManager`.
5. **Graceful Exit:** The logging process continues until interrupted (e.g., via a `KeyboardInterrupt`), ensuring a graceful termination of the logging process.
6. **Closing Log File:** On termination, the log file is properly closed to ensure all recorded data is saved.

F.3.1.2 Components Used

- **FileManager**: Responsible for managing the log files including creation, writing events, and handling file size limits.
- **EventHandler**: Captures mouse events and sends the data to FileManager for logging.
- **os.path** and **os**: Used for handling file and directory paths.

F.3.1.3 Entry Point

The script is intended to be executed as a standalone program. The **main()** function creates an instance of MouseLogger and starts the event logging process.

F.3.2 FileManager.py

The **FileManager** class is responsible for managing the log files used to store mouse event data. It handles file creation, writing events to the file, and managing file size.

F.3.2.1 Attributes and Constants

- **log_folder** (str): Directory where log files will be stored.
- **log_filename** (str): Name of the current log file.
- **file** (file object): File object for the current log file.
- **writer** (csv.DictWriter): Writer object for writing data to the log file.
- **MAX_FILE_SIZE** (int): Maximum size of a log file in bytes, set to 2MB.

F.3.2.2 Methods

- **__init__(log_folder)**: Initialises the FileManager with the specified log folder. Sets the log filename and initialises file and writer attributes to None.
- **get_resource_path(relative_path)**: Returns the absolute path to a resource, necessary when running with PyInstaller.
- **create_log_folder()**: Creates the log folder if it does not exist and handles any exceptions.

- **get_log_file_path(session_number=1)**: Generates a new log file path, incrementing the session number to find an unused filename.
- **open_log_file()**: Opens a new log file for writing and sets up a `csv.DictWriter` for event data.
- **close_log_file()**: Closes the current log file if open.
- **file_switch()**: Switches to a new log file if the current file size exceeds the maximum limit.
- **get_file_size(filename)**: Returns the size of a given file in bytes.
- **write_event(event_data)**: Writes an event data dictionary to the log file and switches files if necessary.

F.3.2.3 Usage

`FileManager` is used within the mouse logger application to manage the creation and handling of log files where mouse events are recorded. It ensures that event data is systematically logged and stored in appropriately sized files.

F.3.3 `EventHandler.py`

The `EventHandler` class is responsible for handling mouse events, such as clicks and movements, and recording these events using the provided `FileManager`.

F.3.3.1 Attributes

- **file_manager**: An instance of `FileManager` used to write event data to a log file.
- **last_move_time** (float): Timestamp of the last mouse movement event handled.
- **last_click_time** (float): Timestamp of the last mouse click event handled.

F.3.3.2 Methods

- **__init__(file_manager)**: Initialises the `EventHandler` with a file manager to handle event logging.
- **on_click(x, y, button, pressed)**: A callback method for handling mouse click events. It records the event details, including the position of the click, the button used, and whether it was pressed or released.

- **on_move(x, y)**: A callback method for handling mouse movement events. It records the position of the mouse at regular intervals.
- **start_listening()**: Starts listening for mouse events and processes them using the on_move and on_click callback methods. This method initiates the event handling loop.

F.3.3.3 Usage

EventHandler is used within the mouse logger application to capture real-time mouse events. It works in conjunction with **FileManager** to ensure that all mouse interactions are systematically recorded and stored in log files. The class provides functionality to handle different types of mouse events and manage the frequency of recording for movements.

F.4 Testing and Benchmark

Unit testing and benchmarks for performance were undertaken on the software. The results of which can be seen below. These unit tests and benchmarks can be run by following the relevant sections in the **README.md** within the root directory.

Intensity	Avg. CPU	Peak CPU	Avg. RAM	Peak RAM
Low	0.63%	3.90%	0.21%	0.23%
Medium	0.89%	3.60%	0.23%	0.23%
High	1.26%	3.70%	0.19%	0.23%

TABLE F.1: Average and Peak CPU and RAM usage expressed as a percentage of the total availability at different intensities

Benchmarking was developed using the PyAutoGUI and psutil libraries [88, 89] and was split into three intensities, with a delay between mouse movements decreasing as intensity increased. Each movement was also ended with a click event. This gave a benchmark testing of three intensity scenarios where events were recorded at rates of 1 per second (low), 5 per second (medium) and 20 per second (high) and ran for 10 minutes. These benchmark tests were run on MacOS, where a separate empty desktop can be created easily. This ensured that the benchmark testing did not inadvertently open any other applications. There is a 'safe zone' where the mouse is limited to in order to reduce this risk, however, if an emergency stop of the benchmark test is needed

the user can manually override by moving the mouse cursor to the edge of their screen, halting the programme.

F.5 Development Guide

The requirements to run the service are listed in the software's '*requirements.txt*' and can be seen below. Should a developer wish to expand on this software these requirements are to be noted. This project was developed in the PyCharm IDE and Git was used for version control, no branch name conventions are specified, but going forward would be advisable. The PEP 8 [\[77\]](#) coding style standard for Python was followed for the duration of development, with Object-Oriented Programming and SOLID principles used as a guide.

- altgraph ~ 0.17.3
- cx-Freeze ~ 6.15.7
- macholib ~ 1.16.2
- MouseInfo ~ 0.1.3
- Pillow ~ 10.1.0
- psutil ~ 5.9.6
- PyAutoGUI ~ 0.9.54
- PyGetWindow ~ 0.0.9
- pyinstaller ~ 5.13.2
- pyinstaller-hooks-contrib ~ 2023.8
- PyMsgBox ~ 1.0.9
- pynput ~ 1.7.6
- pyobjc-core ~ 9.2
- pyobjc-framework-ApplicationServices ~ 9.2
- pyobjc-framework-Cocoa ~ 9.2
- pyobjc-framework-Quartz ~ 9.2
- pyperclip ~ 1.8.2
- PyRect ~ 0.2.0
- PyScreeze ~ 0.1.30
- pytweneing ~ 1.0.7
- rubicon-objc ~ 0.4.7
- six ~ 1.16.0

There is a testing package titled 'testing' in the root directory which covered unit tests for functions as well as a benchmark test for CPU and memory performance at different usage intensities. *Low*, *medium* and *high* intensity scenarios were emulated to gauge performance expenditure, detailed further in the 'Testing and Benchmark' section of

this document. Testing and bench-marking should be elaborated along with any future expansion of the software.

Docstring conventions as outlined PEP 257 [78] were adhered to in this project in order to adequately document Python classes, methods, variables and modules.

F.6 Source Code with Documentation

F.6.0.1 MouseLogger.py

```
1 from FileManager import FileManager
2 from EventHandler import EventHandler
3 import os
4
5
6 class MouseLogger:
7     """
8     A class for logging mouse events.
9
10    This class is responsible for initiating the mouse event logging
11    process, including creating
12    and managing log files and handling mouse events.
13
14    Attributes:
15        log_folder (str): Path to the folder where mouse event logs are
16        stored.
17        file_manager (FileManager): Manages the creation, opening, and
18        closing of log files.
19        event_handler (EventHandler): Captures and processes mouse events.
20    """
21
22    def __init__(self):
23        self.log_folder = os.path.join(os.path.expanduser("~"), "Documents"
24        , "Mouse.logger.logs")
25        self.file_manager = FileManager(self.log_folder)
26        self.event_handler = EventHandler(self.file_manager)
27
28    def run(self):
29        """
30        Starts the mouse logging process.
31
32        Manages the creation of log files, initiates listening for mouse
33        events,
34        and gracefully handles interruptions and errors.
35        """
```

```
31         try:
32             self.file_manager.create_log_folder()
33             self.file_manager.open_log_file()
34             self.event_handler.start_listening()
35         except KeyboardInterrupt:
36             print("Exiting gracefully...")
37         except Exception as e:
38             print(f"Error: {e}")
39         finally:
40             self.file_manager.close_log_file()
41
42
43 if __name__ == "__main__":
44     mouse_logger = MouseLogger()
45     mouse_logger.run()
```

CODE INSERT F.1: Mouse logger Source Code: MouseLogger.py

F.6.0.2 FileManager.py

```

1 import os
2 import csv
3 import sys
4
5
6 class FileManager:
7     """
8     A class to manage log files for storing mouse events.
9
10    Attributes:
11        log_folder (str): The directory where log files will be stored.
12        log_filename (str): The current log file's name.
13        file (file object): The file object for the current log file.
14        writer (csv.DictWriter): A writer object for writing data to the
15        log file.
16
17    Constants:
18        MAX_FILE_SIZE (int): Maximum size of a log file in bytes. Default
19        is 2MB.
20    """
21    MAX_FILE_SIZE = 2 * 1024 * 1024 # 2 megabytes
22
23    def __init__(self, log_folder):
24        """
25        Initialises the FileManager with the specified log folder.
26        It also initialises the log filename and sets the file and writer
27        attributes to None.
28
29        Args:
30            log_folder (str): The directory where log files will be stored.
31        """
32        self.log_folder = log_folder
33        self.log_filename = self.get_log_file_path()
34        self.file = None
35        self.writer = None
36
37    def get_resource_path(self, relative_path):
38        """
39        Get absolute path to a resource, which is needed when running the
40        application with PyInstaller.
41
42        Args:
43            relative_path (str): The relative path to the resource.
44
45        Returns:
46            str: The absolute path to the resource.

```

```

43
44     Source:
45     https://stackoverflow.com/questions/22472124/what-is-sys-meipass-in
46     """
47     base_path = getattr(sys, '_MEIPASS', os.path.dirname(os.path.
abspath(__file__)))
48     return os.path.join(base_path, relative_path)
49
50 def create_log_folder(self):
51     """
52     Creates the log folder if it does not exist. Handles any exceptions
and prints relevant messages.
53     """
54     try:
55         os.makedirs(self.log_folder, exist_ok=True)
56         print(f"Log folder created: {self.log_folder}")
57     except Exception as e:
58         print(f"Error creating log folder: {e}")
59
60 def get_log_file_path(self, session_number=1):
61     """
62     Generates a new log file path. Increments the session number until
an unused filename is found.
63
64     Args:
65         session_number (int): The starting session number for the log
file.
66
67     Returns:
68         str: The path for the new log file.
69     """
70     while os.path.exists(os.path.join(self.log_folder, f"
mouselog-session-{session_number}.csv")):
71         session_number += 1
72     return os.path.join(self.log_folder, f"mouselog-session-{
session_number}.csv")
73
74 def open_log_file(self):
75     """
76     Opens a new log file for writing. Sets up the csv.DictWriter for
writing event data.
77     """
78     self.file = open(self.log_filename, 'w', newline='')
79     fieldnames = ['Timestamp', 'Event', 'Button', 'Action', 'Position-X
', 'Position-Y']
80     self.writer = csv.DictWriter(self.file, fieldnames=fieldnames)
81     self.writer.writeheader()

```

```
82
83     def close_log_file(self):
84         """
85         Closes the current log file if it is open.
86         """
87         if self.file:
88             self.file.close()
89
90     def file_switch(self):
91         """
92         Switches to a new log file if the current file size exceeds the
93         maximum file size.
94         """
95         if self.get_file_size(self.log_filename) > FileManager.
96         MAX_FILE_SIZE: # If size > 2MB
97             self.close_log_file() # Close the current file
98             self.log_filename = self.get_log_file_path() # Get a new log
99             filename
100             self.open_log_file() # Open a new log file
101             print("New log file created")
102
103     def get_file_size(self, filename):
104         """
105         Gets the size of a file.
106
107         Args:
108             filename (str): The name of the file whose size is to be
109             determined.
110
111         Returns:
112             int: The size of the file in bytes.
113         """
114         return os.path.getsize(filename)
115
116     def write_event(self, event_data):
117         """
118         Writes an event data dictionary to the current log file.
119
120         Args:
121             event_data (dict): A dictionary containing event data to be
122             logged.
123         """
124         self.writer.writerow(event_data)
125         self.file_switch()
```

CODE INSERT F.2: Mouse logger Source Code: FileManager.py

F.6.0.3 EventHandler.py

```

1 from pynput.mouse import Listener
2 import time
3 from datetime import datetime
4
5 class EventHandler:
6     """
7     A class for handling mouse events and recording them using the provided
8     file manager.
9
10    Attributes:
11        file_manager: An instance of FileManager used to write event data
12        to a log file.
13        last_move_time (float): Timestamp of the last mouse movement event
14        handled.
15        last_click_time (float): Timestamp of the last mouse click event
16        handled.
17    """
18    def __init__(self, file_manager):
19        """
20        Initialises the EventHandler with a file manager.
21
22        Args:
23            file_manager: An instance of FileManager used to write event
24            data to a log file.
25        """
26        self.file_manager = file_manager
27        self.last_move_time = time.time()
28        self.last_click_time = time.time()
29
30    def on_click(self, x, y, button, pressed):
31        """
32        Callback for handling mouse click events.
33
34        Args:
35            x (int): The x-coordinate of the mouse click.
36            y (int): The y-coordinate of the mouse click.
37            button: The button that was clicked.
38            pressed (bool): True if the button was pressed, False if it was
39            released.
40
41        Source:
42        https://stackoverflow.com/questions/25848951/python-get-mouse-x-y-
43        position-on-click
44        """
45        action = "Pressed" if pressed else "Released"
46        timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S.%f")[:-3]

```

```

40         self.file_manager.write_event({ 'Timestamp': timestamp, 'Event': '
Mouse Click', 'Button': button,
41                                     'Action': action, 'Position_X': x, '
Position_Y': y})
42
43     def on_move(self, x, y):
44         """
45         Callback for handling mouse movement events.
46
47         Args:
48             x (int): The x-coordinate of the mouse position.
49             y (int): The y-coordinate of the mouse position.
50         """
51         current_time = time.time()
52         if current_time - self.last_move_time >= 0.04:
53             timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S.%f")
54             self.file_manager.write_event({ 'Timestamp': timestamp, 'Event':
'Mouse Movement', 'Button': '',
55                                         'Action': '', 'Position_X': x, '
Position_Y': y})
56             self.last_move_time = current_time
57
58     def start_listening(self):
59         """
60         Starts listening to mouse events and handles them using the defined
callbacks.
61         """
62         with Listener(on_move=self.on_move, on_click=self.on_click) as
listener:
63             listener.join()

```

CODE INSERT F.3: Mouse logger Source Code: EventHandler.py

F.6.0.4 Benchmark.py

```

1 import os
2 import random
3 import time
4 import threading
5 import psutil
6 import pyautogui
7 import csv
8
9 duration = 600 # Global variable (10 minutes)
10
11 def is_mouse_outside_safe_zone(safe_zone, screen_size):
12     """
13     Checks if the mouse pointer is outside the designated safe zone.
14
15     Args:
16         safe_zone (tuple): Width and height of the safe zone in pixels.
17         screen_size (tuple): Width and height of the entire screen in
18         pixels.
19
20     Returns:
21         bool: True if the mouse is outside the safe zone, False otherwise.
22     """
23     x, y = pyautogui.position()
24     safe_zone_width, safe_zone_height = safe_zone
25     screen_width, screen_height = screen_size
26     return x < safe_zone_width or x > screen_width - safe_zone_width or y <
27     safe_zone_height or y > screen_height - safe_zone_height
28
29 def simulate_mouse_events(intensity, duration, safe_zone, stop_event):
30     """
31     Simulates mouse events based on a specified intensity for a given
32     duration.
33
34     Args:
35         intensity (str): The intensity of the mouse movements ('low', '
36         medium', 'high').
37         duration (int): The duration in seconds for which to simulate mouse
38         events.
39         safe_zone (tuple): The width and height of the safe zone in pixels.
40         stop_event (threading.Event): Event to signal the end of the
41         simulation.
42
43     The function will stop early if the mouse leaves the safe zone.
44
45     Source:
46     https://pyautogui.readthedocs.io/en/latest/

```

```

41     """
42     delay = {"low": 1, "medium": 0.2, "high": 0.05}[intensity]
43     end_time = time.time() + duration
44     screen_width, screen_height = pyautogui.size()
45
46     while time.time() < end_time and not stop_event.is_set():
47         # User can manually override the test, by moving the cursor to the
48         edge fo the screen
49         if is_mouse_outside_safe_zone(safe_zone, (screen_width,
50 screen_height)):
51             print("Mouse left the safe zone. Exiting program.")
52             stop_event.set()
53             break
54             new_x = random.randint(safe_zone[0], screen_width - safe_zone[0])
55             new_y = random.randint(safe_zone[1], screen_height - safe_zone[1])
56             pyautogui.moveTo(new_x, new_y, duration=delay)
57             pyautogui.click()
58             time.sleep(delay)
59
60 def monitor_resource_usage(pid, shared_data, duration, stop_event):
61     """
62     Monitors and records the CPU and memory usage of a given process.
63
64     Args:
65         pid (int): The process ID of the program to monitor.
66         shared_data (dict): Dictionary to store CPU and memory usage data.
67         duration (int): Duration in seconds to monitor the process.
68         stop_event (threading.Event): Event to signal the end of monitoring
69         .
70
71     The function records CPU and memory usage at one-second intervals.
72
73     Sources:
74     https://psutil.readthedocs.io/en/latest/
75     """
76     process = psutil.Process(pid)
77     for _ in range(duration):
78         if stop_event.is_set():
79             break
80             cpu = process.cpu_percent(interval=1)
81             memory = process.memory_percent()
82             shared_data['cpu'].append(cpu)
83             shared_data['memory'].append(memory)
84
85 def save_results_to_csv(cpu_usage, memory_usage, filename="
benchmark_results.csv"):
86     """
87     Saves the CPU and memory usage data to a CSV file.

```

```

85
86     Args:
87         cpu_usage (list): List of CPU usage percentages.
88         memory_usage (list): List of memory usage percentages.
89         filename (str, optional): Name of the CSV file to save the data.
90
91     The function writes the average and peak CPU and memory usage to the
92     CSV file.
93     """
94     with open(filename, 'w', newline='') as csvfile:
95         fieldnames = ['Average CPU', 'Peak CPU', 'Average Memory', 'Peak
96         Memory']
97         writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
98         writer.writeheader()
99         writer.writerow({
100             'Average CPU': sum(cpu_usage) / len(cpu_usage),
101             'Peak CPU': max(cpu_usage),
102             'Average Memory': sum(memory_usage) / len(memory_usage),
103             'Peak Memory': max(memory_usage)
104         })
105
106 def run_scenario(intensity, duration):
107     """
108     Runs a scenario that simulates mouse events and monitors resource usage
109     .
110
111     Args:
112         intensity (str): The intensity of the mouse movements ('low', '
113         medium', 'high').
114         duration (int): Duration in seconds for the scenario.
115
116     The method sets up the simulation and monitoring threads.
117     """
118     shared_data = {'cpu': [], 'memory': []}
119     safe_zone = (300, 300) # Size of the safe zone in pixels
120     stop_event = threading.Event()
121
122     mouse_thread = threading.Thread(target=simulate_mouse_events, args=(
123     intensity, duration, safe_zone, stop_event))
124     monitor_thread = threading.Thread(target=monitor_resource_usage, args=(
125     os.getpid(), shared_data, duration, stop_event))
126     mouse_thread.start()
127     monitor_thread.start()
128
129     mouse_thread.join()
130     monitor_thread.join()
131
132     if not stop_event.is_set():

```

```
127         save_results_to_csv(shared_data['cpu'], shared_data['memory'])
128
129 if __name__ == "__main__":
130     run_scenario("low", duration) # Pass the global variable
```

CODE INSERT F.4: Mouse logger Source Code: Benchmark.py

F.6.0.5 test_event_handler.py

```

1 import unittest
2 from unittest.mock import MagicMock, patch
3 from EventHandler import EventHandler
4
5 class TestEventHandler(unittest.TestCase):
6
7     def setUp(self):
8         self.mock_file_manager = MagicMock()
9         self.event_handler = EventHandler(self.mock_file_manager)
10
11     @patch('EventHandler.datetime')
12     def test_on_click(self, mock_datetime):
13         """Test the on_click method."""
14         mock_datetime.now.return_value.strftime.return_value = "2021-01-01
15 12:00:00.001"
16         x, y, button, pressed = 100, 200, 'left', True
17
18         self.event_handler.on_click(x, y, button, pressed)
19
20         expected_event_data = {
21             'Timestamp': '2021-01-01 12:00:00.',
22             'Event': 'Mouse Click',
23             'Button': 'left',
24             'Action': 'Pressed',
25             'Position-X': x,
26             'Position-Y': y
27         }
28         self.mock_file_manager.write_event.assert_called_with(
29             expected_event_data)
30
31     def setUp(self):
32         self.mock_file_manager = MagicMock()
33         self.event_handler = EventHandler(self.mock_file_manager)
34
35     @patch('EventHandler.datetime')
36     @patch('EventHandler.time')
37     def test_on_move(self, mock_time, mock_datetime):
38         """Test the on_move method with a delay to trigger logging."""
39         mock_datetime.now.return_value.strftime.return_value = "2021-01-01
40 12:00:00.000"
41         mock_time.time.return_value = 100.05 # Mock current time
42
43         x, y = 300, 400
44
45         # Simulate the previous time to ensure logging occurs
46         self.event_handler.last_move_time = 100.0 # Mock last move time

```

```
44         self.event_handler.on_move(x, y)
45
46         expected_event_data = {
47             'Timestamp': '2021-01-01 12:00:00.',
48             'Event': 'Mouse Movement',
49             'Button': '',
50             'Action': '',
51             'Position_X': x,
52             'Position_Y': y
53         }
54         self.mock_file_manager.write_event.assert_called_with(
55             expected_event_data)
56
57 if __name__ == '__main__':
58     unittest.main()
```

CODE INSERT F.5: Mouse logger Source Code: test_event_handler.py

F.6.0.6 test_file_manager.py

```

1 import unittest
2 from unittest.mock import MagicMock, mock_open, patch
3 from FileManager import FileManager
4
5
6 class TestFileManager(unittest.TestCase):
7
8
9     @patch('FileManager.open', new_callable=mock_open, read_data="data")
10    @patch('FileManager.csv.DictWriter')
11    def test_open_log_file(self, mock_dict_writer, mock_file_open):
12        """Test the open_log_file method."""
13        file_manager = FileManager('log_folder')
14        file_manager.log_filename = 'log_file.csv'
15        file_manager.open_log_file()
16
17        mock_file_open.assert_called_with('log_file.csv', 'w', newline='')
18        mock_dict_writer.assert_called_once()
19
20    def test_close_log_file(self):
21        """Test the close_log_file method."""
22        file_manager = FileManager('log_folder')
23        file_manager.file = MagicMock()
24        file_manager.close_log_file()
25
26        file_manager.file.close.assert_called_once()
27
28    @patch('FileManager.os.path.getsize')
29    def test_file_switch(self, mock_getsize):
30        """Test the file_switch method."""
31        file_manager = FileManager('log_folder')
32        file_manager.log_filename = 'log_file.csv'
33        file_manager.open_log_file = MagicMock()
34        file_manager.close_log_file = MagicMock()
35
36        # Test without switching
37        mock_getsize.return_value = 1 # Size less than MAX_FILE_SIZE
38        file_manager.file_switch()
39        file_manager.close_log_file.assert_not_called()
40
41        # Test with switching
42        mock_getsize.return_value = 3 * 1024 * 1024 # Size greater than
43        MAX_FILE_SIZE
44        with patch('builtins.print') as mock_print:
45            file_manager.file_switch()
46            file_manager.close_log_file.assert_called_once()

```

```
46         file_manager.open_log_file.assert_called_once()
47         mock_print.assert_called_with("New log file created")
48
49
50 if __name__ == '__main__':
51     unittest.main()
```

CODE INSERT F.6: Mouse logger Source Code: test_file_manager.py

F.6.0.7 test_mouse_logger.py

```

1 import unittest
2 from unittest.mock import MagicMock, patch
3 from MouseLogger import MouseLogger
4
5 class TestMouseLogger(unittest.TestCase):
6
7     def setUp(self):
8         self.mock_file_manager = MagicMock()
9         self.mock_event_handler = MagicMock()
10        self.mouse_logger = MouseLogger()
11        self.mouse_logger.file_manager = self.mock_file_manager
12        self.mouse_logger.event_handler = self.mock_event_handler
13
14    def test_run_success(self):
15        """Test the run method under normal conditions."""
16        try:
17            self.mouse_logger.run()
18        except KeyboardInterrupt:
19            pass
20
21        self.mock_file_manager.create_log_folder.assert_called_once()
22        self.mock_file_manager.open_log_file.assert_called_once()
23        self.mock_event_handler.start_listening.assert_called_once()
24        self.mock_file_manager.close_log_file.assert_called_once()
25
26    @patch('builtins.print')
27    def test_run_keyboard_interrupt(self, mock_print):
28        """Test the run method when a KeyboardInterrupt is raised."""
29        self.mock_event_handler.start_listening.side_effect =
KeyboardInterrupt
30
31        self.mouse_logger.run()
32
33        mock_print.assert_called_with("Exiting gracefully ...")
34        self.mock_file_manager.close_log_file.assert_called_once()
35
36    @patch('builtins.print')
37    def test_run_exception(self, mock_print):
38        """Test the run method when an exception is raised."""
39        self.mock_event_handler.start_listening.side_effect = Exception("
Test Error")
40
41        self.mouse_logger.run()
42
43        mock_print.assert_called_with("Error: Test Error")
44        self.mock_file_manager.close_log_file.assert_called_once()

```

```
45
46 if __name__ == '__main__':
47     unittest.main()
```

CODE INSERT F.7: Mouse logger Source Code: test_mouse_logger.py

Appendix G

User Hardware

User	Display	Mouse
A	2 display monitors - both landscape in orientation	Anker Ergonomic Optical USB Wired Vertical Mouse 1000
B	2 display monitors - one landscape, one portrait in orientation	Logitech M185
C	1 display - laptop screen	Apple Magic Mouse 2

TABLE G.1: Summary of each user's hardware set-up

Appendix H

Proposed User Survery

H.1 [Proposal] User Hardware Setup Survey

1. What type of pointing device do you primarily use with your computer?

- ☐ Mouse
- ☐ Trackpad
- ☐ Both
- ☐ Other (please specify): _____

2. What is the brand and model of your pointing device?

3. What operating system do you use on the machine you will partake this study on?

- ☐ Mac OS
- ☐ Windows
- ☐ Linux
- ☐ Other: _____

4. How many active displays do you regularly use with your computer setup?

- ☐ 1 (Would include a single-screen monitor set-up or just a laptop screen)
- ☐ 2
- ☐ 3
- ☐ 4 or more

5. What are the primary purposes for which you use your computer? (Select all that apply)

- ☐ Work or Professional Tasks
- ☐ Gaming
- ☐ Browsing the Internet
- ☐ Creative tasks (ie: Graphic Design, Video Editing, Music Production)
- ☐ Programming or Software Development
- ☐ Other: _____

6. On average, how long are your continuous work sessions on the computer?

- ☐ Less than 1 hour
- ☐ 1 to 2 hours
- ☐ 2 to 4 hours
- ☐ 4 to 6 hours
- ☐ More than 6 hours

7. Do you regularly change your work setup (e.g., moving between different workstations or locations)?

- ☐ Yes
- ☐ No

8. If you answered 'Yes' to the previous question, how often do you change your work setup?

- ☐ Daily
- ☐ Several times a week
- ☐ Weekly
- ☐ Monthly
- ☐ Rarely

9. During weeks of high usage, allocate a percentage of time spent over the following tasks (the total should add up to 100%):

- Work/Professional Tasks: ____ %
- Gaming: ____ %
- Browsing the Internet: ____ %
- Creative tasks (ie: Graphic Design, Video Editing, Music Production) ____ %
- Programming/Software Development: ____ %
- Other (please specify: _____): ____ %
- Other (please specify: _____): ____ %

10. Additional comments or details about your hardware setup: (optional)

Appendix I

CONMAN Service Raw Data Output

This appendix contains the model outputs for the 25 iterations as well as the Shea model. **Please note that the raw mouselog session data will be available in the code submission.**

I.1 Overall Training/Testing set of Performance Metrics for each iteration of the CONMAN service

Iteration	Accuracy	Precision_Macro	Recall_Macro	F1_Macro	ROC_AUC_OvR
1	0.99666667	0.996667983	0.996666667	0.996666162	0.999947167
2	0.998333333	0.998333665	0.998333333	0.998333333	0.999969667
3	0.99766667	0.997667999	0.997666667	0.997667166	0.999915667
4	0.996333333	0.9963363	0.996333333	0.996333325	0.999919667
5	0.99666667	0.99666964	0.996666667	0.99666666	0.99989
6	0.995333333	0.995366203	0.995333333	0.995333217	0.999919
7	0.998333333	0.998333665	0.998333333	0.998333333	0.999890333
8	0.997333333	0.997345238	0.997333333	0.997333309	0.999928167
9	0.997333333	0.997334665	0.997333333	0.997333833	0.999961667
10	0.995333333	0.995347855	0.995333333	0.995335467	0.999961
11	0.99766667	0.997674942	0.997666667	0.997666652	0.999903
12	0.99666667	0.996672954	0.996666667	0.996667657	0.9999035
13	0.99766667	0.997682886	0.997666667	0.997666638	0.999923333
14	0.99866667	0.998667995	0.998666667	0.998666665	0.999953833
15	0.997333333	0.997333664	0.997333333	0.997333333	0.999888833
16	0.99766667	0.997665999	0.997666667	0.997666166	0.999965833
17	0.99866667	0.998667995	0.998666667	0.998666665	0.999959833
18	0.99766667	0.997667331	0.997666667	0.997666833	0.999944
19	0.997333333	0.997333333	0.997333333	0.997333333	0.999898333
20	0.99566667	0.995693318	0.995666667	0.995666579	0.999946167
21	0.998	0.998001325	0.998	0.997999998	0.999927833
22	0.99866667	0.998671979	0.998666667	0.998666661	0.999993833
23	0.995333333	0.995346514	0.995333333	0.995334799	0.999963
24	0.998	0.998001991	0.998	0.99799983	0.999941833
25	0.996333333	0.996362732	0.996333333	0.996335941	0.999969
MEAN	0.99722667	0.997233927	0.997226667	0.997226942	0.99993538
STD	0.00103159	0.001025275	0.00103159	0.001031238	2.89277E-05
MEDIAN	0.997333333	0.997345238	0.997333333	0.997333833	0.999941833
MIN	0.995333333	0.995346514	0.995333333	0.995333217	0.999888833
MAX	0.99866667	0.998671979	0.998666667	0.998666665	0.999993833

FIGURE I.1: A copy of the raw overall data outputs from the CONMAN service, models 1 to 25.

I.2 User-centric Training/Testing set of Performance Metrics for each iteration of the CONMAN service

Iteration	User	Precision	F1	ROC_AUC	TPR	TNR	FAR	FRR	True Negative	False Positive	False Negative	True Positive
1	A	0.994011976	0.995004995	0.9999185	0.996	0.997	0.003	0.004	1994	6	4	996
	B	0.999000999	0.99950025	1	1	0.9995	0.0005	0	1999	1	0	1000
	C	0.996990973	0.99549324	0.999923	0.994	0.9985	0.0015	0.006	1997	3	6	994
2	A	0.997002997	0.997501249	0.9999565	0.998	0.9985	0.0015	0.002	1997	3	2	998
	B	1	1	1	1	1	0	0	2000	0	0	1000
	C	0.997997998	0.997498749	0.9999525	0.997	0.999	0.001	0.003	1998	2	3	997
3	A	0.996003996	0.996501749	0.9998765	0.997	0.998	0.002	0.003	1996	4	3	997
	B	1	0.99949975	0.9999995	0.999	1	0	0.001	2000	0	1	999
	C	0.997	0.997	0.999871	0.997	0.9985	0.0015	0.003	1997	3	3	997
4	A	0.993020937	0.994508238	0.999884	0.996	0.9965	0.0035	0.004	1993	7	4	996
	B	1	1	1	1	1	0	0	2000	0	0	1000
	C	0.995987964	0.994491738	0.999875	0.993	0.998	0.002	0.007	1996	4	7	993
5	A	0.994017946	0.99550674	0.999849	0.997	0.997	0.003	0.003	1994	6	3	997
	B	0.999	0.999	0.9999985	0.999	0.9995	0.0005	0.001	1999	1	1	999
	C	0.996990973	0.99549324	0.9998225	0.994	0.9985	0.0015	0.006	1997	3	6	994
6	A	0.988118812	0.993034826	0.9998745	0.998	0.994	0.006	0.002	1988	12	2	998
	B	1	1	1	1	1	0	0	2000	0	0	1000
	C	0.997979798	0.992964824	0.9998825	0.988	0.999	0.001	0.012	1998	2	12	988
7	A	0.997002997	0.997501249	0.9998505	0.998	0.9985	0.0015	0.002	1997	3	2	998
	B	1	1	1	1	1	0	0	2000	0	0	1000
	C	0.997997998	0.997498749	0.9998205	0.997	0.999	0.001	0.003	1998	2	3	997
8	A	0.99304175	0.996011964	0.9998895	0.999	0.9965	0.0035	0.001	1993	7	1	999
	B	1	1	1	1	1	0	0	2000	0	0	1000
	C	0.998993964	0.995987964	0.999895	0.993	0.9995	0.0005	0.007	1999	1	7	993
9	A	0.996	0.996	0.9999375	0.996	0.998	0.002	0.004	1996	4	4	996
	B	1	0.99949975	0.9999995	0.999	1	0	0.001	2000	0	1	999
	C	0.996003996	0.996501749	0.999948	0.997	0.998	0.002	0.003	1996	4	3	997
10	A	0.990059642	0.993020937	0.999938	0.996	0.995	0.005	0.004	1990	10	4	996
	B	0.998998999	0.99849925	0.999999	0.998	0.9995	0.0005	0.002	1999	1	2	998
	C	0.996984925	0.994486216	0.999946	0.992	0.9985	0.0015	0.008	1997	3	8	992
11	A	0.994029851	0.996508728	0.999866	0.999	0.997	0.003	0.001	1994	6	1	999
	B	1	1	1	1	1	0	0	2000	0	0	1000
	C	0.998994975	0.996491228	0.999843	0.994	0.9995	0.0005	0.006	1999	1	6	994
12	A	0.993027888	0.99500998	0.999819	0.997	0.9965	0.0035	0.003	1993	7	3	997
	B	1	0.99949975	0.9999995	0.999	1	0	0.001	2000	0	1	999
	C	0.996990973	0.99549324	0.999892	0.994	0.9985	0.0015	0.006	1997	3	6	994
13	A	0.993048659	0.996512207	0.9999135	1	0.9965	0.0035	0	1993	7	0	1000
	B	1	1	1	1	1	0	0	2000	0	0	1000
	C	1	0.996487707	0.9998565	0.993	1	0	0.007	2000	0	7	993
14	A	0.997005988	0.998001998	0.9999195	0.999	0.9985	0.0015	0.001	1997	3	1	999
	B	1	1	1	1	1	0	0	2000	0	0	1000
	C	0.998997996	0.997997998	0.999942	0.997	0.9995	0.0005	0.003	1999	1	3	997
15	A	0.996003996	0.996501749	0.999807	0.997	0.998	0.002	0.003	1996	4	3	997
	B	0.999	0.999	0.9999995	0.999	0.9995	0.0005	0.001	1999	1	1	999
	C	0.996996997	0.996498249	0.9998605	0.996	0.9985	0.0015	0.004	1997	3	4	996
16	A	0.996996997	0.996498249	0.9999465	0.996	0.9985	0.0015	0.004	1997	3	4	996
	B	0.999000999	0.99950025	1	1	0.9995	0.0005	0	1999	1	0	1000
	C	0.997	0.997	0.999951	0.997	0.9985	0.0015	0.003	1997	3	3	997
17	A	0.997005988	0.998001998	0.999939	0.999	0.9985	0.0015	0.001	1997	3	1	999
	B	1	1	1	1	1	0	0	2000	0	0	1000
	C	0.998997996	0.997997998	0.9999405	0.997	0.9995	0.0005	0.003	1999	1	3	997
18	A	0.996003996	0.996501749	0.9999115	0.997	0.998	0.002	0.003	1996	4	3	997
	B	0.999	0.999	0.9999995	0.999	0.9995	0.0005	0.001	1999	1	1	999
	C	0.997997998	0.997498749	0.999921	0.997	0.999	0.001	0.003	1998	2	3	997
19	A	0.996	0.996	0.999848	0.996	0.998	0.002	0.004	1996	4	4	996
	B	1	1	1	1	1	0	0	2000	0	0	1000
	C	0.996	0.996	0.999847	0.996	0.998	0.002	0.004	1996	4	4	996
20	A	0.989098117	0.993529119	0.9999265	0.998	0.9945	0.0055	0.002	1989	11	2	998
	B	1	1	1	1	1	0	0	2000	0	0	1000
	C	0.997981837	0.993470618	0.999912	0.989	0.999	0.001	0.011	1998	2	11	989
21	A	0.996007984	0.997002997	0.9998675	0.998	0.998	0.002	0.002	1996	4	2	998
	B	1	1	1	1	1	0	0	2000	0	0	1000
	C	0.997995992	0.996996997	0.999916	0.996	0.999	0.001	0.004	1998	2	4	996
22	A	0.996015936	0.998003992	0.999999	1	0.998	0.002	0	1996	4	0	1000
	B	1	1	1	1	1	0	0	2000	0	0	1000
	C	1	0.997995992	0.9999915	0.996	1	0	0.004	2000	0	4	996
23	A	0.990059642	0.993020937	0.999941	0.996	0.995	0.005	0.004	1990	10	4	996
	B	1	0.99949975	1	0.999	1	0	0.001	2000	0	1	999
	C	0.995979899	0.993483709	0.999948	0.991	0.998	0.002	0.009	1996	4	9	991
24	A	0.996007984	0.997002997	0.999912	0.998	0.998	0.002	0.002	1996	4	2	998
	B	0.999000999	0.99950025	1	1	0.9995	0.0005	0	1999	1	0	1000
	C	0.998996991	0.997496244	0.9999135	0.996	0.9995	0.0005	0.004	1999	1	4	996
25	A	0.990089197	0.994524639	0.999947	0.999	0.995	0.005	0.001	1990	10	1	999
	B	0.998998999	0.99849925	0.999999	0.998	0.9995	0.0005	0.002	1999	1	2	998
	C	1	0.995983936	0.999961	0.992	1	0	0.008	2000	0	8	992

FIGURE I.2: A copy of the raw user-centric data outputs from the CONMAN service, models 1 to 25.

I.3 User-centric Training/Testing set of Performance Metrics - statistical summary for each iteration of the CONMAN service

User											
A	Precision	F1	ROC_AUC	TPR	TNR	FAR	FRR	True Negative	False Positive	False Negative	True Positive
MEAN	0.994187331	0.99588853	0.99990114	0.9976	0.99708	0.00292	0.0024	1994.16	5.84	2.4	997.6
STD	0.002720888	0.0015366	4.4861E-05	0.001296148	0.0013761	0.001376081	0.0012961	2.752162786	2.75216279	1.29614814	1.29614814
MEDIAN	0.996	0.99649825	0.999912	0.998	0.998	0.002	0.002	1996	4	2	998
MIN	0.988118812	0.99302094	0.999807	0.996	0.994	0.0015	0	1988	3	0	996
MAX	0.997005988	0.99800399	0.99999	1	0.9985	0.006	0.004	1997	12	4	1000
B											
MEAN	0.99968004	0.99961993	0.99999974	0.99956	0.99984	0.00016	0.00044	1999.68	0.32	0.44	999.56
STD	0.000466418	0.00047512	0	0.000637495	0.0002332	0.000233238	0.0006375	0.466476152	0.46647615	0.637495098	0.637495098
MEDIAN	1	1	1	1	1	0	0	2000	0	0	1000
MIN	0.998998999	0.99849925	0.9999985	0.998	0.9995	0	0	1999	0	0	998
MAX	1	1	1	1	1	0.0005	0.002	2000	1	2	1000
C											
MEAN	0.99783441	0.99617237	0.99990526	0.99452	0.99892	0.00108	0.00548	1997.84	2.16	5.48	994.52
STD	0.001256712	0.00143464	4.5667E-05	0.002578682	0.0006274	0.000627375	0.0025787	1.254750971	1.25475097	2.578681834	2.578681834
MEDIAN	0.997981837	0.99649123	0.9999135	0.996	0.999	0.001	0.004	1998	2	4	996
MIN	0.995979899	0.99296482	0.9998205	0.988	0.998	0	0.003	1996	0	3	988
MAX	1	0.997998	0.9999915	0.997	1	0.002	0.012	2000	4	12	997

FIGURE I.3: A copy of the raw user-centric data outputs statistical summary from the CONMAN service, models 1 to 25.

I.4 Overall Model 22 (Shea) performance vs. Unseen dataset

vs Unseen	Accuracy	Precision_Macro	Recall_Macro	F1_Macro	ROC_AUC_OvR
Model_22	0.99621381	0.992320892	0.994254996	0.993283437	0.999910477

FIGURE I.4: A copy of the raw overall data outputs from the CONMAN service, model 22 vs Unseen dataset

I.5 User-centric Model 22 (Shea) performance vs. Unseen dataset

22 vs Unseen	Precision	F1	ROC_AUC	TPR	TNR	FAR	FRR
User_A	0.99691517	0.996505293	0.999898936	0.996095757	0.99635347	0.00364653	0.003904243
User_B	0.99893843	0.998635537	0.99999285	0.998332828	0.999383911	0.00061609	0.001667172
User_C	0.98110908	0.98470948	0.999839643	0.988336403	0.99810177	0.00189823	0.011663597

FIGURE I.5: A copy of the raw user-centric data outputs from the CONMAN service, model 22 vs Unseen dataset

I.6 Confusion Matrices: Model 22 (Shea) performance vs. Unseen dataset

22 vs Unseen	True Negative	False Positive	False Negative	True Positive
User_A	8197	30	38	9695
User_B	11355	7	11	6587
User_C	16300	31	19	1610

FIGURE I.6: A copy of the confusion matrix data outputs from the CONMAN service, model 22 vs Unseen dataset

Bibliography

- [1] Maria Papathanasaki, Leandros Maglaras, and Nick Ayres. Modern authentication methods: A comprehensive survey. *AI, Computer Science and Robotics Technology*, 2022.
- [2] Mohammed Aziz Al Kabir and Wael Elmedany. An overview of the present and future of user authentication. In *2022 4th IEEE Middle East and North Africa COMMunications Conference (MENACOMM)*, pages 10–17, 2022.
- [3] Syed W. Shah and Salil S. Kanhere. Recent trends in user authentication – a survey. *IEEE Access*, 7:112505–112519, 2019.
- [4] Maryline Laurent and Samia Bouzefrane. *Digital identity management*. Elsevier, 2015.
- [5] The Open Worldwide Application Security Project. Owasp cheat sheet series - authentication cheat sheet. 2021. URL https://cheatsheetseries.owasp.org/cheatsheets/Authentication_Cheat_Sheet.html. [Date accessed: 10-07-2023].
- [6] National Cyber Security Centre. Top tips for staying secure online. December 2018. URL <https://www.ncsc.gov.uk/collection/top-tips-for-staying-secure-online>. [Date accessed: 10-07-2023].
- [7] Google / Harris Poll. Online security survey. February 2019. URL https://services.google.com/fh/files/blogs/google_security_infographic.pdf. [Date accessed: 09-06-2023].
- [8] Alex Hern. Breached data: Largest collection ever seen with 773 million emails and 21 million passwords hacked. 2019. URL <https://www.theguardian.com/technology/2019/jan/17/breached-data-largest-collection-ever-seen-email-password-hacking>. [Date accessed: 09-07-2023].
- [9] Anti-Phishing Working Group Inc. 2022. URL <https://apwg.org/trendsreports/>. [Date accessed: 02-07-2023].

- [10] Tailgating attack: Examples and prevention — fortinet — fortinet.com. URL <https://www.fortinet.com/resources/cyberglossary/tailgating-attack#:~:text=A%20malicious%20actor%20can%20also,such%20as%20malware%20or%20phishing>. [Date accessed: 27-12-2023].
- [11] William Newhouse, Brian Johnson, Sarah Kinling, Jason Kuruvilla, Blaine Mu-lugeta, and Kenneth Sandlin. Multifactor authentication for e-commerce: Risk-based, fido universal second factor implementations for purchasers. 2019.
- [12] Sanam Ghorbani Lyastani, Michael Schilling, Michaela Neumayr, Michael Backes, and Sven Bugiel. Is fido2 the kingslayer of user authentication? a comparative usability study of fido2 passwordless authentication. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 268–285, 2020.
- [13] Sandeep Kumar, Sukhwinder Singh, and Jagdish Kumar. A comparative study on face spoofing attacks. In *2017 International Conference on Computing, Communication and Automation (ICCCA)*, pages 1104–1108, 2017.
- [14] Tarang Chugh, Kai Cao, and Anil K. Jain. Fingerprint spoof buster: Use of minutiae-centered patches. *IEEE Transactions on Information Forensics and Security*, 13(9):2190–2202, 2018.
- [15] Apple Support. Use face id on your iphone or ipad pro, 2023. URL <https://support.apple.com/en-gb/HT208108>. [Date accessed: 09-07-2023].
- [16] Liukui Chen, Xiaoxing Wang, Haiyang Jiang, Li Tang, Zuojin Li, and Yao Du. Design of palm vein platform and pattern enhancement model based on raspberry pi. In *2021 IEEE International Conference on Emergency Science and Information Technology (ICESIT)*, pages 495–498, 2021.
- [17] Hoang Thien Van, Chien Minh Duong, Giang Van Vu, and Thai Hoang Le. Palm vein recognition using enhanced symmetry local binary pattern and sift features. In *2019 19th International Symposium on Communications and Information Technologies (ISCIT)*, pages 311–316, 2019.
- [18] Shivangi Mahto, Takayuki Arakawa, and Takafumi Koshinak. Ear acoustic biometrics using inaudible signals and its application to continuous user authentication. In *2018 26th European Signal Processing Conference (EUSIPCO)*, pages 1407–1411, 2018.
- [19] Josh Taylor. Major breach found in biometrics system used by banks, uk police and defence firms. *The Guardian*, August 2019. URL <https://tinyurl.com/5d7fmkej>. [Date accessed: 09-07-2023].

- [20] Giacomo Giorgi, Andrea Saracino, and Fabio Martinelli. Using recurrent neural networks for continuous authentication through gait analysis. *Pattern Recognition Letters*, 147:157–163, 2021.
- [21] Yunze Zeng, Parth H Pathak, and Prasant Mohapatra. Wiwho: Wifi-based person identification in smart spaces. In *2016 15th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, pages 1–12. IEEE, 2016.
- [22] Yu Zhong, Yunbin Deng, and Anil K. Jain. Keystroke dynamics for user authentication. In *2012 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, pages 117–123, 2012.
- [23] Yimin Chen, Jingchao Sun, Rui Zhang, and Yanchao Zhang. Your song your way: Rhythm-based two-factor authentication for multi-touch mobile devices. In *2015 IEEE conference on computer communications (INFOCOM)*, pages 2686–2694. IEEE, 2015.
- [24] R.A.J. Everitt and P.W. McOwan. Java-based internet biometric authentication system. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(9):1166–1172, 2003.
- [25] Roman Yampolskiy and Venu Govindaraju. Behavioural biometrics: A survey and classification. *International Journal of Biometrics*, 1, January 2008.
- [26] Ahmed Awad E Ahmed and Issa Traore. A new biometric technology based on mouse dynamics. *IEEE Transactions on dependable and secure computing*, 4(3):165–179, 2007.
- [27] Nyle Siddiqui, Rushit Dave, Mounika Vanamala, and Naeem Seliya. Machine and deep learning applications to mouse dynamics for continuous user authentication. *Machine Learning and Knowledge Extraction*, 4(2):502–518, 2022.
- [28] Penny Chong, Yi Xiang Marcus Tan, Juan Guarnizo, Yuval Elovici, and Alexander Binder. Mouse authentication without the temporal aspect—what does a 2d-cnn learn? In *2018 IEEE Security and Privacy Workshops (SPW)*, pages 15–21. IEEE, 2018.
- [29] Payal Chhabra and Dr. Sonali Goyal. A thorough review on deep learning neural network. In *2023 International Conference on Artificial Intelligence and Smart Communication (AISC)*, pages 220–226, 2023.
- [30] Laith Alzubaidi, Jinglan Zhang, Amjad J Humaidi, Ayad Al-Dujaili, Ye Duan, Omran Al-Shamma, José Santamaría, Mohammed A Fadhel, Muthana Al-Amidie,

- and Laith Farhan. Review of deep learning: Concepts, cnn architectures, challenges, applications, future directions. *Journal of big Data*, 8:1–74, 2021.
- [31] Weili Fang, Peter ED Love, Hanbin Luo, and Lieyun Ding. Computer vision for behaviour-based safety in construction: A review and future directions. *Advanced Engineering Informatics*, 43:100980, 2020.
- [32] Vishal Passricha and Rajesh Kumar Aggarwal. End-to-end acoustic modeling using convolutional neural networks. In *Intelligent Speech Signal Processing*, pages 5–37. Elsevier, 2019.
- [33] Sin-Ye Jhong, Po-Yen Tseng, Natnuntnita Siriphockpirom, Chih-Hsien Hsia, Ming-Shih Huang, Kai-Lung Hua, and Yung-Yao Chen. An automated biometric identification system using cnn-based palm vein recognition. In *2020 international conference on advanced robotics and intelligent systems (ARIS)*, pages 1–6. IEEE, 2020.
- [34] Arslan Bromme. A classification of biometric signatures. In *2003 International Conference on Multimedia and Expo. ICME’03. Proceedings (Cat. No. 03TH8698)*, volume 3, pages III–17. IEEE, 2003.
- [35] Margit Antal and Norbert Fejér. Mouse dynamics based user recognition using deep learning. *Acta Universitatis Sapientiae, Informatica*, 12:39–50, 07 2020.
- [36] Penny Chong, Yuval Elovici, and Alexander Binder. User authentication based on mouse dynamics using deep neural networks: A comprehensive study. *IEEE Transactions on Information Forensics and Security*, 15:1086–1101, 2019.
- [37] Serkan Kiranyaz, Onur Avci, Osama Abdeljaber, Turker Ince, Moncef Gabbouj, and Daniel J. Inman. 1d convolutional neural networks and applications: A survey. *Mechanical Systems and Signal Processing*, 151:107398, 2021.
- [38] Sumit Saha. A comprehensive guide to convolutional neural networks — the eli5 way. 2018. URL <https://saturncloud.io/blog/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way/>. [Date accessed: 31-07-2023].
- [39] Hugo Gamboa and Ana Fred. A behavioral biometric system based on human-computer interaction. In *Biometric Technology for Human Identification*, volume 5404, pages 381–392. SPIE, 2004.
- [40] J. Brownlee. *Machine Learning Mastery with Python: Understand Your Data, Create Accurate Models and Work Projects End-to-end*. Jason Brownlee, 2016.

- [41] Maja Pusara and Carla E Brodley. User re-authentication via mouse movements. In *Proceedings of the 2004 ACM workshop on Visualization and data mining for computer security*, pages 1–8, 2004.
- [42] Chao Shen, Zhongmin Cai, and Xiaohong Guan. Continuous authentication for mouse dynamics: A pattern-growth approach. In *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2012)*, pages 1–12, 2012.
- [43] Fülöp, A., Kovács L., Kurics T., Windhager-Pokol E. Balabit mouse dynamics challenge data set. <https://github.com/balabit/Mouse-Dynamics-Challenge>. [Date accessed: 27-07-2023].
- [44] Athul Harilal, Flavio Toffalini, John Castellanos, Juan Guarnizo, Ivan Homoliak, and Martín Ochoa. Twos: A dataset of malicious insider threat behavior based on a gamified competition. In *Proceedings of the 2017 International Workshop on Managing Insider Security Threats*, page 45–56, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450351775.
- [45] Nyle Siddiqui, Rushit Dave, and Naeem Seliya. Continuous user authentication using mouse dynamics, machine learning, and minecraft. In *2021 International Conference on Electrical, Computer and Energy Technologies (ICECET)*, pages 1–6. IEEE, 2021.
- [46] P.P. Sarangi, M. Panda, S. Mishra, B.S.P. Mishra, and B. Majhi. *Machine Learning for Biometrics: Concepts, Algorithms and Applications*. Cognitive Data Science in Sustainable Computing. Elsevier Science, 2022. ISBN 9780323852098.
- [47] sklearn.preprocessing.minmaxscaler — scikit-learn.org. <https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html>, . [Date accessed: 02-11-2023].
- [48] Api reference — scikit-learn.org. <https://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics>, . [Date accessed: 12-10-2023].
- [49] NumPy Documentation. Numpy reference; numpy v1.26 manual — numpy.org. <https://numpy.org/doc/stable/reference/>, 2014. [Date accessed: 15-07-2023].
- [50] Pandas Team. User guide; pandas 2.2.0 documentation — pandas.pydata.org. https://pandas.pydata.org/docs/user_guide/index.html, 2023. [Date accessed: 11-07-2023].
- [51] Teng Hu, Weina Niu, Xiaosong Zhang, Xiaolei Liu, Jiazhong Lu, Yuan Liu, and Jiageng Chen. An insider threat detection approach based on mouse dynamics and deep learning. *Sec. and Commun. Netw.*, 2019, 2019. ISSN 1939-0114.

- [52] IEEE. Ieee recommended practice for software requirements specifications. *IEEE Std 830-1998*, pages 1–40, 1998.
- [53] J. Patton and P. Economy. *User Story Mapping: Discover the Whole Story, Build the Right Product*. O'Reilly Media, 2014. ISBN 9781491904886.
- [54] IEEE. Ieee standard for information technology–systems design–software design descriptions. *IEEE STD 1016-2009*, pages 1–35, 2009.
- [55] M. Lutz and an O'Reilly Media Company Safari. *Learning Python, 5th Edition*. O'Reilly Media, Incorporated, 2013.
- [56] R.C. Martin. *Clean Architecture: A Craftsman's Guide to Software Structure and Design*. Prentice Hall, 2018.
- [57] Keras Team. Keras documentation: The sequential class — keras.io. <https://keras.io/api/models/sequential/>, . [Date accessed: 12-06-2023].
- [58] Oracle. Mouseevent (java platform se 7) documentation. 2020. URL <https://docs.oracle.com/javase/7/docs/api/java/awt/event/MouseEvent.html>.
- [59] Python Software Foundation. os — miscellaneous operating system interfaces. 2023. URL <https://docs.python.org/3/library/os.html>.
- [60] Python Documentation. sys; system-specific parameters and functions; python 3.9.17 documentation — docs.python.org. <https://docs.python.org/3.9/library/sys.html>, . [Date accessed: 01-10-2023].
- [61] subprocess — subprocess management — docs.python.org. <https://docs.python.org/3/library/subprocess.html>. [Date accessed: 01-10-2023].
- [62] Moses Palmér. pynput documentation. 2018. URL <https://pynput.readthedocs.io/en/latest/index.html>.
- [63] Isaac Da Silva Beserra, Lucas Camara, and Marjory Da Costa-Abreu. Using keystroke and mouse dynamics for user identification in the online collaborative game league of legends. In *7th International Conference on Imaging for Crime Detection and Prevention (ICDP 2016)*, pages 1–6, 2016.
- [64] Using matplotlib 2014; matplotlib 3.8.2 documentation — matplotlib.org, 2014. URL <https://matplotlib.org/stable/users/index>. [Date accessed: 12-07-2023].
- [65] PyInstaller Development Team. Pyinstaller documentation. n.d. URL <https://pyinstaller.org/en/stable/>.

- [66] Tensorflow v2.15.0.post1 — tensorflow.org. https://www.tensorflow.org/api_docs/python/tf, May 2023. [Date accessed: 09-05-2023].
- [67] Keras Team. Keras documentation: Conv1d layer — keras.io. https://keras.io/api/layers/convolution_layers/convolution1d/, . [Date accessed 05-06-2023].
- [68] tf.keras.optimizers.adam — tensorflow v2.15.0.post1 — tensorflow.org, . URL https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/Adam. [Date accessed: 12-07-2023].
- [69] Keras Team. Keras documentation: Layer weight regularizers — keras.io. <https://keras.io/api/layers/regularizers/>, . [Date accessed: 30-07-2023].
- [70] sklearn.model_selection.train_test_split — scikit-learn.org, . URL https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html. [Date accessed: 12-10-2023].
- [71] Xiao-jun Chen, Fei Xu, Rui Xu, S. M. Yiu, and Jin-qiao Shi. A practical real-time authentication system with identity tracking based on mouse dynamics. In *2014 IEEE Conference on Computer Communications Workshops (INFOCOM WK-SHPS)*, pages 121–122, 2014.
- [72] Margit Antal and Elöd Egyed-Zsigmond. Intrusion detection using mouse dynamics. *IET Biometrics*, 8(5):285–294, 2019.
- [73] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, page 41–48, New York, NY, USA, 2009. Association for Computing Machinery. ISBN 9781605585161.
- [74] Harshul Garg, Tanupriya Choudhury, Praveen Kumar, and Sai Sabitha. Comparison between significance of usability and security in hci. In *2017 3rd International Conference on Computational Intelligence Communication Technology (CICT)*, pages 1–4, 2017.
- [75] Nur MERDANOĞLU and Pınar ONAY DURDU. A systematic mapping study of usability vs security. In *2018 6th International Conference on Control Engineering Information Technology (CEIT)*, pages 1–6, 2018.
- [76] European Committee for Electrotechnical Standardization. En 60839-11-1:2013/ac:2015 - alarm and electronic security systems - part 11-1: Electronic access control systems - system and components requirements. 2015.
- [77] Pep 8; style guide for python code — peps.python.org — peps.python.org, 2013. URL <https://peps.python.org/pep-0008/>. [Date accessed: 10-06-2023].

- [78] Pep 257; docstring conventions — peps.python.org — peps.python.org. URL <https://peps.python.org/pep-0257/>. [Date accessed: 11-06-2023].
- [79] Python Documentation. os; miscellaneous operating system interfaces; python 3.9.17 documentation — docs.python.org. <https://docs.python.org/3.9/library/os.html>, . [Date accessed: 24-10-2023].
- [80] Python Documentation. csv; csv file reading and writing; python 3.9.17 documentation — docs.python.org. <https://docs.python.org/3.9/library/csv.html>, . [Date accessed: 29-09-2023].
- [81] time; time access and conversions; python 3.9.17 documentation — docs.python.org. <https://docs.python.org/3.9/library/time.html>, 2014. [Date accessed: 24-09-2023].
- [82] Python Documentation. datetime; basic date and time types; python 3.9.17 documentation — docs.python.org. <https://docs.python.org/3.9/library/datetime.html>, 2014. [Date accessed: 24-09-2023].
- [83] Margit Antal and Lehel Denes-Fazakas. User verification based on mouse dynamics: a comparison of public data sets. In *2019 IEEE 13th International Symposium on Applied Computational Intelligence and Informatics (SACI)*, pages 143–148. IEEE, 2019.
- [84] Keras Team. Keras documentation: Activation layer — keras.io. https://keras.io/api/layers/core_layers/activation/, . [Date accessed: 12-06-2023].
- [85] Keras Team. Keras documentation: Callbacks api — keras.io. <https://keras.io/api/callbacks/>, . [Date accessed: 24-11-2023].
- [86] Python Documentation. logging; logging facility for python; python 3.9.17 documentation — docs.python.org. <https://docs.python.org/3.9/library/logging.html>, . [Date accessed: 19-07-2023].
- [87] tf.keras.utils.to_categorical — tensorflow v2.15.0.post1 — tensorflow.org, . URL https://www.tensorflow.org/api_docs/python/tf/keras/utils/to_categorical. [Date accessed: 05-10-2023].
- [88] Welcome to pyautogui019 documentation; pyautogui documentation — pyautogui.readthedocs.io, 2019. URL <https://pyautogui.readthedocs.io/en/latest/>. [Date accessed: 29-12-2023].
- [89] psutil documentation 2014; psutil 5.9.8 documentation — psutil.readthedocs.io, 2014. URL <https://psutil.readthedocs.io/en/latest/>. [Date accessed : 15-01-2024].