Detection and Identification of Context Switch events through privacy-preserving historical behavioural PC Users data

VI VIET HOANG NGO
SID: 510064233

Supervisor: John Stavrakakis

This thesis is submitted in partial fulfilment of
the requirements for the degree of Bachelor of Advanced Computing
(Honours)

School of Computer Science
The University of Sydney
Australia

30 May 2024

Student Plagiarism: Compliance Statement

I certify that: Vi Viet Hoang (Henry) Ngo

I have read and understood the University of Sydney Student Plagiarism: Coursework Policy and Procedure.

I understand that failure to comply with the Student Plagiarism: Academic Board Policy: Academic Dishonesty and Plagiarism can lead to the University commencing proceedings against me for potential student misconduct under the Academic Dishonesty and Plagiarism in Coursework Policy.

This Work is substantially my own, and to the extent that any part of this Work is not my own I have indicated that it is not my own by Acknowledging the Source of that part or those parts of the Work.

Name: Vi Viet Hoang (Henry) Ngo

Signature: [illegible] **Date: 30/05/2025**

Abstract

Everyday computer use involves frequent shifts of attention between multiple software applications. Most studies in human-computer interaction have focused on predefined tasks with clear boundaries, such as document editing or multimedia consumption. However, real-world workflows are more fluid and unstructured, with users moving unpredictably among activities. The thesis examines whether application-level context switches can be detected automatically by analysing privacy-assured one-minute aggregated behavioural data and treating each change in the active window as an indicator of a workflow transition. Behavioural signals, which include mouse movements, keyboard events and system resource usage, are drawn from the privacy-assured BEHACOM dataset, which provides behavioural PC records from knowledge workers. Five detection methods are evaluated: a naive approach that computes statistical features over fixed look-back intervals; a majority-based contextual clustering method that applies K-means clustering to select representative intervals; a proximity-based contextual clustering method that uses K-means then, greedily selects clusters closest to recent data; a long short-term memory network (LSTM) that learns temporal dependencies in user behaviour; and a hidden Markov model (HMM) that represents behavioural sequences as transitions among latent states. Each method is tested across a range of window lengths and classification back-ends, including gradient boosting, random forest and extra trees. Evaluation metrics include accuracy, precision, recall and F1 score for both switch detection and destination-application identification. Experimental results indicate that sequence-based models outperform clustering methods and the naive baseline, with the long short-term memory network achieving the highest recall and precision for detection and the hidden Markov model delivering robust identification of the destination application. Dynamic window selection from the clustering-based methods also improves clustering performance but requires careful parameter tuning. Analysis of feature importance shows that patterns of typing intensity, mouse movement entropy and CPU usage shifts provide reliable signals of upcoming context switches. These findings contribute new insights into digital work patterns and support the design of adaptive systems capable of anticipating shifts in user focus.

Acknowledgements

I would like to express my deepest gratitude to my parents for their incredible support both emotionally and financially through every step of this journey. Their belief in me made this work possible. I am also immensely thankful to my professor, Dr John Stavrakakis, for expert guidance, patient mentorship and critical feedback. Finally, I owe a special thanks to my friends, whose constant companionship and cheerful encouragement lifted my spirits during challenging times. Each of these contributions has been invaluable to the completion of this thesis.

Contents

List of Figures

List of Tables

1. Introduction

1.1 Problem statement

In today's fast-paced digital world, users constantly juggle multiple applications, shifting their attention from one task to another in a matter of seconds. Yet, most research on context switching in human-computer interaction has focused on pre-identified tasks, such as writing reports, editing photos or watching films, where user tasks and their boundaries are neatly defined in advance. This controlled approach, however, overlooks the messy reality of everyday computer use, where users move fluidly between activities and task boundaries are rarely obvious. In the real world, a simple change in the active application often signals a shift in the user's focus, as each application brings its own set of activities and goals. This thesis explores whether it is possible to detect these context switches, in this case, application switches, by analysing behavioural data, such as mouse movements, keyboard activity, and system resource usage, where 1) using only data aggregated into time intervals; and 2) without relying on predefined task categories. This thesis will base on the assumption that each application is responsible for a particular user's work and transitioning from an application to another will produce a context switch – a switch in user's workflow. By treating each change between applications as a context switch, labels can be generated directly from the chosen BEHACOM data. This approach opens the door to training and evaluating machine learning models that aim to recognize and detect these subtle transitions, offering new insights into the rhythms of digital work to help identify current users' workflow.
The thesis introduces four approaches of detecting switches, and these approaches will be compared against the base line Naïve approach.
The approaches are:

- The Naïve approach, acts as a baseline, looks back at fixed-time window period, capturing the means of numeric values and the entropy of the past app usage and use that to detect context switch.
- MCC (Majority-based Contextual Clustering approach), a proposed method of clustering time series data and choosing the majority cluster, in order to identify

the windows that are related to a switch, and also capturing the means of numeric values and the entropy of the past app usage.

- PCC (Proximity-based Contextual Clustering approach), a proposed method of clustering time series data like the MCC but aims to divide the past windows smaller and greedily choosing the cluster that is closest to the nearest window, in order to identify the windows that are related to a switch, and also capturing the means of numeric values and the entropy of the past app usage.
- LSTM (Long-Short Term Memory), a proposed method of modelling sequential data through LSTM neural network in order to detect context switch and identify destination application.
- Hidden Markov Model (HMM), a proposed method of modelling sequential data through states in order to detect context switch and identify destination application.

All in all, this leads to this thesis' research question:

Can context switches be detected by analysing users' one-minute aggregated historical behavioural data, and how well do different look-back windows and classification methods affect both the detection of switches and the identification of the destination application?

Alongside this research questions come the hypotheses:

- There exists an optimal selection of minutes within a look-back, which would be based on a dynamic selection of minutes from behavioural patterns window that maximizes the accuracy, precision, recall, F1 of application switch detection and destination identification.
- The sequence-based modelling approaches (LSTM and HMM) will outperform context-clustering approaches (PCC and MCC) in detecting application switches and identifying destinations, as they better preserve and utilize the temporal dependencies in user behaviour by maintaining the order of events and learning how behaviour evolves over time. Moreover, clustering approaches may lose important sequential patterns by selecting minutes based on similarity rather than temporal order.

- Both sequence-based modelling approaches and contextual clustering approaches perform better than the baseline Naïve approach.

1.2 Research Objectives

This research aims to achieve and evaluate on the different objectives:

- Investigate and evaluate whether the combinations behavioural patterns of previous minutes before the switch (for example: random typing bursts, concentrated mouse movements, app usage patterns) can determine the optimal look-back window by comparing different approaches: Naïve fixed window (last N minutes) and dynamic selection through clustering (K-Means) approaches.
- Investigate and evaluate whether sequential approaches (through HMM - Hidden Markov Model and LSTM - Long Short-Term Memory) differ with the context-clustering approaches (PCC and MCC).
- Evaluate different classification models (Gradient Boosting, Random Forest and Extra Trees)
- Develop a cascaded pipeline that first detects whether a switch will occur, then identifies the destination of the switch.
- Evaluate methodologies based on standard metrics (accuracy, precision, recall, F1-score) between different users.
- Analyse the feature importance of each model and their adaptivity to different levels of noise.

This research aims to address a gap in literature by focusing on identifying context switches, in this case, application switch, without the known activities through the BEHACOM dataset, which provides rich, longitudinal behavioural data from knowledge workers. While individual behavioural markers such as sudden changes in mouse activity or typing patterns, may not reliably signal a context switch in isolation, it is hypothesized that a combination of these features, learned as patterns through sequence-based approaches, can provide a reliable indicator. By understanding these patterns, the research can better predict when context switches will occur, identify the likely destination of the switch, potentially mitigate the cognitive costs associated with context switching. The key insight is a context switch can only be identified after it has occurred, the combination of behavioural patterns leading up to the switch. When properly modelled through sequence-based or context-clustering approaches, this can provide reliable predictive power for both switch detection and destination identification.

1.3 Thesis Structure

The thesis is structured as follows: the literature review surveys existing work and datasets; the methodology details data processing and experimental design; the results present model performance and analysis; the discussion reflects on findings, limitations, and future directions and the conclusion summarizes and ends the research. This thesis presents a comprehensive literature review examining behavioural datasets available for analysing knowledge workers when working on personal computer environments, with particular emphasis on the BEHACOM dataset and its selection rationale. The review encompasses current research on context switch detection, including the datasets employed, performance metrics of various classifiers, and applicable approaches for BEHACOM analysis. Additionally, the review explores time series clustering methodologies, in combination with sliding windows, as the specific activities preceding context switches (application transitions) remain undefined in the BEHACOM dataset. This investigation aims to establish a foundation for understanding user behaviour patterns and developing effective context switch detection models. The methodology section of this thesis provides an examination of data processing techniques, experimental design, and the theoretical foundations underlying each approach. The results section presents a detailed analysis of model performance across multiple chosen users in the BEHACOM dataset, including feature importance metrics and model robustness under varying noise conditions. The discussion section evaluates the research outcomes, identifies limitations in the current approach, and proposes potential directions for future research. Finally, the conclusion part summarizes the thesis and confirms the finding of thesis.

The next chapter will introduce literature review for this thesis.

2. Literature Review

2.1 Problem Statement

As discussed in the introduction, detecting context switches, especially application switches, is a challenging problem in the study of digital work. Previous research has often focused on tasks that are clearly defined, but this thesis aims to address the more complex reality where users move fluidly between activities and task boundaries are not always obvious. The main challenge is to identify these context switches using behavioural data, without relying on predefined task categories. To tackle this, it is important to have access to datasets that reflect real user behaviour, define the user-centric, task-centric and goal-driven approaches in HCI as well as to consider different methods for classification and for clustering time series data. The next sections will review the available behavioural datasets, examine relevant classification approaches, and explore unsupervised methods for analysing sequential user activity. This review will help explain the choices made in the methodology for this research.

2.2 Related HCI Datasets

One of the central challenges at the outset of this research was identifying a dataset capable of supporting fine-grained user behaviour analysis and task inference. This research aims to infer high-level activities and task structures from low-level digital interactions such as keystrokes, mouse movements, application usage, and system resource metrics. To enable this, the ideal dataset needed to provide synchronized, timestamped data across multiple modalities, while capturing authentic, unconstrained user behaviour over extended periods.

2.2.1. Challenges Encountered

For this research, developing or acquiring a suitable dataset has been significantly challenging. Initially, efforts involved creating a custom dataset through a task-logging application. This approach required institutional ethics approval due to privacy concerns associated with detailed interaction logging, such as keystrokes and mouse movements. Ethical challenges were considerable, focusing on risks associated with identifying individuals through their behaviour patterns. Moreover, the technical complexity of reliably capturing diverse user inputs, which may include keystrokes, mouse actions, active applications, and system resource metrics, added logistical burdens. Moreover, attempts to contact other researchers for existing datasets have been made and were unsuccessful, reflecting broader difficulties in acquiring high-quality behavioural data, often restricted by proprietary constraints, unresolved privacy concerns, or lack of supplementary shared materials.

To address this challenge, an extensive search was undertaken to identify a dataset that aligns with the study's research problem and objectives. Most publicly available datasets reviewed were not suitable due to insufficient granularity or limited modality. Many lacked sufficient granularity in time or behavioural detail, or they were limited to only one type of interaction. For instance, the Bogazici Mouse Dynamics Dataset (Kılıç et al., 2021) consists of mouse interaction logs collected from 24 users and is mainly intended for behavioural biometrics and security applications. While the dataset captures fine mouse dynamics, it does not include keyboard or application data, limiting its use for multitasking inference. Similarly, the IKDD Keystroke Dynamics Dataset (Tsimperidis et al., 2024) focuses exclusively on keystroke dynamics for the purpose of user authentication and classification. Although the keystroke timing data is precise, the dataset lacks contextual, application-level, and cross-modal behavioural information needed for task modelling. The Attentive Cursor Dataset (Leiva & Arapakis, 2020) provides high-resolution mouse cursor trajectories, aimed at analysing user attention and cognitive effort. However, it does not include other behavioural channels such as typing or application focus, which are critical for task-level analysis. The Mouse Dynamics Dataset examined by Kuric et al. (2024) investigates the reliability of mouse dynamics in behavioural studies and includes features like click and movement patterns. However, it lacks synchronized application context, session segmentation, or labelled task episodes, reducing its suitability for workflow inference.

2.3.2 Comparative Evaluation of Candidate Datasets

To support the development of machine learning models aimed at detecting context switches, particularly transitions between different application types, three potential datasets that have been used in previous studies of user behaviour, are evaluated: SWELL, RLKWiC, and BEHACOM. Each dataset offers specific advantages, but only BEHACOM aligns closely with the requirements of this project, especially in terms of temporal resolution, modality coverage, and data realism.

The SWELL Knowledge Work (SWELL) dataset was created to examine how stress impacts user behaviour in a semi-natural office environment (Koldijk et al., 2014). The dataset includes data from 25 participants who completed structured knowledge work tasks like writing reports and preparing presentations under three conditions (neutral, under time pressure, and with interruptions). It features a rich mix of modalities, including physiological sensors, facial expression tracking, posture data, and computer interaction logs which represent 3 hours of work for each user. While useful for studying stress-related responses and human-computer interaction under pressure, due to the inadequate data logged (only 3 hours for each user), SWELL is less applicable for training models to detect real-world context switching.

The RLKWiC dataset (Bakhshizadeh et al., 2024) captures real-world digital work through a combination of user context annotations and desktop-level interaction data such as window switches and file operations. While the inclusion of semantic metadata supports context-aware applications, the dataset lacks the detailed input activity, such as keystroke or mouse behaviour, needed for modelling transitions between tasks. Its emphasis on semantic context over behavioural mechanics limits its utility for detecting nuanced, short-term context shifts, which are central to this research.

The BEHACOM dataset (Sánchez Sánchez et al., 2020) stands out as the most suitable dataset for context-switch detection. To clarify, the dataset was collected over 55 days from 12 participants in unconstrained, real-world environments, captured user behaviour across four comprehensive dimensions: keyboard activity, mouse usage, application context and system-level metrics All data are aggregated into fixed one-minute windows, offering a privacy-preserving yet sufficiently fine temporal resolution to model rapid transitions. As a result, this fixed-window design facilitates context switch detection while still maintaining anonymity. BEHACOM's strength lies in its passive data collection approach and avoidance of experimental task constraints or user labelling. It records authentic behaviour as it naturally unfolds, thereby enabling the development of scalable, unsupervised, and supervised models. Furthermore, its open access and availability of the associated logging tools via GitHub enhance its value for reproducibility and future adaptation.

In summary, while SWELL and RLKWiC contribute useful perspectives on user stress and semantic context, respectively, BEHACOM is best suited for the aim of this project: detecting context switches based on low-level interaction logs. Its scale, structure, and modality diversity make it a strong choice for developing models that can segment and label context changes in real-world usage.
The following section will look into the definitions of user-centric, task-centric and goal-driven approaches in HCI and what defines a context switch in each approach.

2.3 User-centric, Task-centric, and Goal-driven approaches in HCI

2.3.1 User-Centric and Task-Centric Approaches

Understanding the distinction between user-centric and task-centric approaches is critical for designing systems that detect context switches in digital environments. These paradigms differ in how they define tasks, model transitions, and interpret behaviour, shaping the methodologies suitable for context detection. This section examines foundational studies that define these paradigms, compares how they operationalize context switches, and reflects on how these insights relate to the core problem of this thesis.

Definition of User-Centric and Task-Centric

Task-centric, also referred to as activity-centric approaches, aim to model tasks as objective and structured units of work composed of observable digital interaction traces (Granitzer et al., 2009). Similarly, Mirza et al., (2015) stated that the approaches typically assume that tasks exhibit identifiable patterns such as application usage or temporal interaction bursts. These patterns are treated as input to machine learning models that are trained to detect or classify user activities. Task-centric methods are commonly used in studies where tasks are pre-labelled or follow standard workflows, which makes them effective in building generalizable models that apply across users (Granitzer et al., 2009).

In contrast, user-centric approaches prioritize individual differences and contextual subjectivity in how tasks are defined, performed, and interpreted (Jahanlou et al., 2023; Granitzer et al., 2009). Rather than relying on standardized task definitions, user-centric approaches let users define their own activity boundaries, often retrospectively. These approaches aim to reflect the user's own intentions and perceptions of what constitutes a meaningful task or context, even when there is no externally recognizable structure. This makes them valuable for capturing personal workflows but also difficult to scale across users without considerable manual input (Jahanlou et al., 2023).

For reference, Granitzer et al. (2009) illustrate both paradigms in a comparative study. In their task-centric experimental condition, participants completed a series of predefined tasks that were labelled and known in advance. The researchers trained classifiers such as Naïve Bayes, k-Nearest Neighbors (KNN), and Support Vector Machines (SVM) using features including window title, application name, and user interaction frequency. The classification models achieved relatively high accuracy in identifying these routine tasks across users. In the user-centric scenario, however, the same interaction logs were clustered per participant, and each participant assigned their own labels to the discovered clusters (Granitzer et al.,2009). This enabled the modelling of personalized tasks, such as ad-hoc browsing or informal communication, that were not included in the predefined task set. While this approach increased the relevance of the task labels to each user, it significantly reduced the ability to generalize the models across the study population. The contrast highlights the trade-off between precision in modelling individual behaviour and the challenge of developing a system that works for many users.

Mirza et al. (2015) also compare the two paradigms by applying a time-distance-based segmentation approach to infer user activity blocks. In their task-centric condition, they used shared categories like reading, writing, or browsing and trained models using features such as typing rate, dominant application, and switching frequency. These task blocks were derived algorithmically and labelled using predefined taxonomies. In the user-centric version, they allowed participants to label their own segments after reviewing the logs. Although user-centric labels improved within-user accuracy, the resulting models showed limited utility when applied to other users. This reflects how personalized understanding of activities introduces variability that makes it difficult to train general models. Importantly, the authors concluded that dynamic behaviour features, particularly those related to application usage, were more predictive than static attributes such as time of day.

What makes up a context switch in User-centric or Task-centric?

In task-centric models, a context switch is defined as a transition from one goal-directed digital task to another. This can be observed when the user shifts from one application to another that serves a different purpose, for example, moving from an email client to a code editor. This transition typically aligns with a measurable change in interaction patterns, such as changes in typing behaviour, window configuration, or active process. In studies like Granitzer et al. (2009), such switches are indirectly inferred by changes in classification predictions over time, which indicate that the underlying activity type has changed. Mirza et al. (2015) operationalize context switches through segmentation boundaries that are derived based on interaction metrics, such as inactivity gaps or shifts in application focus. These segments are then classified into activity types, and a change from one label to another is treated as a context switch. However, these models rely on structured, labelled data to define when and how context changes occur.

On the other hand, user-centric approaches define context switches as subjective transitions based on individual perception. Jahanlou et al. (2023) provide qualitative insights into this concept through a study involving interviews with professionals from different domains. Participants described switching applications not only when changing tasks but also due to tool limitations, collaboration needs, or even personal comfort with different software interfaces. In many cases, what the system might classify as two different tasks was considered a single cohesive activity by the user.

Conversely, users might perceive distinct phases within one application session as separate tasks. This suggests that context switches, from a user-centric perspective, are influenced by factors such as workflow planning, external interruptions, and mental fatigue.

Tashman and Edwards (2012) add to this understanding with WindowScape, a task-centric window manager that captures desktop states as snapshots, allowing users to return to previous configurations. These snapshots are triggered based on system heuristics, such as significant window and application state changes. Although their system does not explicitly define or detect context switches, the captured transitions serve as useful approximations. When users revisit or mark a snapshot, they are effectively acknowledging a task boundary, which supports the idea that application changes are useful proxies for context transitions.

Reflection on the Problem Statement

The literature shows there is a clear trade-off between scalability and personalization. Task-centric models, like those used by Granitzer et al. (2009) and Mirza et al. (2015), are easier to scale because they use pre-labelled tasks and features that work across different users. However, they often miss the complexity of how people work in real life. On the other hand, user-centric models give a much deeper view of individual behaviour and context, but they rely heavily on manual labelling and user input, which makes them harder to scale. Another key difference lies in how each approach defines a context switch. Task-centric methods usually detect them based on clear changes in behaviour or system usage, while user-centric methods look at more personal, cognitive, or situational factors. This makes it difficult to design one model that can do both well, work across many users and still be sensitive to individual habits.

Both perspectives are valuable for understanding how people interact with digital systems. task-centric models are good at using machine learning to detect general patterns, while user-centric models are better at capturing the more personal and nuanced aspects of task switching. This thesis addresses this gap by proposing a pipeline on the BEHACOM dataset of real-life knowledge workers with the assumption that each app used by the user, is responsible that user's task, which sits in the middle of both user-centric and task-centric approaches.

2.3.2 Goal-Driven Approaches

Broader than user-centric and task-centric approaches, understanding user behaviour from a goal-driven perspective provides a higher-order framework for interpreting digital interaction sequences. Rather than viewing actions as isolated events, goal-driven models assume that behaviour is guided by latent intentions and objectives. This section explores foundational works on goal inference and recognition in human-computer interaction and examines how these models define user goals, recognize transitions, and relate to context-switch detection.

Defining Goal-Driven Behaviour

Goal-driven approaches model user behaviour as the result of intentional action sequences directed toward achieving specific goals. These models emphasize the latent structure behind observable actions and treat goals as either explicit objectives or inferred mental states. User behaviour is interpreted in terms of plans and strategies, where each action is meaningful only in the context of a larger intention. The central hypothesis is that understanding a user's goal enables better support, prediction, or adaptation (Papadimitriou et al. ,2016).

Papadimitriou et al. (2016) proposed a comprehensive framework for goal-aware systems that distinguishes three key phases: goal modelling, goal recognition, and goal exploitation. Goals are formally modelled using environment states and variables and can be represented as either hard goals (Boolean targets) or soft goals (scored preference functions). Different modelling strategies are proposed depending on the availability of action data which include plan libraries that encode mappings between goals and action sequences, consistency graphs that identify feasible paths to goals, and action-centric planners that describe goals based on preconditions and effects (Papadimitriou et al., 2016). In the recognition phase, deterministic and probabilistic methods are employed to infer user goals from observed actions and techniques such as Bayesian networks and Markov models are used to estimate goal probabilities over time (Papadimitriou et al., 2016).

Armentano and Amandi (2007) provide a survey of plan recognition models in interface agents, distinguishing between consistency-based methods and probabilistic models. Consistency-based approaches eliminate goal hypotheses inconsistent with the observed behaviour, often using plan libraries and hierarchies. In contrast, probabilistic methods assign likelihoods to goal hypotheses and are better suited for noisy, interleaved, or incomplete data. Their review confirms the effectiveness of probabilistic reasoning, including Abstract Hidden Markov Models (AHMM), for dynamic goal recognition in interactive systems.

Yan et al. (2021) present Tessera, a segmentation framework designed to infer goal changes during data analysis workflows. Tessera segments exploratory visual analytics sessions into discrete goal-directed blocks by combining user interaction logs with metrics about data coverage and transformation functions. The system uses similarity calculations across temporal windows to detect when user intent has shifted, approximating context changes at the analytical task level. Evaluation using precision, recall, and F1-score against ground-truth annotations shows Tessera's effectiveness at detecting transitions between cognitive states (Yan et al., 2021).

What Defines a Context Switch in Goal-Driven Systems?

In goal-driven systems, context switches occur when the user transitions from one latent goal to another. This may correspond to observable shifts in behaviour, such as changing the data subset under analysis, switching applications, or abandoning a prior plan. For example, in Tessera, a context switch is detected when changes in interaction similarity suggest a shift in analytical focus (Yan et al., 2021). Papadimitriou et al. (2016) view context switches more implicitly, where changes in environment states or plan progress suggest a move from one goal state to another. Similarly, Armentano and Amandi (2007) emphasize that recognizing interleaved or suspended plans is critical, suggesting that real-world users do not pursue goals in isolation but switch among them based on need or interruption.

Reflection on the Problem Statement

The core difference between traditional goal-driven models and this thesis lies in the level of abstraction. Most goal-aware systems, such as those by Papadimitriou et al. (2016) and Yan et al. (2021), aim to identify high-level goals like "searching for help" or "analysing data" to support user intent. These systems typically assume either access to predefined goal libraries or semantic annotations to support inference.
In contrast, this thesis focuses on detecting micro-level transitions such as context switches between application types, inferred from behaviour traces alone, with no predefined goal hierarchies. Goal-driven is an interesting concept to look at when identifying tasks' meaning and workflow, however, the needs for a clear defined task or semantic labelling makes it hard to implement and achieve using the BEHACOM dataset.

The following part of literature review investigate which machine learning approaches and methodologies the past papers used to detect a context switch.

2.4 Context Switch Detection Methodologies

Detecting context switches in user behaviour requires analysing subtle shifts in interaction patterns across multiple applications, input modalities, and temporal structures. This section critically examines eight relevant studies, focusing on the types of data used, the methodologies employed, the performance of these methods, and their alignment with the objective of real-time, low-intrusion context switch detection from passively logged behaviours.

2.4.1 Types of data used across papers

All reviewed studies rely on passive logging of user interactions such as keyboard activity, mouse events, window transitions, and application focus changes. Most datasets were mostly gathered in real or lightly controlled settings to make sure it reflects how people use computers in everyday life. For example, Meyer et al. (2022) collected application usage and interaction logs via a background tool called ‘PersonalAnalytics’ during both observational and experience sampling studies. Similarly, Pellegrin et al. (2021) used ‘TaskPit’ to collect timestamped records of application usage and input events. Granitzer et al. (2009) and Mirza et al. (2015) logged mouse clicks, keystrokes, and window state changes in desktop environments. Satterfield et al. (2020) captured screenshots and OCR-derived text, while Tessera (Yan et al., 2021) focused on user interactions with visual analytic systems. Additionally, Dev and Liu (2017) worked with large-scale logs from a desktop application (10,000 sessions), and Tian et al. (2020) used both manually annotated mobile app logs and large-scale app usage data from millions of users. Similarly, Rath et al. (2011) and Devaurs et al. (2012) collected detailed interaction logs from desktop environments using a mix of system-level monitoring tools and custom application sensors. These logs included mouse activity, keyboard input, clipboard usage, and information about accessed resources, all recorded while university students completed knowledge-intensive tasks in a controlled lab setting.

Despite this diversity, the shared trait across these studies is their reliance on sequences of interaction events that reflect cognitive and behavioural transitions. This reinforces the viability of behaviour-based detection in context switching from logged behavioural data in digital environments.

2.4.2 Methodologies Employed

The studies apply a range of approaches including supervised machine learning, Bayesian inference, segmentation algorithms, and plan recognition frameworks.

Meyer et al. (2022) framed switch detection as a change-point detection problem. They engineered 84 features covering user input such as typing rates, application transition frequency, and lexical similarity of window titles. These were then used in segment-based classification models trained using Random Forests, with SMOTE applied to balance classes. In this study, the authors treated switches as transitions across two application focus segments, optimizing window length and feature sets to detect both the timing and type of switch.

In another paper, Pellegrin et al. (2021) used a hierarchical Bayesian framework that separated frequent and infrequent tasks. To clarify, their system worked by estimating the likelihood of different tasks based on how users interacted with applications, including keystrokes and how long they stayed active. As new data came in, the model updated its understanding over time, allowing it to adapt to changing behaviour. When compared to models like KNN, SVM, and rule-based approaches, the Bayesian method performed better, especially for rare tasks that did not have many labelled examples, however, the research expressed the needs for a clear labelled task datasets as inputs for training models.

Mirza et al. (2015) used a time-distance segmentation approach where a new activity was marked whenever the time gap between two actions was longer than a set threshold. This threshold was customized for each user to reflect their individual interaction patterns. After segmenting the sessions, each activity was labelled using a voting method that gave more weight to rare applications, helping the system better capture less common user behaviours. The researchers then extracted three groups of features: temporal features for the time of the activities and the longest pause between actions. interactional features for applications used most during each activity and content-based features for words found in window titles. These features were used to train five types of machine learning models: Naïve Bayes, Decision Trees (J48), k-Nearest Neighbors (IBk), and Support Vector Machines. The models were tested using stratified 10-fold cross-validation, and Decision Trees turned out to perform best, especially when personalized for each user. This showed that the tree-based method was good at recognizing individual behaviour patterns.

Similarly, Granitzer et al. (2009) processed raw desktop interaction logs by first aggregating low-level events into “event blocks”. These blocks were defined based on logical groupings such as continuous activity within the same window or application. Each block was then enriched with attributes including the application name, the content of the window, and semantic type labels like reading or editing. To prepare the textual features, window titles and other text fields were cleaned and normalized using stopword removal and case folding. Subsequently, Term Frequency–Inverse Document

Frequency (TF-IDF) was applied to convert the cleaned text into numerical vectors. For ranking the relevance of individual features, the authors used information gain, selecting those that contributed most to task differentiation. The final classification was performed using three algorithms: Naïve Bayes, k-Nearest Neighbours (KNN), and Support Vector Machines (SVM). These models were trained and evaluated using stratified 10-fold cross-validation on both task-centric and user-centric datasets. In the task-centric setup, Naïve Bayes achieved the highest performance with an accuracy of 83.5%, particularly when using application name, window title, and content features.

Dev and Liu (2017) introduced a method called “membership-based cohesion” to detect frequent user tasks by analysing over 10,000 real-world user sessions from a desktop photo editing application, which included 1,497 event types and a high level of noise and variability. Instead of segmenting tasks by fixed time windows, their approach minimized the number of unrelated actions, also referred to as outliers, that interrupted core task patterns. They began by mining frequent task sets using a low support threshold to ensure rare but meaningful action sets were captured. Each candidate set was then scored using a cohesion function that accounted for how consistently the set appeared together, penalizing scattered or interrupted occurrences. As the result, this method tolerated flexible ordering and repetitions, allowing for more natural patterns to be captured (Dev & Liu, 2017). Finally, expert judges reviewed the top-ranked task patterns and confirmed their quality, with the top 16 patterns achieving a precision of 1.0.

Satterfield et al. (2020) designed a method to identify information-seeking tasks by collecting over 40,000 screenshots from 17 developers, who completed six predefined tasks in a controlled lab setting where screen captures were taken every second. To clarify, optical character recognition (OCR) was applied to extract visible UI text, which was then cleaned and processed using ‘RAKE’ - Rapid Automatic Keyword Extraction (Satterfield et al.,2020). Subsequently, like Granitzer et al. (2009), task segments were transformed into vector representations using Term Frequency (TF), Term Frequency–Inverse Document Frequency (TF-IDF), and Word2Vec. Cosine similarity was used to compare segment vectors with known task descriptions. Among the methods tested, TF-IDF yielded the highest task-matching accuracy at 70.6%

(Satterfield et al., 2020). They also produced visual word clouds for each segment and evaluated their interpretability through a user study, in which participants achieved 67.9% accuracy when manually identifying tasks from the generated word clouds.

In another related work, Tian et al. (2020) treated task identification as two binary classification problems: detecting task boundaries between events and determining whether two logs belonged to the same task. They used mobile interaction data from the UbiqLog and Flurry datasets, which, while not directly focused on desktop context switching, offered useful methods applicable to broader behaviour modelling. Their feature set included time intervals between events, TF-IDF similarity scores based on app descriptions from the Google Play Store, and sequential log relationships such as 'Pointwise Mutual Information' (PMI). These features were input into classifiers like Random Forest and Logistic Regression. For the result, logistic regression performed best, achieving an F1 score of 0.89, and significantly outperformed traditional time-based segmentation methods. This work shows that even in mobile contexts, combining semantic, temporal, and sequential features can effectively improve task segmentation accuracy.

To address the limitations of models that rely only on basic, isolated features like window titles or app names, some researchers have developed more structured approaches to understand user behaviour in context. Rath et al. (2011) and Devaurs et al. (2012) proposed a two-part framework for detecting user tasks using the User Interaction Context Ontology (UICO), which organizes user behaviour into five structured categories: action, resource, user, application, and information need. Both studies started by collecting detailed desktop interaction data using sensors at both the system and application level. These sensors recorded events such as mouse clicks, keystrokes, clipboard activity, window titles, and the names of running applications. The captured events were grouped into "event blocks" based on logical relationships and timing rules, and then further combined into larger task units. This layered structure, called the event aggregation pyramid, helped convert low-level activities into meaningful representations of user tasks (Devaurs et al., 2012). To support classification, the authors created around 50 features based on interaction patterns, content details, resource use, and how tasks changed over time. These features were tested using machine learning models including Naïve Bayes, J48 decision trees, Support Vector Machines, and k-Nearest Neighbors. Across both studies, the models performed very well, achieving classification accuracy of up to 95.5 percent when using features such as window titles, resource interactions, and visible user interface content.

In an old paper yet interesting paper, Armentano and Amandi (2007) conducted a conceptual survey of plan recognition for interface agents. They categorized methods into consistency-based (goal elimination via plan hierarchies) and probabilistic such as Bayesian networks and Abstract Hidden Markov Models, which estimate goal likelihoods under uncertainty. Although their paper did not implement or evaluate these models empirically, it offers valuable theoretical insights into how agents and Hidden Markov modelling can infer user intent from partial or noisy observations.

Among the models, the Random Forest approach by Meyer et al. (2022) stood out with 84% accuracy and 1.6-minute lead time. Dev and Liu (2017) achieved 1.0 precision in identifying task patterns via cohesion-based scoring. Pellegrin et al. (2021) outperformed standard baselines using Bayesian inference, especially for infrequent task detection. Tian et al. (2020) demonstrated the value of combined temporal and behavioural features, while Tessera (Yan et al., 2021) achieved superior task segmentation F1 scores by modelling similarity decay and event proximity.

Context Switch Detection Methods - Comparative Summary with their best results

(Excluding Armentano and Amandi (2007) as no implementation or evaluation on proposed models was found)

Table 1 Comparision of papers in context switch detection

Study	Data Source	Key Features	Methodology	Performance and Evaluation (Standout result)
Meyer et al. (2022)	‘PersonalAnalytics’ logs; 2 studies	Mouse and keyboard activity, app	Random Forest + SMOTE, change-point detection	84% accuracy; 1.6 min lead; mouse and keyboard

	(observational + sampling)	focus, 84 engineered features		features are the dominant
Pellegrin et al. (2021)	'TaskPit' desktop logs from software employees	Clicks, keystrokes, window titles, usage time	Hierarchical Bayesian estimation (frequent/infrequent task)	Achieved highest accuracy on infrequent tasks (up to 0.95); real-time inference.
Mirza et al. (2015)	Desktop logs of 5 graduate students, 1 week	Time intervals, focused apps, app sequences, window titles	Time-distance segmentation + feature-based ML classification	81% user-specific accuracy (Decision Trees); interaction best
Granitzer et al. (2009)	Synthetic and real logs from knowledge workers	Window title, semantic type, app usage, content	Block clustering + Naïve Bayes/KNN/SVM with Information Gain selection	83.5% accuracy (Naïve Bayes with active window class features)
Dev & Liu (2017)	10,000 user sessions from photo editing software	Frequent pattern sets, cohesion scores, outlier minimization	Membership-based cohesion score ranking from item sets	Top 16 patterns had 1.0 precision (Normalized Discounted Cumulative Gain - NDCG = 0.95)
Satterfield et al. (2020)	Controlled lab study; screenshot-based logs	OCR-extracted keywords from screenshots, TF-IDF, Word2Vec	Keyword vector matching (TF-IDF) for screenshot segments	70.6% TF-IDF accuracy; >80% precision on some tasks
Tian et al. (2020)	Mobile logs from 'UbiqLog' and Flurry datasets	Time gaps, TF-IDF similarity, log sequences.	Supervised boundary detection with Logistic Regression	F1 score of 0.89; scalable to large corpora
Yan et al. (2021)	Exploratory visual analytics logs (PoleStar)	Statistical, semantic, temporal query similarities	Sliding window similarity + segmentation using decay scores	Highest F1 and compression vs graph and hierarchy baselines
Rath et al. (2011)	Lab study with routine and	50 ontology-based features: actions,	Event block aggregation + supervised ML	Up to 95.5% accuracy (routine

	complex knowledge tasks	resources, apps, metadata	(Naïve Bayes, J48, SVM, KNN)	tasks); 86.4% (complex tasks)
Devaurs et al. (2012)	Desktop logs via sensors for Word, Excel, browser use	Action-resource relations, ontolo structure, accessibility dat	Ontology mapping + 50 features + multi-class ML classification	Achieve 94.85% accuracy (top 6 features); strong generalizability

2.4.3 Reflections of task/context switch detection to problem statement

These methodologies reflect growing interest in passively detecting task shifts from digital behaviour, yet few address application-type context switches from unknown activities as most of the task have already been defined by controlled data logging (users in these papers have told what to do and the start time and end time are determined). While Meyer et al. (2022) and Dev and Liu (2017) approach real-time inference, most methods either classify past sessions or require manually curated task labels (Granitzer et al., 2009; Satterfield et al., 2020). Even ontology-based approaches such as those by Rath et al. (2011) and Devaurs et al. (2012), though powerful in modelling rich semantic structure, still depend on semantics task categories, making them less suitable for scalable deployment in passive environments like BEHACOM. Plan recognition frameworks (Armentano & Amandi, 2007) offer flexible modelling of intention but assume known goal libraries, which contrasts with the unsupervised nature of this thesis. However, the classification models being used, and their results would be beneficial and interesting to test such as Random Forest by (Meyer et al. (2022), Hidden Markov Models approach by Armentano and Amandi (2007) or Hierarchical Bayesian estimation by Pellegrin et al. (2021), suggesting meaningful ways to classify tasks effectively. To address the undefined tasks, the next part of literature review will look into unsupervised learning methods to determine actions done before a context switch.

2.5 Unsupervised Machine Learning Methods on Time Series Data

Unsupervised machine learning offers practical solutions for detecting patterns and transitions in time-series behavioural data without requiring explicit instructions from a controlled logging experiment or session. In the context of this thesis, which focuses on context switch detection from unspecified-task interaction logs - BEHACOM, unsupervised segmentation and clustering techniques are essential for identifying meaningful behavioural boundaries and latent task structures. This section reviews several representative methods that contribute to this objective.

2.5.1 Sliding Window Segmentation and Temporal Clustering

Zinkovskaia et al. (2024) proposed a framework called Temporally Aligned Segmentation and Clustering (TASC) that aims to discover recurring patterns, or motifs, within continuous behavioural logs. The method begins by applying a sliding window over the input sequence to generate overlapping subsequences of behaviour. These subsequences are then embedded into a low-dimensional space using 'Uniform Manifold Approximation and Projection' (UMAP), which helps to preserve local structure and temporal relationships in a compact form. Once embedded, the segments are clustered to find behaviourally similar groups. A key contribution of TASC is its iterative refinement process: cluster assignments are aligned over time, and boundaries are adjusted by merging segments that show high similarity. This allows the model to capture recurring behaviours even if they vary slightly in form or timing. The method was evaluated using clustering quality metrics like Silhouette score or Davies-Bouldin Index, showing strong performance. Overall, TASC's emphasis on temporal alignment and clustering makes it a valuable approach for identifying recurring digital routines and optimizing cluster selection through the Silhouette score and sliding windows, particularly in human-computer interaction datasets like BEHACOM.

Similarly, Kasliwal and Katkar (2016) developed a hybrid segmentation approach that combines K-Means and DBSCAN to analyse user web sessions without supervision. K-Means is first used to generate an initial partition of the interaction logs, effectively capturing global structure based on centroid similarity. Then, DBSCAN is applied to the K-Means clusters to refine the segmentation by detecting dense regions and filtering outliers, allowing the model to handle varied workflow lengths and subtle transitions.

Their evaluation focused on metrics such as entropy and variance of user behaviour across segments, demonstrating that the method could adapt to noisy and variable-length data effectively. This approach is particularly useful in contexts like BEHACOM, where digital behaviour is often unstructured and heterogeneous, requiring models that can flexibly adjust to differing session lengths and activity intensities.

Perea and Harer (2015) introduced a method called SW1PerS that uses sliding windows, and a mathematical tool called persistent homology to find repeating patterns in time series data. The method works by turning a sequence into a shape in high-dimensional space using sliding windows, then measuring how circular that shape is. A strong circular shape means the data is likely periodic. They tested this on synthetic and biological data, like gene expression, and found it was better at handling noise than older methods. While their focus was on detecting cycles, the idea of using shape and structure to understand behaviour can also support clustering by revealing repeated patterns that may not align in time but share similar forms. This is especially useful when traditional features fail to capture subtle context switches. This sliding-window approach can be especially helpful for identifying hidden structure in behaviour leading up to a context switch, allowing clustering methods to uncover the most representative activity patterns before a transition occurs.

Truong et al. (2020) conducted a detailed benchmark of change point detection (CPD) methods, which aim to identify points in a time series where statistical properties change. This technique is well-suited to continuous interaction logs like BEHACOM, where context switches are not labelled but may show up as gradual shifts in behaviour. The study compared kernel-based methods that detect distributional changes through similarity functions, window-based approaches that evaluate differences between sliding segments, and dynamic programming techniques such as PELT, which finds optimal segmentation efficiently. They also evaluated different search strategies, including binary segmentation and bottom-up merging, using precision, recall, and F1 score to assess accuracy. This work provides valuable insights for this thesis, as it shows that CPD techniques can effectively detect behavioural transitions using only changes in data patterns, without relying on labelled tasks.

In an older paper, Liao (2005) presented a foundational survey on time series clustering methods, organizing them into three main categories, these are: raw-data-based, feature-based, and model-based approaches. Raw-data methods use distance metrics like Dynamic Time Warping (DTW) or Euclidean distance to compare sequences directly, however, it is important to note that they can be sensitive to noise and variations in time alignment. Feature-based methods convert the time series into a set of representative characteristics, such as wavelet coefficients or symbolic encodings, which can make patterns more stable and clustering more effective. Furthermore, model-based methods use statistical models like Hidden Markov Model (HMM) to describe the underlying structure of the data before clustering. The paper highlights that transforming time series into feature-rich representations often improves clustering performance, especially when the raw data is noisy or inconsistent. This insight is inspiring for this thesis, where behavioural sequences from the BEHACOM dataset could be segmented using sliding windows and used clustering for task extraction to detect context changes without predefined task.

2.5.2 Reflecting on problem statement

Together, the reviewed approaches highlight a rich landscape of unsupervised methods that can extract meaningful patterns and segment behavioural time-series data without requiring labelled tasks. This is particularly important for datasets like BEHACOM, where user sessions are unstructured and diverse, and task boundaries are not predefined. Techniques such as sliding window segmentation combined with clustering algorithms, such as K-means or DBSCAN, offer a powerful foundation for detecting context switches by converting raw activity streams into analysable subsequence and grouping them based on similarity as in (Zinkovskaia et al., 2024) and (Kasliwal & Katkar, 2016). In theory, clustering algorithms then help uncover latent behavior categories or modes by grouping similar sequences, without requiring any prior knowledge of task types or user intentions. In this thesis, methods like K-Means and hybrid clustering are useful because they scale well and adapt to different behaviours and by applying them to overlapping sliding windows, the flow of user actions would still be kept while capturing short-term patterns that may signal a context switch. Methods like TASC (Zinkovskaia et al., 2024) further inspire the temporal alignment and refinement of clusters, which is valuable for identifying gradually emerging context switches and CPD techniques (Truong et al., 2020) add another layer by statistically validating transitions through changes in distribution. Ultimately, this thesis leverages these insights to design a hybrid segmentation pipeline that integrates sliding windows, feature extraction and clustering to filter out noise regarding the minutes up to the switch of application. This approach hypothetically enables better detection of context switches in daily, unstructured human-computer interaction logs, without needing controlled user studies, or predefined task schemas.

2.6 Summary of Literature Review

This literature review explored key challenges and approaches in detecting context switches from digital behaviour data. A major issue identified was the lack of suitable datasets. Many existing datasets were either too limited in modality or collected in controlled settings. The BEHACOM dataset stood out as the best fit for this thesis, offering multi-modal, real-world interaction logs that support unsupervised analysis. The review also compared user-centric, task-centric, and goal-driven models. Task-centric methods are easier to generalize but can miss how people actually work, and user-centric approaches capture more realistic behaviour but rely heavily on manual input. Interestingly, goal-driven models offer a broader perspective but usually depend on structured goals, which this study does not assume. This thesis aims to strike a balance, using behaviour traces to infer app changes without predefined tasks. Finally, the review looked at various detection methods. Supervised models are discussed and review such as Random Forest, Naïve Bayes or Decision Tree showed strong results, and unsupervised methods, especially with sliding windows and clustering, were found suitable for this thesis. Overall, this thesis builds on those ideas by designing a hybrid pipeline using BEHACOM to detect context switches from user behaviour, without needing controlled experiments or labelled tasks, and from that detection, identify where the users would switch to.

3. Methodology

3.1 Overview

To answer the research question of whether context switches can be detected from previously one-minute aggregated and undefined behavioural tasks data, a multi-step experimental pipeline was developed using the BEHACOM dataset. Since the dataset does not contain ground-truth labels for context switches, a set of heuristic rules was first designed to approximate when a context switch occurs. These rules identify changes in the dominant application category from one minute to the next, while also considering factors such as the duration of foreground activity and the stability of user behaviour, in order to avoid wrong labelling for a switch. After generating these proxy labels, features describing user behaviour leading up to each detected switch are extracted.

For each detected switch event, the methodology described below is used to extract and summarize the behavioural context preceding the switch. The methodologies include both clustering-based pipelines, which summarize behavioural features over past sliding windows, and sequence modelling approaches, including Long Short-Term Memory (LSTM) networks and Hidden Markov Models (HMMs), which operate directly on sequential data. All developed methods are systematically compared across different users, with performance evaluated against both a naïve baseline. This comparative framework enables a comprehensive assessment of each approach's effectiveness in detecting context switches. Depending on the specific modelling approach, these pre-switch and no-switch segments are either aggregated into feature blocks or retained as sequential data.

The resulting labelled dataset is then used to train and evaluate models for context switch detection. In the case of the two-stage classification pipeline, the first stage predicts the occurrence of a context switch, while the second stage predicts the destination application category. Model performance is assessed using standard metrics such as accuracy, precision, recall, and F1-score, with feature importance analyses providing further insight into model behaviour. All evaluations are conducted relative to proxy labels generated by heuristic rules, rather than true ground truth, and this limitation is explicitly considered in the analysis. Despite these constraints, the approach enables a systematic investigation of context switch detectability in real-world behavioural data and facilitates the identification of patterns that typically precede such switches.

3.2 About BEHACOM

As discussed in the literature review, the BEHACOM dataset (Sánchez Sánchez et al., 2020) offers a comprehensive, minute-level record of real-world user interactions with desktop computers. The dataset encompasses behavioural traces from 12 users over a continuous 55-day period, resulting in a total of over 160,000 minutes of interaction data. No specific tasks were assigned to participants, ensuring that the logs reflect authentic, unconstrained digital activity in daily life, which is also one of the limitations. Each user's behaviour is characterized along four primary dimensions: keyboard activity, mouse dynamics, application usage, and resource consumption. In total, more than 12,000 features were extracted across these dimensions, including metrics such as total keystrokes, average press-release durations, digraph patterns (i.e., timing between consecutive key pairs), mouse click counts, mouse movement direction histograms, positional mouse heatmaps, foreground application names, switch counts, process statistics, CPU and memory usage per application, and network traffic in bytes. All features are computed in one-minute intervals to maintain user privacy and ensure consistent temporal granularity. These feature vectors are labelled with anonymous user identifiers (User 0 to User 11) and stored in CSV format, with one file per user. Analysis of the dataset reveals substantial variability in user activity: some users contributed over 50,000 feature vectors, while others produced fewer than 10,000.

3.3 Data Cleaning

3.3.1 Separating Sessions

The BEHACOM dataset is shared as one long log file per user that covers around fifty days of activity. Although each person's data may have originally come from multiple separate logs, the final dataset has them all joined together without any clear markers to show when one work session ends or begins. If this isn't handled properly, long breaks like lunch, errands, or even sleep might be mistaken for context switches, leading to false context switches. To prevent these errors, the timeline is divided into distinct sessions by detecting absent intervals longer than 30 minutes between consecutive records, ensuring that no context switch is mistakenly identified due to logs concatenated together. Any time such a gap is found, a new session ID is assigned. This approach confines feature extraction and switch detection to uninterrupted stretches of activity, ensuring that only genuine shifts in task context are identified while natural breaks are treated as session boundaries.

3.3.2 Data Cleaning

To prepare the BEHACOM dataset for accurate study, data cleaning is crucial to handle invalid or inconsistent values. Since this dataset includes many system-level and user-level activity features, it's important to ensure that these values make sense before using them for modelling or feature extraction.

The data overall is well-logged, no N/As have been found, however, questionable data points having negatives value have been found. These negative values should not exist as they are the logs of the number of mouse, keyboard interactions or computer resources. This issue might stem from an error in the data logging implementation of the dataset.

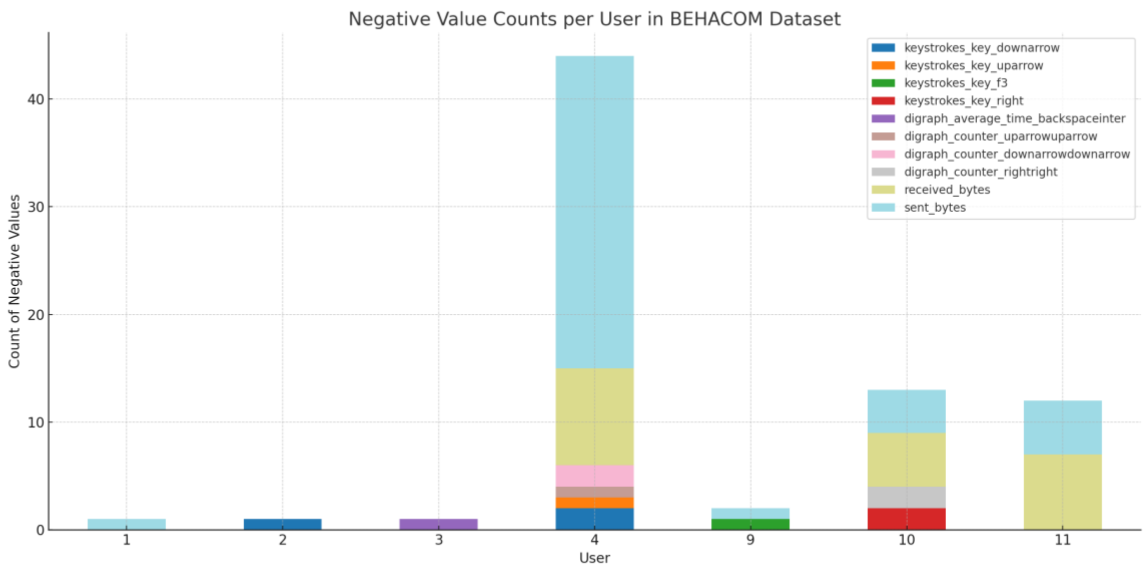


Figure 1 Negatives Value found in Behacom

For data cleaning process, negative data points regarding to count of mouse and keyboard interactions will be replaced with 0. Some negative data points regarding to “press_release_average” or “diagraph_average_time_*” (average time of key/key combination presses), will be replaced with the average of the nearest 10 data points. That means 5 minutes before and after will be looked at and averaged to replace the negative value (“sent_byte” and “receive_byte”). Altogether, these steps restore the natural non-negative constraints on counts and durations and fill in spurious dips in network traffic with a simple, robust local average, leading to a cleaner, more reliable feature matrix for modelling.

3.5 Feature Transformation and Engineering

The raw BEHACOM dataset contains 12,051 features, resulting in a high-dimensional space that poses challenges for machine learning models and classifiers. Many of these features, such as the full set of possible digraphs between key pairs, are sparsely populated and contribute little meaningful information (mostly 0). To address this, the feature engineering process focuses on compressing the data into a smaller set of interpretable metrics by aggregating related columns through summation, averaging, and ratio calculations, while retaining the essential behavioural signals required for detecting task shifts. Specifically, targeted feature engineering is applied across four core dimensions of user interaction: keyboard activity, mouse usage, system resource consumption

3.5.1 Keyboard Metrics

The keyboard-related features were engineered to reflect both the quantity and the nature of text input. Total key presses and word counts were used to derive indicators such as ‘erase_rate’, ‘words_per_key’, and ‘total_key_presses’. These features captured overall text density and editing behaviour.

Key usage was further decomposed into categories, for example: vowels, consonants, number row characters, and special characters, to identify input types associated with different task domains (e.g., writing, coding, numerical entry). Interaction-specific keys such as tab, space, and enter were also counted and contextualized by their frequency and their association with word counts. Additional features included arrow key usage, the frequency of repeated character presses, and the use of shortcut key combinations (e.g., ctrl+v, ctrl+c). To capture motor fluency and typing rhythm, timing features such as average and standard deviation of press-press and press-release intervals were included. The occurrence of certain symbols (e.g., the '@' character) and punctuation density were also incorporated to provide indicators of communication or composition-related activities.

Keyboard-related features were engineered to capture both the volume and characteristics of text input. Metrics such as total key presses, word counts, erase rate, and words per key were computed to reflect overall text density and editing behaviour. Key usage was further categorized into vowels, consonants, number row characters, and special characters, enabling the identification of input types associated with different task domains, such as writing, coding, or numerical entry. Interaction-specific keys, which include tab, space, and enter, were quantified and contextualized by their frequency and association with word counts. Additional features included arrow key usage, the frequency of repeated character presses, and the use of shortcut key combinations (e.g., ctrl+v, ctrl+c). Typing rhythm and motor fluency were assessed through timing features, such as the mean and standard deviation of press-press and press-release intervals. Additionally, indicators of communication or composition activities were incorporated by measuring the occurrence of specific symbols (e.g., '@') and punctuation density.

3.5.2 Mouse Metrics

Mouse interaction features were derived from six categories of actions, each corresponding to a specific type of mouse event as defined in the BEHACOM dataset (Sánchez Sánchez et al., 2020): 0 for left button single click, 1 for right button click, 2 for left button double click, 3 for scroll action, 4 for mouse pointer movement, 5 for drag or selection action, and 6 for middle button click. Each action type was quantified per minute, and several normalized ratios, such as click-to-scroll and drag-to-click ratios, were calculated to distinguish between browsing, editing, and interaction-heavy behaviours. Spatial interaction was assessed using histogram-based metrics that compared activity in screen edge regions versus central zones, resulting in an edge-to-center ratio that approximates the distribution of attention across the screen. Total mouse activity was compared to keyboard usage to compute the keyboard-to-mouse ratio, reflecting the dominant input modality. A combined navigation intensity score, based on scrolls and clicks, served as an indicator for overall user engagement.

3.5.3 Resource Metrics

Resource consumption features were designed to identify periods of high system load, which may be associated with context switching. CPU and memory usage were normalized against system-level baselines to produce ‘cpu_usage_ratio’ and ‘memory_usage_ratio’. Stability indices, calculated as the ratio of standard deviation to mean, were used to detect variability in resource consumption. Dynamic thresholds, based on the 95th percentile of each user’s session, flagged high CPU and memory peaks. Network activity was represented by sent and received bytes, their sum, and their ratio (network_activity_ratio), with bursts in network usage flagged as potential indicators of high-demand activities. Composite indices, such as ‘resource_demand_score’ and ‘resource_engagement_index’, were computed to reflect combined load across CPU, memory, and network dimensions. The correlation between system CPU usage and application switching was also included to assess whether resource spikes coincided with behavioural shifts.

3.5.4 Application/Context Metrics

Application-level features provided categorical and temporal context for each minute-level sample. Application names were mapped to broader categories (e.g., Web, Office, Development), enabling the extraction of current and penultimate application categories to represent the user’s current and prior task contexts. Key behavioral indicators included ‘changes_between_apps’, capturing the frequency of application switches within a given window, and ‘current_app_foreground_time’, measuring the duration a single application remained active. The ‘app_transition_freq’ metric was derived to indicate session stability or fragmentation. Additional system and application resource metrics were retained to enrich context understanding, supporting the distinction between transient exploratory behaviours and sustained task engagement.

3.5.5 Application Category Mapping

Since the preferences and usage patterns of applications could vary significantly between users, a mapping has been introduced to group individual application executables into broader, semantically meaningful categories. This mapping assigns each application name or process (such as chrome.exe, spotify.exe, or notepad.exe) to a higher-level category like Web, Development, Communication, Office, or Music. The category naming aims to be generalized yet keeping the basics of the purpose of that application, the categories included: Web, Development, Gaming, Communication, Network, Cloud, Video, Office, Reading, File Tools, Calculator, System, Downloads, Graphics, Finance Academic and Music.

One limitation of this approach is that if a user switches from Chrome to Firefox, both of which are mapped to the "Web" category, the context switch will not be captured, as the mapping treats both applications as belonging to the same activity type. However, this consolidation helps to reduce fragmentation in the data and results in a more balanced set of categories for model training and the analysis can focus on user behaviour at the activity or context level. By merging similar applications into broader categories, the model benefits from having more samples per class, which can improve training stability and overall performance. This trade-off prioritizes general patterns of user behaviour over fine-grained distinctions between individual applications.

Overall, after the feature engineering part, the data now consists of 77 features and is ready for experiments.

3.6 Setting Up the Experiments

3.6.1 Defining context switch

In BEHACOM, the temporal aggregation introduces complexity, as the "current app" recorded for each minute may not always correspond to the true application in use, particularly if multiple switches occur within a single minute or only switch at the latest seconds of that one-minute interval. The 'define_context_switches' function is designed to identify correct app switches within the BEHACOM dataset. A confidence-based approach is employed to determine whether a switch has occurred, taking into account three main factors: the proportion of the minute spent in the current app (foreground time factor), the number of changes between apps within the minute (changes factor), and the stability of app usage as indicated by whether the penultimate app category matches the previous app category (stability factor).

A confidence score is calculated for each potential switch by adding the three factors, and a switch is confirmed and not be changed if this score exceeds a predefined threshold (in this thesis 0.5), and the foreground time meets a minimum requirement (20 seconds). The penultimate app of that minute will replace the current app if confidence threshold is below 0.5. Due to the one-minute aggregation, even after the function, the final "current app" may not always provide a fully accurate reflection of user activity, especially in cases of rapid or frequent switching. Nevertheless, the function may not capture all scenarios, particularly those involving, rapid switching between several applications within a minute, but such cases are relatively uncommon, and the function is designed to effectively handle the majority of typical user behaviours.

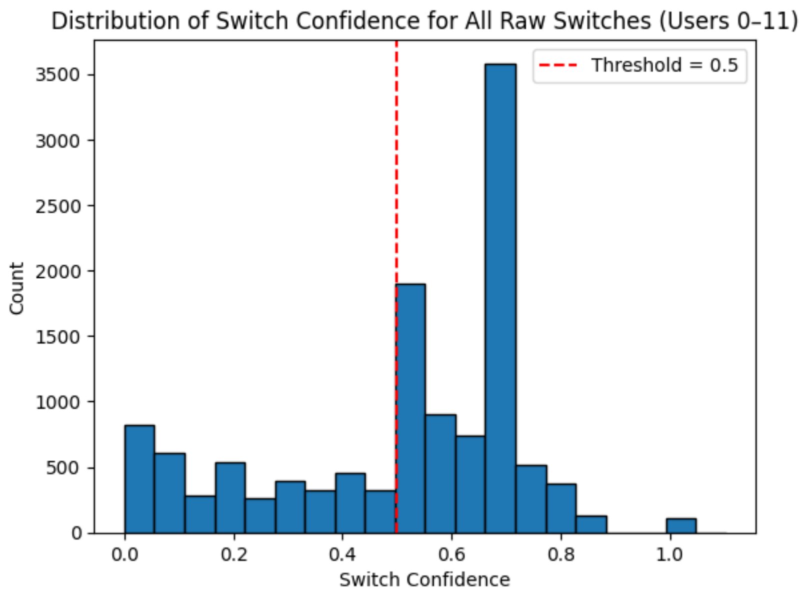


Figure 2 Switch confidence threshold for all user

As shown in the graph, most switches do not need to be redefined. For the evaluation of results section, only users whose majority of switches are above the threshold will be evaluated. This is done to avoid incorrect destination labels, since the destination application does not need to be changed in these cases. The users who meet this condition are User 1, 3, 7, 10, and 11.

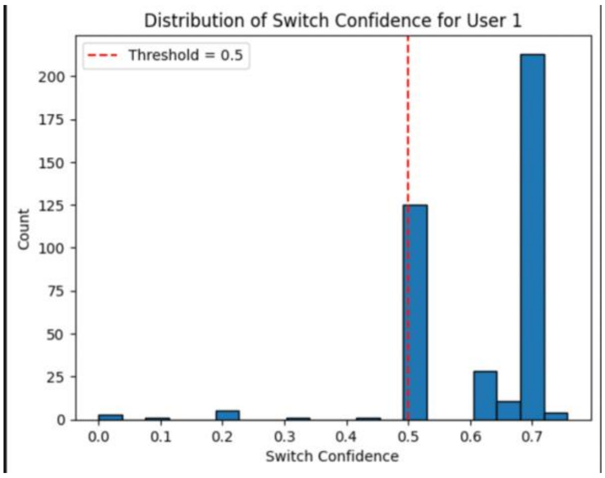
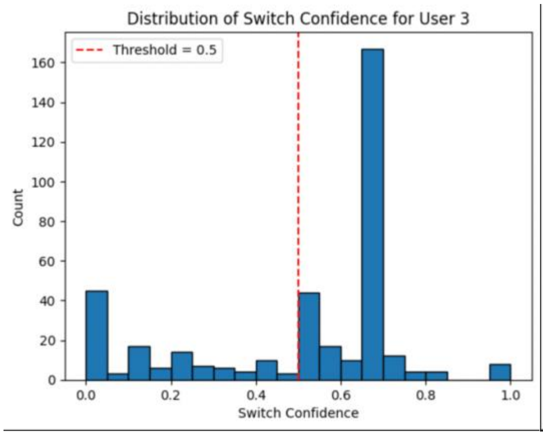
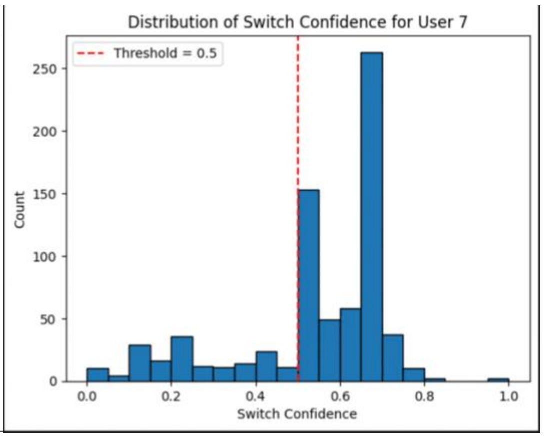
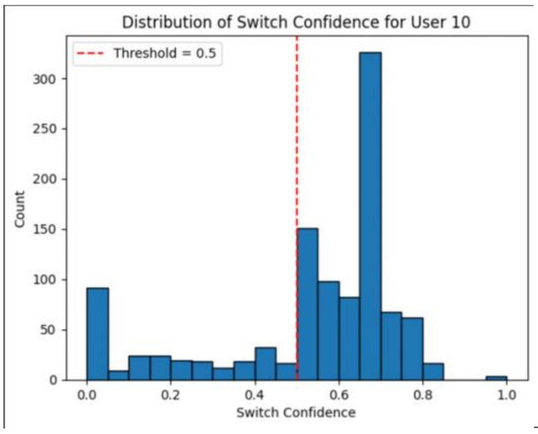
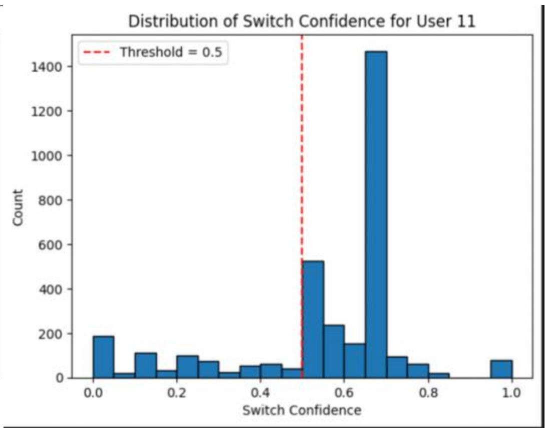


Figure 3 Switch confidence threshold for User 1,3,7,10,11









3.6.2 Entropy for Application:

Entropy is used for representing active applications and is calculated using the Shannon entropy formula (Shannon,1948):

$$H = -\sum_i p_i \log_2(p_i)$$

where p_i represents the probability of observing app category *i* in the block, measured as the proportion of minutes spent in each app category.

This entropy metric quantifies the diversity of app categories used in the past block of time. The calculation process involves counting the frequency of each app category, converting these counts to probabilities through normalization, and applying the entropy formula to these probabilities. Entropy serves as a measure of the diversity or unpredictability of app usage within a look-back block, where a low entropy value indicates highly focused usage in a single app category, while a high entropy value suggests diverse usage across multiple app categories. To prevent bias in context switch detection, the current app, at the time of context switch, is excluded from the entropy calculation to avoid label data leakage, ensuring that the model's detection and identification are based solely on historical patterns rather than obvious current state information. This entropy calculation replaces the need for categorical variable in the models and is be used for all approaches.

3.7 Naïve Approach

The Naïve approach is implemented to detect context switches and extract relevant data blocks from the BEHACOM dataset, serving as a baseline for comparison with more advanced methodologies. Operating on one-minute aggregated application usage data, this method employs a straightforward procedure for both switch detection and block extraction.

Block Extraction:

For both detected switches and periods without a switch, blocks of data are extracted by looking back over a fixed window of minutes (in this study – 9 minutes). Each block is summarized into a single record by calculating the mean of numeric features and the entropy of app categories, capturing the behavioural context within the window. For NoSwitch blocks, an additional condition is applied: only blocks where the last 2 minutes prior current app is spent in the same application category, are retained, ensuring that these negative examples represent sustained, uninterrupted activity. The resulted data is then passed to the chosen classifiers to evaluate on the performance of detecting context switches and the identifying the destination app.

By providing a simple and transparent baseline, the naive approach highlights the challenges of working with aggregated behavioural data and establishes a reference point for evaluating more advanced methods. The resulting proxy labels and summarized blocks enable systematic comparison of different context switch detection strategies.

3.8 Clustering Based Modelling

3.8.1 Proximity-Based Contextual Clustering (PCC)

This approach builds upon the methodologies and assumptions previously defined, with a special focus on the look-back range when encountering a switch. Context switches are identified using the ‘define_context_switches’ function, as detailed in Section 3.6.1. As described earlier, this algorithm computes a confidence score for each potential switch based on foreground time, changes between apps, and stability factors, aiming to accurately define the correct application switch when switch occurs.

For NoSwitch blocks (periods without context switches), the algorithm scans for runs of at least two consecutive minutes within the same application category. Blocks of up to the same length as positive blocks are extracted (like the Naïve approach, this look-back length is 9 minutes for the study), ensuring NoSwitch block's stability while still capturing what happens before the stability.

For each detected context switch, a block of behavioural data preceding the switch is extracted using a proximity-based clustering approach. Specifically, up to a fixed horizon of 18 minutes prior to the switch (twice the minimum required block length of 9 minutes) is considered. This interval is divided into overlapping sliding windows (3-minute windows with a 1-minute stride). Each window is summarized by averaging numeric features and taking the mode of categorical features (penultimate app and current app), with the current app at the minute of switching removed to avoid label leakage. The resulting window representations are then encoded, scaled, and reduced in dimensionality using PCA. To determine the optimal number of clusters, K-Means clustering is applied across a dynamic range of k values, and the best k is selected based on the silhouette score.

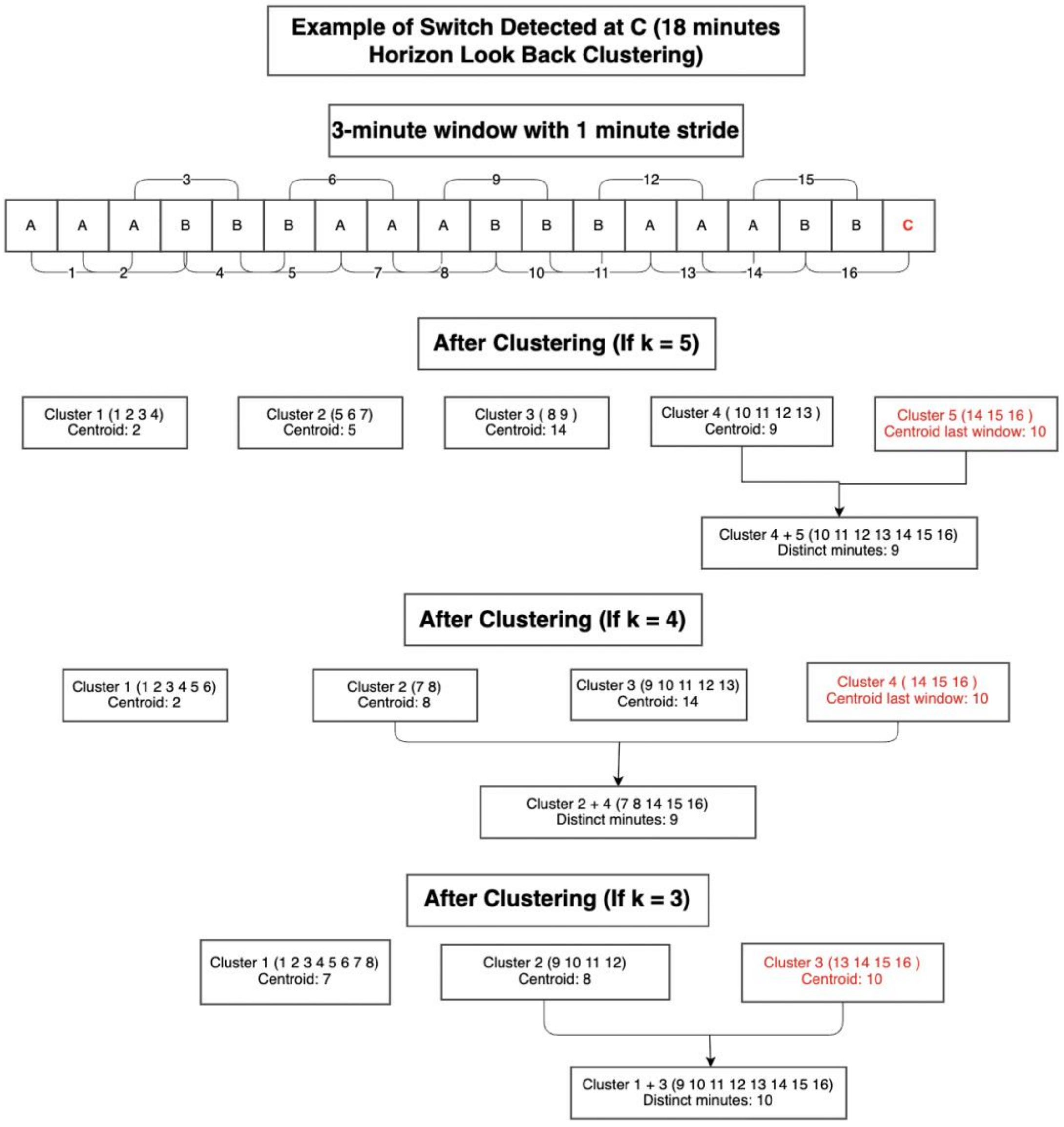

Example of Switch Detected at C (18 minutes Horizon Look Back Clustering)
3-minute window with 1 minute stride
A A A B B B A A A B B B A A A B B C
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
After Clustering (If k = 5)
Cluster 1 (1 2 3 4) Centroid: 2
Cluster 2 (5 6 7) Centroid: 5
Cluster 3 (8 9) Centroid: 14
Cluster 4 (10 11 12 13) Centroid: 9
Cluster 5 (14 15 16) Centroid last window: 10
Cluster 4 + 5 (10 11 12 13 14 15 16) Distinct minutes: 9
After Clustering (If k = 4)
Cluster 1 (1 2 3 4 5 6) Centroid: 2
Cluster 2 (7 8) Centroid: 8
Cluster 3 (9 10 11 12 13) Centroid: 14
Cluster 4 (14 15 16) Centroid last window: 10
Cluster 2 + 4 (7 8 14 15 16) Distinct minutes: 9
After Clustering (If k = 3)
Cluster 1 (1 2 3 4 5 6 7 8) Centroid: 7
Cluster 2 (9 10 11 12) Centroid: 8
Cluster 3 (13 14 15 16) Centroid: 10
Cluster 1 + 3 (9 10 11 12 13 14 15 16) Distinct minutes: 10

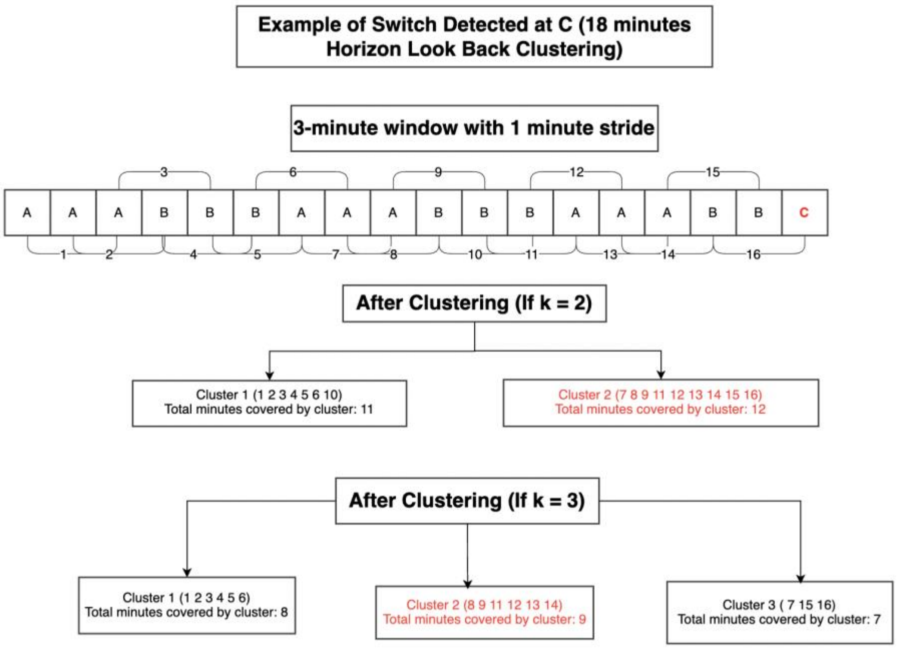
Figure 4 PCC Clustering graph

After clustering, the cluster whose centroid is closest to the most recent window (immediately before the switch) is identified as the main cluster. Windows from this main cluster are greedily aggregated using mean, prioritizing those most similar to the last window centroid, until the minimum required block length (9 unique minutes) is reached. If the main cluster alone does not provide enough unique minutes, additional windows are added from other clusters, selected in order of their centroid's proximity to the last window centroid, again prioritizing windows most similar to the last window centroid. This process continues until the block contains at least the required number of unique minutes (at least 9 minutes). This approach, along with the use of mean values for numerical features, ensures that the constructed block captures the most relevant behavioural context leading up to the switch, without introducing bias with respect to the temporal distribution of the look-back period.

To ensure the integrity of the model training process, it is crucial to avoid label data leakage, which occurs when information that should not be available at detection time is inadvertently included in the training data. In the context of context switch detection, this means excluding certain metrics from the current minute when constructing the feature block. These include average CPU and memory usage, app changes, current application CPU and memory. Finally, features for the block are summarized, including the entropy of app categories as a measure of behavioural diversity. This approach will answer the question of whether the specifically chosen minutes before a switch influences the result.

3.8.2 Majority-Based Contextual Clustering (MCC)

This approach builds on the methodologies and assumptions described above but introduces a different strategy for selecting the most representative behavioural context prior to a detected switch. As with the proximity-based method (PCC), context switches are identified using the define_context_switches algorithm (see Section 3.6.1), and NoSwitch blocks are extracted using runs of at least two consecutive minutes within the same application category, with a look-back length of 9 minutes for stability.



Example of Switch Detected at C (18 minutes Horizon Look Back Clustering)
3-minute window with 1 minute stride
A
A
A
B
B
B
A
A
A
B
B
B
A
A
A
B
B
C
After Clustering (If k = 2)
Cluster 1 (1 2 3 4 5 6 10) Total minutes covered by cluster: 11
Cluster 2 (7 8 9 11 12 13 14 15 16) Total minutes covered by cluster: 12
After Clustering (If k = 3)
Cluster 1 (1 2 3 4 5 6) Total minutes covered by cluster: 8
Cluster 2 (8 9 11 12 13 14) Total minutes covered by cluster: 9
Cluster 3 (7 15 16) Total minutes covered by cluster: 7

Figure 5 MCC clustering graph

The majority-based clustering (MCC) approach also employs 3-minute sliding windows with 1-minute stride but differs from PCC in its window selection strategy. Rather than using proximity to the last window centroid, MCC identifies the cluster with the most unique minutes. MCC deals with sliding windows like PCC by summarizing each window through averaging numeric features and taking the mode of categorical features, then encodes, scales and finally PCA (dimension reduction) for clustering. After applying K-Means clustering to a smaller and defined range (k = [2,3]) and choosing the better k value based on silhouette score, the algorithm identifies the majority cluster by counting the unique minutes each cluster covers. As the result, the cluster with the most unique minutes is selected as the majority cluster. The algorithm calculates the coverage of the selected majority cluster and uses all the distinct minutes from that cluster to aggregate by means to create a block, making up that Switch. Similar to the previous approach, certain metrics from the current minute are excluded to avoid data leakage, which includes memory usage, app changes, current application CPU and memory at the minute of detection. Finally, the resulted data then is passed for classification. This approach will answer the question of whether the specifically chosen look-back length before a switch influences the result.

3.9 Sequential Modelling

3.9.1 LSTM-based Contextual Sequence Modelling

Building upon the methodologies previously described, a Long Short-Term Memory (LSTM) network is employed to detect context switches and predict the destination application category. LSTM is a form of recurrent neural network that uses gated memory cells to preserve error signals over long sequences and so can learn long-range dependencies across hundreds or even thousands of time steps, making it especially well-suited to tasks like sequence prediction and time-series analysis (Hochreiter and Schmidhuber, 1997). This approach leverages the sequential nature of the behavioural data to capture temporal dependencies and patterns that precede a context switch.

Data Preparation and Feature Extraction

The process begins by extracting numeric features from the behavioural data for each user session. Features that could cause data leakage, such as memory usage, CPU usage, and app transition frequency at the minute of detection, are explicitly excluded. The data is segmented into sequences of fixed length of 9 minutes, with each sequence representing a window of user behaviour leading up to a potential switch. For each time step in the sequence, the entropy of app categories over the preceding 9 minutes before the switch is calculated (including the current minute) and added as an additional feature, providing the model with information about how varied or focused the user's activity was in the recent past. All features are standardized using a StandardScaler to ensure consistent normalization, which is important for neural network training.

Model Architecture

The LSTM-based methodology employs two distinct neural network architectures, each tailored to a specific stage of the prediction pipeline: switch detection and destination prediction. For the switch detection stage, the model consists of a single-layer, one-directional LSTM network. The input to this model is a sequence of feature vectors, each representing one minute in the look-back window. The LSTM processes the sequence and outputs a hidden state summarizing the temporal dynamics of the user's recent behaviour. This hidden state is then passed through a fully connected (linear) layer, followed by a sigmoid activation function, to produce a probability indicating whether a context switch will occur in the next minute.

In this research, the modelling of LSTM was constructed as follows. Given an input sequence of feature vectors:

$$X = [x_1, x_2, \ldots, x_T]$$

Where each $x_t \in R^d$, with d features per time step, and T = SEQ_LEN=9

The LSTM processes the sequence step by step:

$$(h_t, c_t) = LSTM(x_t, h_{t-1}, c_{t-1})$$

Where h_t is the hidden state (short term at step t) and c_t is the cell of that time t (long term at step t).

After the last time step (t = T), the final hidden state h_T is passed through a fully connected layer and a sigmoid activation:

$$p = \sigma(w^T h_T + b)$$

where w and b are the weights and bias of the output layer, and σ is the sigmoid function.

Finally, the prediction: If $p \geq 0.5$, predict Switch, else NoSwitch.

For the destination prediction stage, a more complex architecture is used. This model features a two-layer, bi-directional LSTM, which allows the network to capture dependencies in both forward and backward directions within the sequence. Bidirectional LSTMs offer an elegant way to capture both past and future context in sequence data, by running two parallel LSTM layers over the same input, one in the forward direction and one in reverse, then concatenating their hidden states at each time step (Ma & Hovy,2016). By processing the input in both temporal directions, the bi-directional LSTM can leverage a more complete context, which is particularly beneficial for accurately predicting the destination application category. The output from the final hidden states of both directions is concatenated and passed through a series of fully connected layers with ReLU activations and dropout regularization (with a dropout rate of 0.3) to prevent overfitting. The final output layer uses a SoftMax activation to produce a probability distribution over the possible destination application categories. This design enables the model to learn richer representations of user behaviour, utilize temporal context more effectively, and generalize better to unseen data.

Given same input sequence type like Switch LSTM – X, T = 9 (SEQ_LEN):

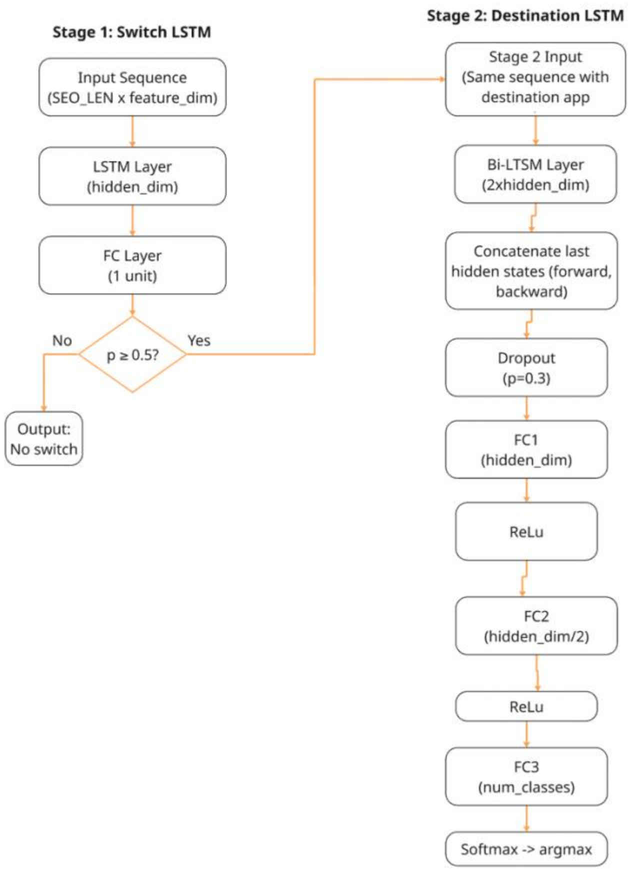
A bi-directional LSTM processes the sequence in both directions (from start to end and from end to start):

- Forward LSTM: $\overrightarrow{h_t}$ (from *t*=1 to *T*)
- Backward LSTM: $\overleftarrow{h_t}$ (from *t*=T to *1*)

After that, the final hidden states are concatenated from both directions (h_{final}). This concatenated vector is passed through a series of fully connected layers with ReLU activations and dropout. Finally, the SoftMax function is applied to get class probabilities and make prediction on the destination app:

$$z_1 = \text{Dropout}(h_{\text{final}})$$
$$z_2 = \text{ReLU}(W_1\, z_1 + b_1)$$
$$z_3 = \text{Dropout}(z_2)$$
$$z_4 = \text{ReLU}(W_2\, z_3 + b_2)$$
$$o = W_3\, z_4 + b_3$$
$$p = \text{softmax}(o)$$

For context: FC = Fully Connected Layer = z, W_x= matrix of learnable parameters (weights) and b = bias vectors



Stage 1: Switch LSTM
Input Sequence (SEQ_LEN x feature_dim)
LSTM Layer (hidden_dim)
FC Layer (1 unit)
p ≥ 0.5?
No
Yes
Output: No switch
Stage 2: Destination LSTM
Stage 2 Input (Same sequence with destination app
Bi-LTSM Layer (2xhidden_dim)
Concatenate last hidden states (forward, backward)
Dropout (p=0.3)
FC1 (hidden_dim)
ReLu
FC2 (hidden_dim/2)
ReLu
FC3 (num_classes)
Softmax -> argmax

Figure 6 LSTM modelling

Training Process

The training process is carefully structured to ensure robust model performance and to prevent overfitting. The dataset is first split into training, validation, and test sets using stratified sampling to maintain class balance, especially important for the typically imbalanced nature of switch/no-switch events. For the switch detection model, the binary cross-entropy loss function is used, as the task is a binary classification problem. The model is trained using the Adam optimizer, provided from Pytorch library, which adaptively update this neural network's weights during training, helping the model learn from data by minimizing the loss function. Moreover, during training, the model processes mini-batches of data (16 in this research), and after each epoch, its performance is evaluated on the validation set. Early stopping is implemented: if the validation loss does not improve for a set number of epochs (15 in this thesis), training is halted to avoid overfitting.

For the destination prediction model, the training procedure closely follows that of the switch detection model, but with adaptations for multi-class classification. Cross-entropy loss with class weights is used to address class imbalance, and the AdamW optimizer, also from the PyTorch library, introduces weight decay for additional regularization, which penalizes large weights in the model by shrinking them slightly during each update, discourages the model from relying too heavily on any single feature, and helps to prevent overfitting and improving the model's ability to generalize to unseen data. A learning rate scheduler and gradient clipping are applied to further stabilize training, and dropout layers are included in the model architecture (like Switch LSTM) to help prevent overfitting. Finally, early stopping is again used, with training halted if validation loss does not improve for 15 epochs.

3.9.2 HMM-Augmented Sliding Window Feature Extraction

Building upon the methodologies previously described, a Hidden Markov Model (HMM) is employed to detect context switches and predict the destination application category. A Hidden Markov Model consists of an unobserved finite-state Markov process whose hidden state at each time step produces an observation drawn from a state-specific probability distribution (Rabiner, 1989). The HMM approach offers a probabilistic perspective on context switch detection, leveraging the sequential nature of behavioural data to identify underlying states that may indicate a switch. This approach is distinct from the clustering and LSTM methods, as it models the data as a sequence of hidden states, each associated with a probability distribution over observable feature.

Preparation and Feature Extraction:

Like LSTM and other approaches, the process begins by labelling context switches in the dataset using the 'define_context_switches' function. Numeric features are extracted from the behavioural data, with care taken to exclude any features that could cause data leakage, such as memory or CPU usage and app transition frequency at the minute of detection (t-0). Similar to the methods above, the data is then organized into sequences of 9 minutes, representing windows of behaviour leading up to potential switches and the entropy of app categories, excluding the current app minute (t-0), is calculated to quantify the diversity of app usage.

HMM Model Training and State Assignment

A Gaussian HMM (from Python 'hmmlearn' library) is defined with three hidden states (n=3), reflecting the aims of representing states where user is in switching state, focus state or mixed state, and also is a common and interpretable choice for capturing distinct behavioural patterns without overcomplicating the model. The HMM used from the 'hmmlearn' Python library, is trained on the scaled feature data using the Expectation-Maximization algorithm, which iteratively adjusts the model parameters to maximize the likelihood of the observed sequences. After training, the HMM assigns each minute in the dataset to one of the three hidden states, based on the learned statistical properties of each state. These state assignments are then added to the dataset as a new feature. The resulting dataset is then passed into classifiers and continue with the cascaded detection of switches.

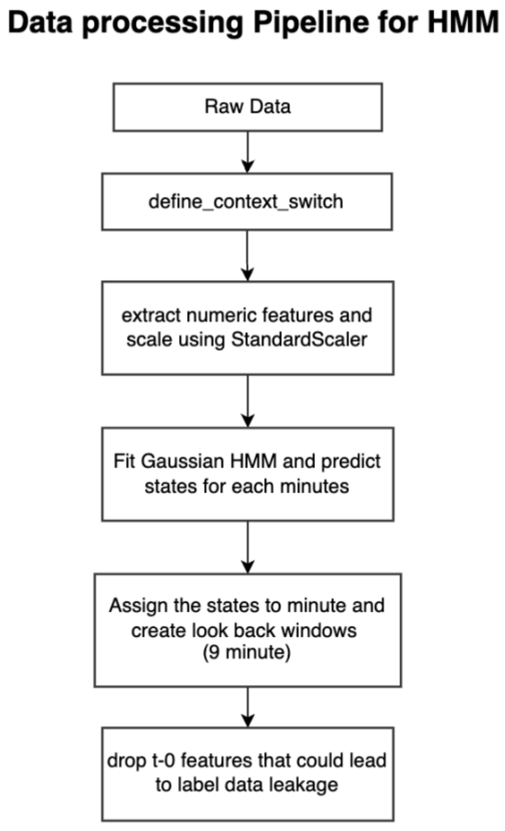

Data processing Pipeline for HMM
Raw Data
define_context_switch
extract numeric features and scale using StandardScaler
Fit Gaussian HMM and predict states for each minutes
Assign the states to minute and create look back windows (9 minute)
drop t-0 features that could lead to label data leakage

Figure 7 HMM graph

3.10 Cascaded Pipeline and Choice of Supervised Classifiers

3.10.1 Classifiers choice

This section looks into the choice of classifier for the Naïve, MCC, PCC, HMM approaches. The LSTM model presented in Section 3.9.1, inherently performs sequence classification by transforming input sequences into class predictions within its own network structure, eliminating the need for a separate classifier.
The literature review suggests that J48 decision trees (Rath et al., 2011) ,Random Forest (Meyer et al, 2022) or other tree-based classifications might be potential for contexts switch detections through high results. Hence, this research will look into the choices of tree-based classifiers for the approaches listed in Methodology. Random Forest, Gradient Boosting and Extra Trees are the chosen classifiers.

Random Forests build an ensemble of decision trees by training each tree on a randomly sampled subset of the data and features, then aggregating their votes for classification (Breiman, 2001). As the number of trees grows, the generalization error converges to a stable limit determined by the average strength of the individual trees and their mutual

correlations (Breiman, 2001). By selecting a random subset of features at each split, Random Forests achieve accuracy and offer greater robustness to noise.
The Extra Trees method builds an ensemble of decision trees by randomly selecting both which features to split on and where to split, rather than optimizing these choices (Geurts et al. ,2006). By tuning a single randomness parameter, it strikes a balance between bias and variance, delivering fast, robust performance without sacrificing accuracy (Geurts et al. ,2006). Unlike Random Forests, which search for the best split at each node, Extra-Trees picks cut points completely at random, leading to faster training and greater diversity among trees.
Finally, Gradient Boosting formulates model fitting as an optimization in function space, building an additive series of learners by performing gradient-descent steps on a chosen loss at each iteration (Friedman, 2001). This general framework accommodates various loss functions, such as least-squares or least absolute deviation, and specializes naturally to "TreeBoost" when those learners are regression trees (Friedman, 2001). The resulting tree-based gradient boosting models are both highly robust to noisy data and relatively interpretable, making them a powerful choice for real-world predictive tasks.

3.10.2 Cascaded Pipeline

This research proposes a cascaded classifier pipeline, or a two-stage classification system, that first detects whether a switch occurs, if the classifier predict True and it is truly a switch, the results are then passed to a different classifier (same type as the first) to identify the destination app. To ensure robust and fair evaluation, the dataset is split into training (70%), validation (15%), and testing (15%) sets. Stratification is used during splitting to preserve the proportion of key classes (such as Switch/NoSwitch or destination categories) across all subsets. This is crucial for imbalanced datasets, as it prevents the training, validation, or test sets from being dominated by one class, which could bias the model or evaluation. Also, unknown destination classes (represented as '-' in the raw dataset – no app is presented on the screen), are removed before classification.

For Naïve, MCC, PCC and HMM approaches:

In the cascaded classifier pipeline, Stage 1 focuses on Switch/NoSwitch classification, where the model predicts whether a context switch will occur. The input to this stage consists of a set of carefully engineered features capturing application usage, system resource consumption, keyboard activity, and mouse behaviour. The output is a binary label indicating either a Switch or NoSwitch event. In Stage 2, for those instances predicted as Switch in the first stage, a second classifier is applied to predict the specific destination application category. This stage uses the same family of models as Stage 1, such as Random Forest, Gradient Boosting or Extra Trees, and relies on a similar feature set, but only considers cases where a switch is detected. The output of Stage 2 is a multi-class label corresponding to the predicted destination category, enabling a more granular understanding of user context transitions.

For LSTM approach:

The LSTM (Long Short-Term Memory) approach, as described in 3.9.1, leverages deep learning to model sequential data, making it particularly well-suited for capturing temporal patterns in user behaviour. In this framework, the input consists of sequences of features, constructed as look back windows, that provide the model with context over time. In Stage 1, the LSTM processes these sequences to directly predict whether a context switch (Switch/NoSwitch) will occur, effectively learning from the temporal dependencies present in the data. For those instances where a switch is predicted, Stage 2 employs either a second LSTM or a dense layer to predict the specific destination application category. One of the key advantages of the LSTM approach is its ability to automatically learn complex feature interactions and temporal relationships, significantly reducing the reliance on manual feature engineering.
The results of both pipelines are the classification report which shows accuracy, precision, recall and F1 for the two stages. The next section will look at the results produced by all approaches across chosen users.

4. Results

The results section consists of four parts: Comparison of methodologies between multiple users, evaluation of methods' robustness to noise, feature importance analysis and comparison of classifiers. The following table shows the total minutes blocks of NoSwitch and Switch for each user for a better understanding of class imbalances. To recap, sessions are separated by an inactivity time (logged data gap of 30 minutes).

Table 2 Users data overview

User	Total Sessions	Switch	No Switch
User 1	3	388	41606
User 3	41	314	2521
User 7	6	730	54575
User 10	86	944	13381
User 11	173	1022	11611

It can be seen from the table that the skewness appears in all chosen users, they all rarely switch. The least data is User 3, while User 1 and User 7 have the most data, they have the most skewed data (Switch vs NoSwitch). User 10 and User 11 are the most balanced one, with user 10 will be chosen as the main user for evaluating results as user 10 have less sessions, meaning less variability.

4.1 Comparison of Methodologies between Multiple Users

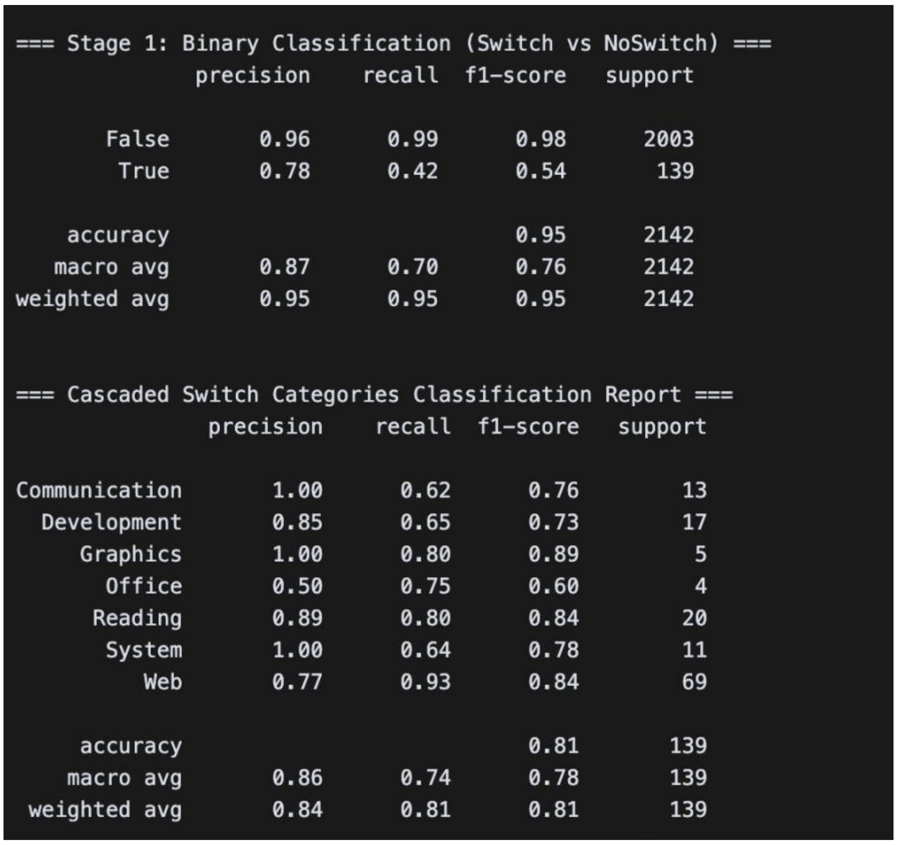
Since the classes are highly imbalance, accuracy may not accurately assess the performance of the methodologies, with the aim to detect context switch. Hence, only precision and recall are assessed.
Powers (2020) defined precision and recall as:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \quad , \quad \text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

S1 refer to Stage 1 of detecting switch or no switch and S2 refer to Stage 2 of identifying the destination app. NS in Stage 1 refers to NoSwitch and S refers to Switch. The classifier used for Naïve, PCC, MCC and HMM is Gradient Boosting.

The below tables of precision and recall are the summarized data from the output of each user in each method. Here is an example table, for user 10, the method in the picture is MCC.

Table 3 Example of output



```
=== Stage 1: Binary Classification (Switch vs NoSwitch) ===
              precision    recall  f1-score   support

       False       0.96      0.99      0.98      2003
        True       0.78      0.42      0.54       139

    accuracy                           0.95      2142
   macro avg       0.87      0.70      0.76      2142
weighted avg       0.95      0.95      0.95      2142


=== Cascaded Switch Categories Classification Report ===
               precision    recall  f1-score   support

Communication       1.00      0.62      0.76        13
  Development       0.85      0.65      0.73        17
     Graphics       1.00      0.80      0.89         5
       Office       0.50      0.75      0.60         4
      Reading       0.89      0.80      0.84        20
       System       1.00      0.64      0.78        11
          Web       0.77      0.93      0.84        69

     accuracy                           0.81       139
    macro avg       0.86      0.74      0.78       139
 weighted avg       0.84      0.81      0.81       139
```

4.1.1 Precision:

Table 4 Precision of User 1,3,7,10,11

		Naive	PCC	MCC	LSTM	HMM
User 1	S1	NS: 0.99 S: 0.2	NS: 1 S: 0.45	NS:0.99 S: 0.47	NS:1 **S: 0.87**	NS:0.99 S: 0.68
	S2	0.89	0.95	0.83	0.53	**0.97**
User 3	S1	NS: 0.94 S: 0.54	NS: 0.97 S: 0.62	NS: 0.94 S: 0.59	NS: 0.98 **S:0.88**	NS: 0.94 S: 0.73
	S2	0.79	0.78	0.71	0.37	**0.83**
User 7	S1	NS: 0.99 S: 0.42	NS: 0.99 S: 0.66	NS: 1 S:0.82	NS: 1 **S: 0.90**	NS: 0.99 S: 0.75

	S2	0.86	0.85	0.67	0.4	**0.90**
User 10	S1	NS: 0.95 S: 0.44	NS: 0.95 S: 0.81	NS: 0.96 S: 0.78	NS: 0.98 **S:0.85**	NS: 0.96 **S:0.85**
	S2	0.88	0.82	0.84	0.39	**0.92**
User 11	S1	NS: 0.85 S: 0.51	NS: 0.88 S: 0.8	NS: 0.9 **S: 0.85**	NS:0.93 **S: 0.84**	NS: 0.9 S:0.82
	S2	0.82	0.84	0.7	0.36	**0.89**

General Observations

- NoSwitch (NS) precision is consistently high (often >0.9) across all methods and users, reflecting the class imbalance as most blocks are no-switch.
- Switch (S) precision varies much more and is the key differentiator between methods and users.
- LSTM and HMM generally achieve the highest S precision and more stable throughout the datasets, especially for users with more balanced or richer data.
- PCC and MCC offer a substantial improvement over Naive, especially for S precision, but their gains depend on user data characteristics.

User-Specific Insights

Stage 1: Switch or NoSwitch?

For User 1 and User 7, who is highly focused and almost never switches, NS precision is perfect or near perfect for all methods, Switch (S) precision is very low for Naive (0.2/0.42) but improves with PCC (0.45/0.66) or MCC (0.47/0.82) and is highest for LSTM (0.87 or 0.9) and HMM (0.68 or 0.75). The extreme class imbalance makes S precision hard to improve for simple models. LSTM, with its ability to learn subtle patterns, achieves a dramatic gain in S precision. HMM also improves S precision, but not as much as LSTM. PCC/MCC provide moderate gains, showing the context switch capability even in imbalanced settings.

User 3 who has least data and is also heavily on NoSwitch, NS precision remains high for all. S precision is moderate for Naive (0.54), improves for PCC/MCC (0.62/0.59), and is best for LSTM (0.87) and HMM (0.73). Even with little data, LSTM and HMM outperform context-based and naive methods for S precision. PCC/MCC still provide a boost over Naive, but the performance is worse than for users with more data.

User 10 and User 11 have the most moderate data, with User 10 being more stable since having less sessions, NS precision is also high for all. With more balanced data, all methods perform better for Switch (S) precision. LSTM (0.85/0.84) achieves the highest Switch precision, showing its strength with balanced and sufficient data. PCC (0.81/0.8) and MCC (0.78/0.85) are also strong, indicating that context is highly informative for these users. Like others, HMM performs well for user 10 and 11 with 0.85 and 0.82 precision.

Stage 2: *Identifying the destination?*
This second stage is highly influenced by the first part since the classifiers are cascaded, which means that only true predicted switch from stage 1 will be evaluated by stage 2. One noticeable in performance is LSTM, which performs badly, even with regards to more corrected switch predictions being made, the results are still poor across the users and could be due to overfitting of the model. HMM on the other side shows high precision suggesting the hidden states are efficient in determining the destination app. Naïve approach shows good precision at stage 2, however it is mainly due to its poor ability to detect the true predicted switch. PCC and MCC approaches show acceptable destination identification, with PCC has better precision metric than MCC.

4.1.2 Recall:

Table 5 Recall of User 1,3,7,10,11

		Naive	PCC	MCC	LSTM	HMM
User 1	S1	NS:1 <u>S:0.11</u>	NS:1 <u>S:0.17</u>	NS: 1 <u>S: 0.35</u>	NS:1 <u>**S:0.72**</u>	NS:1 <u>S:0.42</u>
	S2	0.9	0.94	0.81	0.15	**0.96**
User 3	S1	NS: 0.96 <u>S: 0.44</u>	NS: 0.97 <u>S: 0.47</u>	NS: 0.96 <u>S: 0.47</u>	NS:0.99 <u>**S:0.79**</u>	NS:0.98 <u>S: 0.53</u>
	S2	0.79	0.77	0.7	0.23	**0.79**

User 7	S1	NS: 0.99 S: 0.32	NS: 1 S: 0.33	NS: 1 S: 0.62	NS: 1 **S: 0.7**	NS:1 S: 0.65
	S2	0.86	0.8	0.62	0.22	**0.88**
User 10	S1	NS: 0.98 S: 0.22	NS: 1 S: 0.31	NS: 0.99 S: 0.42	NS: 0.99 **S:0.76**	NS: 0.99 S:0.45
	S2	0.87	0.82	0.81	0.18	**0.9**
User 11	S1	NS: 0.93 S: 0.30	NS: 0.97 S: 0.46	NS: 0.98 S: 0.7	NS:0.97 **S: 0.72**	NS: 0.96 S: 0.61
	S2	0.75	0.72	0,61	0.24	**0.9**

General Observations

- Like precision, NoSwitch (NS) recall is extremely high (often 0.96–1.0) for all methods and users, reflecting the class imbalance.
- Switch (S) recall is much lower and varies significantly between methods and users. This is the key metric for evaluating the ability to detect actual switches.
- LSTM and HMM generally achieve the highest S recall, especially for users with more data or more balanced classes.
- PCC and MCC offer moderate improvements over Naive, but their gains are less dramatic than for precision.

User-Specific Insights

Stage 1: Switch or NoSwitch?

Again, for users who switch rarely, User 1 and User 7, NS recall is perfect (1.0). S recall is very low for Naive (0.11 or 0.32), slightly better for PCC (0.17 or 0.33), and improves with MCC (0.35 or 0.62). LSTM (0.72 or 0.7) is much higher, and HMM (0.42 or 0.65) is moderate. The extreme class imbalance makes it very difficult for any method to recall actual switches, with false negatives greatly affect the recall. LSTM's ability to learn subtle patterns allows it to recall more switches, but even so, recall is not perfect as precision (0.72/0.7). HMM and MCC provide moderate improvements, but the challenge remains.

User 3 with least data, Noswitch recall is very high for all methods. Switch recall for user 3 are Naive (0.44), PCC and MCC performing slightly better (0.47/0.47). LSTM, again performs best (0.79) and HMM is also acceptable (0.53).

For a more moderate Switch/NoSwitch distribution, User 10 and User 11's NS recall is very high for all methods. S recall: Naive (0.22 or 0.3), PCC (0.31 or 0.46), MCC (0.42 or 0.7), LSTM (0.72 for both), HMM (0.41 or 0.61). LSTM achieves the highest S recall, showing its strength with more balanced data. MCC and PCC provide moderate improvements over Naive. HMM is similar to MCC for these users.

Stage 2: *Identifying the destination?*
Again, this second stage is based by the first part as the classifiers are cascaded. Like precision, LSTM are still poor across the users. HMM's high recall once again implying that the hidden states are efficient in detecting the destination app. For PCC and MCC approaches show acceptable destination identification with regards to stage 1 recall being acceptable.

4.1.3 Run Time:

Table 6 Runtime for different methods

	Naive	PCC	MCC	LSTM	HMM
User 1	1m59s	2m53s	2m13s	19m42s	20m12s
User 3	10.8s	44.6s	18.9s	1m23s	1m29s
User 7	2m40s	4m1s	3m7s	27m34s	25m21s
User 10	50s	2m41s	1m81s	7m19s	7m42s
User 11	1m9s	3m56s	2m36s	7m11s	7m31s

For context: all models are performed running on a MacBook Pro (2021) with Apple M1 Pro chip and 16gb of RAM. The runtime analysis reveals a clear trade-off between model complexity and computational efficiency. The Naive approach is consistently the fastest due to the simple algorithm, but its predictive performance is limited. Contextual clustering-based methods introduced in this research, PCC and MCC, incur higher computational costs, with PCC being the slowest due to its broader k range evaluation when clustering using K-means searches, while MCC's majority voting runs faster for its limited k range (2,3). LSTM and HMM, while offering superior predictive performance in many cases, are by far the most computationally demanding, with runtimes that scale with data size.

4.2 Evaluation of Methods' Robustness to Noise

To evaluate all methodologies robustness to noise, Gaussian noise addition is used. Gaussian noise is random variation added to data where each noise value is sampled from a bell-shaped normal distribution defined by its mean (often zero) and standard deviation, simulating the kind of natural fluctuations (Çoban et al., 2012). Kim & Chung (2024) also used the Gaussian noise of = 0.3 × σ to inject into data test the adaptability of time series data classification.

To accurately add noise, the training data will remain clean and will be used to train the Gradient Boosting classifier. In contrast, the test data will be modified by adding Gaussian noise, where the noise standard deviation varies from 0 (no noise) up to 100% the standard deviation of each feature. The graphs below will show all models' robustness to noise addition.

4.2.1 MCC (User 10)

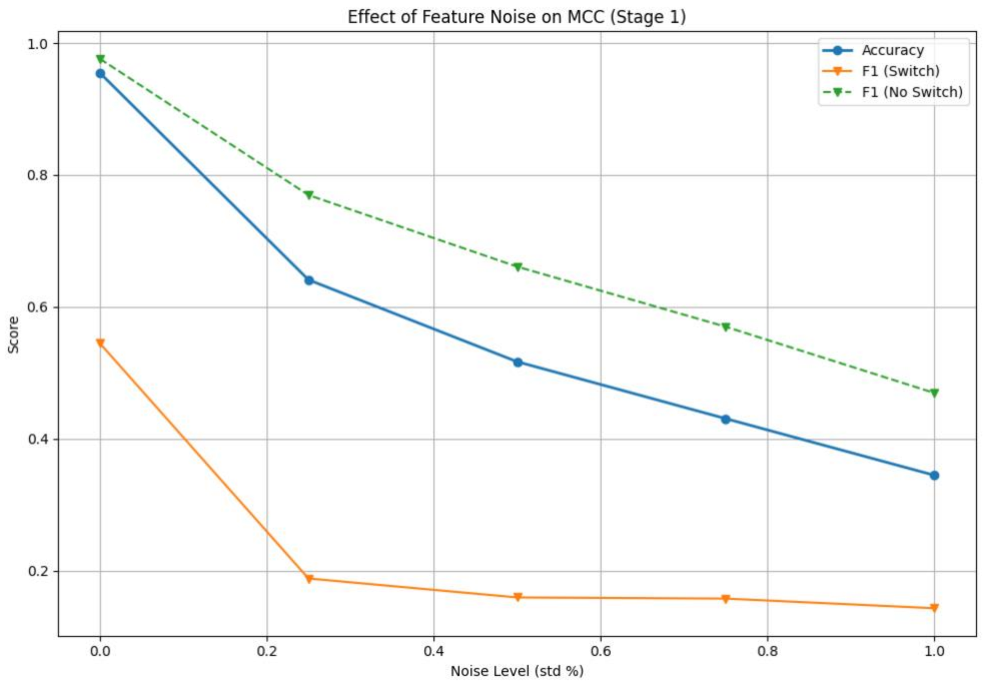


Figure 8 MCC Noise Stage 1

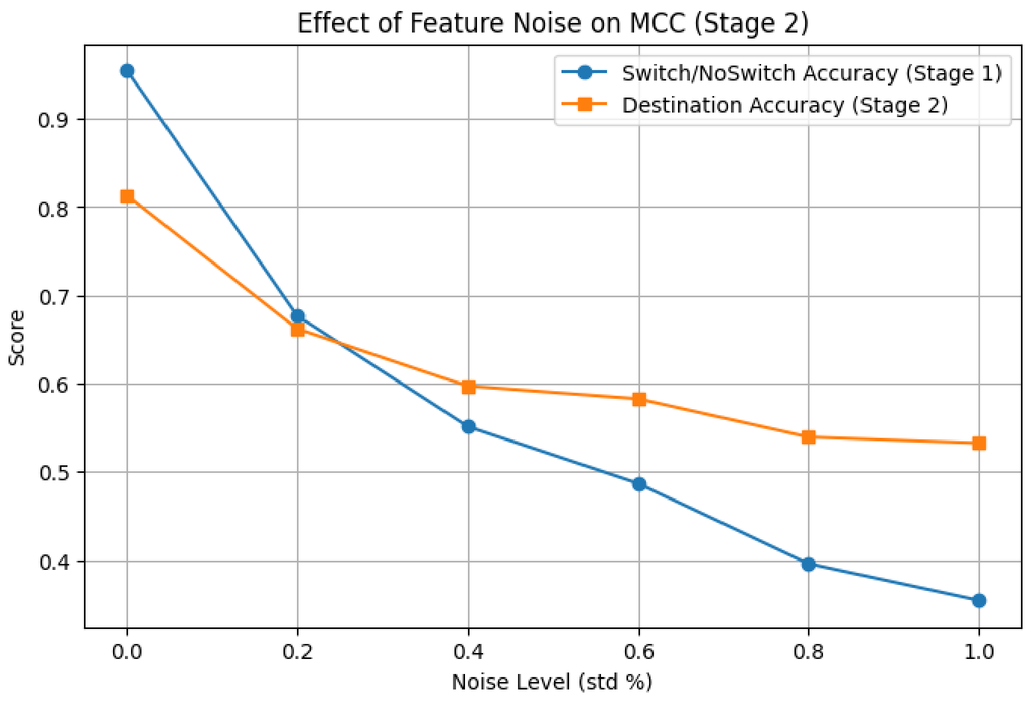


Figure 9 MCC Noise Stage 2

When Gaussian noise is added to the features, the MCC's ability to tell Switch from NoSwitch in Stage 1 drops quickly - accuracy and F1 scores fall as noise goes up, with the Switch F1 taking the biggest hit and the NoSwitch F1 staying higher since it's the majority class. In Stage 2, destination prediction accuracy also goes down with more noise, but not as steeply as Stage 1 accuracy, which shows that picking out switch events is more sensitive to noisy data than predicting where they switch to. Overall, the MCC works best on clean data and becomes much less reliable as additive noise approaches the features' full standard deviation.

4.2.2 PCC (User 10)

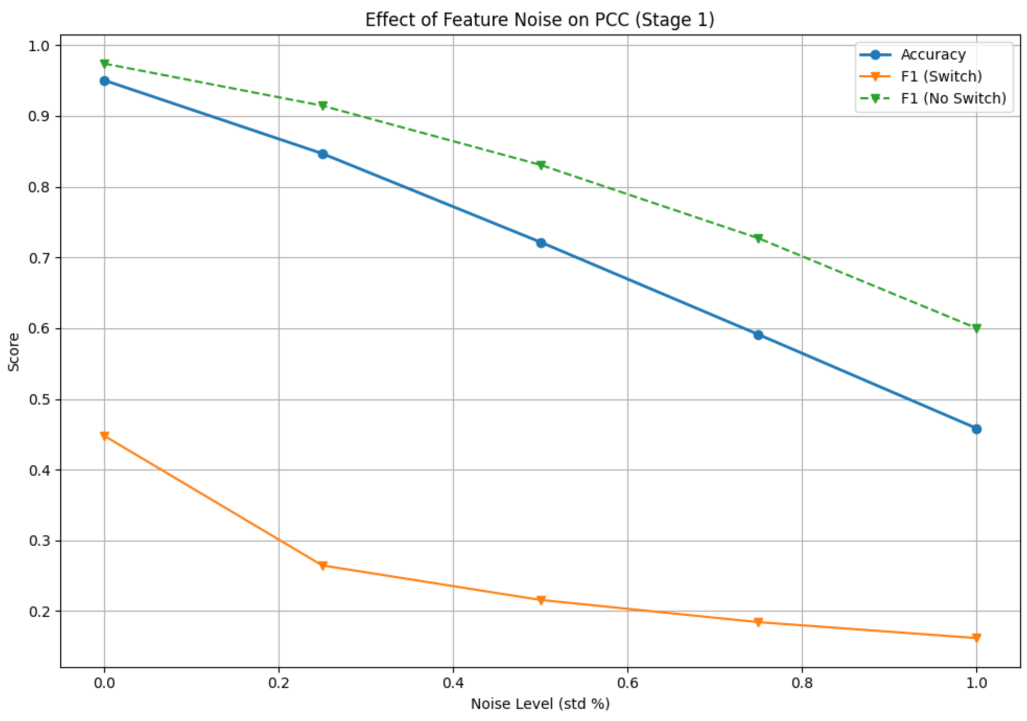


Figure 10 PCC Noise Stage 1

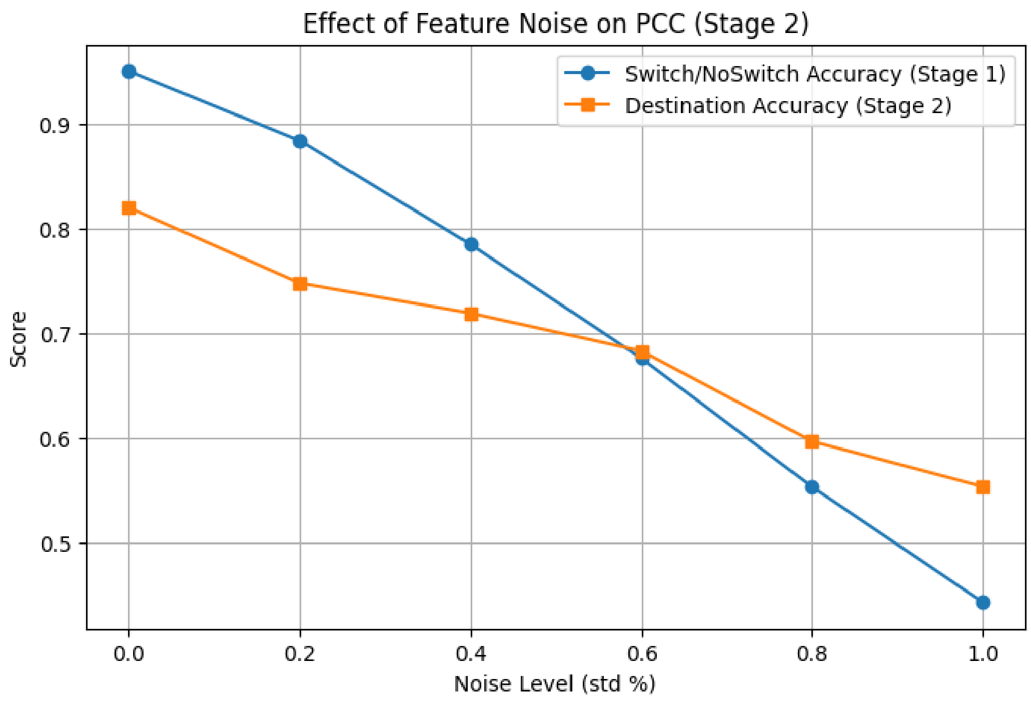


Figure 11 PCC Noise Stage 2

In Stage 1, with accuracy and F1 scores for both Switch and No Switch going down as noise increases, and the Switch F1 suffers the most because it's the minority class. Stage 2 destination prediction accuracy also falls with more noise, but not as steeply as Stage

1 accuracy, showing that PCC needs clean, reliable features (especially for detecting switches) to work well.

4.2.3 LSTM (User 10)

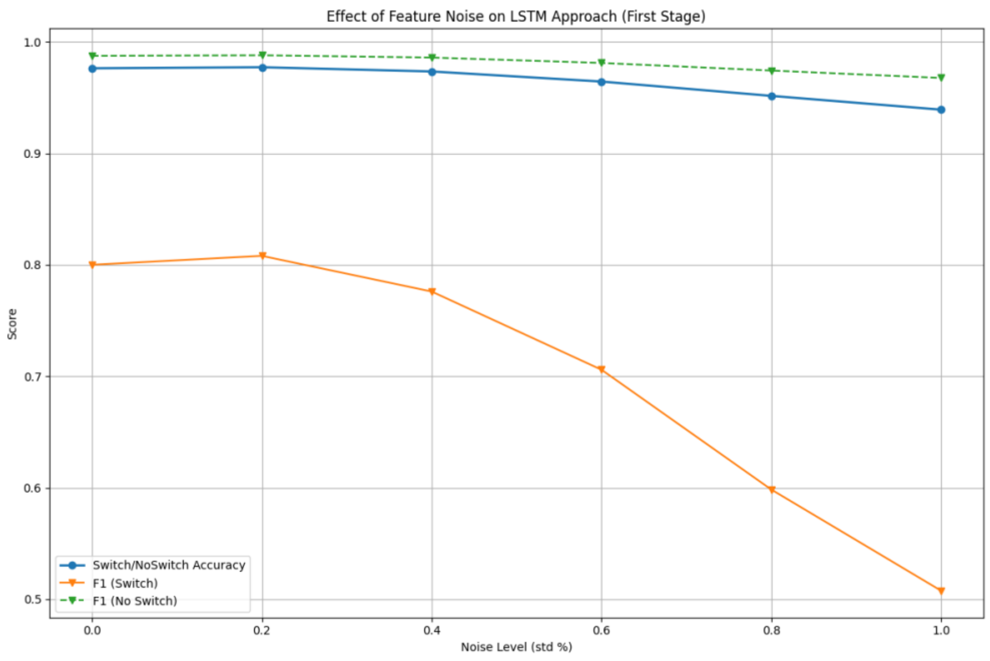


Figure 12 LSTM Noise Stage 1

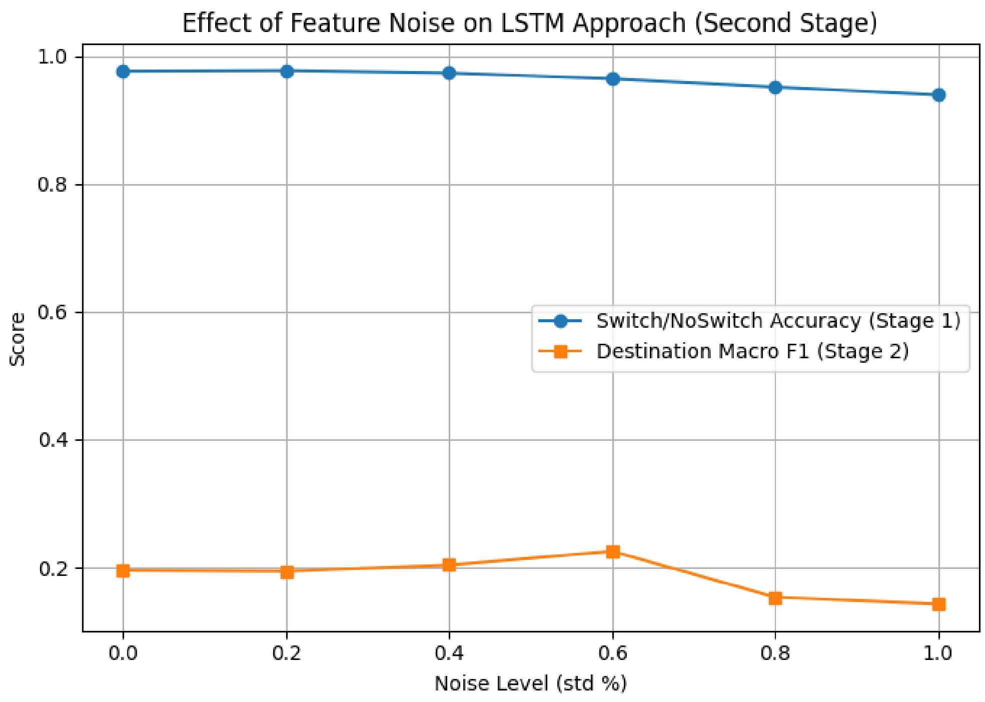


Figure 13 LSTM Noise Stage 2

The LSTM stays very accurate and maintains a high F1 for the NoSwitch class even as feature noise rises, but its F1 for the minority Switch class falls off sharply past moderate noise levels, suggesting that the LSTM finds it really hard to detect switch in a noisy test set, though it still performs much better than MCC and PCC. Its destination prediction is weak to begin with and gets slightly worse under noise, showing that while the LSTM is great at the main switch/no-switch task, it still struggles with rare events and detailed destination categories even when data is not noisy.

4.2.4 HMM (User 10)

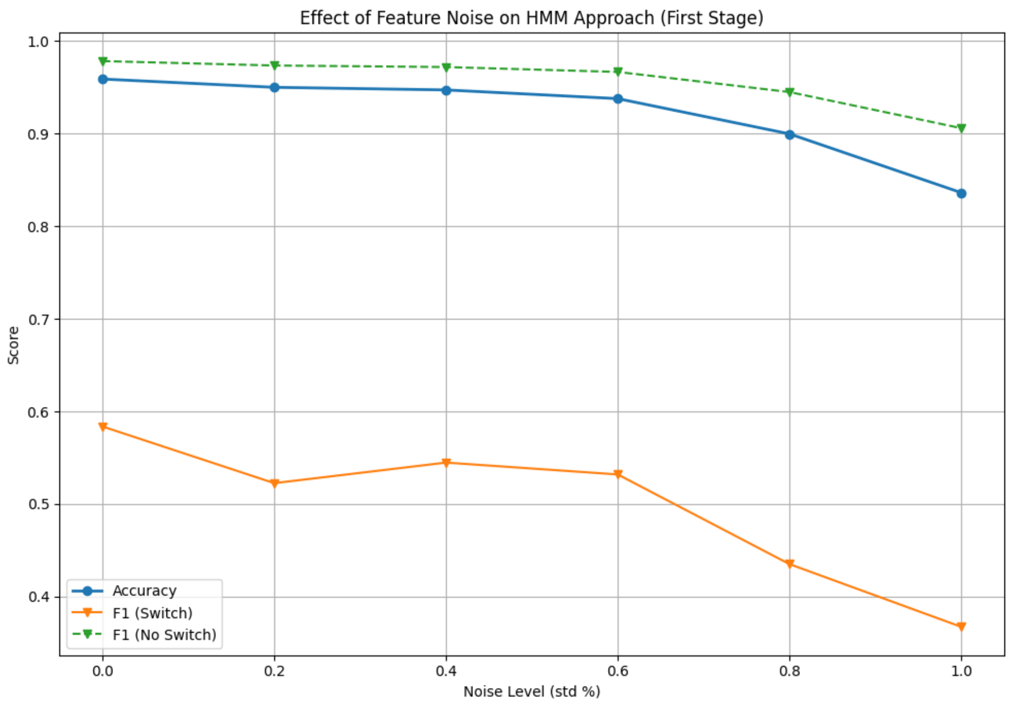


Figure 14 HMM Noise Stage 1

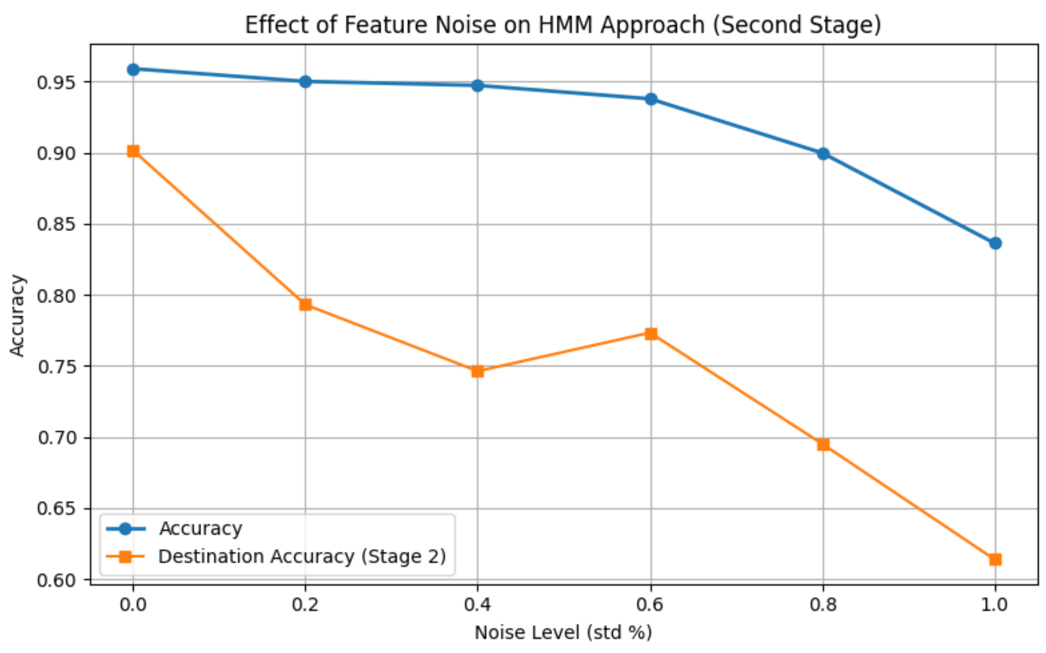


Figure 15 HMM Noise Stage 2

The HMM approach maintains very high accuracy and F1 for the NoSwitch class up to moderate noise levels (around 0.6 additive noise level) and only starts to decline noticeably at high noise (≥ 0.8 additive std noise), while its F1 for the minority "Switch" class is lower from the start and drops more sharply only when noise becomes severe, showing that switch detection is more sensitive. In the second stage, Switch/NoSwitch accuracy remains stable across all noise levels, but destination accuracy falls off rapidly as noise increases which illustrates that HMMs are robust for the main classification task yet struggle with detailed destination prediction under noisy conditions. However, even with huge level of noise, HMM's 2nd stage accuracy is still acceptable.

4.3 Feature Importance

This section will look into the importance of features in different methods' classification (Gradient Boosting and LSTM). To recap, It is important to note that features regarding to current app, system memory and CPU, or changes between app and current app foreground time at the time of detection, were carefully dealt with, by removing the current value from HMM and LSTM or by removing from the aggregation by means of PCC and MCC.

Features importances are calculated using feature_importances_ attribute from sklearn.ensemble library, defined as the sum of these purity gains across all splits in all trees and normalized so that the importances of all features sum to one (Breiman, 2001).

4.3.1 Stage 1 - Switch or NoSwitch?

Table 7 Stage 1 Feature Importance

PCC	MCC
app_category_entropy	app_category_entropy
changes_between_apps	changes_between_apps
current_app_foreground_time	app_transition_freq
app_transition_freq	vowel_key_ratio
memory_usage_ratio	current_app_foreground_time
resource_demand_score	special_character_rate
system_average_mem	system_stddev_mem
system_average_cpu	memory_usage_ratio
received_bytes	enter_key_count
system_stddev_mem	consonant_key_ratio

LSTM	HMM
click_scroll_ratio	current_app_foreground_time
right_clicks_ratio	app_category_entropy
inter_key_ratio	changes_between_apps
tab_key_count	memory_usage_ratio
app_transition_freq	current_app_average_mem
number_row_usage	app_transition_freq
press_release_interval_std	cpu_usage_ratio
communication_@	resource_demand_score
tab_key_context	current_app_average_cpu
high_mem_peak	edge_center_ratio

In Stage 1 (Switch vs NoSwitch classification), the most important features span application usage, system resources, keyboard, and mouse activity. Application-related features such

as *app_category_entropy*, *changes_between_apps*, *current_app_foreground_time* are the most direct indicators of context switching, as they capture how frequently and diversely users interact with different applications.
Resource features like *memory_usage_ratio*, *resource_demand_score*, and *system_average_mem* are also highly relevant, since spikes or instability in system resources often precede or follow a switch event. Keyboard features (for example: *vowel_key_ratio, inter_key_ratio, tab_key_count*) and mouse features (for example: *click_scroll_ratio*, *right_clicks_ratio*, *edge_center_ratio*) provide additional behavioral context, reflecting changes in user interaction patterns that may signal an impending switch. Overall, while application and resource metrics are the strongest predictors, keyboard and mouse activity offer valuable supplementary signals for robust switch detection.

4.3.2 Stage 2 - Destination

Table 8 Stage 2 Feature Importance

PCC	MCC
system_average_mem	cpu_stability_index
network_activity_ratio	system_stddev_mem
cpu_stability_index	pointer_movement_per_min
memory_usage_ratio	resource_demand_score
current_app_average_cpu	cpu_usage_ratio
total_network_bytes	system_stddev_cpu
left_clicks_per_min	network_activity_ratio
mem_stability_index	current_app_average_mem
pointer_movement_per_min	memory_usage_ratio
system_stddev_mem	system_average_mem

LSTM	HMM
resource_spike	current_app_average_mem
scroll_events_ratio	memory_usage_ratio
mem_stability_index	system_average_mem
memory_usage_ratio	total_network_bytes

system_average_mem	edge_center_ratio
system_average_cpu	resource_engagement_index
network_activity_ratio	system_stddev_cpu
changes_between_apps	pointer_movement_per_min
system_stddev_cpu	click_scroll_ratio
current_app_stddev_cpu	received_bytes

In Stage 2 (Destination prediction), the most important features are predominantly related to system resources and user interaction patterns. Features such as *system average mem*, *cpu stability index*, *memory usage ratio*, *system stddev mem*, and *system stddev cpu* reflect the overall and dynamic state of system memory and CPU usage, which can influence or indicate the type of application a user is likely to switch to. Network activity features like *network_activity_ratio* and *total_network_bytes* capture the intensity of data transfer, which may be characteristic of certain destination app categories (for example: communication or web apps). User interaction features such as *pointer_movement_per_min*, *left_clicks_per_min*, and *scroll_events_ratio* provide insight into how actively the user is engaging with their device, which can also be predictive of the next app category. Overall, resource and interaction metrics are key for distinguishing between different destination categories, as they encapsulate both the system's operational context and the user's behavioural patterns at the time of switching.

Next part, different classifiers will be looked at and see how implementing different classifiers combining with different methodologies change the results.

4.4 Classifiers Comparison

For this part, three classifiers, described in 3.9.2 (Extra Trees, Random Forest and Gradient Boosting) will be compared across different approaches (except for LSTM since LSTM itself is doing the classification).

Precision:

Table 9 Precision of different classifiers (User 10)

User 10		Extra Trees	Random forest	Gradient boosting
PCC	S1	S: 0.55	S: **0.85**	S: 0.81
	S2	0.79	0.72	**0.82**
MCC	S1	S: 0.6	S: **0.9**	S: 0.78
	S2	0.7	0.68	**0.84**
HMM	S1	S: 0.79	S: 0.79	S:**0.85**
	S2	0.93	0.95	**0.92**

Recall:

Table 10 Recall of different classifiers (User 10)

User 10		Extra Trees	Random forest	Gradient boosting
PCC	S1	S: 0.48	S: **0.68**	S:0.31
	S2	0.79	**0.68**	0.82
MCC	S1	S: 0.59	S: **0.71**	S:0.42
	S2	0.69	**0.63**	0.81
HMM	S1	S: 0.19	S: 0.42	S: **0.45**
	S2	0.92	0.88	**0.90**

F1:

Table 11 F1 of different classifiers (User 10)

User 10		Extra Trees	Random forest	Gradient boosting
PCC	S1	S: 0.52	S: **0.68**	S: 0.45
	S2	0.79	**0.68**	0.82
MCC	S1	S: 0.59	S: **0.8**	S: 0.54
	S2	0.69	**0.62**	0.81
HMM	S1	S:0.31	S: 0.55	S: **0.58**
	S2	0.93	0.87	**0.89**

Interestingly, the choice of classifiers influences the results a lot, especially with the Contextual Clustering-based approaches. Gradient Boosting, which have been used for evaluating between performance of models in different users, adaption to noised and feature importance, perform worse than Random Forest when combine with Contextual Clustering-based approaches. The Precision, Recall and F1 support that, for example: F1 switch of PCC and MCC improves from 0.45 and 0.54 to 0.68 and 0.8 (from Gradient Boosting to Random Forest), suggesting that these methods work well with Random Forest. On the other hand, HMM approach, which mostly out-perform the two cluster-based approaches for Gradient Boosting, has lower F1 score in Random Forest. For Stage 2 – identifying the switch, HMM is still on top with around 0.9 F1 score in all classifiers. Out of the three classifiers, Extra Trees seem to perform the worst, when combining with all methodologies.
The stage two can only be accurately and meaningfully assessed alongside stage 1, that is the reason why Random Forest in stage 2 for PCC and MCC is in bold, even though it is less than Gradient Boosting. As the stage one's total detections become less, the resulting metrics in stage 2 increase, as explained in many parts above, hence Stage 1 detection is prioritized if the gaps are considerable.
Next part, all the results will be summarized into key take-away insights.

4.5 Summary of Results

Across all five users and methods, detecting the rare Switch events in Stage 1 proved to be the greatest challenge. The LSTM consistently achieved the highest precision and recall for Switch detection by learning subtle temporal patterns, while HMM offered a robust middle ground and PCC/MCC showed capability of detecting context switch through moderate improvements over the Naive baseline. In Stage 2 - destination prediction, HMM's hidden-state representation translated most effectively into precise and reliable app-category forecasts, whereas LSTM, despite its strong switch-detection performance, struggled to identify where users switched. When test data were corrupted with Gaussian noise addition, HMM remained the most resilient in both stages, PCC and MCC degraded rapidly on Switch detection, and LSTM retained high NoSwitch scores but lost Switch performance sharply under moderate noise. Runtime measurements highlighted the trade-off between accuracy and efficiency: the Naive

method ran fastest, PCC/MCC incurred moderate cost, and LSTM/HMM were substantially slower. Moreover, feature-importance analyses revealed that Stage 1 models relied most heavily on application-usage and system-resource metrics, supplemented by keyboard and mouse signals, while Stage 2 forecasts were driven primarily by resource stability and mouse and keyboard interaction pattern features. Finally, when replacing Gradient Boosting with Random Forest in the clustering-based approaches, Random Forest often yielded higher F1 scores for MCC and PCC, suggesting that bagged decision trees may pair better with PCC and MCC approaches. However, HMM preferred Gradient Boosting as it shows worse results when combined with Random Forest. These results inform the choice of method according to whether priority is given to switch detection accuracy, destination prediction fidelity, noise-robustness, or computational efficiency.
The discussion will list out the contributions of this thesis, the limitations this research faces and the possible future work.

5. Discussion

5.1 Contributions

This research overall has addressed the gaps for detecting context switches and identifying the destination application in a privacy-assured aggregated logged dataset like BEHACOM, where behavioural features are aggregated into one minute and the actions leading to the switch are undefined. The research has developed sets of engineered features that are derived from original dataset, in order to reduce the huge dimension of the BEHACOM from over 12000 features to 77 features while still maintaining the nature of the dataset.

Methodologies such as Proximity-Contextual clustering and Majority-Contextual clustering have answered objectives regarding whether the combinations behavioural patterns of previous minutes before the switch can determine the optimal look-back window. That is, in an obscured data in one-minute-interval like BEHACOM and with consideration to the skewed nature of the dataset, when combining with Gradient Boosting and preferably Random Forest, the two approaches of PCC and MCC can still detect with F1 score around 0.5 to 0.8 on the first context switch detection part and accuracy of 0.63 to 0.87 in the destination application. Those two clustering-based modelling have been compared to the Naïve baseline approach and have shown significant improvement demonstrated through the result section. Proximity-based (PCC) contextual clustering and Majority-based Contextual clustering (MCC) has acceptable metrics on both 1st stage and 2nd stage for User 10 and 11 (least skewed data), with MCC mostly perform better on 1st stage due to its ability to choose the optimal past window length while PCC mostly perform better on second stage due to its ability to greedily choose the window based on the centroid of nearest windows.

Additionally, the clustering-based modelling methodologies (Proximity-based Contextual Clustering - PCC and Majority-based Contextual Clustering - MCC) are also compared to the sequential-based modelling, such as Hidden Markov Model (HMM) and Long-short Term Memory (LSTM). The results in 4.1.1 and 4.1.2 indicate that the sequential-based approaches perform better. Each model has shown its own strengths and weaknesses, for example: LSTM, with its sequential learning model, is superior to others when it comes to context detection at first stage but performs poorly on second, likely due to overfitting or insufficient data. On the other hand, HMM, preferably with Gradient Boosting classifier, with its hidden states is strong when identifying the destination application, while having acceptable metrics at the first stage of detection.

In least skewed user 10 and 11's activity logs, the four proposed methodologies all perform better than the Naïve approach and shows that context switch can be detected through high precision and an acceptable recall. Research of models have used the same parameters and background so that comparison of models can be as meaningful and accurate as possible and avoid potential biases towards any model.
The research also proposed the cascaded pipeline which 1) detect if the context switch occurs and 2) identify the destination application, only predicted and true switches from the first part are evaluated in the second part. For the results, research compared different supervised classification methods (Gradient Boosting, Decision Tree and Extra Trees) and found out that models can perform better with different models, for example: PCC and MCC performs better with Random Forest while HMM performs

better with Gradient Boosting. The results also analysed on each models' robustness to noise through Gaussian noise addition method, and found out that, although all methodologies are influenced by high level of noise, clustering-based approaches like PCC and MCC are much more vulnerable to noisy data than sequential based approaches like HMM and LSTM.

5.2 Limitations

While this research presents a detailed and comprehensive approach to context switch detection in a privacy-preserving data by minute-aggregation, using different machine learning methods, several limitations should be acknowledged.

5.2.1 BEHACOM data

As the BEHACOM data only has the aggregated log data due to privacy, this research's reliance on hand-crafted rules and thresholds for data labelling, means that the detection criteria may not generalize well to all users or contexts, and the absence of user-annotated ground truth (actions before the switch and how long those actions are) restricts the ability to rigorously train and evaluate supervised classifiers. However, the research has addressed that by contextual clustering approaches to determine the relevant actions before a switch, or a fixed look back length for sequential models.

The skewness of data also troubles the four proposed methodologies as the Switch events is much less than the NoSwitch events. Again, the lack of ground knowledge within each raw application data such as Chrome or Firefox, leading to limited details for what the users are actually doing in that particular application. As a result, the assumption of the users' activity in one app is different from the other app does not consider the semantic meaning of app transitions, for example: switching from a Web type application to Office type application may be a meaningful context change, but that is not always the case, as it can be simply the action of copying and pasting data Web to Word document, while the main action is in Word.

5.2.2 Handcrafted Rules and Heuristic-Based Switch Definition

The re-definition of context switches relies on hand-crafted rules and thresholds (foreground time, number of app changes, stability factor). This threshold is somewhat arbitrary and may not accurately capture correctly the destination app, although users with most valid switches (above the confidence threshold of 0.5) are chosen for training and evaluating the results, to avoid changing the much of the data. Moreover, the results are sensitive to the chosen value of parameter such as N look-back window for sequential based modelling approaches like HMM and LSTM, or HORIZON_MINS (the minutes to look back to choose - 18 minutes) for clustering-based methods (PCC, MCC). Small changes in these values can significantly affect the number and type of switches detected.

5.2.3 Potential Bias towards Switch block and two minutes of same application for NoSwitch blocks

Although handled with care and aggregated using means, for Majority-based Contextual Clustering, a larger look back minutes (in length) can somehow have slight benefit on detecting switches in the first part classification. However, it does prove that hypothesis that different a larger look back time can help in detecting context switches. Another limitation is how the NoSwitch blocks for Naïve, PCC and MCC are created To ensure the reliability of NoSwitch blocks for Naïve, PCC and MCC approaches, NoSwitch segments are defined as three latest consecutive stable minutes, rather than just two like Switches (this minute' current app is different from last minute's current app). This criterion matches the assumption that NoSwitch block should have a little level of stability and leads to more robust classification. Also, using three-minute blocks also improves the performance of Naïve metrics, making results in the second stage more meaningful and comparable to other proposed methodologies, preventing unrealistic outcomes in destination prediction that could occur if the first stage fails to accurately identify NoSwitch events.

5.3 Future Work

5.3.1 Applying the pipeline to better datasets

As discussed in the literature review and limitations, the BEHACOM is not perfect because of its privacy-assured aggregated data in one-minute intervals. Initial effort of this research is to create a logger and apply consent form to ethics committee to get the permission to record the ideal data. The ideal data would capture data for users each of the users' action on their PC sequentially, alongside other information, for opened file name (not just application), URL accessed (web page), clicking/scrolling/dragging coordinates or sequentially logged keystrokes. Such data is hard to find, and all the papers experimenting on that kind of data in the literature review, do not publish them, potentially due to the privacy issue. If there had been such a dataset, PCC and MCC would get some ground truth to validate the clustering windows, or LSTM and HMM approaches can look at predefined actions in a task, which makes switch detection more robust and meaningful, rather than just application switches.
If gathering or publishing such kind of data is hard, BEHACOM paper itself published a logger that enables logging data, which is also what the research would have done if given more time. Through self-logged data, researchers can control and know the activities performed before a switch, which would help further justify the methodologies proposed.

5.3.2 Broader definition of context switch

The current research primarily focuses on features derived from app usage and user interaction, such as foreground time and the number of app change. However, context switches in real life are often influenced by a broader set of factors. Future work could enhance detection accuracy by incorporating additional contextual signals, such as the web link, precise mouse keyboard usage, etc. By engineering new features or building ideal datasets mentioned above, the model could gain a more holistic understanding of the user's context, leading to more accurate and meaningful switch detection. Future research could also consider conducting user studies where participants label their own context switches or validate the model's predictions. This would not only provide a more rigorous basis for evaluating model performance but also offer valuable insights into how users perceive and define context changes. Understanding user intent and perception is crucial for developing models that align with real-world needs and expectations. Future research could aim to segment higher-level tasks or workflows, perhaps using sequence modelling or clustering techniques. By combining app usage data with semantic analysis of user activity (such as document editing or web browsing topics), the system could provide a richer and more meaningful understanding of user context.

5.3.3 Other methodologies of segmenting time-series data

Due to time constraints, the research could not implement some wanted methods. These includes change point detection (Truong et al. ,2020), DBSCAN (Kasliwal and Katkar ,2016) or Agglomerative Clustering ("Agglomerative Hierarchical Clustering," 2013) for segmenting tasks before a context switch, which are found in the literature review. Other potential approaches that could have been explored are Hidden Semi-Markov Models (Yu, 2016) for more flexible state duration modelling, spectral clustering for uncovering latent behavioural patterns, and the use of attention-based neural architectures such as Transformers (Lim et al., 2021) to capture complex temporal dependencies.

5.3.4 Predicting the Context Switch

While this research has focused on the detection of context switches and destination categories, future work could explore proactive prediction. Developing predictive models would allow the system not only to recognize when a context switch has occurred, but also to anticipate upcoming switches and likely destination categories in advance. This could be achieved by integrating sequential forecasting models, such as recurrent neural networks (like LSTM in the methodologies) or attention-based architectures like Transformers from Lim et al., (2021), and by applying change point detection and clustering methods discussed earlier to identify early signs of transitions. Moving from detection to prediction would enable more adaptive and intelligent user support, improving the responsiveness and usefulness of context-aware applications.

5.3.5 HCI Applications

Future work could extend the switch detection framework beyond fixed rule-based thresholds by developing fully data-driven and adaptive models. Subject to ethical approval for collecting user-annotated labels, supervised learning algorithms (random forests), long short-term memory networks (LSTM) and transformer-based architectures, could be trained to recognize subtle behavioural patterns that simple heuristics fail to capture. Personalization mechanisms would allow the system to adjust detection criteria dynamically to individual work habits, thereby improving robustness across a diverse range of user profiles. Implementing context-switch detection in real time represents a critical step toward practical applications in productivity analytics and digital wellbeing. Additionally, integration with productivity dashboards, wellbeing platforms and adaptive user interfaces would enable detected switches to trigger personalised interventions, for example, break reminders after prolonged multitasking or summaries of focus patterns when attention fluctuates. By combining data-driven, adaptive detection with real-time processing and seamless integration for existing tools, future systems will be able to anticipate user attention shifts and deliver targeted support within everyday workflows.

6. Conclusion

This thesis has demonstrated that application-level context switches can be detected by analysing aggregated behavioural data without relying on predefined task labels. The BEHACOM dataset was reduced from over 12000 raw features to a concise set of 77 engineered variables, preserving essential patterns in mouse movements, keyboard activity and system resource usage. Treating each change in the active window as a proxy for a workflow transition allowed for generation of switch labels directly from application logs.

The literature review surveyed four key areas that informed the methodology: behavioural datasets for digital work analysis; definition of user-centric, task-centric and goal-driven approaches; classification methods for context switch detection and time series clustering for sequential user activity. Examining existing datasets highlighted the suitability of BEHACOM for studying knowledge worker behaviour. Reviewing classification approaches clarified the advantages of the reviewed supervised classifiers and out-performing classification techniques. Exploring clustering techniques for time series data through sliding windows motivated the development of proximity-based and majority-based contextual clustering methods.

Four novel detection methods were evaluated alongside a naïve fixed-window baseline. Proximity-based contextual clustering (PCC) and majority-based contextual clustering (MCC) use K-means to select relevant look-back intervals dynamically. These approaches, combined with Random Forest, achieved F1 scores between 0.50 and 0.80 for switch detection and destination-application accuracies of 0.63 to 0.87 on the least skewed users. Sequence-based models (Long short-term memory-LSTM and hidden Markov models-HMM) captured temporal dependencies more effectively. The LSTM produced the highest recall and precision score for detecting context switches, while the HMM paired with gradient boosting achieved the strongest performance in identifying the destination application.

A two-stage pipeline was introduced to isolate detection from destination identification. Only correctly detected switches in the first stage were passed to the second stage classifier. Comparative analysis across random forest, gradient boosting and extra trees classifiers revealed that PCC and MCC performed best with Random Forest, whereas HMM benefited more from Gradient Boosting. Noise-robustness experiments, using Gaussian noise addition, showed that sequence-based models degrade more gracefully under high noise levels than clustering approaches. The discussion section highlighted the limitations, through section 5.2 and possible future works, through section 5.3.

Reference List

Agglomerative hierarchical clustering. (2013). In W. Dubitzky, O. Wolkenhauer, K.-H. Cho, & H. Yokota (Eds.), *Encyclopedia of Systems Biology* (p. 17). Springer New York. https://doi.org/10.1007/978-1-4419-9863-7_100033

Armentano, M. G., & Amandi, A. (2007). Plan recognition for interface agents. *Artificial Intelligence Review*, *28*(2), 131-162. https://doi.org/10.1007/s10462-009-9095-8

Bakhshizadeh, M., Jilek, C., Schröder, M., Maus, H., Dengel, A., & Shuliang, L. (2024). Data collection of real-life knowledge work in context: The RLKWiC dataset. *Information Management*, *2102*, 277-290. https://doi.org/10.1007/978-3-031-64359-0_22

Breiman, L. Random Forests. *Machine Learning* 45, 5–32 (2001). https://doi.org/10.1023/A:1010933404324

Çoban, G., Büyüklü, A. H., & Das, A. (2012). A linearization based non-iterative approach to measure the gaussian noise level for chaotic time series. *Chaos, Solitons & Fractals*, *45*(3), 266-278. https://doi.org/10.1016/j.chaos.2011.10.011

Dev, H., & Liu, Z. (2017). Identifying frequent user tasks from application logs. *Proceedings of the 22nd International Conference on Intelligent User Interfaces*, 263-273. https://doi.org/10.1145/3025171.3025184

Devaurs, D., Rath, A. S., & Lindstaedt, S. N. (2012). Exploiting the user interaction context for automatic task detection. *Applied Artificial Intelligence*, *26*(1-2), 58-80. https://doi.org/10.1080/08839514.2012.629522

Friedman, J. H. (2001). Greedy function approximation: a gradient boosting machine. *Annals of statistics*, 1189-1232. https://doi.org/10.1214/aos/1013203451

Geurts, P., Ernst, D., & Wehenkel, L. (2006). Extremely randomized trees. *Machine learning*, *63*, 3-42. https://doi.org/10.1007/s10994-006-6226-1

Granitzer, M., Rath, A. S., Kröll, M., Seifert, C., Ipsmiller, D., Devaurs, D., Weber, N., & Lindstaedt, S. (2009). Machine learning based work task classification. *Journal of Digital Information Management*, *7*(5), 306–313. https://hal.science/hal-00872101

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, *9*(8), 1735-1780. https://doi.org/10.1162/neco.1997.9.8.1735

Jahanlou, A., Vermeulen, J., Grossman, T., Chilana, P. K., Fitzmaurice, G., & Matejka, J. (2023). Task-Centric Application Switching: How and Why Knowledge Workers Switch Software Applications for a Single Task. *Graphics Interface 2023*. https://www.research.autodesk.com/app/uploads/2024/02/Task-Centric-Application-Switching_GI23.pdf

Kasliwal, A. D., & Katkar, G. S. (2016). Web usage mining for comparing user access behaviour using clustering technique. *'Research Journey' International E-Journal*.

Kim, G. I., & Chung, K. (2024). Extraction of features for time series classification using noise injection. *Sensors*, *24*(19), 6402. https://doi.org/10.3390/s24196402

Koldijk, S., Sappelli, M., Verberne, S., Neerincx, M. A., & Kraaij, W. (2014). The SWELL knowledge work dataset for stress and user modeling research. *Proceedings of the 16th International Conference on Multimodal Interaction*. https://doi.org/10.1145/2663204.2663257

Kuric, E., Demcak, P., Krajcovic, M., & Nemcek, P. (2024). Is mouse dynamics information credible for user behavior research? An empirical investigation. *Computer Standards & Interfaces*, *90*, 103849. https://doi.org/10.1016/j.csi.2024.103849

Kılıç, A. A., Yıldırım, M., & Anarım, E. (2021). Bogazici mouse dynamics dataset. *Data in Brief*, *36*, 107094. https://doi.org/10.1016/j.dib.2021.107094

Leiva, L. A., & Arapakis, I. (2020). The attentive cursor dataset. *Frontiers in Human Neuroscience*, *14*. https://doi.org/10.3389/fnhum.2020.565664

Liao, T. W. (2005). Clustering of time series data - a survey. *Pattern Recognition*, *38*(11), 1857-1874. https://doi.org/10.1016/j.patcog.2005.01.025

Lim, B., Arık, S. Ö., Loeff, N., & Pfister, T. (2021). Temporal fusion transformers for interpretable multi-horizon time series forecasting. *International Journal of Forecasting*, *37*(4), 1748-1764. https://doi.org/10.1016/j.ijforecast.2021.03.012

Ma, X., & Hovy, E. (2016). End-to-end sequence labeling via bi-directional lstm-cnns-crf. *arXiv preprint arXiv:1603.01354*. 10.18653/v1/P16-1101

Meyer, A. N., Satterfield, C., Zuger, M., Kevic, K., Murphy, G. C., Zimmermann, T., & Fritz, T. (2022). Detecting developers' task switches and types. *IEEE Transactions on Software Engineering*, *48*(1), 225-240. https://doi.org/10.1109/tse.2020.2984086

Mirza, H. T., Chen, L., Hussain, I., Majid, A., & Chen, G. (2015). A study on automatic classification of users' desktop interactions. *Cybernetics and Systems*, *46*(5), 320-341. https://doi.org/10.1080/01969722.2015.1012372

Papadimitriou, D., Koutrika, G., Mylopoulos, J., & Velegrakis, Y. (2016). The goal behind the action: Toward goal-aware systems and applications. *ACM Transactions on Database Systems*, *41*(4), 1-43. https://doi.org/10.1145/2934666

Pellegrin, F., Yücel, Z., Monden, A., & Leelaprute, P. (2021). Task estimation for software company employees based on computer interaction logs. *Empirical Software Engineering: An International Journal*, *26*(5). https://doi.org/10.1007/s10664-021-10006-4

Perea, J. A., & Harer, J. (2014). Sliding Windows and persistence: An application of topological methods to signal analysis. *Foundations of Computational Mathematics*, *15*(3), 799-838. https://doi.org/10.1007/s10208-014-9206-z

Powers, D. M. W. (2020). Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation. *arXiv.Org*.

Rabiner, L. R. (1989). A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, *77*(2), 257-286. 10.1109/5.18626

Satterfield, C., Fritz, T., & Murphy, G. C. (2020). Identifying and describing information seeking tasks. *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*, 797-808. https://doi.org/10.1145/3324884.3416537

Sánchez Sánchez, P. M., Jorquera Valero, J. M., Zago, M., Huertas Celdrán, A., Fernández Maimó, L., López Bernal, E., López Bernal, S., Martínez Valverde, J., Nespoli, P., Pastor Galindo, J., Perales Gómez, Á. L., Gil Pérez, M., & Martínez Pérez, G. (2020). Behacom - a dataset modelling users' behaviour in computers. *Data in Brief*, *31*, 105767. https://doi.org/10.1016/j.dib.2020.105767

Shannon, C. E. (1948). A mathematical theory of communication. *The Bell system technical journal*, *27*(3), 379-423. 10.1002/j.1538-7305.1948.tb01338.x

Tashman, C., & Edwards, W. K. (2012). WindowScape: Lessons learned from a task-centric window manager. *ACM Transactions on Computer-Human Interaction*, *19*(1), 1-33. https://doi.org/10.1145/2147783.2147791

Tian, Y., Zhou, K., Lalmas, M., & Pelleg, D. (2020). Identifying tasks from mobile app usage patterns. *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. https://doi.org/10.1145/3397271.3401441

Truong, C., Oudre, L., & Vayatis, N. (2020). Selective review of offline change point detection methods. *Signal Processing*, *167*, 107299. https://doi.org/10.1016/j.sigpro.2019.107299

Tsimperidis, I., Asvesta, O., Vrochidou, E., & Papakostas, G. A. (2024). IKDD: A keystroke dynamics dataset for user classification. *Information*, *15*(9), 511. https://doi.org/10.3390/info15090511

Yan, J. N., Gu, Z., & Rzeszotarski, J. M. (2021). Tessera: Discretizing data analysis workflows on a task level. *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, 1-15. https://doi.org/10.1145/3411764.3445728

Yu, S. (2016). General hidden Semi-Markov model. *Hidden Semi-Markov Models*, 27-58. https://doi.org/10.1016/b978-0-12-802767-7.00002-4

Zinkovskaia, E., Tahary, O., Loewenstern, Y., Benaroya-Milshtein, N., & Bar-Gad, I. (2024). Temporally aligned segmentation and clustering (TASC) framework for behavior time series analysis. *Scientific Reports*, *14*(1), 14952. https://doi.org/10.1038/s41598-024-63669-6