Developing a Mental Health Toolkit for Social Media

Lewis Buttle, Matthew Jameison, Daisy Kennedy and Patrick Michaels

May 2022

Fourth Year of Study

Supervised by Prof. Mike Joy

Department of Computer Science

University of Warwick

Abstract

This report details the successful development of a toolkit that can be used as a resource to reduce the negative impacts social media has on the mental health of its users. Four tools were developed to reduce a user's screen-time. One tool was developed to provide effective moderation. All tools are open-source. A guide is included as an appendix, which explains how to use and integrate each of the tools. Developers are invited to add their own relevant tools to the toolkit, to help with the goal of creating an exhaustive, open-source resource for tackling the mental health problems caused by social media.

Keywords

- Social Media
- Mental Health
- Screen-Time
- Content Moderation
- Machine Learning
- Natural Language Processing
- Data Visualisations
- Android OS

Contents

1 Introduction

1.1 Background

The amount of time that we spend everyday staring at digital screens has increased rapidly over the last decade, with more recent years seeing an exponential increase due to the Covid19 pandemic. [31] This surge in time spent online has been linked to increases in sedentary behaviour, anxiety levels, depression rates, and suicide rates. [19][44]

A major culprit for the sudden increase in screen-time is social media. We dedicate almost a sixth of the day to scrolling – three hours and 40 minutes. [13] When you take into account that the recommended amount of sleep is eight hours per night, that figure becomes a quarter of our day that is spent on social media. Therefore, any impacts from social media, good or bad, greatly affect our day and our lives. This project explores the bad, specifically the negative implications on mental health.

UK Parliament has published many reports about the negative mental health effects of social media, which are written by reputable and relevant institutions. One such report, from the Royal College of Paediatrics and Child Health, expresses deep concern with how little research is going into the harms of social media. [38] Another, from the House of Commons Science and Technology Committee, recommends the introduction of new regulations for social media companies, to facilitate easier analysis of the impacts of prolonged use of such platforms. [16] The last example is from the ESRC International Centre for Lifecourse Studies in Society and Health, who suggest that the technology industry should spend more time investigating solutions to the mental health problems caused by social media platforms. [14] These calls-to-action inspired this project, and the team, to research and develop countermeasures for these under-explored issues.

1.2 Motivation

The breadth of the mental health problems caused by social media means that it is a difficult area to tackle exhaustively. Some platforms choose to ignore the existence of the problems entirely, while others do employ some well-being tools. However, the users do not get to benefit from all of these companies' well-being developments because they are not all available on every platform they use. Companies do not share their tools with competitors for obvious reasons, as such they are restricted to only using the tools that they develop in-house. Exhaustively tackling all of the issues in-house would be time-consuming and provides almost no benefit to the business; implementing these tools is essentially an act of philanthropy towards the users to protect their mental health. As a result, there is no incentive to implement every tool possible, and so many platforms just implement a few. Social media companies may be more willing to mitigate all of

these issues on their platform if it was easier to do i.e. if there was a free, open-source set of resources available to tackle the negative mental health issues caused by the platforms. This is the motivation behind developing a mental health toolkit for social media.

1.3 Problem Definition & Scope

This report explains how the social media related mental health issues stem from: (1) valuable and positive offline activities being displaced by the disproportionate amount of time spent online; (2) the exposure to negative and offensive content online. The project explores and implements technical strategies to solve these issues (reducing screen-time and providing effective moderation), and produces a toolkit of resources for developers and users to exploit with the purpose of improving user mental health within the context of social media.

The toolkit produced is not exhaustive because, as mentioned, this would be extremely time-consuming. The number of tools implemented is limited by the amount of time available to complete the project. The toolkit provided by this project is simply a foundation on which other developers, with the same non-profit goal of improving the mental health of social media users, can build upon.

This project provides tools that span the realms of reducing screen-time and providing effective moderation. These areas are explained and justified in the research section. On a high-level though, they were chosen because other options were based more on sociology, which the team has limited expertise in. The screen-time reduction tools and the moderation tool were better suited to the team as they could be designed and implemented from a more technical standpoint.

Delving deeper into the scope of this project, the definition of screen-time must be addressed because social media can be used from both desktop computers and mobiles, which have very different issues. The majority of social media visits are done from mobile. For example, only 20% of Twitter visits are done via a desktop computer or laptop, and 1.7% for Facebook. [13] Due to this, the project focuses solely on mobile users, so as not to overlook the difference in how users interact with social media on desktop vs mobile. Specific tools for desktop users are left as an extension to the project.

Now we consider how the success of the toolkit can be measured within the time-constraints of this project. To rigorously *prove* that the toolkit is useful for social media developers and for reducing a social media's impact on mental health, the tools would need to be integrated into an existing social media platform, or even integrated into a 'dummy' social media platform that would be developed by the team. This would facilitate testing the tools in situ and so, the efficacy of them for their respective goals could be thoroughly evaluated. However, testing the tools on such platforms would require months of monitoring users and their interactions with the platforms and the tools. This project is limited by time and so, testing in this way is infeasible. Due to this, integrating the tools with a platform would not be fruitful enough to warrant the

time-consuming task of performing the set-up of the platforms and the integration itself. The development team felt that the time would be better spent focusing on creating a breadth of tools, to cover as much ground in this field as we could. In short, integrating the tools into a single platform is not within the scope of this project.

The tools are instead unified by the construction of a detailed guide document for how to use the tools, which includes suggestions and resources for integrating the tools with a platform. Further projects are encouraged to use the guide to integrate the tools into a platform and analyse deeply the efficacy of the tools over time when integrated with a platform. However, in terms of this project's scope, the success of the tools is to be measured by validating their implementation against well-researched tool requirements that are based on a sound, theorised potential of reducing mental harm to users. Additionally, the tools' success can be measured on their ability to be clearly communicated to software developers within the manual-style guide.

1.4 The Deliverable

This project develops open-source tools to reduce users' screen-time on social media, and automate the moderation of content to reduce users' exposure to offensive content. A manual-style guide is produced for software developers to be able to use and integrate the tools within their own platforms. The toolkit and guide are designed as a modular resource that can be added to by other non-profit developers wanting to help to improve the mental health of social media users.

2 Research & Tool Derivation

Due to the expansive nature of this project, the team agreed that a sizeable amount of research was essential for deeply understanding the existing mental health issues caused by social media platforms. This research is used to justify which tools were implemented by this project, to form the foundation of the toolkit that can be expanded upon and used by social media developers.

This section details the research path taken by the team to break down the problem as a whole, to derive these tool ideas. To achieve this robustness, the following research process was followed:

1. Study the background of the entire problem and gather specific sources which detail the features of social media responsible for mental health issues.
2. Sift through these sources to identify which of these problematic features could have technical, implementable solutions (essentially disregard issues which likely require sociological or psychological solutions).
3. Further research this shortlist of issues and derive coherent tools which could serve as potential solutions.
4. Research the technical feasibility of these tools (such as existing, similar solutions), to justify that they could be produced within this project's scope.

Alongside this tool derivation, research was made into similar technical guides and toolkits, to inform the design of the ulterior deliverable: integrating the tools into a single toolkit document.

2.1 Social Media and Mental Health

Excessive Screen-time

A major issue with the interactions with social media is the sheer amount of time that we spend on the platforms. Spending so much time online displaces other important activities like work, exercise and in-person social activities. This leads to poor work performance; poor physical health; and under-developed social skills and loneliness. [32] These all contribute to damaging mental health, so it is not surprising that statistics show a strong link between excessive screen-time and poor mental health. For example, some studies have found that people who use social media for more than two or three hours per day have lower self-esteems, poorer mental health, and increased risk of suicide than those who use the platforms for less time. [23][44][15] Combine this with the statistic that on-average, globally, we spend three hours and 40 minutes per day using social media on through a phone, and you see just how important and widespread this issue is; the average person uses social media to the point that they are at an increased risk of developing mental health issues.

It is important to note that using social media is not explicitly a bad thing. Social media provides many people with important social information needed in a modern world. It allows marginalised communities, like LGBTQ+ youth, to explore their identities in safe, virtual spaces. [24][39] And it empowers exhibitionism of negative experiences, helping people to share their struggles and understand them better with the help of others in the space (e.g. with '#Depression' on Instagram). [29] Disregarding these benefits would be ignorant to the needs of society.

Furthermore, using social media can be positive, but over-using it has the opposite effect. One study describes a Goldilocks approach must be employed here, which is an important concept for any tools developed for this issue of excessive screen-time. Tools in this area should not aim to dissuade people from using social media entirely, but to limit people's exposure to it in order to create a balanced experience. [2]

Unhealthy Comparisons

One issue with social media is how people make unhealthy comparisons to others, based on their social media presence. Many social media users do not present their authentic self online, tending to post only the positive aspects of their life. Comparing one's own life, which is full of ups and downs, with a social media presence of purely positive experiences, can leave us feeling like life is lacking and underdeveloped. Young people may aspire to these unrealistic expectations, under the assumption that they are achievable. Therefore, when the user inevitably underachieves in adulthood, they are more likely to experience mental health issues related to depression and suicide. [19]

This problem is a sociological struggle that is difficult to solve with just a technical tool, and so is out of the scope of this project. However, it is worth mentioning that new social medias are emerging that try and combat this problem. For example, BeReal is an app where at a random time each day, users are given two minutes to take one photo and share it with their friends for the day. This platform encourages users to share what they do on a normal day-to-day basis, rather than just the good parts of their life. [8]

Exposure to Offensive Content

Offensive content and hate speech refer to the online targeting of users based on personal characteristics, in order to spread hate and discrimination. People of colour, women and sexual minorities are the most common victims of online hate speech. [25] Exposure to this abuse and bullying naturally impacts a user's mental health and can make them feel unsafe online. These victim's of cyber-abuse are often told to "just log off", as though being abused online is a choice. Unfortunately, in the modern world, online abuse is unavoidable. A European study found that 42% of users are regularly exposed to this type of harmful content. [21] Moreover, the European Commission co-funded a project called SELMA, which found that online hate is "an inevitable

part of [user's] everyday social media experience". [42]

The SELMA project tackled the problem of online hate speech by providing tools to encourage respect and tolerance for other users. The SELMA report notes how little attention is given to the prevention of online hate speech, with regulators tending to focus on legal action instead. The SELMA project provides a toolkit for education about hate speech and encouraging social cohesion. Their approach focuses on preventing online hate speech through education of young people. [43] However, the current social media users need to be protected from uneducated, bigoted users now. An immediate solution is needed. This is where moderation comes in. Hate speech can be prevented from reaching victims online from a technical stand-point by moderating what a user can post to a platform. If a user cannot post hate speech, then vulnerable users are protected.

2.2 Specific Issues

Screen-Time Reduction

When considering the goal of screen-time reduction, one must question how much to reduce it by? What is a *healthy* amount of screen-time? While the background research for screen-time did suggest that more than two to three hours per day was excessive, further research found that such an explicit cut-off is not so provably defined. The Corsham Institute, a non-profit whose research aims to provide a fair and inclusive digital space for all, posit, in a report published by UK Parliament, that more research is needed in order to define such explicit thresholds for moderate and excessive use of social media. [11] This research area is limited because, as The House of Commons Science and Technology Committee affirms, social media companies are not regulated in the same way as other industries, meaning that information like the screen-time of their users does not have to be disclosed. As a result, research into screen-time has to be gathered by surveys. [16] This causes an issue with defining explicit screen-time thresholds because the data gathered can lack breadth and reliability. Due to this, the project was limited in that the objectives for reducing screen-time should not be bounded by specific amounts of time. Instead, the project had to be able to measure and define the ability of screen-time reduction in a different way.

Excessively using social media can be seen as a bad habit. Indeed, companies like Android treat excessive screen-time as a bad habit and design digital well-being tools around this notion. [5] The goal for screen-time reduction is therefore being able break the habit. A Harvard article decomposes habits into three components [9]:

Trigger

A trigger is a reminder to start the habit. For example, the ping of a notification from a social media app triggers the user to unlock their phone and check their social media.

Behaviour

The behaviour is the main activity of the habit. For example, excessively scrolling on social media.

Reward

The reward is what reinforces the habit behaviour. For example, the dopamine rush of occasionally enjoyable social media content reinforces how good it feels to scroll on social media platforms. This feeds into a habit cycle.

Removing any part of this habit cycle would break the habit and therefore aid screen-time reduction. From a project perspective, the reward part would be hard to remove as a social media without enjoyment for the user would not be used. Hence, the we focus on removing the trigger (e.g. the notifications) and the habit behaviour (e.g. the excessive screen-time spent on social media platforms). Breaking the trigger could be done by reducing the number of notifications a user is sent. Breaking the behaviour could be done by interrupting the behaviour before the user experiences too much of a reward. To summarise, this project measures success of screen-time reduction related tools by their ability to break an area of the habit cycle.

Moderation

Moderation is a fundamental feature within any platform which encourages user-based content. Social media platforms fit this description, with platforms like Facebook having over one billion pieces of content uploaded each day. [26] The concept of moderation is the act of ensuring that user-based content is user friendly to a specific degree, and having an effective way to perform this moderation is crucial in ensuring that users of a social media platform are able to interact with each other in a healthy way.

Human moderation is currently the gold standard for moderation, but this is often not a scalable solution for social media platforms due to the amount of new content that is being created constantly. As a result, the need for effective automatic moderation is ever-growing, creating a market for auto-moderation products. Pattr [33], Preamble [34], WebPurify [48], Moderate-Content [28], Mobius Labs [27], and Amazon Rekognition [1] are just some of the purchasable auto-moderation tools available to developers. Unfortunately, the list of free, open-source auto-moderation tools is not so long, with most only being free for a short trial period. Hence, there is space for an effective, free and open-source auto-moderation tool. Such a tool would allow smaller moderation teams to employ moderation more easily, in turn allowing and encouraging them to better protect the mental health of their users through effective moderation.

2.3 Tool Derivation

This section discusses how tools to combat the issues of reducing screen-time and providing effective moderation were derived. Each tool aims to reduce the severity of the chosen issues.

Humane by Design

One resource that helped derive the tools, in a general sense was the documentation detailing the principles of humane design [50]. Humane by Design is described as "a resource that provides guidance for designing ethically humane digital products through patterns focused on user well-being", which can be understood as having a very similar goal to the project, hence the source was used within this project's derivation of tools. The seven humane design principles are standards that any digital product should generally adhere to, if they want to effectively consider the mental well-being of their users. These principles are summarised as:

Empowering A product should prioritise the value it gives to a user, and not the revenue it can generate.

Finite A product maximizes the *quality* of the time a user spends with it, not the quantity.

Inclusive Design products for disabilities first, with a focus on developing within diverse teams.

Resilient Products focus on the well-being of the most vulnerable users, and anticipates the potential for online abuse.

Respectful Products are designed to prioritise user's time, attention and overall well-being.

Thoughtful Products aim to prevent abuse, protect privacy, and steer people toward healthier digital habits.

Transparent All features within a product are clear about intentions, and honest in actions.

The team felt that these principles aligned closely with the project's goals, and were used to help inspire and justify the idea for the tools. These principles served as an effective background source for this project, as the principles are effectively existing research into categorizing theory-based solutions to the mental well-being related issues with social media. Hence this project was able to use these as direct, reliable starting points for deriving and justifying the efficacy of tool deliverables. It was thought that if proposed tools that relate to these principles were thought of as a 'product', the criteria these principles are proposing could be applied to these tools as a means of building the requirements for the individual tool's success criteria, as such these principles were used as primary success validators for some tools within this project.

Notification Spam Filter

Justification

One trigger in the habit cycle of social media overuse is push-notifications. A hypothesis was made that there may be a positive relationship between the number of notifications a user receives, and the amount of time they spend on their phone. To test the hypothesis, during the period 20/12/2021 to 12/01/2022, one of the team members recorded their daily notification and screen-time data provided by 'Digital Well-Being and parental controls' in the Android settings app. *The data is included within Appendix D at the end of this document.* The tool provides the number of notifications, the hours and minutes of screen-time, and the number of unlocks daily (see Figure 1).

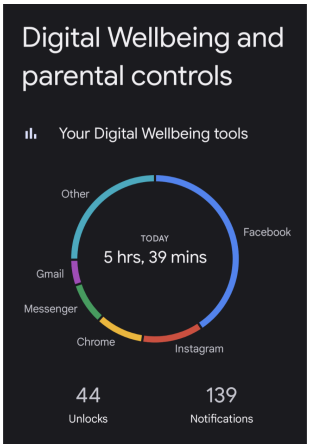


Figure 1: Screenshot of 'Digital Well-Being and parental controls' feature in the Android settings app.

The goal was to analyse the relationship between the following three variables: notifications, screen-time, and unlocks. Unlocks were included because from team discussion it was found that some people unlock their phone, read a notification and close it, while other people unlock their phone for a notification and proceed to scroll on social media for a period of time (for example, 20 minutes). Hence, the link between notifications and screen-time would differ greatly for users. And so by looking at notifications and phone unlocks, the team could analyse the trigger more the just the varying behaviour.

The Android tool provided the data in relation to general apps. So, some entries for screen-time were skewed by days where the user watched content on streaming services like Netflix. Those days naturally had more screen-time, but that does not imform much about the trigger of notifications. Hence, the team manually processed the daily data to store screen-time from non-streaming apps, and notifications from social media apps. This allowed for a more confident data analysis because the data was more specific to the hypothesis.

The attributes used in the analysis are daily:

- Number of Unlocks (*Unlocks*)
- Non-streaming screen-time minutes (*Screen-time*)
- Number of Social Media Notifications (*Notifications*)

Weka[47] was used to produce linear regression models from pairs of the three variables. Weka was used because the team has experience with it and easy to use for simple linear regression and data analysis, which is all that is needed for comparing viewing relationships between three numeric variables.

Using Weka, the following three linear regression models were produced:

1. Screen-time = 0.9635 * Notifications + 155.4153

   with Correlation Coefficient: 0.3428

2. Unlocks = 0.4243 * Notifications + 27.3193

   with Correlation Coefficient: 0.6145

3. Screen-time = 1.521 * Unlocks + 165.4414

   with Correlation Coefficient: 0.3645

The models produced all have a positive gradient, which means that all the variables have a positive relationship with each other. That is, as the number of social media notifications increases, the number of unlocks increases and so does the amount of screen-time. This is in-line with the hypothesis.

Equally important to the gradient is the correlation coefficient. The correlation coefficient of a linear regression model is a metric of the confidence we can have in the model, on a scale of 0 to 1, where 1 is complete confidence. Model 1 and Model 3 have fairly low correlation coefficients, both around 0.35. This suggests that the positive link between screen-time and notifications, and the link between screen-time and unlocks cannot be fully trusted. This may be because of the issue mentioned earlier in this section about how the reactant behaviour to a notification trigger may vary depending on external factors, like who the user is. These models were made with only one user's data, but there are still factors that affect the way a user reacts to notifications. For example, some days are more busy than others, and so regardless of the number notifications, a user may not use their phone for a whole day. This variance in daily screen-time is why the number of unlocks was included when making models. This feature aims to explore how to stop the notification trigger, and not how to control the behaviour. And so, while we cannot have confidence in Model 1 and Model 3, the tool is still viable.

Finally, Model 2 has a reasonable correlation coefficient of 0.61. This gives reasonable confidence that the more notifications a user receives, the more times they unlock their phone. This

explains that reducing the number of notifications would help the user to break the habit cycle by not triggering them to unlock their phone.

The amount of data used for the models was limited, but the conclusions do align with the natural intuition of the relationships. As a result, it was decided that reducing the number of notifications a user receives would help to reduce the screen-time of a user, and thereby improve the user's mental health.

Existing Solutions

Blocking all notifications is the obvious, but extreme solution. This would reduce the number of notifications, but it comes at a cost of the user missing out on events or information about them or their friends. The project aims to provide a balance; reduce notifications while still notifying the user of notifications that are important to them.

A comparable existing solution for this Notification Spam Filter tool is the type of spam filtering used in emailing systems. 'A Study on Email Spam Filtering Techniques' [46] explores the ways in which these email filters are designed. The author defines spam as unsolicited emails with a message general enough to be sent to many recipients, regardless of their personal identity. This has parallels with notifications because some notifications are as broad as 'Bob Bobbington posted a photo'. One's personal identity does play some role here, in that Bob must be someone you are friends on Facebook with. However, Bob may have thousands of friends who could all be sent that same notification; Bob's posting of a photo does not have any direct link to you as a user. Hence, we could consider this type of generic notification as spam that could be filtered out. However, the notification from Bob Bobbington blurs the line on the spam criteria of being unsolicited. The user is friends with Bob, and so this could be considered as a request to get notifications about Bob's posts, whether they are important to the user or not. This subjective importance of a notification is what may differ email spam from notification spam.

To elaborate, a user may not find a post update notification about Bob important, but they may find the exact same notification important when it is about a different user (say their best friend, Alice). The user is triggered by both notifications to check their phone and increase screen-time, whether or not the notification is swiped away; whether or not the notification was important to them. Spam in the context of notifications can be considered being more about the receiver than the subject(s) of the notification; is the notification important to the user?

This research suggests that there are two ways to consider spam notifications. The first is as certain types of messages. The second is as certain types of messages about certain users. The design section covers why the second definition is used by this tool.

'A Study on Email Spam Filtering Techniques' [46] proceeds to describe some of the techniques used for email spam filtering. The first technique described is 'Distributive Adaptive Blacklists', which is essentially a lookup system where new emails are checked against known spam messages, and labels it as spam accordingly. This is useful with emails because the same message

can be sent out again and again to new victims. However, with notifications, the labelling is not so black and white, as preferences can change over time.

Another technique described is 'Rule Based Filtering'. This involves checking for textual patterns in a new email, and each match affects the email's score. If the email's score exceeds a threshold, then its is filtered out as spam. [46]

The remaining filters suggested by this study are machine learning techniques. For example, Bayesian Classifiers, K-Nearest Neighbours and Support Vector Machines. The discussion about them in this study was limited to specific implementations rather than conceptual explanations and comparisons. This study does not cover the techniques in enough detail for this project to choose an approach for the notification spam filter. However, it was useful for defining spam in the notification context, as well as guiding this tool towards machine learning techniques, which were ultimately used for this tool.

In contrast, 'Machine learning for email spam filtering: review, approaches and open research problems' [12] explores the use of machine learning in the context of spam email detection on a very deep level. Most importantly for this project, the study compares the strengths and weaknesses of the machine learning approaches used in spam filtering. The study concludes with the hope that the paper will be used for further exploration and research into machine learning for spam filtering. This paper is a valuable resource for the project's experimentation with building a predictor for notification spam. The models and findings of this paper are used in the design section to influence and justify design choices for the project.

Resources

When choosing classifiers for the filter, 'Machine learning for email spam filtering: review, approaches and open research problems' is an invaluable resource. [12]

Scikit-learn is a Python library with tools for quick implementation of most machine learning classifiers.[1]

Notification Postbox

Justification

The justification for this tool overlaps heavily with the justification for the spam filter tool; reducing the number of notification a user receives will help to reduce screen-time. This tool is also heavily inspired by the humane design principles' heavy emphasis on returning control to the user. The tool aims to supplement notification management features already existing, and better enable the user to control the behaviour of apps that make this control difficult.

[1] `https://scikit-learn.org/stable/supervised_learning.html#supervised-learning`

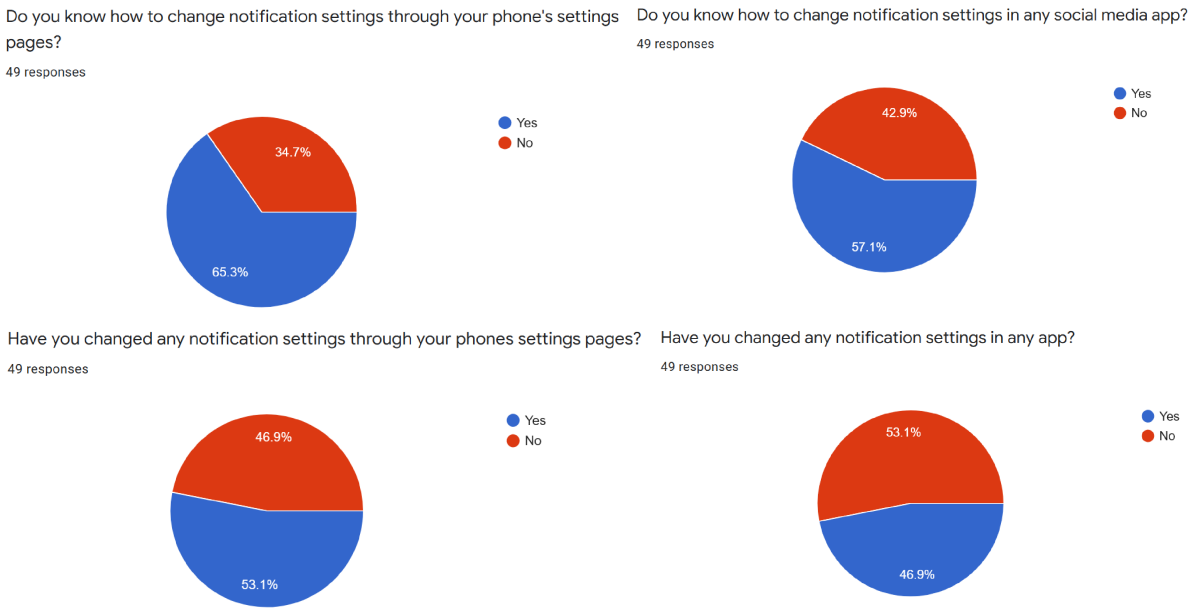


Figure 2: Survey responses related to existing notification management

The premise is that notifications are difficult to manage regardless of the mechanisms in Android, iOS, and applications themselves give users control. To ensure the validity of this premise an anonymous survey was conducted with 49 respondents to evaluate how effective the existing measures are and if push notifications trigger undesirable user behaviour or even app usage.

To justify the tool three things must be found. That current notification management techniques are not sufficient or accessible. That notifications trigger app usage. That users receive unwanted amounts of notifications. An additional question of whether notifications have directly caused stress in respondents would further enrich the justification but is not necessary provided the other three conditions are met.

When asked only 57% of respondents know how to change notification settings in any social media app they had installed and 65% knew how to change notification settings through their phone's settings pages. 53% of respondents had changed settings through the phone's settings pages and only 47% had changed notification settings in any app. This indicates that a large number of typical users do not know how to use the existing mechanisms for managing notifications.

Users were asked to rate how they felt about the number of notifications they received from one to five, with one being too few, three the right amount, and five too many. No respondent answered with a one or two, meaning all respondents felt they received sufficient or too many notifications. 41% of respondents answered with a three – an ideal number of notifications – meaning 59% of respondent felt they received too many notifications and 25% answered with a five suggesting they received far more notifications than they would prefer.

When examining Figure 4 it is also clear than not all notifications are interacted with, but the

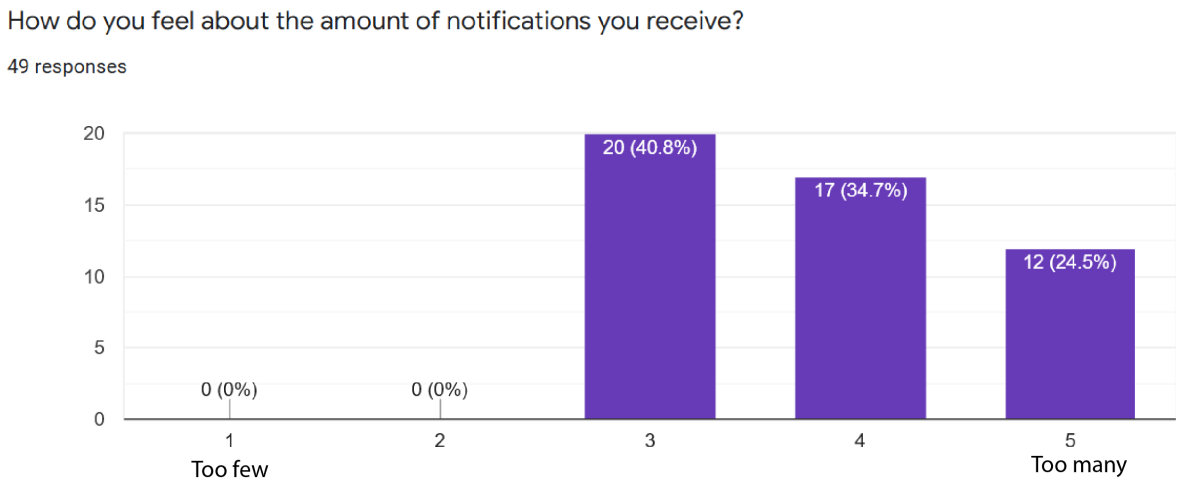


Figure 3: Survey responses related to the number of notifications received

number of respondents who frequently open apps after receiving a notification with great frequency (63% at least daily, compared to 80% who use apps at that rate) indicates that push notifications act as a trigger for social media app use.

Finally when asked if they ever found notifications stressful 82% of respondents answered yes. Considering the lack of time frame and specificity in the question, this is not as strong a statistic as initially indicated. However, the combination of all of these factors justify the exploration of a tool intended to manage notifications.

All together the survey was strongly indicative that push notifications are triggers of app usage, as they are designed to be. Additionally it showed that users receive many notifications and often more than they would wish, and that push notifications are a source of stress for some users.

Existing Solutions

The existing Android and iOS notification management tools are both granular and powerful. They allow for notifications to be enabled or disabled on an app-by-app basis as well as granting users control over more specific control over the types of notifications permitted -– on Android through the notification channel settings. Additional controls allow users to specify how intrusive different kinds of notifications should be. Whether the notification should ever play noise or cause vibration or if it should simply silently appear. Managing notifications through phone settings is the easiest way to do so and offers the most options but users must be able to navigate to the settings page for each app and the survey suggests this knowledge is not as common as would be desired.

YouTube has a feature called scheduled notification digest. It collates all the notifications for the day and lets you set a time to deliver them all at once. [17] This inspired the open-source postbox tool for the project. The postbox would aid in reducing the number of notifications

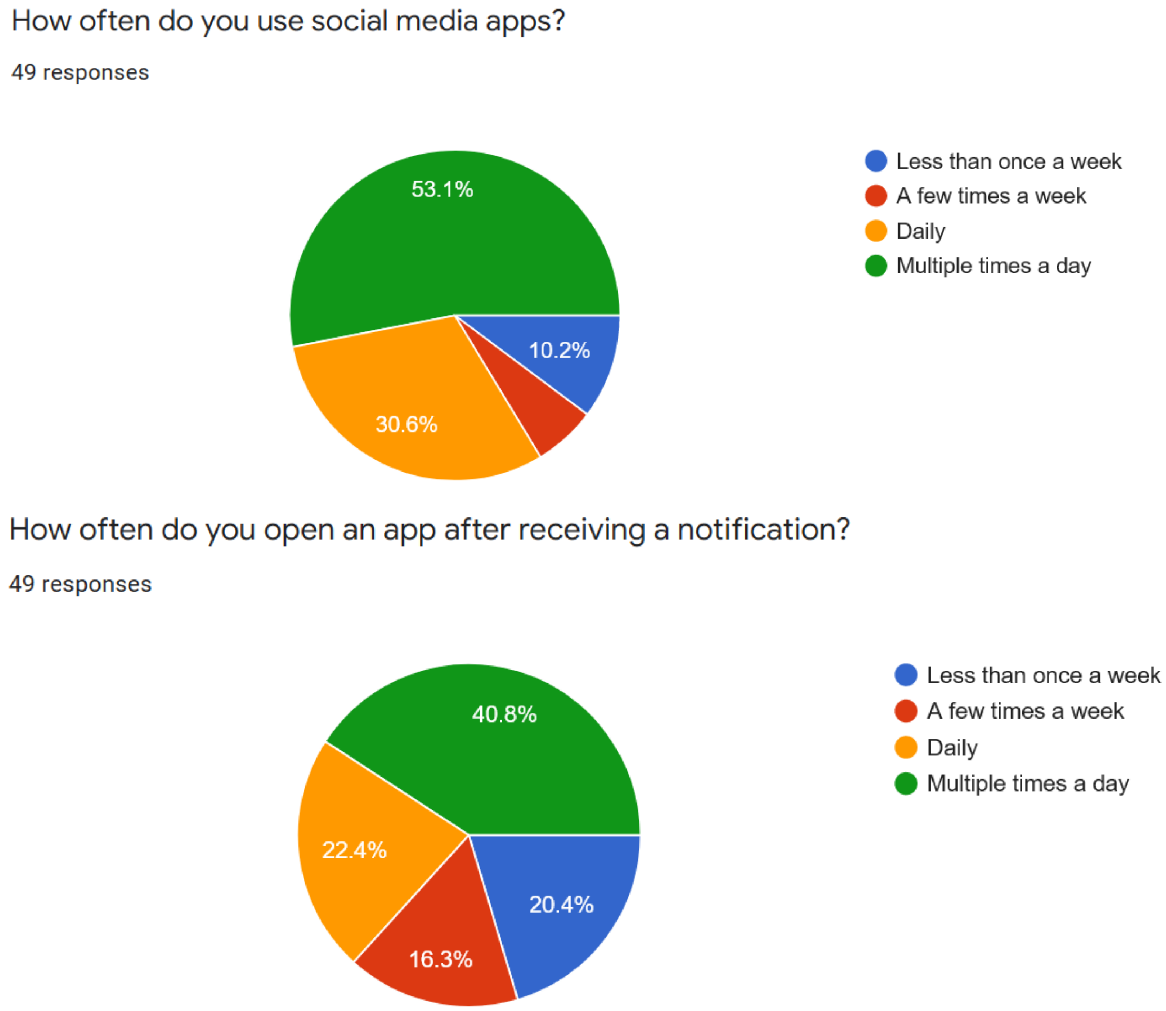


Figure 4: Survey responses regarding social media use and notifications acting as entry points to app usage

triggers received by a user because it will block notifications from being immediately being sent to a user. Blocking these triggers breaks the habit cycle and reduces the amount of time a user spends online. This feature only applies to notifications YouTube would send, however, and while a good feature for developers to include, it only aids users with the apps that implement is and, based on the survey responses, users would still need to know of the feature to use it.

A similar solution is status feeds. When an event occurs it is added to the feed and the feed can be filtered to only show important events or only certain types of event. The proposed app works along similar lines except managing notifications instead, which must allow the user to remove them from the feed. In a sense, the app is an implementation of this feed for notifications on mobile devices. The unique properties of notifications (they can be updated and removed) is distinct and ensures some degree of novelty.

Exit Points

Justification

A key concept in humane design is making the experience finite. If we present content as infinite, people tend to consume more of it. They lose track of how much they are really consuming, and don't gain any satisfaction from it. On the other hand, by making it finite, a person will have a full awareness of how much they have consumed, thus consuming less content and gaining the same satisfaction from it. A related study, the bottomless soup study, shows that when a person is provided with unlimited soup (a bowl which will always refill itself from the bottom), they would consume 73% more soup compared to from a non-bottomless bowl. This is exactly the effect social media has on people when they are given an infinite scrolling feed. Exit points can act as the bottom of the bowl; an end to the infinite feed of social media. Infinite content will be available to the user, but they will be fully aware of the amount of content consumed and their level of satisfaction with it. Exit points provide a clear option for a user to put down whatever platform they are interacting with. This can help curb unhealthy social media habits and help keep a user actively engaged on the platform, rather than absentmindedly scrolling through its content.

Existing Solutions

There are examples of exit points in other industries that aim to help a user plan their time spent engaging with the content provided by these industries. A good example of this is within the video game industry. Some video games utilise addictive design ideas, such as gambling, to maximise the amount of time their users play the games and increase their likelihood of spending money within the game. This can lead to unhealthy relationships with the game that ultimately causes a users trust in the game to fall and drives them away when they realise that it is causing them harm. The game makes addiction easy. Video game addiction is a problem which some

of the video game industry has tried to address, as they want their users' to come back to game after a session rather than quit it. This also reduces the chance of a user buying any more content that is associated with the game they quit as they would expect the same negative experiences. Therefore, they found it beneficial to embed exit points within a user's experience, these are breaks in the action of the video game, allowing a player to take a breather and logically be able to set down the game at these points and plan a session length dependent on these exit points. A common example of exit points in video games are a central hub area that a player can interact with between the main elements of gameplay, when a player enters this area, they may reflect on the amount of time they have been playing the game and then make the conscious decision to move on to the next part of the game or stop their session. They may decide that when they get to the hub area, they are going to end their session. This allows the exit point to act as a measure of the amount of game played, providing a finite experience where they don't burn themselves out by playing the game too much, so are then more likely to finish it.

Exit points are also used within streaming platforms. An example most people are familiar with is used by Netflix. Netflix has an auto play function, this could end up creating an infinite stream of content, as when a series is over it auto plays a recommended series or movie. To make this more finite, if a user has not interacted with the video player for a few videos and has allowed the auto play feature to play each new video then Netflix will provide a prompt to the user asking if they're still watching, this requires the user to clear the message if they want to keep watching. This acts as an exit point as it asks an important question to the user if they are still engaged with the content. The user can then make the conscious decision to stop watching shows on Netflix if they are done with the session, rather than consuming more than then initially intended.

Time Transparency

Justification

Through the conducted research into the humane design principles, the 'Transparency' and 'Empowering' principles were deemed an interesting basis for a tool aiming to reduce a user's screen-time. These principles served as the primary inspiration for the proposed tool: rather than forcefully managing a user's screen-time, effective means of managing their own screen-time can be given to them.

The transparency design principle stands for a design ideology that is "clear about intentions, honest in actions" [51]. In context, a feature which follows this principle will utilise a user's data for the sole good of maximizing their experience with a platform, whilst being as honest with the user as possible with the collection and results of their data. This principle relates to the screen-time reduction issue as it highlights that many existing addictive designs are not transparent with a user's *time* data, where the data of the length of time a user spends interacting with a platform is either not recorded or unavailable to be viewed by a user. From this lack of transparency, a user is unable to be directly aware of their screen-time and furthermore cannot take action

on any unhealthy screen-time habits, as any data suggesting it is problematic is unavailable to them. This research proposes that being transparent with a user's screen-time data(recording their time spent on a platform, and making this data available to them to see) would help reduce their screen-time by empowering them with the knowledge of their screen-time data, and hence enabling them to take personal action in reducing an unhealthy amount of screen-time.

The humane design principle of empowerment was also studied to extend the research into transparency design, as the empowerment principle overlaps with transparency in some ways and further gives thought on how to involve the user more in a positive social media experience. Specifically, empowering designs focus on maximizing the "value they provide to people over the revenue it can generate" [49], which relates to giving users as much control as possible when interacting with the algorithms and data within a platform, to *empower* the user with control over their own user experience. This principle is applicable to the screen-time reduction issue, as a user spending time on a social media platform means they should be *empowered* to choose both *how long* they spend on the platform, and *what* they are engaging with. Hence, this principle inspired that the tool should not only make users aware of their browsing habits, but empower them to be able to change these browsing habits once presented.

Research into both of these principles justified a solution to managing screen-time from a user-sided perspective, forming a basis for a tool which records a user's time spent on a platform and presents this information to the user (transparency), and then giving them the option to manage their screen-time directly in some fashion as a reaction to the recorded data (empowerment). The initial research into principles was effective in forming the fundamental idea for this tool and justifying that it would be effective at reducing a user's screen-time, however, this research alone did not serve well for the initial technical details of this tool, as there were still a few major questions about this tool idea before it could begin being designed:

- What specific time data will be recorded?
- How will the recording data be communicated to the user?

In essence, this initial tool idea; if utilised by a developer, would provide them with a means of cooperating with their users by being transparent with their screen-time data, to empower their users to make informed choices about their screen-time habits.

Existing Solutions

To answer this tool's further questions that came from the initial research and gain insight into the general technical area, research was made into existing solutions for similar problems.

Android Well-Being Kit - App Timer A relevant source of a similar solution was Android's "Well-Being Kit", which aimed to give users mobile-specific tools for managing their "digital well-being" [5]. The source presents a number of Android smart phone features for a non-technical,

user-based audience, using images and concise explanations to describe each feature. Android's solution deals with a similar issue as this project, as it is aiming to consider and account for the general well-being of their user base, whilst this project aims to deliver the abstract tools for any social media developer to achieve this same goal. It is worth considering that Android's motivation differs as their kit is embedded specifically for *smart phone* related issues for *users*, whilst this project is dealing with *social media* specific tool solutions for *developers* to implement. However, due to the overlap that many social media interactions occur on smart phones, this allowed Android's solution to become a worthwhile source throughout this project specifically for gaining insight into existing ideas on solving well-being issues.

The "App Timer" feature in the Android well-being toolkit was used to inform the design of the transparency tool. This similar solution operates by displaying the recorded times a user spends on different mobile apps per day (Fig. 5), and gives the user the option to *limit* any of these apps, where a limited app will be paused and unable to communicate with the user: hence disallowing the app from further taking time away from the user. This solution succeeds in following principles similar to those research through humane design to solve a similar issue: reducing a user's time spent on their smart phone. Importantly, this solution was able to answer the questions asked from the original research in Section 2.3:

- **What specific time data will be recorded?**
  The time data *per app, per day* is recorded.
- **How will the recording data be communicated to the user?**
  This similar solution communicates the recorded data through clear bar charts, representing the *total* time spent per day, alongside the individual times spent on each app below the bar chart.

This project's transparency tool's design benefited from adapting certain aspects of this existing solution.

The method in which this solution was storing time data *per app, per day* was an important identified feature as it is effectively breaking down the user's screen-time habits even further, and is able to effectively communicate more of the user's data, meaning it is able to be even more transparent than just sharing the user's total time spent. As this data is displayed *per app, per day* because of the solution being based around smart phone usage, this idea was adapted to a social media context via the form of recording time spent per *content tag*, instead of *per app, per day*. A *content tag* is a term loosely applied to online interactions, but in the context of social media it is a "term or keyword attached to content that identifies characteristics of the content" [45]: essentially a string token attached to content that can help identify what is involved with the content(examples of content tags include items such as "Gaming", "Sci-Fi" or "Cooking", if the content included themes relating to these tags). Recording and displaying a user's time spent viewing content tags would better suit the context of social media, as social media is central

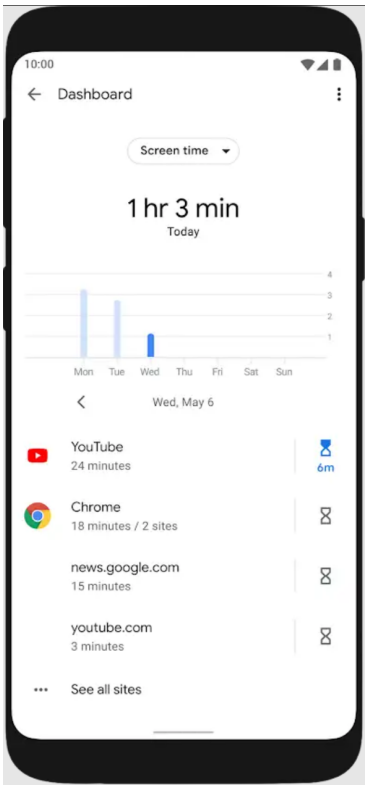


Figure 5: The Android App Timer feature[5]: displaying user times per app, and allowing user-chosen limits.

around content and the content viewed by a user is extremely refined to their interests: hence a breakdown of their screen-time *per content tag* is a relatively high level of transparency, and gives the user a much more refined break down of their screen-time.

To further justify the decision of timing by *content tag*, the social media context is vastly different from the smart phone specific context, as general app usage on a smart phone can be extremely variable depending on the app (for example, it would be normal to spend hours on an app for phone calls, but irregular to spend these same number of hours on the calculator app), whilst content on social media is on a much more even field of importance for a user(for example, there is no technical reason that a user will spend more screen-time browsing "Cooking" tagged content over "Gaming" tagged content, other than purely the user's interest in these topics). Hence, due to this fundamental difference between the mobile based solution and the social media based solution, this change to visualising time *by content tag* was deemed a more transparent solution, and hence more desirable for the social media context.

Additionally, the idea of using a bar chart to visualise and communicate a user's time data was adapted for this tool as this visualisation method seemed effective at communicating the data concisely. Hence, this feature was adapted for this solution and the transparency tool communicated the user's time data per content tag via a bar chart.

Resources The technical implementation of the transparency tool required a limited amount of resources to implement, as the finished implementation for this tool only utilised simple com-

ponents from within Python. Hence, this tool's development made use of Python documentation [36] as a general source for writing code, specifically relating to the *Time* library built into Python. Additionally, for plotting the time visualisation bar charts, the documentation for *matplotlib* [20] was used. Overall, these two basic sources served the technical aspect of this tool sufficiently.

Moderation Tool

Justification

Effective moderation is a necessity for a social media platform to remove content that could exist which would be deemed unacceptable to host on that platform. Moderation ensures that this content is removed before it is available to be seen by other users or shortly after it is available to be seen by other users. It allows platforms to remain a safe place for their user base to engage with each other, with a set of ground rules on that engagement.

A healthy social media platform should seek to remove content that is offensive, abusive, and hateful. Such content may affect other users' mental health, whether they are targeted directly or not by it. Through moderation it is possible to improve most users' experiences on a platform by ensuring they feel welcome and safe within it.

For this reason, this abusive content should be filtered out of a social media platform as soon as possible. Whether other types of non-hateful content are deemed offensive (e.g., content that is not safe for work), would be dependent on the platform and its desired audience, and is not something this tool seeks to address. Human moderation is often seen as the highest quality form of moderation, but its main downside is the lack of scalability, as a large amount of resources need to spent to hire people to cope with a large amounts of new content that can be posted daily. As a result, automated solutions are a necessity for platforms which experience a high quantity of new posts and comments every day. It also remains a very useful tool for smaller platforms that could manage to moderate all the new posts with a team of people.

Automated moderation allows for near instant filtering of content, far faster than a human could keep up with. A single moderation tool can be allocated more computational power to allow it to handle a higher number of posts and comments, which is easier to do than hiring a larger human work force, as well as cheaper. This allows human moderators to be reserved for edge cases or used to check some of the automated moderator's decisions to check it is behaving as desired.

This allows unacceptable content to be swiftly removed from a platform at a rate that is fast enough to cope with a large amount of new potentially dangerous content being posted.

Existing Solutions

There are many moderation systems which exist already for social medias, however most of these systems are not available to the public to use for free. The only options for a new social media platform are to develop a bespoke moderation system or outsource the work to a third party company. Despite the existence of these solutions, there is not a free to use tool available for developers to use for any social media platform they build.

Toolkit Guide

A variety of tools were chosen to address the problems the project is focused on. The project needed a single resource to combine these tools into a single comprehensive guide to solving the problems that the project addresses. This guide can then be used to give clear instruction to developers on how to approach creating a social media platform that values humane design by being able to easily implement the solutions presented to them within the guide. It also enables a clear linking between the tools to establish how tools can be used in conjunction with each other to improve a user's experience on the platforms they end up developing.

A single clearly written guide for all the tools makes each tool approachable increasing the likelihood of a developer understanding the problems presented and be more able to address them following the implementations shown with the guide and the code repository that the guide points towards.

Guidance and Inspirations

The guide takes inspiration and guidance from other user guides and documentation for other software tools or resources. The guide takes what works well from these and uses it to ensure the guide is well presented, easily navigable, and easy to understand. The main resource used for developing the guide was from the article "The Eight Rules of Good Documentation" [41], this provided a large amount of advice that the guide aimed to follow to ensure that it was effective and providing the information in it the most convenient way possible to maximise its effectiveness. We aimed to write the guide so that it fulfilled all eight rules described in the article.

One example of documentation looked at was the overleaf documentation [30]. This shows a good example of a easily navigable and clear documentation. It covers a much larger breadth of content then the guide covers so not everything is transferable. Having each tool and feature separate to allow for an easily searchable document and one that can be used quickly for references was a key feature that we aimed to emulate.

2.4 Software Development Environments

Python Notebook

All of the tools within the guide, except for the notification postbox, were created as python notebooks. This was as it allows for very readable solutions which can easily be tested and ran by anyone who has access to conventional software development tools, or a browser. Python is a very readable language and is well supported by many libraries that make the development of the tools for the guide much more efficient. This allows the code developed to be easily read by most people who have some familiarity with code, allowing them to then understand how to take these tools and convert them to work within their own platforms. Additionally, if someone wanted to try using the tools to see how they worked before converting the code for their platform, by creating a python notebook the code can be opened in Google Colab without any specialist software, using a normal web browser. This gives the solutions high levels of readability and portability, allowing their utility to be maximised.

Notification Postbox

Developing a mobile application for Android offers two primary possibilities. The first is to utilise the C++ Qt library for UI and the second is to use the JVM (Java Virtual Machine) development environment offered with Android Studio. Qt is a multi-platform UI framework and is widely used for C++ projects whilst Android Studio is specific to Android app development and offers a wider range of tools for that purpose. Since the app will only need to function on Android then using Android Studio and its wider range of features makes more sense. Additionally, the documentation of Android gives examples in JVM languages.

The following question is which source languages to use. The two primary choices are Kotlin and Java and whilst the two are inter-operable only one language will be utilised to maintain a cleaner source and reduce the time needed for a team member unfamiliar with the code to get up to date. Kotlin offers features that Java does not that greatly aid in Android development, such as coroutines easing multi-threaded programming, though the language is fairly abstract and can be difficult to learn. In contrast, Java is a language that the entire team is familiar with from prior experience and alternatives exist for Kotlin exclusive features, though they may be conceptually different or more verbose. Since the app is small, and the benefits of Kotlin over Java and Java over Kotlin are marginal at that scale, the decision was made to use Java as the team already have a good understanding of the language.

3 Requirements Specification

For each major module of the deliverable, a set of requirements were established and justified based on the conducted research, to ensure they were representative of the project's objectives. These requirements served as the primary motivator throughout the design and implementation stages, where these stages could effectively be finished once these requirements were met.

Requirements have been given specific labels pertaining to the tool they are a requirement for, for the sake of better organising them and demonstrating their relevance during testing.

3.1 Notification Spam Filter

SF1. To output a trained model with 10, 100, 1000, 10000 training notifications within 1 minute, with a specificity>0.5 and sensitivity>0.5.

SF2. To be able to output a trained model with an extreme number of training notifications, 100000, within 3 minutes, with a specificity>0.5 and sensitivity>0.5.

SF3. To be able to make a prediction of whether a notification is important to the user or not within 1 second.

SF4. To produce models with specificity>0.5 and sensitivity>0.5, within 1 minute, training with various user's notification data.

Requirements Justification

SF1. The model should be scalable and not take long to build as this would be inefficient for a platform to repeat on a regular basis. The performance metrics should be better than randomly guessing the class of the notifications (0.5).

SF2. The model should be able to cope with an extreme user who has an unexpected amount of notifications. This is expected to take longer than the usual cases, but the tool should not run into an error.

SF3. Making predictions will be the most frequently used part of the tool. So, the tool should scalably be able to predict the importance of notifications quickly.

SF4. The model should be able to produce good models for various users, with performance metrics better than randomly guessing the class of the notifications (0.5).

3.2 Notification Postbox

The notification postbox tool is primarily targeted towards end users with less value offered to developers unlike the other tools developed. The reason for this is that it was conceived of and worked on before the scope was redefined with an emphasis on tools for developers to utilise. That is not to say that it does not have any utility for a developer – only that it functions more as an example or proof of concept rather than a directly usable tool that requires minimal modification for use.

Overview

The tool takes the form of an app that aids users in managing notifications. Notifications broadly act as an entry point to app use and the inbuilt tools for managing notifications are not widely known and therefore not widely used. The app should supplement these tools to provide additional options for managing notifications.

Following the principles of humane design that underpin the project the features offered by this app must remain under the control of the user and the user interface of the app must ensure that the user remains in control of what the app does and retain the ability to shut it off easily.

In the specification some terms are used that are specific to this tool, the following definitions are meant:

Intercept notification This specifically refers to adding to the app's notification store the notifications that are listened to and meet the requirements to be added.

System UI This refers to the host devices native UI, for instance home pages or swipe down system bars.

Cancelling a notification This is the process by which a notification is removed from the device. Typically this removes it from the system UI so that it is not longer displayed to the user.

Functional Requirements

PF1. The app listens to notifications sent by the system and ”intercepts” them if they meet user defined requirements.

PF2. Intercepted notifications are blocked from the system UI so that they are only visible in the app.

PF3. Intercepted notifications are added to a store of notifications that persists when the app isn't open or the device restarts.

PF4. The app does not need to be open to intercept notifications; it intercepts notifications regardless of if it is open or not.

PF5. The user can view details about intercepted notifications.

PF6. The user can remove notifications from the store so that they are no longer stored by the app or displayed.

PF7. The user can remove all notifications from the store.

PF8. The user can toggle whether the app will listen for notifications – disabling its collection of notifications.

PF9. The user can toggle whether the app will block notifications from the system UI – so that notifications are visible in the app and on the system UI.

PF10. The user can toggle whether cancelling a notification in the system UI also removes it from the app.

PF11. The app selectively intercepts notifications, if enabled, on an app-by-app basis which is controlled by the user.

Non-functional Requirements

PNF1. The app is designed in accordance to the UI conventions of modern apps.

PNF2. Users are guided on how to use the app through the design and explicit guidance.

PNF3. The app supports light and dark mode.

3.3 Exit Points

Requirements

EP1. This tool must facilitate the breaking up of infinite scrolling to allow user to limit consumption of new media

EP2. This tool must allow for the frequency of exit points to be customised by a developer

EP3. This tool should provide different options to a developer to allow them to select the most appropriate

Justification

The major problem that exit points seek to address is making the user experience more finite, this tool provides a developer with a frame work to base how often they should insert exit points within their platform, between the content of the platform. Without doing this the tool does not have the impact that it was intended for.

Providing customisability to a developer allows the tool to be tweaked to be as effective as possible within a platform and also allows for a developer to have as much control over how the exit points would affect the look and feel of their platform.

Similarly, providing a different options to a developer allows for exit points to have a frequency that varies greatly between potential solutions. This accommodates most use cases of that this tool could be used within. This ensures that a generic social media platform should be capable of picking up this tool and implementing it within their platform smoothly and in a way that maintains their desired user engagement expectations while also providing a more finite experience for their users.

What to use as an exit point is not given to developers as this will be a highly contextual decision depending on how a social media platform intends to show content to users, the types of content users see, and how the user interacts with this content, which is why this is not seen as a requirement for this tool.

3.4 Time Transparency

Requirements

TT1. To utilise a single, permanent data structure that contains content tag names at a maximum of 20 characters long with their respective seconds: an integer between 0-86400.

TT2. To use an update function that takes between 1 and 100 unique content tags and their respective timings, and update the existing timing for the corresponding tags. This function will take at most 3 seconds to compute for any valid data amount.

TT3. In the case that an input tag does not have an existing entry, the update function adds a new string tag to the permanent data structure. This function will take at most 3 seconds to compute for any valid data amount.

TT4. To use a function which takes the permanent data structure as an input, and outputs a bar chart plot of the top 10 highest to lowest tags, with the tag name being the x axis labels, and time spent as the y axis labels. This function takes less than 3 seconds to run on any number of valid data.

TT5. To demonstrate at least 1 method of allowing a user to choose a content tag to limit.

Justification

For the time transparency tool, requirements were established based on the research conclusions from section 2.3. For this tool to justifiably operate on a technical level, it would need to demonstrate the specific features that were identified. These features were reflected formally in the requirements above, and the justification for each requirement is as follows:

TT1. The research specified a need for storing time recordings, so a simple, effective method of storing this kind of data must be demonstrated: hence a single data structure. Each content tag is specified as 20 characters long as content tags are generally single words or short phrases, so 20 characters was deemed generally good enough to encapsulate this size. The time value as a maximum of 86,400 is because there is 86,400 seconds in a day, which is an effective upper bound to account for when recording daily timings. Seconds was chosen as a time measurement as this is a smaller resolution, which is more appropriate for a being sensitive toward wider range of social media content which can be both very short or long in length.

TT2. With this existing data structure for timings, there must be a means of updating this structure when a user views some content. As a single piece of content may have many tags, this update function must be able to account for this whilst updating the existing timings. ”1-100” tags is chosen as the range, as 100 tags is a reasonable maximum edge cases for the number of tags a single piece of content will have. A maximum of 3 seconds is how long this update function should take to process for any data amount because of general usability, as this tool would be deeply flawed if it could not scale effectively with a much larger social media platform, hence it should be able to scale well computationally.

TT3. Understandably, the update function should be able to handle the anomalous case when an inputted tag does not exist within the data structure.

TT4. As the tool is based on communicating the user’s time data visually, a way of constructing a bar chart using the timings data structure is required. This bar chart needs to adhere to what the research decided, so to specifically plot per *content tag* against the time spent viewing content with this tag. The top 10 highest viewed content tags shown as the user as the user should only be interested in the tags they spend the most time viewing, as these content types are making up the bulk of their screen-time. Only specifically the top 10 are display as to not bombard the user with too much information which will likely be redundant to their decision on what content to limit: to essentially keep the tool as user friendly as possible. Once again, this process should scale for any data structure size so that it can provably scale to any size social media platform.

TT5. A means of collecting the user’s choice on which content to limit is necessary, to complete the empowerment process. A developer should be able to know from this toolkit gener-

ally how to collect a user's choice, resulting from the visualisation, hence at least one way should be demonstrated to the developer.

3.5 Moderation Model

Requirements

M1. Moderation must be effective.

M2. Moderation must be automated.

M3. Moderation should avoid biases.

M4. Moderation should be fast and suitable for a large amount of throughput.

Justification

The moderation model needs to be effective, if it is unable to correctly identify obvious cases of abusive language and comments which use slurs then it will not have any positive impact on the platform it is used within. Poor moderation will have little to no positive impact on a platform.

The process must also be able to be automated, we want to provide something that can be relied on to run by itself and identify any posts or comments that will cause users to be offended or upset with minimal attention. This is as we want to have this tool as something any social media platform could implement or use regardless of the amount of resources they have as good moderation is a necessity to ensuring that a platform is welcoming and friendly place for a user to interact with.

The moderation model should avoid biases as we don't want the moderation tool to allow speak that negatively affect a group of people due to a bias held by the model. This would result in it being seen as okay within the platform's community to target this community as there would be no repercussions to this, therefore ostracising this community and affecting their mental health.

Finally the model should be able to cope with a platform which grows, therefore it needs to be able to work quickly and provide fast responses to a large number of queries. Otherwise the moderation tool risks leaving a platform feeling slow and unresponsive as it works through a buffer of posts waiting to be approved or could leave a large number of offensive posts and comments on the platform as it has not had the time to assess these and remove them from the platform.

3.6 Toolkit Guide

Requirements

G1. The guide must explain how to implement all tools.

G2. The guide must explain why the tools help mental health.

G3. The guide should be clear and easily understandable to a developer.

G4. The guide should offer guidance to a developer for how to implement the solutions on their own platform.

Justifications

The guide must cover all the tools, otherwise a tool that has been developed won't be properly talked about. Without doing this then a developer won't implement this tool into their social media as there would not be any documentation for how to do this.

The guide needs to explain the reason for the tools existing to justify to a developer why they should take the time develop the tools, and so they know what to expect from the implementation of that tool.

If the guide is not clear and understandable it would be harder for a developer to take the tools described in the guide and implement them within their own platforms. The aim of the project is to produce tools that a developer would use within their platforms, so making the process to implement these tools as easy and approachable as possible we are able to increase the likelihood of this occurring.

4 Project Management & Reflection

A project of this scale has many potential points of failure, and so these had to be avoided by carefully planning and managing how the project would be structured to succeed. This section covers the extent of these management considerations, and reflects upon their usage and efficacy throughout the project.

4.1 Customer & Stakeholders

This project is supervised by Mike Joy, a professor at University of Warwick. They are one of the assessors of the project, and as such they have power and influence with regards to the project's success. So, it was important to manage the relationship with them closely. Mike was kept informed of progress within the project during regular meetings with the team. Their opinion on the project was always requested and valued by the team, to provide the team with confidence that we were moving in the correct direction.

The project has a second assessor, Hakan Ferhatosmanoglu, who is also a professor at University of Warwick. Hakan's opinions were important to the team, because they too have the influence over whether the project is deemed a success. The only interaction the team had with Hakan was during a progress presentation in December 2021. They provided feedback and concern over the initial broad and unclear scope of the project. The team listened to these comments and dedicated time to produce diagrams to visually show how the tools fit together in the context of a social media platform, so as to show how the tools for a coherent and unified toolkit. The team hopes that this report explains well enough how the tools we have made relate to each other, and that the scope is clear. Monitoring Hakan's opinion has enabled the team to better achieve success.

Another important stakeholder is the project's customer, Zheng Fang. They are a PhD candidate at University of Warwick, who focuses on research into the development of topic modeling tools in social media. The team chose to work with them because they are knowledgeable and experienced in the area of the project; both with the problems with social media, and with implementing tools for used with social media. The customer helped to define objectives for the project. Additionally, they were kept informed about the project throughout, so that they could provide input about their expectations and opinions on design. Additionally, due to the customer being so knowledgeable in this area, they were used as a point of advice for this toolkit implementation.

4.2 Development Methodology

A development methodology had to be established to effectively structure the project's development. This methodology had to be chosen specifically to suit the project goals and be feasible to follow within the team to maximize the project's chance of success. There are many methodologies and management techniques available for a project, and this section covers which techniques and methodologies were chosen to organise this project.

Decision Making Process

One methodology used by the project was to have a clear decision-making process. This was established to aid pro-activeness of team members, because it would reduce the impact of decision paralysis, as there would a clear strategy for how to tackle a difficult decision with many options. The decision making process design is influenced by the multi-criteria decision analysis technique described in the Guide to the Project Management Body of Knowledge (PMBOK) Section, 8.1.2.4 Decision Making. [35] The design is enumerated below.

1. Choose a set of metrics relevant to the decision, e.g. cost, schedule, risks, and alignment with team values.
2. Weight the metrics.
3. List the options for the decision.
4. Perform multi-criteria analysis.
5. Choose the option with the highest score.

The design is justified below, using the exact process enumerated above – which acts as an exemplar for the decision making process used in this project.

Example of how to make a decision

1. Choose a set of metrics relevant to the decision

The criteria chosen are some team values relevant to this decision. These metrics are defined below:

Communication
: This refers to how much clear and efficient communication the option allows for.

Values Diversity
: This refers to how much an approach enables different perspectives and ideas.

Decision Making

This refers to how well the option meets the team's value of making transparent, efficient and provably effective decisions. [18]

2. Weight the Metrics

The weights of the metrics are shown in the table under each metric name, with x2 being worth twice as much as x1.

3. List the options for the decision

PMBOK's Section 5.2.2.4 Decision Making[35] suggests some decision making techniques, which form the options for this decision:

- **Voting**
  Group discussion of every decision.
- **Autocratic decision making**
  Allowing one individual to take responsibility for a decision.
- **Multi-criteria decision analysis**
  One person making the decision but has to offer any alternatives and justify their choice formally.

4. Perform multi-criteria analysis

The multi-criteria analysis is performed in table 1.

Table 1: Multi-criteria Analysis for Deciding Which Decision Making Process to Follow.

Option	Communication x2	Accountability x2	Values Diversity x1	Decision Making x2	Score
Voting	3/3 Whole group discussion.	1/3 Unclear who's fault a bad decision is.	3/3 Whole group discussion allows multiple views to be considered.	1/3 Time-consuming activity, which means that it is not an efficient technique.	13/21
	1/3	3/3	1/3	3/3	15/21

Autocratic decision making	Have to trust other people's judgement and cannot share input.	People know who has made the decision.	Only one person decides.	Good accountability leads to better standards and quality[35]; nobody wants to be linked to a bad decision. So the method is effective.	
Multi-criteria decision analysis	3/3 Clearly justified decisions explain exactly why a decision was made - avoiding disputes and gossiping.	3/3 People know who has made the decision.	2/3 Considering alternatives aids open-mindedness. But still the decision is only made by one person.	3/3 Transparent and proven to be effective.	20/21

5. Choose the option with the highest score

The option with the highest score is the multi-criteria decision analysis (as previously established). This concludes the example of how decisions were designed to made in this project. Following this technique would have ensured that transparent and provably effective decisions were consistently made throughout the project, aiding the team to be productive and, as such, aiding the project to be successful.

Methodology Choice

This project was a clear candidate for an Agile project management methodology because of the modularity of the toolkit and the fact that scope had to be estimated, while time and resources were fixed. The toolkit was to be made iteratively by adding more tools as the project goes on. Each iteration the project was to have made a complete toolkit, since the tool made in the iteration is independent from others, and so integration is easy.

Agile is an umbrella term of project management methodologies. This section shows the consideration that the team made with regards to which approach under the umbrella was the most suitable fortheproject. The decision was made using the decision making process defined above. The metrics, by which the approaches were compared, are defined below:

Reliability of Workflow refers to how the team has other commitments (like courseworks) throughout the duration of the project. This metric considers the approach's suitability to a team with an inconsistent amount of time to work on the project.

Meeting a Fixed Deadline refers to how well the approaches enable the team to meet the strict deadlines involved in this project. One measure of success for the team is reaching schedule targets, so it is important that the approach can satisfy this.

Sureness of Scope refers to how the team initially had a vague idea of scope. This metric considers the suitability of each approach with regards to handling an unclear set of features.

Measuring Success refers to how easily the approaches allow the team to validate scope.

Bureaucracy refers to the amount of meetings and organisation involved with each approach. The team does not want excessive bureaucracy in order to maximise the flow of work.

The three Agile approaches that were considered for this project were the following. These were chosen because the development team had the most experience with these options.

- **Scrum**
  Work is organised into sprints. Items from the product backlog fuel each sprint backlog, defining how much work should be done in a particular sprint. All work is planned at the start of the sprint.
- **Kanban**
  Work is organised into a pull system, where work is done when resources are available.
- **Scrumban**
  Work is organised into sprints. Work is pulled from the sprint backlog during the sprint, like a Kanban board.

Table 2 shows the multi-criteria analysis performed for the decision of which project management approach was the most suitable for the project.

Table 2: Multi-criteria Analysis for Deciding Which Project Management Methodology to Use.

Option	Reliability of Workflow x2	Meeting Finxed Deadline x2	Sureness of Scope x1	Measuring Success x2	Bureaucracy x1	Score

Scrum	1/3 May not be able to reliably commit to do an amount of work in the sprint, as the team has other commitments (i.e. varying amounts of cswk).	3/3 You plan towards long term goals.	2/3 Scrum needs an initial vague list of features (product backlog), which we have. But, we don't know the sub-scope of each feature, as it will be planned at the start of each sprint, so if a feature turns out to be more interesting or complex during development, then we won't be able to deviate from that specific feature's plan.	3/3 Sprint review meetings allow the team to incrementally measure success. Scope is continually validated throughout development.	1/3 Daily meetings to discuss progress would be time-consuming and possibly futile as the team may not be able to work on it everyday. Sprint review meetings would help the team to continually improve and learn how to function more efficiently as a team.	17/24
	3/3	1/3	1/3	1/3	3/3	14/24

Kanban	Allows the team to work on the project whenever they are able to. This ties with the Lean Muri principle, that people cannot work at an unsustainable pace i.e. avoiding the overburdening of courseworks and the project.	There is no overall long-term goal to reach with Kanban, so meeting a fixed deadline will not be guaranteed.	Hard to estimate how much scope will be completed by the end of the project.	There is no end point/-goal in Kanban, so measuring the success of the project will be difficult, as there is no plan to compare progress to.	No excessive planning or paperwork. Team members just do work when they get time to, rather than wasting time in meetings.	
	2/3	3/3	3/3	2/3	2/3	19/24

Scrumban	Without the sprint being planned, it may be difficult to motivate the team to start work on the sprint goal, when they have other commitments. But the flexibility of Scrumban means that the team members do not have to commit to an amount of work, just do as much as they can in the time-scale, with other members being able to take on extra tasks.	With Kanban, the aim is to streamline the delivery process, so the amount of work in a sprint will be more when compared to the more bureaucratic style of a normal Scrum sprint. Scrumban is a leaner version of Scrum, therefore work can be completed faster than in Scrum – at worst at the same speed.	Scrum is ideal when work is in large vague chunks, like in this project. But the Kanban element gives flexibility during a sprint to take a feature further (or restrict it back more) than perhaps we initially thought we should.	The lack of planning in each spring means that measuring the sprint's success could be difficult, as the work may change throughout the sprint. However, sprint review meetings mean that the team will frequently discuss progress of the product backlog.	Do not necessarily need daily meetings, as work is organised in Kanban. Just need a meeting at the start and end of a sprint for planning and review respectively.	

The best scoring option from Table 2 is Scrumban. This allowed the team to confidently move forward with Scrumban as the agile project management approach for this project. Naturally, the approach has been tailored to suit the needs and the style of the team. The details of this approach are detailed in the following subsection.

Methodology Design

The modified Scrumban approach for this project is illustrated in Figure 7. This section explains the diagram and justifies any changes made to Scrumban.

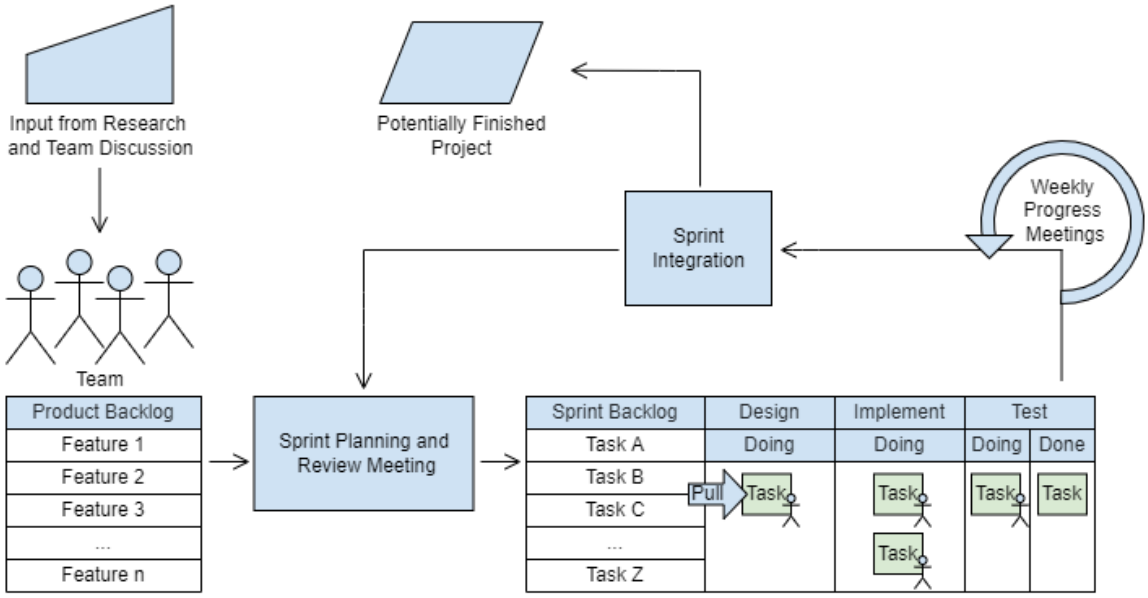


Figure 6: The Scrumban project management methodology to be followed throughout this project.

Input and Product Backlog

The background research about problems within social media fed the product backlog, which contained a list of high-level feature descriptions for this toolkit to include. Once the product backlog was established, the team could start the first sprint.

Sprint Planning Meeting and Sprint Backlog

Each sprint begins with a sprint planning meeting. After the first sprint, this meeting also reviews the previous sprint. The review stage of the sprint meeting involves reviewing the toolkit, suggesting possible improvements where necessary. The review was also to celebrate the achievements of the team during the sprint, so as to motivate the team to continue their efforts.

The planning stage of the sprint involves choosing a feature to work on, as well as breaking that feature down into more precise, yet still generally vague, tasks to be completed during the sprint. This sprint backlog of tasks can be added to or removed from during the sprint if required, though this was discouraged in order to inspire a more thorough planning phase at the start of the sprint.

Kanban Board

The sprint backlog tasks are pulled into action on the sprint's Kanban board. The first phase is to perform the deign work needed for the tool or task. Once this is done, the task is pulled into implementation, either by the same or a different team member. After implementation, each task is tested. For some tasks this involves technical testing of some code implementation. For other tasks, this is simply a case of measuring success of the activity by validating objectives. For example, a task of finding and preparing a suitable dataset does not involve any functional testing. The success could instead be measured on how similar the dataset is to the design plan for the required dataset.

The original Kanban board design can be seen in Figure 7. During the first sprint's review meeting, the team decided that the 'done' sections for design and implementation were overly bureaucratic. This is because usually the same person saw the task right through the board. The board was redesigned to remove the 'done' columns, in order to make the team more efficient.

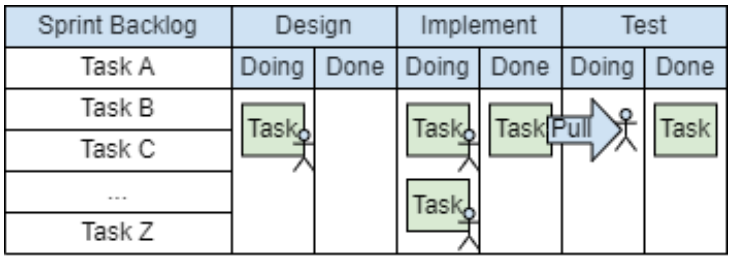


Figure 7: The original Kanban board design of this project.

paragraphWork-in-Progress Limits

The Kanban board design limited the amount of work-in-progress to two tasks. This was to prevent all team members from, for example, pulling all of the tasks from the sprint backlog and designing them, but never implementing or testing them. This would halt progress, and the sprint could end with lots of design work but very few implemented parts of the product backlog feature. The work in progress limit means that no more than two tasks can be waiting to be implemented or tested at a time. This kept the workflow moving, as the team had to pull a task that needed to be implemented before designing any more tasks.

Weekly Progress Meetings

One of the modifications made to Scrumban for this project is that there are no daily stand-up meetings during the sprint. The reason for this is that the team had an unreliable workflow, with other commitments sometimes taking precedence over the project. This meant that the team could not working on the sprint backlog everyday. As a result, the meetings would be a waste of resources, as there would be no progress to discuss on most days. However, the team did instead have weekly meetings to talk about progress and motivate ourselves for the following week of the sprint.

Sprint Integration

The work performed during each current sprint had to be integrated with the rest of the project. This involved making sure tools were complete and adding them to the relevant toolkit directory. This also involved writing a short manual for how to use the tool.

Potentially Finished Project

Each sprint ended with polished set of work where code had to have been added to the toolkit directory, commented and formatted to a professional standard.

4.3 Team Management

Team Roles

The team for this project consisted of only four members: Lewis Buttle, Matthew Jameison, Daisy Kennedy, and Patrick Michaels. For this reason, the team decided to not be overly bureaucratic with the assignment of roles within the team, choosing to avoid assigning roles for activities in which all team members took part. For example, establishing that all members were on the development team was unnecessary; the responsibility for each team member to work on the toolkit is to be assumed.

One role that was assigned was that of the project manager, which was volunteered for by Daisy. Having good team leadership allows a team to know who is accountable for clear team communication. For example, Daisy was held accountable for communication failings, like a team member not knowing what time a meeting was and therefore missing a session. That is not to say that the project manager was responsible for organising meetings; just accountable for any problems with communication surrounding that. Other areas that the project manager was accountable for were:

- Ensuring fairness during the handling of inter-team disputes. For example, during disagreements, votes were called by the project manager so that the problem could be solved quickly and moved on from.
- Making sure that when risks materialised that fast action was taken to minimise the damage.
- Checking in with team members to see if anyone was struggling, and trying to find ways of helping them if they were.

Another necessary role for this project was a scrum master. The scrum master role was more of an active role than project manager because they had to:

- Organise group meetings during sprints.
- Maintain contact with the project supervisor during sprints.

So that all team members could experience a leadership role, the scrum master role was to be circulated between Lewis, Matthew and Patrick.

Project Delegation

A benefit to the project being so modular in its technical deliverable was that the work could be effectively delegated amongst a team, thus parallelising the workload and maximizing value throughput. At the start of the project, the members of the team discussed their individual talents, experience and interests. This information was used to better delegate the work amongst the team, so that each team member was in charge of working on a deliverable they were specifically interested or experienced with. This was done to help motivate the team to complete the tools on time and to a higher standard, as this would be more achievable for a teammate that is interested and experienced with the technical area their work is involved with.

A risk with taking this approach was that if a team member was limited to working on a specific module of the deliverable, then then vastness in technical area of the tools could limit the help that a teammate could receive. This is because the other teammates may not have the research or background knowledge to begin to help a struggling teammate, and the work would become very secular and independent. Hence, great care was put into ensuring that the entire team remained aware of the entire project's scope and deliverables, and the research and design of each of the tools. The overhead of ensuring the team spent the time keeping up to date with every tool in the project was deemed a worthwhile trade off for risk mitigation. In the case where a teammate was halted in their implementation, then another teammate was able to have the expertise to lend a hand with little resistance.

Communications

The team held regular meetings to plan and discuss progress and pace. These meetings were sometimes held in person and sometimes held online. This was mostly dependent on the team members' availability. In preparation for meetings, the team would add to an agenda points that they wished to discuss or that the team as a whole felt needed to be addressed. The team took it in turns to write the minutes, this was an unspoken arrangement but was effective. The minutes are included with this report as Appendix B.

During the second half of the project, the team members had much busier schedules. As a result, the frequent meetings became more of a bureaucratic formality than an effective communication point. During this time, the team began using Slack more heavily to communicate. This was because it was a low-effort and more concise way of updating the team and planning

next-steps. The Slack is separated into channels for relevant conversation topics, such as: #team-updates, #meetings and #proj-presentation (see Figure 8). This helped to keep conversations clear and organised, which was helpful for anyone who was not part of a particular conversation to keep informed.

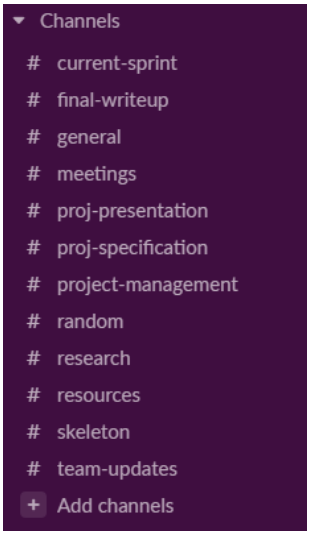


Figure 8: Screenshot of the channels used in the team's Slack.

Many of the team's interactions were formal and work-focused. This can lead to a lack of playfulness, which is known to result in a less productive team. [18] To encourage camaraderie, the team made a channel in the project's Slack specifically for team jokes (#random). This channel allowed team members to laugh about silly mistakes and vent their frustrations about difficult parts of the project. This created a stronger sense of belonging to the team, in turn leading to a more productive team.

Team Development Strategies

The team utilised team exercises and techniques to ensure team cohesion was robust. This benefited the project through better management and better utilisation of the team as a resource. For example, team members often held in-person meetings for short but rigorous work sessions, that involved each team member working on their designated section but within a communal, social setting. This had the benefit of team members feeling more involved in the overall project and preventing the team from feeling disjointed. Additionally, this gave an opportunity for team members to share their progress; chat about the state of the project; and give advice and help to one another. These communal environments served as informal meetings throughout the project, allowing the team to better synergise, and feel more comfortable as a team. This technique also helped balance the project with the busy academic lives of the team, as these sessions could be organised quickly in-between academic responsibilities, as long as the team were close by.

One way the team encouraged personal development was through allowing each member to experience a leadership role. Whether that was project manager or scrum master, the team members go to experience responsibility and learnt the difficulties of managing a group of people. This prepares the the team members to feel empowered to volunteer for leadership roles in the future.

4.4 Scheduling

Scheduling the project's timescale was necessary for planning exactly how the project would be completed, as well as justifying that completing the project was feasible within the given time. This section explains how and justifies why, the schedule changed throughout the project life cycle, mainly to adapt to the change in scope that occurred after the first sprint.

Original Schedule

An original schedule plan was specified at the start of the project in the form of a Gantt chart (see Figure 9), which was designed specifically to complement the chosen scrumban methodology. The schedule suited the chosen scrumban methodology as it broke the work down into modular sprints, with each sprint dealing with a different identified mental health issue with social media. Originally, the first sprint would focus on tools relating to *reducing screen-time*, sprint two would focus on tools relating to *user's engagements with social media* and the final third sprint would focus on *issues with moderation*.

Each sprint was originally allocated a similar length of time for completion. The first sprint was allocated slightly more time to account for the overhead of initially setting up the project and the development environments the project would be based upon. Originally, the scope included developing the tools on existing-open source social media environments. However, this changed early on in development due to the recognition that applying these tools within a social media platform did not suit the project's objectives (justified further in Section 1.2).

To further suit the scrumban methodology, the exact deliverable timespans included within this schedule were intentionally vague, as the methodology planned for the exact deliverables to be researched and created exclusively within the sprint. Hence at the start of the project the exact tool deliverables were unknown. Nonetheless, time was allocated for researching, designing and implementing these tools. To mitigate the obvious risk of this vague scheduling, the concept of *float time* was included within the schedule. Float time served as a buffer between the sprints (coloured in green in the Gantt chart), and helped by accommodating for the unforeseen issues that could occur throughout development, such as a sprint overrunning, as this float time could be borrowed safely to mitigate any overrunning. To reflect, this float time was very effective in allowing the project to stay on-track as it was used as intended. This float time allocation suited a project, like ours, that could potentially change slightly in scope throughout development to

mitigate the overhead of these unforeseen changes.

Overall, the original schedule heavily benefited the management of the project as it gave the team a fundamental, initial idea of how the project would run over the year. From this schedule, the entire project could be justified as feasible to succeed within the project's time frame.

Sprint	Start Date	End Date	Objective	Fundamental Activities
1	01/11/21	13/12/21	Sprint Initiation	Sprint 1 Initiation Meeting
			Preliminary Implementation	Produce API Skeleton
				Initiate Database
			Feature 1 - Reduce Screen Time	Research Objective
				Design Feature
				Implement Feature
				Test Feature Functionality
			Sprint Transition/ Risk Aversion Time	Evaluate Sprint 1
	14/12/21	02/01/22		Reserved Float Time
				Sprint 2 Initiation Meeting
2	03/01/22	31/01/22	Feature 2 - Reduce Unhealthy Engagements	Research Objective
				Design Feature
				Implement Feature
				Test Feature Functionality
			Sprint Transition/ Risk Aversion Time	Evaluate Sprint 2
	01/02/22	13/02/22		Reserved Float Time
				Sprint 3 Initiation Meeting
3	14/02/22	14/03/22	Feature 3 - Enlist Effective Moderation	Research Objective
				Design Feature
				Implement Feature
				Test Feature Functionality
			Sprint 3 Closure/ Risk Aversion Time	Evaluate Sprint 3
	15/03/22	04/04/22		Reserved Float Time
	29/10/21	29/10/21	Module-related Deadline Preperation	Specification Document
	06/12/21	11/12/21		Progress Presentation
	04/05/22	04/05/22		Final Reports (Group & Indiv)
	09/05/22	16/05/22		Final Presentation

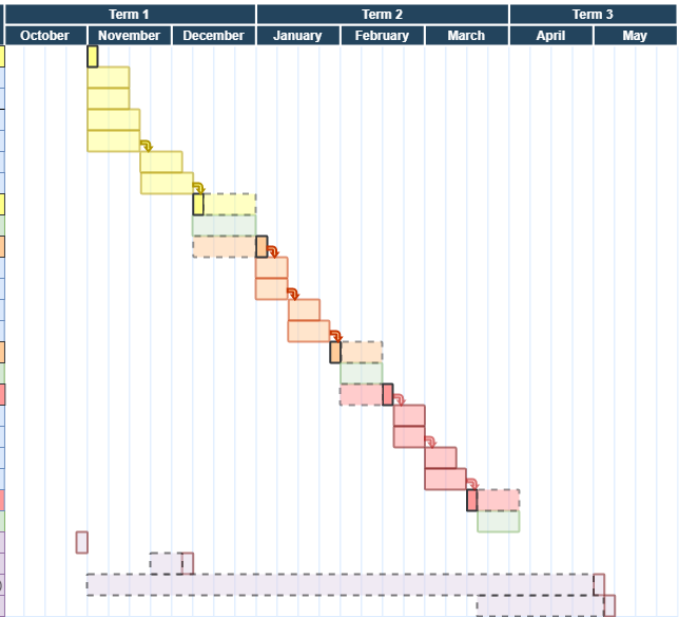


Figure 9: The original proposed project schedule.

Revised Schedule

The original schedule was correctly followed until the end of the first sprint however, following the project's progress presentation in December, the project entered a short phase of reflection to consider and act upon the feedback given by the project supervisors.

One major concern was that the project's success may suffer from focusing on too large a breadth of tool deliverables, without a clear route of how they all link. Additionally, it was felt that focusing on too many separate deliverables would be difficult to manage without extensive management. This management overhead would be damaging due to the project's limited timescale and workforce, as more time would be spent managing the distribution of work on these tools, rather than spending this time actually developing deeper, useful tools that suit the project's motivation. Hence, the team responded to this feedback in three ways:

1. Sprint two and three were merged into a single sprint to encourage focus on having less tool solutions with more technical depth to them, instead of a breadth of tools. Essentially, by merging the sprints, more time was now available to develop more technical tools. Furthermore, this new merged sprint would only deal with the theme of *moderation*. This decision was made due to the team agreeing that the second sprint's theme based on *"improving user engagements"* was vague in its nature, and was a fault during

the project's specification process that this theme was not more thoroughly researched and explained, though by erasing this theme and giving its time to the better understood theme of *moderation*, this encouraged more technical and novel solutions for this project to pursue.

2. For each tool, the individual technical depth of each tool would be considered rather than taking each tool to their maximum depth as originally intended. This depth would be decided based on how useful it would be to demonstrate a tool's core idea. For example, it may be ineffective for a smaller tool to be fully implemented within a social media platform, if the tool could be demonstrated to a developer just as effectively through a less expansive solution. By considering this, the depth of tools could be individually considered to maximize a tool's impact, which helped further manage where time could be spent to maximize the value of the deliverables within this project.

3. Thirdly, the time gained from removing the third sprint was used to partly extend the first sprint, to allow for the newfound revelation of wanting to provide more depth to tools to be applied to the existing tools developed during sprint one so far.

It is worth noting that these large happened a third into the project and did not damage the project's success. This was only possible because of the flexibility allowed via the scrumban approach. As the approach allowed the project to make effective progress towards the final deliverable without needing to fully know the entire consistency of the deliverables, this meant that changes affecting the schedule could be implemented with little loss to the project's entire value. Additionally, it can be seen how the planned float time also helped mitigate this sudden change during development, as it meant the unforeseen overhead required to reschedule the project's sprints were well mitigated within this float time, without wasting any of the project's planned time resources.

In summary, major changes occurred a third way into the project's development to the schedule, acting upon the feedback given by the supervisors, and these changes allowed for the project's entire value to drastically increase, and the overhead of these changes were well mitigated due to the project's methodology and planned float time. Taking the time to re-calibrate the schedule to maximize the project's value in response to feedback, rather than continue down the planned route heavily benefited the outcome of the project, as the final tools were able to be much more potent in their technical impact to solving the overarching project issues.

4.5 Risk Management Strategy

Risk was a key factor to consider when managing the project, as the strict time scale of the project meant that any encountered risk that could prolong the outcome of the project, would highly affect the project's likelihood of succeeding. Hence, risks were closely analyzed and managed

throughout the project to avoid them or at least dampen the damage of any that were encountered.

Risk Analysis & Contingency Planning

At the conception of the project, a risk analysis table (Table 3) was constructed to identify the potential risks that could occur, and detail how these risks could be avoided or mitigated.

Risk	**Risk Type**	**Risk Management**	**Risk Response Type**
A team member is unable to continue working on the project	Team	Reducing the scope of the project to compensate to prevent burn out in remaining team members	**Mitigate**
A planned feature is not as feasible as initially expected	Scope	Ensure a feature is fully researched before committing to its development, to avoid time being wasted on trying to implement a solution that would be out of scope	**Avoid**
Team member does not complete a task by set deadline	Team	Keep tabs on team members and utilise a pull system to make sure members are not overworked. Be prepared to de-scope the project.	**Avoid**
Team member keeps missing meetings and not staying in communication	Team	React quickly and contact them via Email, Slack and Messenger. If still no update, inform the supervisor.	**Accept**
Dataset is not suitable	Resources	Another dataset would have to be found or Gathered. Existing datasets for Social media content exist, by aggregating several we would lessen the impact of a single dataset being unusable for the project	**Mitigate**
Team is unproductive/unmotivated	Team	The project management plan for the team is founded on values proven to be indicative of a productive team	**Mitigate**

Team members have a disagreement that halts progress	Team	Hold an in-person meeting ASAP with the whole team. Define exactly what the problem is. Allow all parties to express their stance, encouraging honesty and clarity. Brainstorm solutions. Choose a fair solution, expect to compromise in some aspects.	**Accept**
A team member is struggling with a task they have started	Team	First, ask for help. The team members should prioritise their own tasks, and if resources are available, aid with the struggling task. If the struggling task is of low priority and there is not an excess of resources available, consider removing it from the scope.	**Accept**
A team member is struggling with a task they have started	Team	Plan to use pair programming for the most difficult tasks, so that the pressure of such a task is not put on a single person.	**Mitigate**
Work is deleted	Resources	A backup should be made of all work that is not made on cloud services, as soon as the work is completed. This should limit the amount of work that could be lost.	**Mitigate**
The scope is completed much sooner than anticipated	Scope	Extra possible scope should be thought of at the start of the project, with the deliverables organised using the MoSCoW method (extra scope is labelled as C – could do if there is enough time).	**Mitigate**
The scope is completed much sooner than anticipated	Scope	Hold a meeting to discuss whether to work more on existing features or to come up with new features.	**Accept**
System requirements are not adequately defined and so the team is unsure what features to develop and how	Scope	Use an Agile project management methodology, to allow for changes to requirements during development	**Mitigate**
Ineffective communication within the team	Team	Assign a project manager and hold them accountable for encouraging and facilitating good team communication	**Mitigate**

A technology being used is difficult to adjust to and eats into development time	Resources	Clarify the technologies that the team are equipped with, and establish everyone's abilities early on. Only plan for activities which use these resources	**Mitigate**
A sprint deadline is approaching but the sprint backlog is still very full because the team moved slower than expected	Resources	Plan for float time after each sprint, with the first sprint having the longest float time because this is the sprint with the most uncertainty. The sprint review meeting will help to estimate the pace of the team. Prioritise the sprint backlog at the start of the sprint so that the most important tasks are completed first. Consider de-scoping the feature.	**Mitigate**

Table 3: The risk analysis table.

Backup and Recovery Strategies

A main point of risk mitigation was version control and keeping the deliverables backed up, to mitigate the possibility of losing work from such issues as hardware failures or accidental deletion. The team considered this issue when deciding upon the development platforms for the tools, in which one software development environment chosen was online (Google Colab). It provided automatic backing up to cloud services throughout development, thus providing a simple means of backing up development work.

For the postbox notification tool, this was developed on a local development platform as the tool was a mobile solution, although the code was backed up using version control via *Git*. Using this version control method allowed for better bug fixing, due to being able to check through previous versions for potential fixes if a new version was having issues with errors.

Risks Mitigated

To reflect, this thorough understanding of the potential project risks helped mitigate risks throughout the project's life cycle. A few examples are given in this section.

A planned feature within sprint 2 was taken out of the project's scope due to the conclusion that the tool was infeasible to be produced effectively to meet the project's goal. Namely, this tool was going to deal with the issue of *slacktivism*: the idea that charities online suffer from interactions

on social media, which causes them to lose funding. However, after researching the background for a tool idea to deal with this issue, it was decided that the resulting tool would be non-technical and not suit the nature of demonstration for developers that this project aimed to achieve, hence this tool was ceased to be worked on and no more time was spent designing or developing the tool. Additionally, the issue's theme did not suit the project, and was straying from the goal of dealing with a general user's mental health, as this tool was considering simply the quality of interactions for a user with no reference to their well-being. Essentially, this decision was made due to the risk response claiming to **avoid** the situation of a planned feature becoming infeasible and wasting project time, by conducting effective research into this tool and spending the time to consider this feasibility. By following this process, the weak slacktivism tool was able to be prevented and more project time was able to be allocated to more valuable development within the project.

Team disagreements were also mitigated, as the analysis was able to be followed correctly to prevent the project development being halted by a disagreement. This was done by following the risk analysis and holding meetings immediately to discuss any issues, and resolving them as soon as possible as a team. An example of this was deciding the languages to develop the tools on, where a meeting was held to agree that Python could be used for the majority of the tools as the team had a synonymous level of experience with the language. By holding this meeting immediately after having this issue, the project's development was able to continue quickly with no damage to the project's outcome.

Risk analysis was also utilised when a team member was having issues implementing their solutions, as the table states that team members should be vocal to the team about issues as soon as possible, where this can be mitigated by putting team members on to the problem to help get past the issue. This occurred numerous times throughout development, especially during sprint 1 where some team members had issues starting to implement their tools. This time waste was mitigated through holding programming sessions, where the team would meet up in person and spend some time each working on individual allocated work, but with the shared resource of the team available in person to ask for help or advice. This was effective at keeping the project moving at a steady pace, as any pertinent issues could be dealt with together as the team during these programming sessions.

4.6 Measuring Project Success

For the project to succeed, the formal definition of project success had to be considered. This sections details this consideration given for project success.

Technical Tests

Due to the methodology of the project, the major tool deliverables were to be identified, researched, designed and implemented throughout the project life cycle. These tools had to be justified to be relevant to solving the social media issue at hand. This was achieved effectively by setting formal requirements for each tool and relating them to the issue research conducted, whereby these requirements are the formal definitions of what the tool should be as to be proven effective at solving a specified issue. These requirements are derived for each tool within Section 3. A formal means of testing how successful these tools were, was to derive a test plan which tests technical functions of the tool, where each test directly related to the derived requirements. This was done as a means of bridging a formal proof that if the tool succeeded in these tests, then it had passed the set requirements and therefore was successful in what it aimed to achieve.

These technical tests served vitally to measuring the success of each tool, and were an effective technique at ensuring the tools were always strictly relevant to the ulterior project goal, as long as the tool could aim to pass these derived requirement tests. These tests are formalized within each tool's relevant design section, within Section 5, and were performed by the team after implementing the tool.

User Acceptance Tests

To have an informed idea of the entire project's success, it was deemed suitable to perform user acceptance testing with the final deliverable: the toolkit guide. This test would gather opinions specifically from university peers with a background understanding in software development as to the efficacy of the toolkit. They are asked about their opinion for each tool about two main aspects:

- How effective the tool *resource* is, if they were to implement the tool themselves.
- How effective the tool *idea* is, and if they feel it would be effective for helping a user's mental health.

As both of these main points link closely with the project's ulterior objectives, the opinions from this overall testing phase were used to evaluate and conclude how effective each tool is for considering mental health, as well as how well the guide has achieved in being a useful resource for developers. The questionnaire provided these questions via a linear 1-10 scale, as to how effective the tester feels the guide has met both of these criteria, per tool. The project aims to receive at least a majority of testers rating each question as 7+ to provide an indication as to which tools within the project succeeded. This testing was the least important for justifying the entire project's success, but is used to evaluate the efficacy of each tool and gain a general idea to the success *per tool*.

The primary indicator of overall project success is based upon the customer's opinion: the final user acceptance test that will be conducted. The project's customer is planned to be presented the final deliverable and asked for their honest feedback. The customer will be asked how successful they believe the toolkit guide is in the project's aims, and if it is what they expected. Additionally, they will be consulted for their opinions on each tool, to see if they have any issues or thoughts on specific tools. This feedback is used to conclude the project's success, as the customer is the primary stakeholder for this project. If they are able to define the deliverable as what they expected, and they believe it is an effective resource, then this would be direct evidence that the project has succeeded.

4.7 Legal, Social, Ethical and Professional issues

Considering tools which are heavily involved with taking control or directly affecting a user's mental health does innately produce some questions about human ethics, such as ensuring the tools are sensitive and never in a position where they can damage a user's mental health. However, due to the project's scope of only treating these tools in an abstract sense where they are not to be used directly or implemented directly within a user base, these ethical questions did not need to be answered during development. For the notification postbox tool, there are a few considerations outlined in its own design section as they are specific to that tool.

The project did not need to consider many other areas of these issues. During the stages of user acceptance testing, all the testers are to be kept anonymous and no personal data is kept. This same concept is applied to the machine learning models constructed throughout the project (namely the moderation and spam filter tools), in which the data sets used to train these models are to be gathered fairly, and data is anonymised where necessary.

5 Design & Implementation

Each of the major deliverable tools were thoroughly designed using the conducted research, to ensure that each tool was effectively on track to meet their individual requirements justified in Section 3. Due to the varying technical fields of study, each tool had a unique design and implementation process, and this section reports this process in detail per tool. Additionally, the design and implementation process of integrating these completed tools into the ulterior toolkit guide is detailed. As well as general design decisions, the feasibility, social and ethical issues, possible expansions, portability and limitations of each major deliverable are considered and explained.

5.1 Notification Spam Filter

Scope

The tool **must** train and output a model using an input of labelled notifications.
The tool **must** take an unlabelled notification and output a prediction for its label.
The tool **should** explain how to gather and format training and test data.
The tool **should** provide guidance on maintaining the tool.
The tool **could** be joined with a front-end tool to produce a complete platform. The tool should before this be a complete, independent tool. Joining with a front-end would be an extension task if there is time.
The tool **would** be tested in-place as a platform if there was enough time to both join it with a front-end and have a long testing period to compare, for example, how the tool performs when the model is updated on a daily or monthly basis. Such testing is unachievable within the timeline of this project.

Tool Design

One of the resources found in the research section was 'Machine learning for email spam filtering: review, approaches and open research problems'. [12] This resource dictates there are two stages to handling the data for email classification: pre-processing and feature selection. Pre-processing involves tokenising the emails. Tokenization is the process of transforming an email into meaningful components, e.g. the header, the subject and the body of the email. Feature selection involves choosing the tokens that best represent the emails and influence their likelihood to be spam. The tokens may be transformed to be more useful by a model. The end result is a feature vector that represents the email. These stages are expected to be good guides for the steps needed to prepare notification data for a machine learning model.

After preparing the notification data, the next stage is to build a model. The resource mentioned above provides a comparison of classifiers used in email classification. This is discussed thoroughly in the model building section below. The resource helps to justify the design choices of that section.

Tokenisation

The meaningful parts of a notification, like 'Bob Bobbington added a new photo', are Bob's name and the fact that he has added a photo. This section discusses the ways in which this tokenisation of notifications can be carried out.

Both Android and IOS assign channels to notifications.[6][7] A channel is essentially a notification type. For example, if you have been tagged in a post it may come under 'Tags'; if you

have been notified about an upcoming event it may come under 'Reminders'; and if you have been notified that a friend has posted a new photo it may come under 'Updates from friends'. This is useful for the tool because it summarises what the notification message is and places it into a category. using this, we can just have a feature for the channel of a notification, without any extra design to be discussed, because the channel name can be grabbed straight from the notification data.

Another token of a notification is the users that it refers to. For example, who tagged you in a post; who posted a new photo; and who invited you to their event. This token is not neatly provided by the notification interface, like the channel token was. This token would need to be extracted from the whole notification message. For example, one would have to extract Bob's name from 'Bob Bobbington added a new photo'. The way that this could be done would vary depending on the platform being used. For example, if a new social media was made which used this tool, then it could output the name(s) directly as a list when a notification is sent. Alternatively, if the tool is used as an exterior add-on to an existing social media, then the friends list of the user could be cross-referenced with the notification message, like a look-up table for names. For the implementation of this tool in this project, automatically grabbing the names and tokenising them is out of scope. This is because they would rely on interference with the front-end of a social media platform, when this tool is meant to be back-end. As a result, tokenising the names used in a notification is left as a implementation-specific extension to the tool, for gathering the input data for the model. For the sake of this tool's implementation in this project, it is assumed that the data has been gathered by an external tool and is provided to this model. In the case of this project, the external tool is manually tokenising notifications.

Table 4 shows some examples how the notifications are, and should be, tokenised for this spam filter tool.

Table 4: Design for Tokenisation of Notification Messages.

Notification Message	**Channel**	**Names**
Bob Bobbington added a new photo.	Updates from Friends	[Bob Bobbington]
Bob Bobbington, Carol Carolton and 5 other people reacted to a post that you shared.	More activity about you	[Bob Bobbington, Carol Carolton, other people]
Reminder: You have an event coming up this week.	Reminders	[ ]

Feature Selection

Recalling one of the research findings, we can describe spam notifications in two ways. One where spam is defined by the type of message it is, i.e. the notification channel. And the other where spam is defined by the type of message it is, as well as the user(s) it is about. The layout of the corresponding feature vectors is given below:

1. [channel]
2. [channel, [names]]

Type one is easier to implement. This is because there is only one feature and it is of static size. In comparison, type two has two features, and there are a dynamic number of names for each notification. With regards to the technical experience of the team, a dynamic length feature has not been implemented by the team members before. Therefore type two is a more difficult choice.

However, from the team's personal experience with using social media, the importance of a notification does tend to rely heavily on who the notification is about. For example, an update about a distant uncle is not something we would want to be notified for. Whereas an update about a close friend is something we would want to be notified for. The subjective importance of a notification does usually depend on whether or not the people involved are relevant to us.

That being said, some notification channels are outright unimportant to some users, regardless of who the notification is about. For example, if a user has not been explicitly tagged in a post, then they may not want to be notified about it. One reason for this is because they may want to reply to notifications to seem polite, but do not care about the notification if they are not obligated to do anything. In this scenario, type one would be sufficient to make a prediction of importance.

Overall, there is an argument for both feature vectors. Type one is simpler and is sufficient in some prediction scenarios. Type two would produce a more complex model that is able to make better predictions on new notifications. An additional benefit to type two is that the team would have the opportunity to learn how to handle dynamic features in machine learning. In the interest of making a model with the best performance (see the notification spam filter requirements about specificity and sensitivity scores), type two is chosen.

Notification Model Trainer Design

The research section sourced a paper which deeply discussed the current machine learning techniques used in email spam filtering.[12] This is used to inform which classifiers should be used in this project for this tool. A summary of the paper's findings about different classifiers and their use within email spam filtering is found below. Along with the classifier descriptions is, where

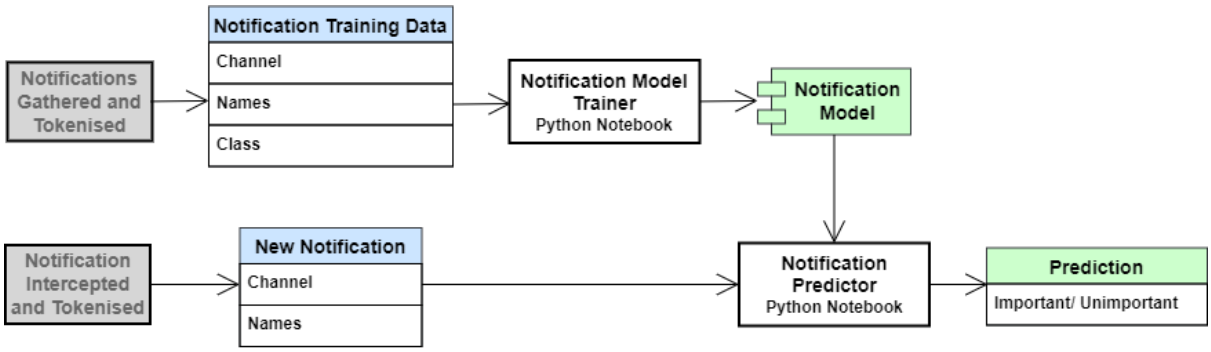


Figure 10: Flowchart design for spam filter tool. Gray areas are pre-processing to be done external to the tool. Blue are inputs to the tool. Green are outputs from the tool.

available, a link to a Python library for potentially implementing the model. Classifiers with libraries are easier to implement than those without, as they do not need to be programmed from scratch. The presence of the library is an influential factor in the design of the model builder.

- K-Nearest Neighbours (KNN)
  KNN has a very fast training phase. However, all training examples must be stored as the model, meaning that the size of the model is the size of the training data.
  https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

- Naive Bayes
  Naive Bayes is effective and robust, as well as computationally efficient.
  https://scikit-learn.org/stable/modules/naive_bayes.html

- Firefly Algorithm
  With Firefly, the model improves with each new email, so the model gets more accurate over time.
  It does not have a Python library for it, so would need manual implementation.

- Rough Set
  Rough Set is efficient as it is not time-consuming for it to extract hidden patterns in the data. It is best for use with imprecise and noisy data.
  Does not have a Python library for it, so would need manual implementation.

- Support Vector Machine (SVM)
  A weakness of SVM is that is not as fast at training as other algorithms. However, it achieves high accuracy because it is able to model multi-dimensional data. The paper recommends performing a grid search on the values for the parameters C and the the kernel.
  https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

- Decision Tree
  Decision Trees are particularly good with non-uniform data. The output is very human-

readable, unlike other models.
https://scikit-learn.org/stable/modules/tree.html

- Neural Networks
  Gmail uses 'state of the art' Neural Networks in its email classification. These can be difficult to implement, especially deep ones, but they perform very well with email classification tasks.
  A simple neural network implementation: https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html

Choosing a classifier for the notification spam filter depends on the situation for the model. Email spam filtering, for which the models above are justified, is generally done using one large model as a predictor for all user's emails. This is because there is an objective definition for email spam. However, with notification spam, the definition is subject to the opinion of each user. As a result, one must consider whether one large, generic model is suitable for this tool or whether having a different model for each user may instead be necessary. The design decision is made using multi-criteria analysis, to compare and rank each of the choices against each other (see Table 5). The metrics consider how much training data is needed for each choice, as the project is limited by how much can be gathered for testing because the notifications have to be manually tokenised by the team. Another metric is about the expected performance and how effective such an implementation would be at reducing a user's trigger to open their phone. The difficulty of implementation is also considered, as the tool should be designed with the future goal of being easily integrated with a social media platform. From Table 5, it can be seen that the better design choice for this tool is to produce **a different model for each user.**

Table 5: Multi-criteria Analysis for Deciding One Spam Filter per User or One Spam Filter for All Users.

Option	**Amount of Training Data Needed x3**	**Implementation Difficulty x1**	**Efficacy of Reducing Habit Trigger x2**	**Score**
	2/2	1/2	2/2	11/12

One per User	The project would only need enough training data for each user. The project will only test on a few users. So, the amount of training data needed is kept to a minimum.	This would involve continuously updating the model based on the user's new interactions, as personal opinion changes over time.	Tailoring to each user would likely result in less false-positives, meaning the user would receive less non-important notifications.	
One for All Users	1/2 The tool would need training on a large population of data to produce the singular large model, which would then be predicted on by a few users in the population. So, the amount of training data needed is much more than in option 1.	2/2 The model would be made once using a training set and then released as an objective model.	1/2 Having this generic model would be less effective because some users could receive notifications that are non-important to them, because the training dataset could be full of a certain demographic.	7/12

An important aspect of making a model is choosing a classifier that suits the data. However, since the spam filter tool is to be used to create a different model for each user, it is impractical to expect each model to have a manually selected best classifier. As a result, the model builder is designed to build multiple models with different classifiers and compare them automatically. The model builder should then output a single model. This decision is feasible because each user will only have a small amount of data (relative to the amount of data for the whole user base), and so training multiple models on each user will not take much time.

In terms of which classifiers to train the spam filter with, it is decidedly those discussed above which have a Python library implementation. All the classifiers would suit this tool because of the overlap between email spam filtering and notification spam filtering. However, to keep implementation time low, classifiers that have to be produced from scratch are being avoided.

To summarise, the model will train on the following classifiers:

- Decision Tree
- KNN
- Naive Bayes
- SVM
- Multi-layer Perceptron

The models trained by the tool are to be compared automatically using multi-criteria analysis, with metrics as specificity and sensitivity. Specificity is weighted as x3, and sensitivity is weighted as x2, to give an overall score out of 5. The comparison is designed like this because the project's **customer** expressed a preference that the spam filter tool focuses more on labeling unimportant notifications correctly, than on labeling important notifications correctly. That is, that the true negative rate (specificity) be a more important factor than the true positive rate (sensitivity) when comparing models. This is in contrast to the preference for emails, because with emails it is better to let through occasional spam ones, than for the user to miss important emails. With notifications though, according to the customer, it is 'less likely for [there to be] serious consequences for not reading social media posts'. The goal for the spam filter is about improving mental health by reducing the notification trigger, so it is better to over-reduce it in this scenario, than under-reduce it.

The spam filter model builder has now been designed in more depth, and so its part of the flowchart, from Figure 10, is able to be broken down further to visually show the new details of the design (see Figure 11).

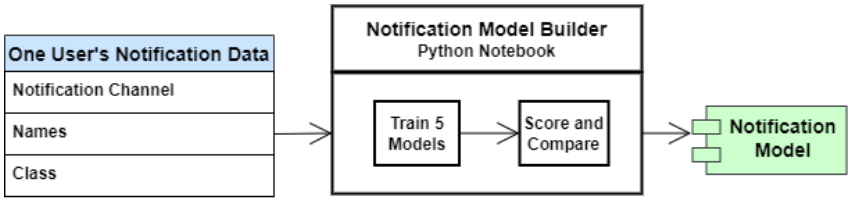


Figure 11: Flowchart for notification spam filter's model builder. Blue is a CSV input to the Python notebook. Green is the output of the Python notebook.

Notification Model Trainer Implementation

The decision tree was implemented using numerically encoded feature vectors of the form X = [channel, name]. The features were encoded so that the Python decision tree library could handle the data, as it can only work with numbers, not words. Notifications with multiple names were split into separate training examples for each name, so that each feature vector only had

one name. The decision tree was trained on some data collected about Daisy's notifications – this was done just for experimentation during the implementation stage. The trained tree was visualised and can be seen in Figure 12. X[0] represents the channel feature, and X[1] represents the name feature. In the visualised tree, statements like, 'X[1] <= 4.5' can be seen. In English, the statement means that the tree should branch if the name is less than 4.5. This suggests that the order of the names affects the importance of a notification, which it does not, since the names are categorical not ordered. I.e. a notification about Alice is not more important than one about Bob because her name is closer to the start of the alphabet. It became apparent that the decision tree was interpreting the encoded data like it was numeric instead of categorical, leading to models that did not make sense and would likely not perform well once deployed. A new approach had to be considered.

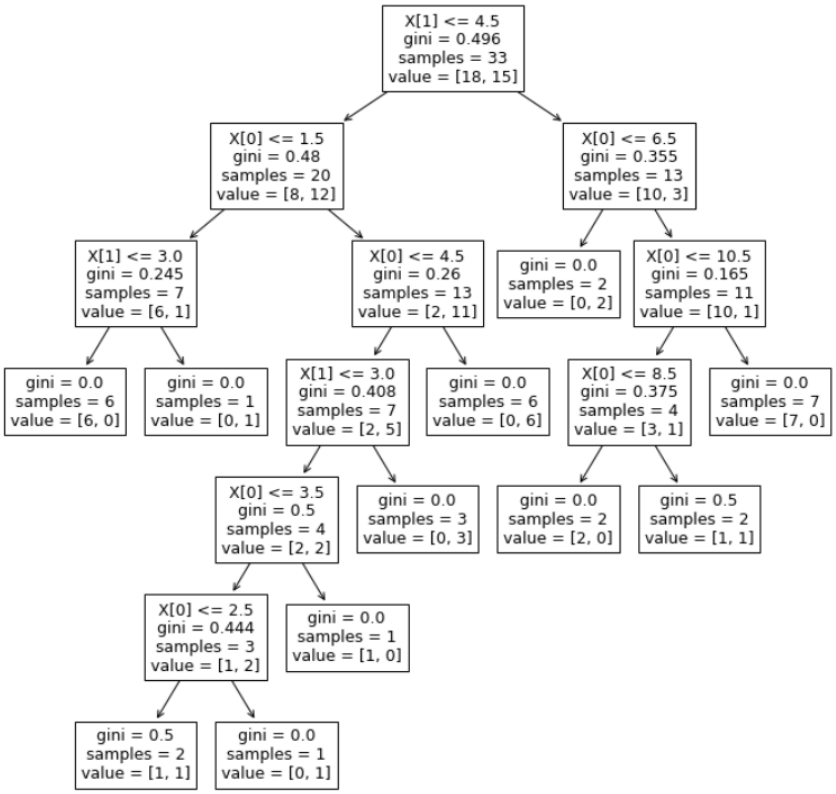


Figure 12: Initial Decision Tree implementation for spam filter tool.

Some research was done into how to handle categorical data. The sklearn.ensemble .HistGradientBoostingClassifier() was found to have native support for categorical features.[40] Unfortunately, the resulting model seemed to predict not-important for all notifications. The data did not suit this classifier either.

Another way of handling categorical data is to restructure the data to have each category as a binary feature. See Figure 13 for the original format of the data, and Figure 14 for the restructured binary features version. To binarise the channel column in Figure 13, pandas.get_dummies() is used because the function employs one-hot encoding, which is ideal because all notifications are only assigned one channel (see the final two columns of Figure 14 for the one-hot channel cate-

gory columns). The names column in Figure 13 is binarised using sklearn.preprocessing.MultiLabel Binarizer() (see the second, third and fourth columns of Figure 14 for the binarised name category columns). This function allows multiple binary columns to be flagged as 1, which is ideal because there can be multiple names in a notification. Restructuring the data into this form allowed the Python classifiers to treat the data as categorical. For example, see Figure 15. The statement 'Tags <= 0.5', in English, means that if a new notification is flagged as the Tag channel, then the notification is important. As a note, whilst the tree says '<= 0.5', it really means '== 0', since the columns are binary; either a 0 or a 1.

Row	Notification Channel	Names
1	Comments	Bob Bobbington. Charles Charlington
2	Tags	Emma Emmington. Bob Bobbington
3	Tags	Charles Charlington

Figure 13: Notification input format to spam filter tool.

Row	Bob Bobbington	Charles Charlington	Emma Emmington	Comments	Tags
1	1	1	0	1	0
2	1	0	1	0	1
3	0	1	0	0	1

Figure 14: Notification input pre-processed for spam filter tool.

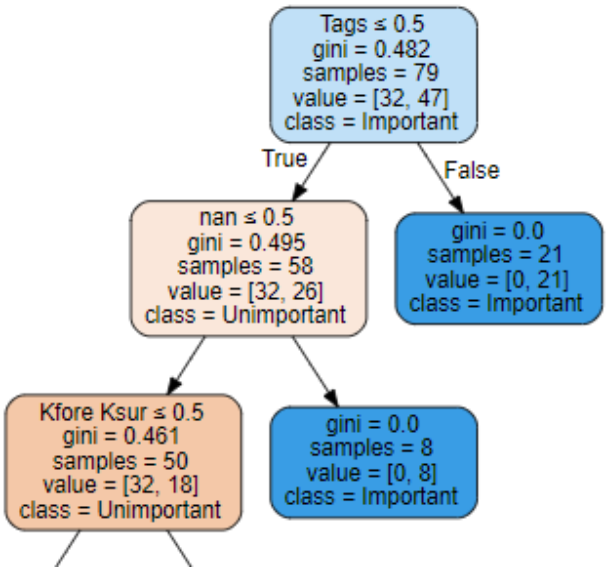


Figure 15: Portion of the improved Decision Tree implementation for spam filter tool. Tags is channel category. Nan is a name category for notifications with no names. Kfore Ksur is a name category of an anonymised name.

After the pre-processing of the data, the Python classifiers from the design are each trained. An additional classfier was included, the sklearn.ensemble.HistGradientBoostingClassifier() which

had native support for categorical features. This is because it sometimes performed better the the other classifiers, and so was an easy addition to the model builder.

As mentioned during design, grid search can be used to improve a model by choosing the best parameters for a classifier. A separate function is used for each classifier to grid search specific parameters. For example, to find the best number of neighbours to use in the KNN. The grid search is performed using sklearn.model_selection.GridSearchCV(). The best model for each classifier type is then compared against the the other classifiers' models, using the designed score metric. Finally, the model is saved in a joblib file using the joblib.dump(). A further detailed flowchart for the implemented model builder portion of the tool can be seen in Figure 16, which visualises the implementation description from this section.

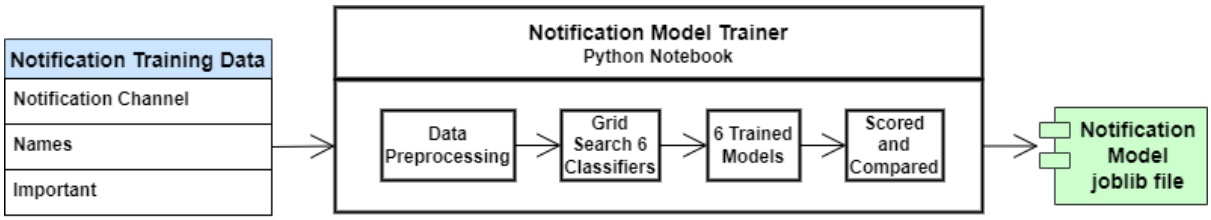


Figure 16: Flowchart for the notification spam filter's implemented model builder. Blue is a CSV input to the Python notebook. Green is the joblib output of the Python notebook.

Notification Predictor Design and Implementation

The predictor part of the tool is relatively simple. The best trained model is loaded into the predictor notebook using joblib.load(). An intercepted notification can then either be loaded from a CSV file or manually defined in the notebook. The new notification is pre-processed to match the feature vector shape of the training data. This is done by using the model's attribute 'feature_names_in' to gather the known categories to set them as the feature vector column headings. The new notification then sets the categories to true or false. Once the feature vector is of the correct format, the trained model is used to predict the importance of the notification. The prediction is outputted to a CSV file. A visualisation of the predictor's implementation can be seen in Figure 17.

Notification Spam Filter Tool Integration and Maintenance Design

The spam filter tool is a back-end tool for predicting notification importance. The guide produced by this project explains details on how to use it and how to integrate it with a social media platform.

As note on maintenance, in the predictor, if the newly intercepted notification includes a channel or name that is unknown to the model, the category is simply ignored because the feature vector of the prediction has to be the same format as the training feature vectors. This is a limitation of the model; it does not predict on unseen features. Hence, the model for each user should

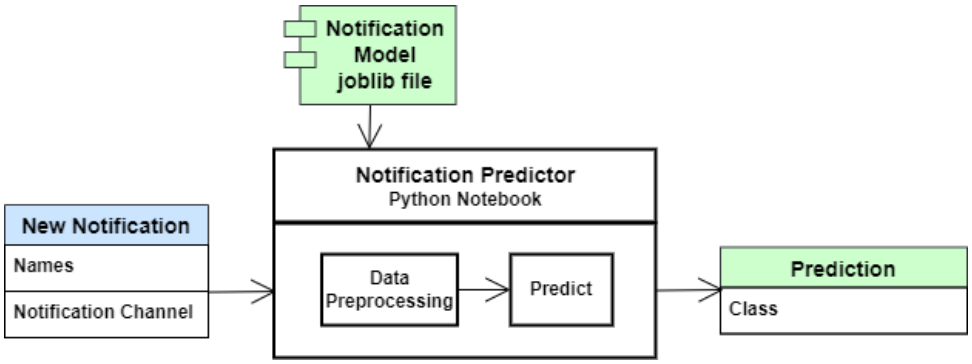


Figure 17: Flowchart for the notification spam filter's implemented notification predictor. Blue is a CSV input to the Python notebook. Green are the joblib outputs of the tool. The joblib file is the output from the model builder. It is used as input for the predictor.

be rebuilt regularly to allow for new data. One design for the maintenance of this tool is to store whether a user opens each notification in the relevant app. If it is not opened, then it is flagged as unimportant. At the end of each month, rebuild the model with the user's month of data. Then use the updated model to make the next month's predictions. This will help to ensure the tool evolves with the user's preferences, leading to a continued successful performance.

Test Plan

The final stage of the tool development is to test its performance. The Spam Filter Tool is to be tested by measuring performance of training the model on various input CSVs. The input CSVs were produced by manually collecting, anonymising and tokenising the social media notification messages received by the members of the team during one day. Daisy's notifications were replicated to produce the larger training sets that will test scalability of the tool. It was infeasible to collect enough notifications to test the scalability of the tool, as requirement SF2 specifies the tool should be scalable up to 100,000 notifications. So, it was deemed acceptable to replicate notifications to make up the numbers. The CSVs still represented real data because lots of notifications received are not unique; users tend to interact with the same people over and over again.

Table 6 shows the specific tests to be carried out to measure the success of this tool. Each test is linked to one of the spam filter tool requirements, as meeting these requirements is the basis for the success of this tool. The tests are designed to cover whether the requirements have been met. The test descriptions refer to the CSV files on which the model is to be trained for that test. These CSV files can be found in the testing sub-directory for this spam filter tool. Some test descriptions also refer to joblib files that are to be used by the predictor to predict the importance of a new notification. The joblib files are produced by the training with the CSV files, and so once testing has begun, they will also be included in the testing sub-directory for this tool.

Table 6: Test Plan for Notification Spam Filter.

ID	Req.	Test Description	Expected Output
1	SF1	Train model with 10 notifications. Using 'Test 10.CSV'.	Takes < 1 minute, Specificity>0.5, Sensitivity>0.5
2	SF3	Use 'Test 10.joblib' model to make a prediction on 'New Notification.CSV'.	Takes < 1 second
3	SF1	Train model with 100 notifications. Using 'Test 100.CSV'.	Takes < 1 minute, Specificity>0.5, Sensitivity>0.5
4	SF3	Use 'Test 100.joblib' model to make a prediction on 'New Notification.CSV'.	Takes < 1 second
5	SF1	Train model with 1000 notifications. Using 'Test 1000.CSV'.	Takes < 1 minute, Specificity>0.5, Sensitivity>0.5
6	SF3	Use 'Test 1000.joblib' model to make a prediction on 'New Notification.CSV'.	Takes < 1 second
7	SF1	Train model with 10000 notifications. Using 'Test 10000.CSV'.	Takes < 1 minute, Specificity>0.5, Sensitivity>0.5
8	SF3	Use 'Test 10000.joblib' model to make a prediction on 'New Notification.CSV'.	Takes < 1 second
9	SF2	Train model with 100000 notifications. Using 'Test 100000.CSV'.	Takes < 3 minutes, Specificity>0.5, Sensitivity>0.5
10	SF3	Use 'Test 100000.joblib' model to make a prediction on'New Notification.CSV'.	Takes < 1 second
11	SF4	Train model with only important notifications, 'Test All Impnt.CSV'. Use 'Test All Impnt.joblib' with 'Impnt Notif.CSV'.	Output'Important'
12	SF4	Train model with only important notifications, 'Test All Impnt.CSV'. Use 'Test All Impnt.joblib' with 'Not Impnt Notif.CSV'.	Output 'Not Important'
13	SF4	Train model with Daisy's notifications, 'Daisy Notifications.CSV'. This contains a redundant column with the whole notification message in, to check the tool does not error when given unusual input.	Takes < 1 minute, Specificity>0.5, Sensitivity>0.5

14	SF3, SF4	Use 'Daisy Notifications.joblib' model to make a prediction on 'New Daisy Notification.CSV'.	Takes < 1 second
15	SF4	Train model with Lewis's notifications, 'Lewis Notifications.CSV'.	Takes < 1 minute, Specificity>0.5, Sensitivity>0.5
16	SF3, SF4	Use 'Lewis Notifications.joblib' model to make a prediction on 'New Lewis Notification.CSV'.	Takes < 1 second
17	SF4	Train model with Patrick's notifications, 'Patrick Notifications.CSV'.	Takes < 1 minute, Specificity>0.5, Sensitivity>0.5
18	SF3, SF4	Use 'Patrick Notifications.joblib' model to make a prediction on 'New Patrick Notification.CSV'.	Takes < 1 second
19	SF4	Train model with Matthew's notifications, 'Matthew Notifications.CSV'.	Takes < 1 minute, Specificity>0.5, Sensitivity>0.5
20	SF3, SF4	Use 'Matthew Notifications.joblib' model to make a prediction on 'New Matthew Notification.CSV'.	Takes < 1 second

Limitations

One limitation of the spam filter tool is that it cannot be tested to the full extent for which it is designed. For example, the design describes that the tool should be updated once a month. The project does not have time to monitor multiple months of notifications on various users, because the tool is a back-end implementation and so all inputs have to be manually recorded and tokenised from devices. The tool is still to be tested thoroughly using the test plan, but completely testing its capability to reduce screen-time is out of the scope of this project.

Extensions

The next steps for this tool are to link this back-end implementation to a mobile application that can intercept and block notifications when they are predicted as unimportant. Resources for this are provided in the tool guide in the further considerations section for this tool.

5.2 Notification Postbox

Whilst the motivation and requirements of this tool have already been laid out this section aims to establish the bounds of those requirements and the limitations in how they can be fulfilled, as well as the path through which they will be achieved and the tools and design patterns that are utilised.

Scope

Whilst the motivation of this tool is split between examining alternative notification systems from a development perspective and providing a means for end-users to better control the notifications they receive, there is only one specific implementable. From the specification, it is clear this implementable is an app that must be able to listen to notifications and interact with them in the specified ways.

There are two main operating systems for mobile devices that can be targeted with this app, Android and iOS. It is not possible to easily implement the core interception behaviour on iOS as the operating system securely handles push notification delivery and sandboxes apps from each other. Potentially this security could be violated at a low level though this would not be a reasonable development path to take.

On the other hand, Android does allow for applications to listen to notifications from other apps. For privacy and security reasons the user must grant this special permission (`BIND_NOTIFICATION_LISTENER_SERVICE`). Since it is not possible to implement core functionality of the app for iOS then development must target Android.

Ethical Considerations

There are a few security and ethics issues to be mindful of, some of which are partially or mostly handled by Android and others that the developer remains solely responsible for. In the first category is user consent for what your app does. Whilst Android is more permissive that iOS for allowing apps access to data it requires the user to explicitly grant permissions for any service that is potentially damaging. Sending push notifications does not require permission (until Android 13) but accessing location data does. Listening to push notifications from other apps falls into the latter category.

When requesting permissions from users Android has a prompt system that makes requesting the permission easy for a developer and recommends, especially for certain permission types, displaying the rationale behind requesting the permission, for example that location data is needed to know which shops are nearby. Provided the developer follows this guidance in requesting permissions then the app cannot act against the bounds the user has allowed, ensuring that the ethical obligations are met.

The other ethical hurdle to consider is that the content of push notifications from apps can be sensitive. Banking apps often notify when standing orders or large transactions occur and fitness and health apps may reveal medical information. This app is required to store notification data and therefore potentially store information that should be kept private. To ensure allowable behaviour there are two requirements that the app should be in compliance with. The first requirement is that privileged data should not be classified, processed, or otherwise utilised outside of the specific purpose of the app – to store notifications for later display.

The second is that data storage and transfer must be secure so that a malicious attacker cannot gain access to it. This will be achieved by not forwarding the data beyond the device. Many Android applications use a data store to persist information between sessions and a common choice is a remote database as it allows user data to be consistent across multiple devices, however this requires sending information across a network creating security concerns. An alternative is to use the device's local storage. Android grants every app storage space that is only accessible to that app. By utilising this space only this app and someone with access to the physical device could extract the data.

Validation

To verify that the behaviour is in line with the specification requires testing. Most of the functional requirements can easily be covered with automated testing that verifies correct implementation. However, not all elements can be adequately mocked for automated unit tests, nor would automated testing completely cover the specified behaviour. An example is component integration and UI elements as both require too many individual components for robust testing. Or more specifically that notifications are blocked from the system UI as automatically fetching this state is challenging.

For these elements manual acceptance testing is necessary. For the UI this is verifying that UI elements feel responsive and that interacting with the UI causes the appropriate response, for example deleting a notification or changing a switch's display state. This testing is aided by the unit testing, as knowing that the disparate parts correctly functions ensures that this tests only the interface between them. Additionally, the UI layer of an Android app should not have business logic and so unit testing is not advised since the setup is more involved and the value lower. [3]

The non-functional requirements of the project specify three things. That light and dark mode are supported is trivially verified and can be ignored for now. The other two requirements relate that the user should be guided on using the app and that the app should follow conventional UI design principles.

The design of the app following good UI conventions will be enforced through the use of standard Android UI libraries, and, when competing UI ideas are present, utilising A/B tests for acceptance. This will be an ongoing process throughout the development of the app. The other

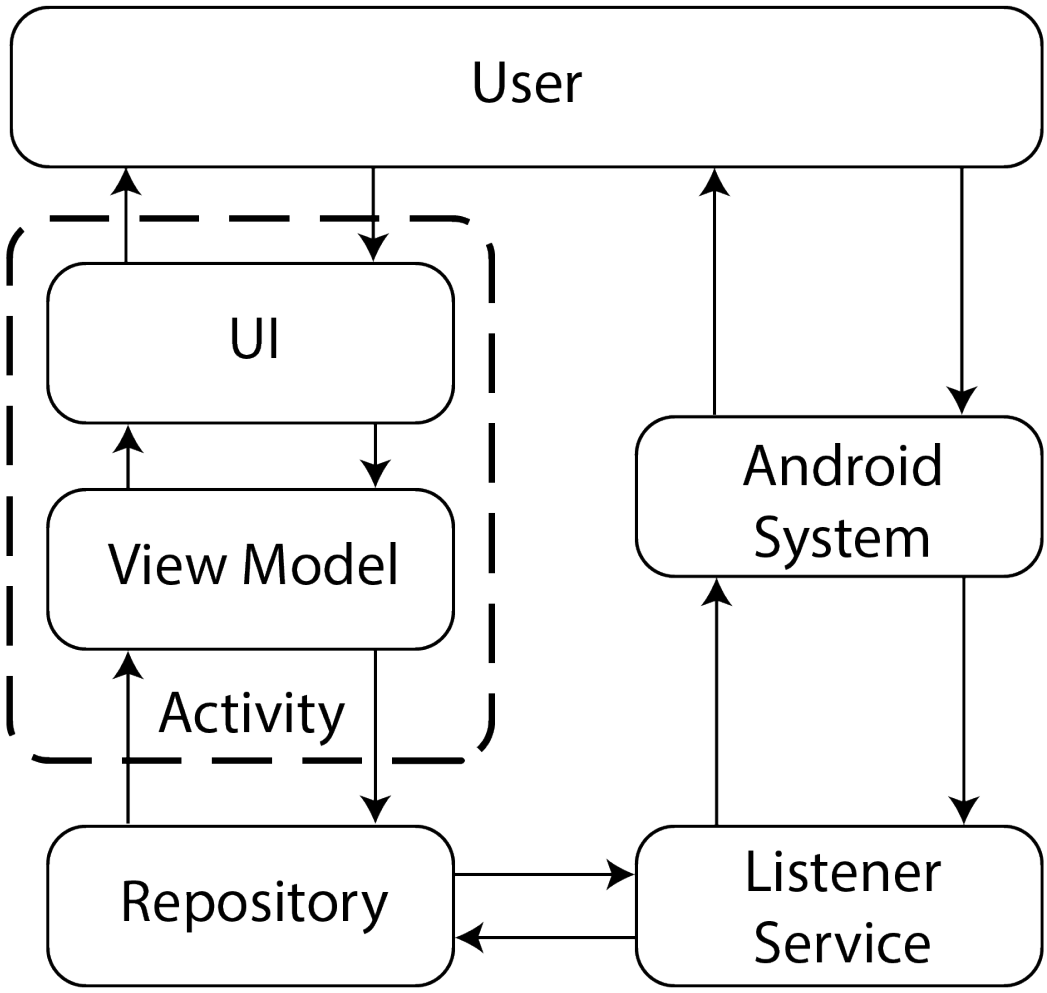


Figure 18: Architectural Overview of the Notification Postbox App

non-functional requirement specifies that users can easily use the app without external guidance. This is most easily tested through providing the app to new users who have not been involved in the development process but are familiar with apps in general to perform the common operations of the app.

Architecture

The app, at a high level, consists of two elements: an 'activity' that the user interacts with; and a service that listens for notifications. The service is distinct in both function and lifecycle to the app as it must run anytime the phone is on to intercept all notifications. The app's lifecycle is much shorter as it's core elements are tied to the activity – that is the app being open – and UI elements may be deleted and recreated as needed by the operating system in that time, therefore an interface must exist between the service and the activity and this is the data repository.

Figure 18 shows the outline of this architecture where user interaction are with the activities' UI and the Android system (e.g. through other apps). The Android system provides notification

data to the listener service which then interacts with the repository. The actual app also interacts with the repository through a view model. This architectural pattern is common is Android development, as it separates concerns leading to a more modular codebase.

The Android documentation suggests conceptualising an app as having layers. A UI layer that displays information to the screen, a data layer that contains the business logic and exposes the data to the application, and an optional domain layer that simplifies the interactions between the UI and data layers (when multiple view models are affected or the business logic is very complex). [4] For this app, the UI layer will consist of the UI classes composing the layouts in the activity and the repository and view model classes expose the data forming the data layer. As the service is outside of the application it interacts with the data layer directly.

The specific realisation of this general architecture is already hinted at by the view model. The notification data will be displayed according to the MVVM (Model, View, View-Model) pattern. This pattern specifies that the data model of stored notifications will be displayed in some view (a UI class) and the two elements will exchange data via a view model, decoupling the components to allow easy interchangability and testability.

The MVVM pattern is well supported by Android and a LiveData class exists to facilitate it. LiveData is intended to store data at the view model level and it is an instance of the observer pattern; when the underlying data is updated (e.g. through the repository) all view that are displaying ('observing') the data have a callback executed to update their view. This way changes to the data in the view model propagate to the UI. The UI and repository can also update the LiveData when needed, such as a notification being cancelled or a notification being intercepted and added.

The repository class acts as an abstract interface for the storage of notifications. The view model ensures that the UI has a clean interface to change the data and ensure its display is up-to-date, but the repository is the interface to persistent storage. This pattern abstracts the actual storage mechanism for notifications so that if the specification changed and notifications needed to be stored in a remote database all changes would be in the repository class. Alternatively, if in the future a developer wanted to extend the application to work across multiple devices the repository would also interface with a web-hosted data store as well as local caching of the data. Whilst implementing these directly into the view model is possible it makes the view model grow beyond its needed scope and having an abstract interface to the underlying storage mechanism also means that the repository can act as a single source of truth.

User Experience & UI Design

Android apps are composed of activities. These activities define how the user interacts with the app for certain purposes. Activities then have a UI set and handle the code callbacks from interaction with the UI. Every app has at least one activity – the 'main' activity which is the activity into which the app is launched, though they can define others that may also act as entry-

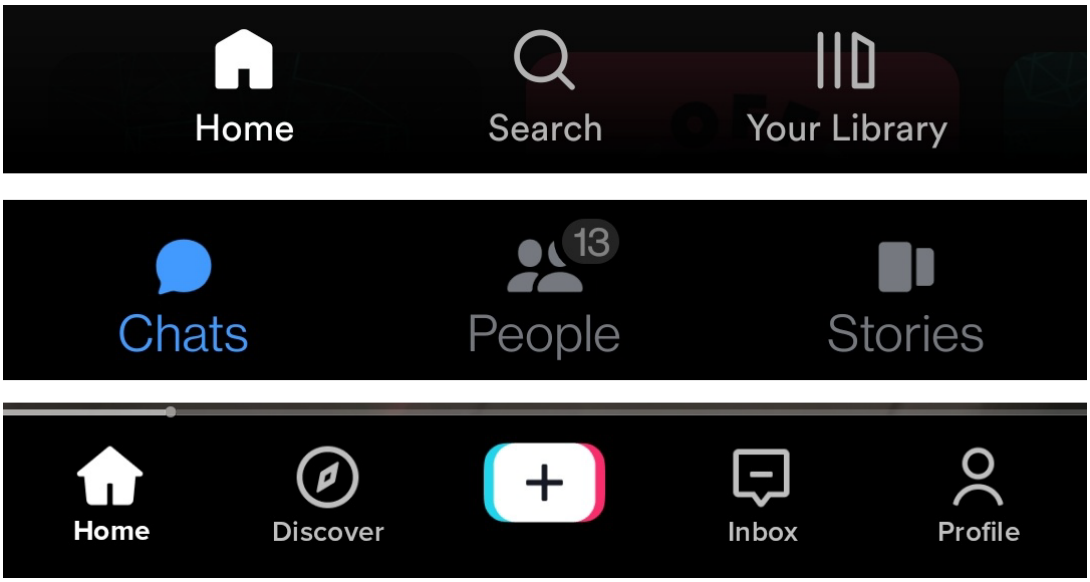


Figure 19: From top to bottom, Bottom Navigation Bars in the Spotify, Facebook Messenger, and TikTok Android apps

points into the app.

Sometimes an activity needs to have distinct UI elements on screen with complex internal behaviour that may be reused. This is where fragments come in. A fragment allows the definition of a UI element or group of elements that must be hosted in an activity. A common use for fragments is when the activity needs to handle several related tasks that do not need to be different activities and so hosts a fragment container and a navigation UI. The fragment container can then host different fragment, showing different content to the user, based on the interaction with the navigation UI.

This pattern will be used for the notification postbox app. The activity will utilise a bottom navigation bar and host fragments for more specific tasks. Bottom navigation bars are a common navigation component used in apps. The bar typically consists of three to five destinations with icons and labels indicating what each one is for.

The three destinations that the app will use are: an inbox, to view the notifications that have been captured; a settings page, to allow the user to alter the behaviour of the app; and a debug page, this is mostly for testing to allow notifications to easily be generated. An alternative that was considered was using the standard top bar to allow users to access a navigation drawer, however the bottom navigation bar is easier to access as it is always visible and is more common than navigation drawers for social media applications so users will be more familiar with it. If the debug screen was removed the bar might be overly sparse though the addition of a new navigation destination would ease this. Mock-ups of the display and settings fragments are shown in Figure 20.

Of the three fragments, the first is the inbox fragment. This is the UI the app should open to and it should display a scrollable list of notifications. At the top is guidance on how to delete

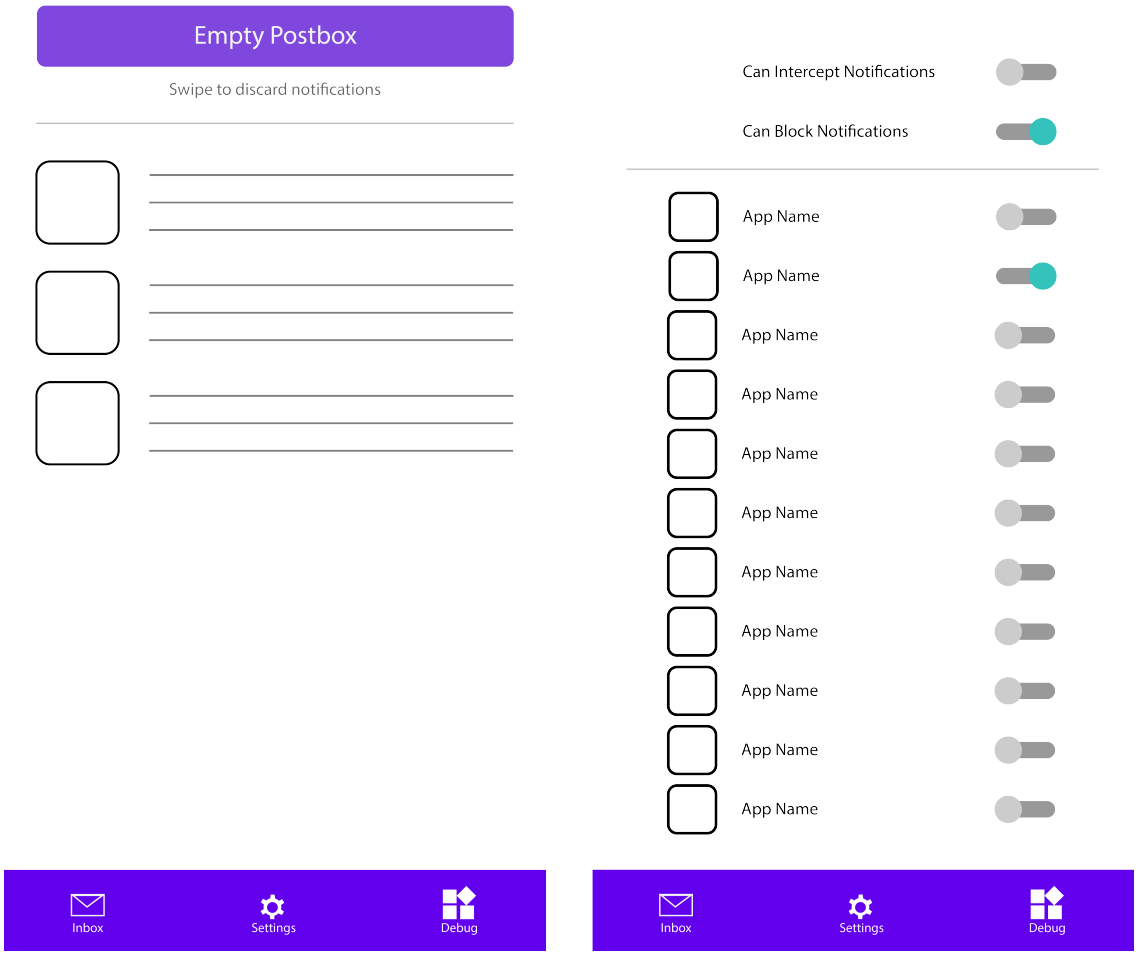


Figure 20: Mockups of the designs for the notification display fragment (left) and the settings fragment (right)

notifications as well as the clear notification button, to satisfy the mass removal specified behaviour. The user can swipe to remove single notifications or click that button to remove all of the notifications. The button is positioned at the top of the screen where it is not easily reachable to minimise the risk of accidentally tapping it as well as to draw attention to it. The top section remains always visible so that only the notifications themselves scroll. This allows a user with a lot of notifications the ability to easily 'bail out' and remove them all instead of having to scroll all the way to the top. The settings fragment displays UI that allows the user to control the behaviour of the app. This includes enabling the interception service and preventing notifications from reaching the system UI. The remainder of the UI should consist of a list of apps with toggles to allow interception on an app-by-app basis. If this list is too long for the screen, then the user can scroll through the list. To allow the user to easily find the apps they want to the list should be sorted alphabetically by app name.

The debug fragment exists solely to allow easy generation of notifications and support other operations for development purposes and therefore has not been designed ahead of time as end-user interaction should not be needed for the use of any app features. There are several final design considerations that underpin the designs above and should be kept in mind throughout the development of the user experience with the app. The humane design principles centre control with the user; they control which features are enabled and the process should be opt-in. Additionally, the user should not have to do extra work to understand the operation of the app; where the method to affect a response is not intuitive or obvious guidance should be given.

In line with the opt-in principles of humane design, when initially installed the app should be inert and not function to listen to notifications or interact with them in any way. The user should only be prompted to give the app permission to intercept notifications so that it may function. Once this permission is granted the app should still be inert as interception is also enabled on a per-app basis and the default off-state means no apps are enabled. Only once the user has then additionally enabled notification interception for a subset of apps should notifications from those apps be captured and displayed by the app.

Hinting the user towards non-obvious behaviour is also important to ensure use is easy. Bottom navigation bars are ubiquitous enough to not warrant a guide but to initially turn on the app features will so a message prompting this behaviour from the user should be displayed. Similarly, users may find swiping notifications to discard non-obvious and text to inform them that this is possible should be displayed. Other than buttons, this forms the main user interactions with the app.

Implementation

This section will outline several of the implementation details of the app, some of the ways that development deviated from the original plan, and go into greater details on the specifics of some implementation details.

UI Implementation

As outlined in the above design sections the interface to the user will consist of a single activity and three fragments. Above this though is the App class which is an abstraction of the entire app that's lifespan encapsulates all of the activities and other interactions. It is used to setup notification channels (for debug and testing purposes) and to provide a means of accessing the application context – which is often needed when interacting with the Android API.

The UI holding classes (activities, fragments, and other custom views) are defined in two parts. The first part is a java class that defined the behaviour of the UI elements and allows the interaction with the other components. The second is a layout that the class inflates to display to the user. This layout is an XML file found in the app's resources directory.

The main activity uses the 'activity_main' layout which consists of a `FragmentContainerView` and a bottom navigation view. The fragment container view, unsurprisingly, acts as a host to fragments. The java side simply sets the listener for the navigation bar so that it changes the displayed fragment, this is a very simple class as it is best to keep UI holding classes lightweight to increase performance.

The debug fragment is also a simple implementation, inflating the 'fragment_debug' layout which consists of some text, two edit texts, and two buttons. The edit texts allow users to enter the title and content for notifications and the buttons generate a new notification with the provided title and content and send it on one of two channels – each button corresponding to a different channel.

The settings fragment is slightly more complex as it must programmatically define some of its UI elements and interact with the shared preferences system provided by Android. The preferences screen is defined by 'root_preferences' which is in an XML file. This automatically ties into the shared preferences of the app to set switches and other UI elements to mirror the stored values when the fragment loads.

Shared preferences are one of the easiest ways for an Android app to implement a settings or user preferences page. The interface automatically stores preference changes in persistent storage and it can easily be queried. Whilst the shared preferences library is a convenient method for storing settings the values stored are unique to an app and so a future developer may wish to instead use a remote database query so that a user can have consistent settings across a range of devices – for example on multiple Android phones or a web page. To abstract the shared preferences library we define a settings interface that mandates that boolean settings and string sets can be stored and retrieved and app features that wish to query or modify user settings utilise this interface. An implementation of the interface using the shared preferences library.

Since components may need to find a settings instance, the interface defines a static method to provide an instance of the shared preferences settings implementation. This acts to allow components to query the interface to fulfil their dependency on a 'settings' object – an implementation of the service locator pattern. Whilst dependency injection would also be a valid

pattern the instantiation of some classes with this dependency is not controlled by application code and so injecting the dependency would be tricky – the service locator pattern worked better in this instance.

The settings fragment utilises this interface directly with the xml defined settings, which should be rectified in the future. The fragment also generates, upon creation, a list of switches for apps installed on the device to enable interception by app, whilst the predefined xml switches contain the settings corresponding to enabling the interception service, blocking notifications from reaching the system UI, and having notification cancellations in the system UI also remove the notification from the app.

The list of apps is generated with the `findAllApps()` method, which queries Android for all launchable apps on the system and converts them to `AppFilterItems`. It then sorts them by app name and removes any duplicate entries. The list of app filter items is created at app launch in the activity class as it takes some time to execute. When the settings fragment is loaded this list is used to generate preference switches. These switches collectively maintain the app filter which is stored as a single set of strings. The `AppFilterItem` class provided the functionality to obtain the relevant app details and generate a corresponding preference switch UI element and define the on change behaviour; it must fetch the old string set, update it, and save it back through the settings interface.

The final UI element to discuss is the inbox (`DisplayNotificationsFragment`). The static UI elements are defined in 'fragment_display_notifications' and consist of two parts. The first is a single text view that has the same size as the fragment – pushing all other elements off-screen. This text view contains either the settings prompt or an empty inbox message. After this is a linear layout container which has the empty inbox button, a text view telling the user how to remove single notifications, a dividing line, and a scroll view containing a recycler view.

The code for the fragment creates a view model for the stored notification data. The view model provides access to a LiveData variable containing the stored notifications and so the fragment registers as an observer of the data with the callback function set as `initRecyclerView`. This callback is called when the LiveData value changes as well as once when the UI is created. If the LiveData contains no notifications (that is it is empty) the text view is made visible and the other UI elements are made to be 'gone' (no screen space is allocated for them and they can't be interacted with). The text view has its text set to either the empty mailbox text or the settings page prompt text depending on whether the user has enabled notification interception.

If there are stored notifications for display the recycler view is provided with a `StoredNotificationAdapter` which converts the stored notification data into UI elements and the recycler view also has an `ItemTouchHelper` assigned, which defines the swipe-delete behaviour of the displayed notifications. Provided with the adapter the recycler view then draws the notifications to the screen.

The adapter inflates a 'stored_notification_item' layout for each item and initialises a view

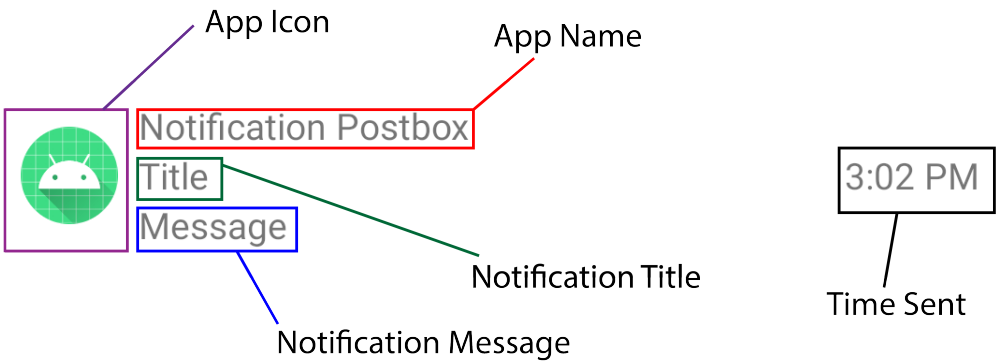


Figure 21: Labelled layout of a single notification displayed in the app.

holder for it. Then each view holder has a stored notification bound to it which is then used to populate the display fields. The conversion of `StoredNotification` objects to UI views is relatively straightforward as each field of the stored notification corresponds to part of the notification view. See Figure 21 for a breakdown of the data shown to the user.

The Interception Service and Notification Abstraction

The notification interception service is a background service that extends the `NotificationListenerService` and it has a dependency on the settings interface. When the service is granted permission to run it will run in the background regardless of whether the app itself is running which is ideal as it ensures that notifications are always able to be intercepted. However, the service requires a special permission to run that, unlike most Android permissions, cannot be granted by a dialogue presented to the user. The service itself is very simple, overriding two methods from the base class.

The first is called when a new notification is intercepted. This notification is provided as a StatusBarNotification object which is a wrapper class around the notification containing some additional data. The most important of these fields at this time is the package of the originating application. If the user has enabled the app, we check that the intercepted notification's origin package matches one of the apps that interception is specifically enabled for. If it is not then the service does nothing, but if this notification should be intercepted it is converted to a `StoredNotification` object and added to the repository. Finally, if the user has specified that system UI notifications should be supressed, the notification is cancelled removing it from the system UI.

The second function that the service implements is called when a notification is cancelled. When this occurs, if the app is enabled, we check if notifications that are cancelled should be removed and that notification blocking is not enabled. If this is the case the notification is converted to its equivalent StoredNotification object and removed from the repository. We must

check that notification blocking is disabled as when it is enabled every time a notification is intercepted it is immediately cancelled, and therefore removed, which would prevent any notification from entering the repository (aside from momentarily).

The `StoredNotification` is an abstraction of notifications. Due to Android's security features the `StatusBarNotifications` are useless as we lack permission to extract images and execute the intended user interaction. Instead, we strip out the relevant details and store them as a plain java object. This also removes dependency on Android classes for the data layer, as recommended by the Android documentation.

The `StoredNotification` class is instantiated with a non-null key and may additionally store the originating package, time sent, title, content, and channel of the modelled notification. `StoredNotifications` are uniquely identified by the key –- if two have the same key they are the same notification –- and so the hashcode and equals function are implemented on this basis. Incidentally, on the Android level a notification is uniquely defined by its originating package and a numerical notification id. This is used to construct the keys used in the abstraction.

A helper `StoredNotificationBuilder` takes a status bar notification intercepted by the service and constructs an equivalent stored notification. All of the fields have values contained within the status bar notification or the raw Android notification that it is a wrapper around. For the eventual UI display the sending apps name and icon is also needed, but this is queried at display time and not stored as it can be obtained from the originating package.

The Data Layer

So far the model and the views have been covered, so to complete the MVVM discussion, we will next cover the view model. The view model is instantiated by Android system code, so in order to pass the repository as a constructor parameter we must use a view model factory. The factory simply passes a reference to the repository to the view model's constructor in its `create()` method. A view model is then obtained through the following code:

```
viewModel = new ViewModelProvider(this, new
    NotificationsViewModelFactory()).get(NotificationsViewModel.class);
```

The view model is the interface between the UI and the repository, it provides access to a LiveData object containing the list of notifications, as well as methods to remove a specified notification, remove all notifications, and add a new notification through calls to the repository. The data the repository provides does not need particular processing to be suitable for display making the view model itself simple and primarily concerned with separating data storage concerns from presentation specific concerns.

The repository itself is defined as a singleton class acting as a single source of truth, through which all non-preference related storage occurs. The class is a singleton to enforce it being a *single* source of truth to prevent multiple different repositories with differing data being accessible.

The repository also own a `StoredNotificationLocalDataStore`. This is an abstraction of the file access that is used to store the list of notifications from the abstract interface that the app uses. The main repository does not need to be concerned with how the data is stored and the mechanics of storage need not be concerned with how the rest of the app may interact with it.

Decoupling the storage mechanism from the interface also permits later developers to modify the storage mechanisms or add online storage with local backups, via having two data stores, without it affecting the interface to the rest of the application – changes would be entirely local to the internal mechanism of the repository class and its dependencies.

The local data storage is flawed. Since the long-term storage of notifications does not need to be immediate it is permitted to be carried out on a background thread at a later time. Firstly, the execution of delayed actions should take place using an Android work scheduler rather than a raw java thread, as in the current implementation, to guarantee eventual execution. Secondly, there is not guarantee writes will execute in order or even in a synchronised fashion. This means that there is no recourse if writing is not possible due to another thread already executing a write operation or if an earlier write overwrites a later one if thread execution was delayed. This is the weakest element of the code base. Whilst these concerns are unlikely for small numbers of notification, if the app is going to function at larger scale, they need to be resolved.

Installing the app

The code is provided as a usable Android Studio project and should be able to be imported without issue, allowing developers to immediately examine the code and build on it. Similarly, the app is fully functional but providing an APK for installation on mobile devices is not feasible as different APKs would need to be build for different phone versions and the process of APK installation and building is non-trivial. If a user wanted to install the app, the easiest method at this time is to install Android Studio and run the program in debug mode on their phone. This causes Android Studio to build and install the app on the device, after which it is fully functional. The official Android developer documentation explains how to do this.

5.3 Exit Points

Scope

The tool **must** give a developer options of how to spread out their exit points.
The tool **must** be clear in how it would be used by a developer to extend their social media platform.
The tool **must** be able to be compatible to most popular social media platforms.
The tool **should** allow for a reasonable amount of customisation by a developer to make the behaviour appropriate for their platform.
The tool **should** give some information on what an effective exit point would look like.
The tool **could** have provided example exit points in addition to how to spread them out if there was more time however a large spread of these would have to be made to ensure this tool remained relevant to all social media platforms.
The tool **would** have been tested in place as a platform if developing a platform to showcase the tool was possible within the timeline of the project.

Tool Design

A user's need for an exit point to help them be aware of their social media usage will vary with the amount of time a user spends on the platform. If a user has just opened social media and started to engage with it, they would likely find it off putting to immediately be met with prompts asking them if they have spent too long on the platform. One way to deal with this would place the exit points sparsely enough to ensure they don't get in a user's way when they first start on the sight, this could be done by making exit points very sparse. However, if a user has lost track of time, after being on the platform for 30 minutes, and has now spent much longer than they initially wanted on the platform they would want to see an exit point to help them be conscientious of their usage, sparse exit points may leave this need unmet for a long time. Therefore, a variable system that increases the frequency of exit points based on the expected need for them allows for exit points that are not annoying by getting in the way but are also able to useful to a user.

A function which takes in a measure of how long the user has been on platform as an input can be used to spread out exit points, this would allow the exit points to be sparse when they are expected to be needed less, and more dense as the need for them increases.

A user's need for an exit point won't increase with the time spent on a platform linearly.It will stay low until they start spending to spend longer on the platform than they had planned to, then their need starts to increase quickly. Therefore, non-linear functions are required to model this changing of needs effectively. Users' needs for an exit point will have a limit, and this limit will vary, a linear approach to modelling this will result in the occurrence of exit points to create far too much friction in the user experience and overtake the need for exit points too quickly. The

functions selected can be seen below.

$freq = log(x)$

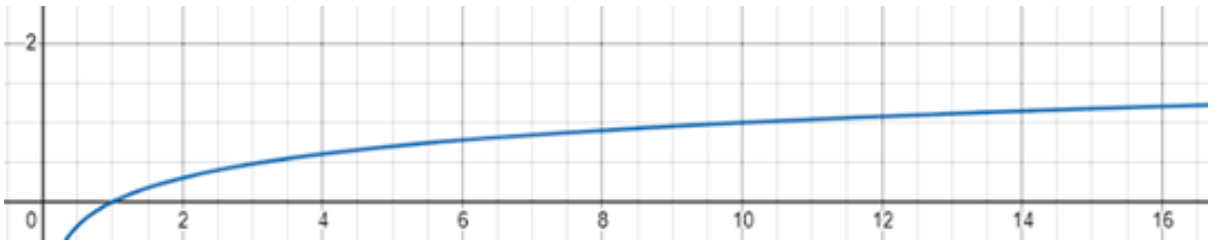


Figure 22: Graph showing the function $log(x)$ change over time as x changes

$freq = x^{0.5}$

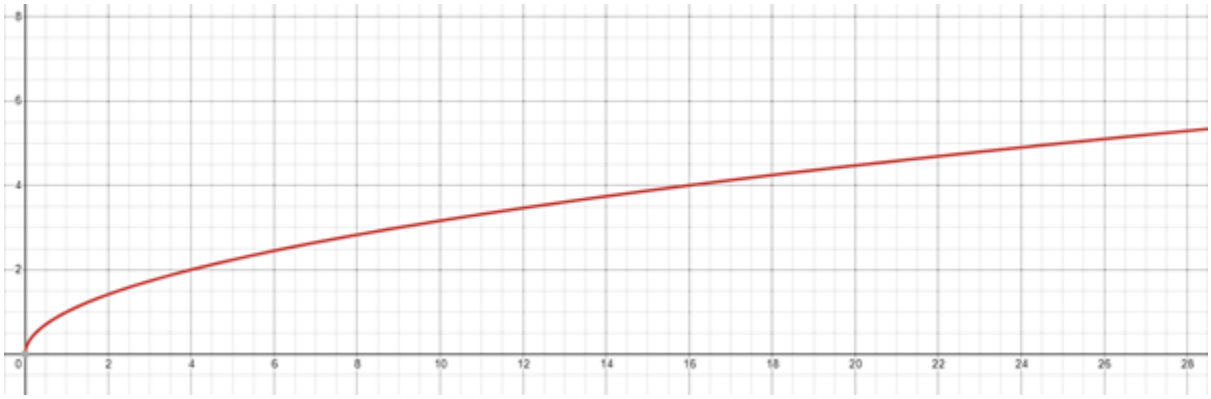


Figure 23: Graph showing the function $x^{0.5}$ change over time as x changes

These two functions follow the trend found for a user's needs. You can see that they increase and start to increase much slowly as the measure of time x increases. This allows for a user's need for an exit point being limited to be met. This also prevents exit points become so prevalent that it then becomes impossible to engage with a social media platform if a user still desires to, if the function did not increase at a slowly rate as time goes on then the platform would become a constant stream of exit points making the platform unusable. The most appropriate function is $f(x) = log(x)$ as this appropriately models a users low need for exit points until their typical session length is met. Then the need rapidly increases to a plateau where the maximum need is met. This function can then be tuned so that the function's x intercept is equal to a typically users session length on the platform.

An alternative option which does not fit entirely with the user testing data is the exponential function, this would be more appropriate for cases where a user spending excessive time on a platform was heavily detrimental to their well-being, an example case could be a social media that would include some form of gambling within it, here a user engaging with this social media excessively could lead to a very serious issue, therefore it would be in the user's benefit to be encouraged to engage in a very limited manor. This function is provided to cater for these more extreme examples as the frequency of exit points would exponentially increase over time. For

some platforms the idea of the time period between exit points may be easier to use and implement, therefore the functions for determining this have also been provided within the tool by modifying the functions that provide the frequency of exit points to make the tool more complete. This can be seen below

$$freq = e^x$$

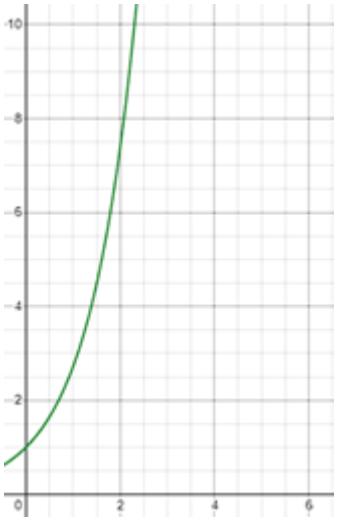


Figure 24: Graph showing the function e^x change over time as x changes

These functions had parameters added to them to allow them to be easily customised by a developer, additionally allowing these parameters to be potentially algorithmically tuned depending on a particular user. Although this would be a very platform dependent task so the specifics of how this would be done falls out of the scope of this tool.

Finally, as these functions are not strictly bounded a limit to them has been added as parameter, which is customisable dependent on the use case of a platform. This allows a developer to guarantee that exit points do not occur more than this parameter allows for, defining the maximum frequency of exit points.

The main limitation of this tool is that it provides everything necessary to space exit points effectively but does not provide the exit points themselves as a ready to use resource. This was decided as to make this tool universal for any potential type of platform to use a large and diverse number of exit points would have to be designed, implemented, and tested. But even with a large collection of exit points, a platform would likely desire a bespoke exit point that would not be within the collection. Therefore, it was deemed that this would be something left to a developer to create with some guidance as to what the exit point should achieve written into this tool.

The functions used so far have all referred to measures of time so far, how this measure of time is decided is something that has been left to a developer to implement, this is as it would be a very platform specific solution to extract this information and some metrics may be more natural to use then other. To ensure that this process is easy the tool recommends a couple of methods that would likely be possible to use an effective measure of the amount of user time spent on

the platform. These being the amount of time a user has spent on the platform in this current session/day, or the number of posts a user has seen/interacted with in this current session/day.

Test Plan

ID	Req.	Test Description	Expected Output
1	EP1	Test a range of input values and check the outputs are expected	The functions should output a valid range of positive real numbers
2	EP2	Test the functions using a range of input values and changing the parameters	The functions should produce an output with any numerical parameter input.
3	EP3	Test each function and compare their behaviour to ensure they provide different outputs	Each function should produce a valid range of positive real numbers

Table 7: Test Plan for Exit points Tool

Choice of Development Platform

Coding these solutions within a python notebook allows a developer to quickly understand the code as python is one of the most readable programming languages, so they would be able to convert the appropriate function for their needs in whichever programming language they are developing in easily. Additionally, by making use of a notebook a developer could play around with this tool without needing any software installed on their machine other than a web browser capable of using Google Colab.

Extensions

A developer would need to extend the tool by producing their own designs for their exit points. This is not provided as this is incredibly platform independent as what makes for a good exit point will vary with how a user engages with the content on the social media platform, therefore it would likely not be possible for any exit points presented in this tool to be able to be used for a developers own social media platform

5.4 Time Transparency

Scope

To ensure the transparency tool was designed to meet the justified requirements, the scope of what the tool was going to achieve was established. To meet the requirements, the tool would need to demonstrate how each component of the proposed transparency system will operate in response to simulated 'viewing content'. For the sake of focusing on the tool, the guide section will abstract and simulate external, preliminary processes which are not directly a part of the tool such as recording timings of content viewings, and actually limiting content the user has chosen to limit. These processes will be acknowledged within the toolkit, but providing implementations for these processes are outside of the tool's scope.

Tool Design

Initial Mock-up

To gain an initial insight into the tool's workings and begin breaking down its components, a mock-up 'overview' diagram was created(Fig. 25). This diagram contextualized each of the five requirements within an imagined mobile solution, and was able to give a basic idea of how the tool system will look if all of the requirements are met. By doing this, it justified that these requirements were sufficient in representing a complete tool. Additionally, it served as an initial work breakdown structure for the tool by giving the abstract overview of this tool, and breaking it down into its fundamental requirements. Producing this diagram made it easier to visualise the tool as a whole, and aided in breaking down the solution into more technical modules, thus helping focus the design process.

This mock-up did not mean to serve as a representation of the final tool, but was rather presented within a mobile context to serve as an example where this tool could be utilised by a software developer, as well as an example where every part of the tool could be simply visualised in one place.

Data Flow Diagram

To extend the high level mock-up design, a more in-depth, technical representation of the tool's breakdown was created in the form of a data flow diagram(Fig. 26). By representing the tool by how the data will travel through it, it identified the key stages the tool is comprised of, and that the data visualisation aspect of the tool can take the form of an input to output process with the time data structure.

Firstly, the data flow diagram shows that the time data must be collected and organised cor-

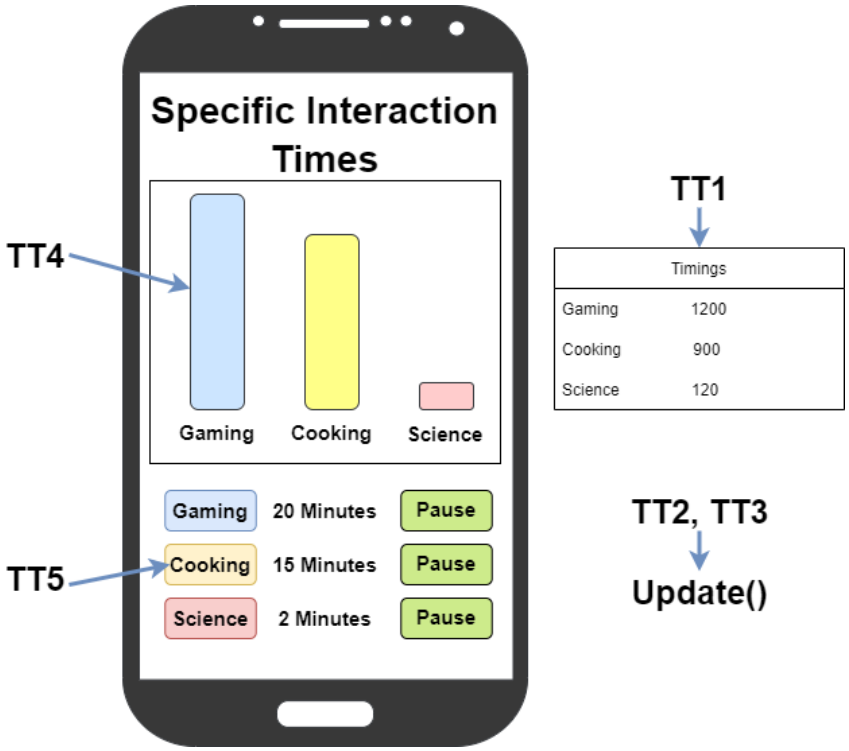


Figure 25: The original mock-up of the components of this tool imagined in context, with labels for each requirement.

rectly *per content tag* within a single data structure, and this entire process can be separated as 'preliminary work', as this process involves preparing the data for its main usage: the visualisation. The diagram was effective at separating how this preliminary process will be processed mainly through the *update* function described within requirements **TT2** and **TT3**. It effectively describes how the update function should take a set of content tags and a timing as an input(specifically, a user watching some content for some amount of time) and should output an updated existing time data structure, storing all the content tags with all their timings up to date. By understanding this preliminary process as a set of inputs and outputs, it made designing further technical aspects of the update function much simpler, and the overall design much more rigorous.

The diagram highlighted the data visualisation process of this tool as the main 'usage' of the data, as it was effectively making this pre-processed data available to the user. This process relates to requirement **TT4**, and shows that as long as the input data structure is available, the data visualisation can be abstracted as a single module to simply display this data.

The final stage of the data flow is simply gathering a user's input as to which content should be limited, as a reaction to the data visualisation. This user's choice acts as the main output the entire tool.

Overall, this diagram allowed for a stronger design process for this tool as it broke the tool down into technical modules with definitions on how they interact, which allowed for a much clearer understanding of how the technical elements of this tool could be developed further. This

diagram was also used within the toolkit guide to give a technical overview to the tool's usage, as it was deemed effective at giving developers insight into the different modules the tool comprises of.

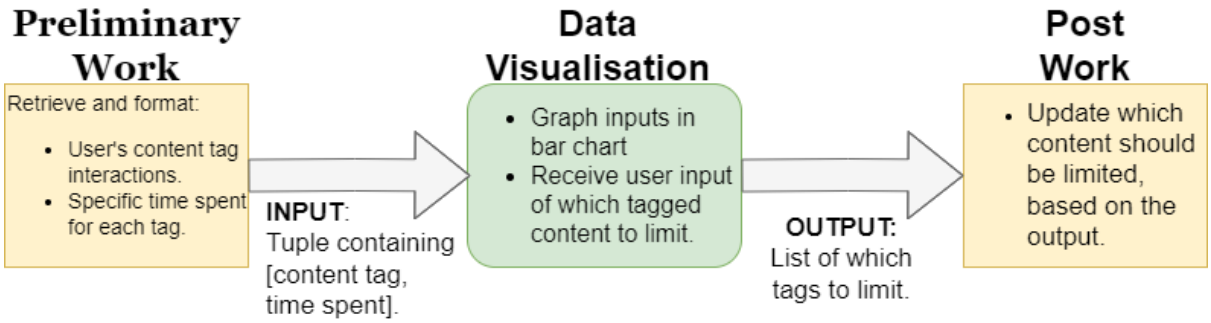


Figure 26: A generalised data flow diagram outlining the transparency tool's entire process.

Update Function Flowchart

The data flow diagram made it clear that the *update* function within the tool's preliminary work section would require a somewhat technical solution, and hence would benefit from rigorous designing. Hence, a flow chart(Fig. 27 was created to justify exactly how the update function would operate to keep the timings data up to date when given a new content tag and timing pair.

Creating the flowchart diagram helped plan how this function would operate. The solution to the *update* function proposed with the flowchart is; upon being given a list of content tag with timings tuples, iterate through each of the input content tags and check if the tag already exists in the existing data structure of content tags and timings. If it already exists the function should simply append the input time to the existing time, but if the tag does not exist then the storage should be updated with a new tag, and the initial time value is the same as the input time value. This solution was able to propose a specific, successful means of creating the desired output from the expected input as planned in the data flow diagram.

A flow chart was an effective choice for planning this function, as it allowed for the planning of this function to the lowest-level, specific even to which data structures will be used throughout the code. By designing this function via a flowchart, it enabled a more robust implementation of the tool as every potential action that should occur within the function had been planned and justified. This also made implementing the function relatively simple, and mitigated the risk of encountering time delays during implementation because no time needed to be spent coming up with a solution for small parts of the function, as they had already been planned.

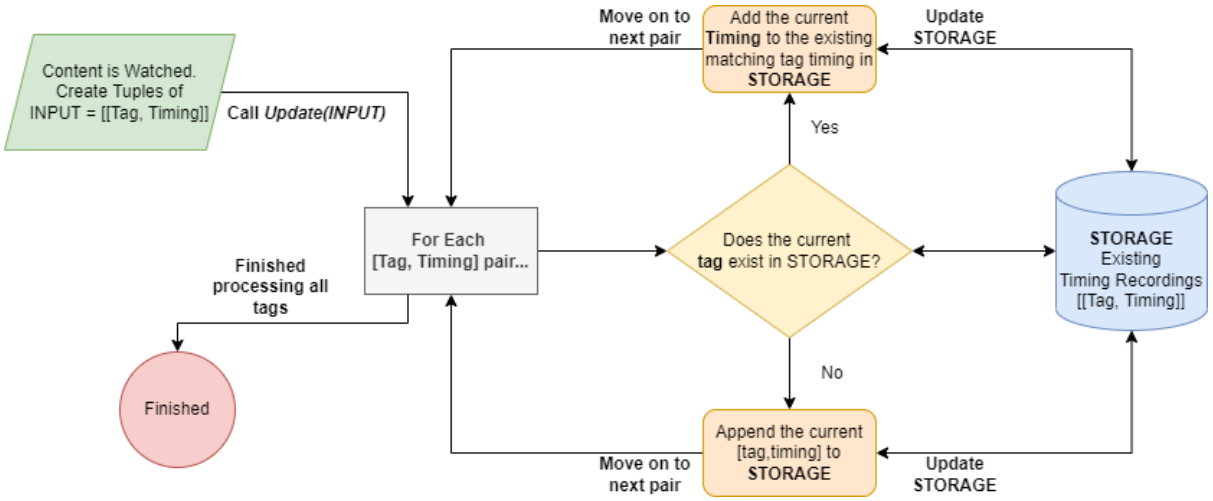


Figure 27: A flowchart detailing the planned processing within the *Update* function.

Test Plan

A formal test plan was constructed before the implementation, as a means of validating the success of the tool against the original requirements. Each test is labelled specific to which requirement it is validating for. The test plan is shown in Table 8.

Table 8: Test Plan for Time Transparency Tool

ID	Req.	Test Description	Expected Output
1	TT1	Verify that both content tags and time data is stored within a single data structure.	Content tags and time data is stored within a single data structure.
2	TT1	Insert ”Gaming, 10”, ”Science, 20” and ”Cooking, 30” into the data structure and print the contents.	The contents should read ”Gaming, 10”, ”Science, 20” and ”Cooking, 30”.
3	TT1	Insert 20 content tags into the data structure, each respectively named ”A”, ”AB” and so on until ”ABCDEFGHIJKLMNOPQRST”, and print the results.	The data structure will be able to store and print all 20 content tags without errors.
4	TT1	Insert 20 content tags into the data structure, each with respective time data in increments of 4320: ”0”, ”4320”, ”8640” and so on until ”86400”, and print the results.	The data structure will be able to store and print all 20 editions of the time data without errors.

5	TT2	Initialise a data structure with 100 unique content tags with a timing of 1. Call the update function with identical content tags, with a timing of 2 as the parameters. Print the structure.	Prints the 100 unique content tags, all with a time value of 3.
6	TT2	Repeat test 5, and record how long the process takes during runtime.	The recorded runtime should take less than 3 seconds.
7	TT3	Initialise an empty data structure. Call the update function with 100 unique content tags, with a timing of 1 as the parameters. Print the structure.	Prints the 100 unique content tags, all with a time value of 1.
8	TT3	Repeat test 7, and record how long the process takes during runtime.	The recorded runtime should take less than 3 seconds.
9	TT4	Initialise a data structure with 100 unique content tags with unique timings, counting up from 1,2,3..100. Pass this data structure to the visualisation function.	A bar chart is produced with an x axis of the content tag names and the y axis displaying time values. The bar chart consists of exactly 10 bars, corresponding to the *largest 10 content tag timings*, namely 100-90, in size order largest to smallest.
10	TT4	Repeat test 9, and record how long the process takes during runtime.	The recorded runtime should take less than 3 seconds.
11	TT5	Verify that at least 1 method of allowing a user to select a content tag is demonstrated.	At least 1 method of allowing a user to select a content tag is shown within the guide.

Implementation

The implementation process for this tool was confidently performed using the resources created via the design process. A single *.ipynb* file was created to demonstrate this tool, where the tool's implementation was broken down and demonstrated individually for each of its three planned major components:

- **Preliminary Work** - The update function was implemented identical to its proposed design (Fig. 28) with no issues. For the sake of the developer understanding the tool's code, a simplified means of simulating 'viewing content' was given in a single cell, so that the developer can interact with the update function directly to understand how the function operates rather than just studying the code. This also allowed for technical testing of the

tool to occur. Essentially, the update function was implemented effectively with no issues due to the thorough design process.

- **Data Visualisation** - Using the resources from mentioned in Section 2.3, Python's *matplotlib* library was utilised to produce a function for abstracting the time data visualisation process, given the correct inputs and outputs. As requirement **TT4** stipulated the data must be displayed in size order, an unforeseen slight difficulty was achieving this ordering within the structured timing data, however, a solution was found through researching how to sort by size a list of tuples by their second value, using the *.sort* function of a list, using an appropriate key[10]. Otherwise, this section of the tool was able to be successfully implemented, and was able to produce visualisations similar to the mock-up designs(Fig.29).

- **Post Work** - To stipulate requirement **TT5**, the tool was successfully able to demonstrate a single means of allowing a user to select which content to limit based on the content tags displayed in the visualisation. A simple, interactive drop down box allowing the selection of a tag to limit was implemented.

Whilst it could be generally identified that the tool met many of its original requirements, a process of testing against the designated test plan occurred to formally ensure the tool had succeeded in its requirements, and cease the implementation phase for this tool. This testing process is documented in Section 6.1.

```
#A function that will update the timings variable accordingly, given a list of name tags and time to add
def updateTimings(contentTags, time):
  #Retrieve only the tag names from timings
  storedTags = [tags[0] for tags in timings]

  #For each tag...
  for tag in contentTags:
    #If the timings does not contain time data for the current tag, add it.
    if (tag not in storedTags):
      timings.append((tag, time))
      #Update storedTags
      storedTags = [tags[0] for tags in timings]

    #Otherwise, update the existing time
    else:
      tagIndex = storedTags.index(tag)
      newTime = timings[tagIndex][1] + time
      timings[tagIndex] = (tag, newTime)
```

Figure 28: The implemented update function, identical to how it was designed.

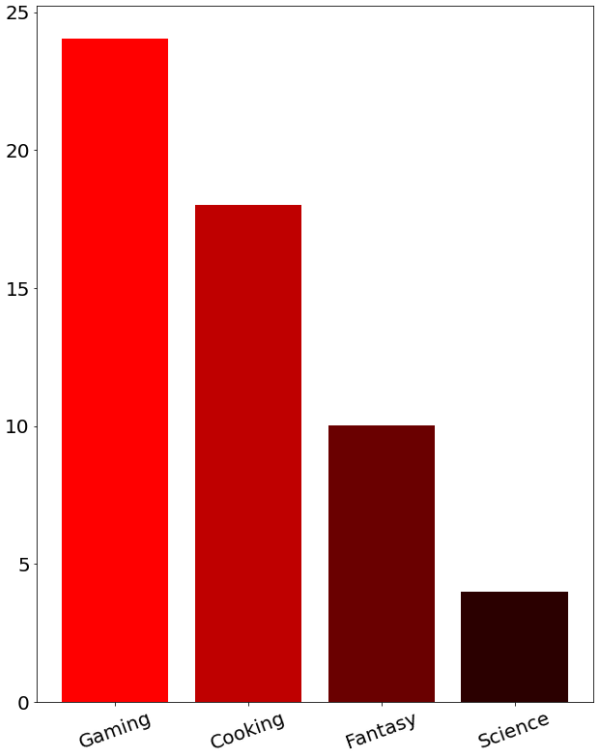


Figure 29: The resulting visualisations from the tool, given some example data.

5.5 Moderation Model

Scope

The moderation model must be effective at identifying offensive language.
The moderation model **must** be fast.
The moderation model **must** be able to scalable.
The moderation model **should** not be biased.
The moderation model **should** be trained on a varied dataset.
The moderation model **could** be able to cope with words that it did not see during training.

Tool Design

The moderation tool needed to be able to understand text posts written by people and how people would interpret and feel about those posts. Then it can filter out the posts that it has identified to be offensive. Machine learning and natural language processing techniques were used to achieve this. Using natural language processing it is possible to distil text posts into a numerical representation that is can then be given to a machine learning model to learn how to classify text posts into two categories, offensive and not offensive, therefore being able to filter out posts that are not suitable to be hosted on a social media platform.

The first step to creating this tool was to find a suitable dataset that could be used as input to the machine learning model to allow it to learn how to classify posts and comments. There were two main options of how to get a dataset to use for this, either creating it by scraping social media platforms for posts and then manually going through those posts and classifying them or finding a dataset that has already been created and ready to use. Making the dataset ourselves would be very time consuming, result in a much smaller dataset then would be possible by using a third-party dataset, and would inherit any biases held by the team. Therefore, finding a large and high-quality dataset that labels text posts from social media based on how offensive, abusive, and or hateful those posts are was decided upon. This decision led to choosing the Measuring Hate Speech dataset described in Kennedy et al. (2020) [22] for training the machine learning model. This dataset is made up of 39,565 comments from social media platforms, like Twitter, which was labelled by 7,912 annotators. This provides a sufficiently large enough dataset to create accurate models, as well as a robust enough dataset that should not have inherited any one particular bias from a single annotator.

With a dataset selected it needed to be modified before it could be used to train the moderation model. The dataset did not use categorical labelling of if a post is acceptable or unacceptable, instead giving a numerical range called a hate speech score. The more positive a score is the more it is deemed hateful, while the more negative the score is the less hateful it is deemed. A score of zero correlates to a post that is somewhat borderline on being hateful or not. By thresholding this value, so that any post with a hate speech score higher than zero would be labelled as unacceptable, and any post score less than or equal to zero being labelled as acceptable, the dataset can be converted into having the categorical labelling that we wanted the dataset we used to have. The dataset we then used consists of the text post and if it is deemed acceptable or unacceptable.

The next step was determining how to represent the text posts so they could be used as inputs to a machine learning model.

The first method used was TF-IDF (term frequency-inverse document frequency), which gives a statistical measure of how important a word is within a post or comment. This importance is determined by a measure of how often that word occurs in one post, multiplied by how much information that word provides (e.g. how common the word is in all posts within the dataset). This creates a large and sparse matrix which has every word that's within the dataset as a column, and every post as a row. Position $i * j$ will have the value depending on how often the ith word occurs within the jth dataset, multiplied by how often the ith word occurs in all the posts in the dataset. Each row of the TF-IDF matrix can be used as a feature vector for the problem.

The second method used was Word2Vec. This uses a neural network to learn a vector embedding for each word in the dataset. The words vectors produced give an idea of similarity with the angle between vectors indicating the similarity of the words. Taking a post and taking the average word vector of the post can then be used as a feature vector for that post . The third and final method used was FastText. This is an extension to the Word2Vec method which can

find the embeddings for parts of words. This allows it to get a better embedding for words that are not common within the dataset by splitting the word into sub words and finding the common embedding for those sub words. For example, if the word 'scarcity' is uncommon within a dataset, the algorithm could identify the embedding of 'scarc' using the word 'scarce' and find the embedding of 'ity' by using the embedding found for 'ity' within 'rarity'. This allows for a more accurate embedding for the word scarcity from other common words in the dataset. This produces word vectors as a final output like the Word2Vec method that can be used as feature vectors. It is also capable of producing sentence vectors, which we use for the feature vectors.

Three machine learning models were used to test the feature vectors created by the various natural language processing techniques. These were K Nearest Neighbour (KNN), Support Vector Machine (SVM) and Neural Networks (NN). The general pattern across all these models was that TF-IDF was outperformed by Word2Vec, which was outperformed by FastText. KNNs and SVMs performed very similarly to each other 82% accuracy and 83% accuracy respectively. The neural networks performed better with a 87% accuracy, but the FastText library's neural network model with the FastText representation performed significantly better with an accuracy of 92%. Based on these results for the moderation models, the FastText text representation method was chosen as well as the FastText classification model.

The word embedding created by the FastText model is visualised below using the UMAP library to reduce the dimensionality of the vector that represents the word embedding from 300 dimensions to 2.

The blue dots are all the words which are unacceptable according to the model while the red ones are those that are acceptable. There are several 'strands' of words which have been created here, a strand contains words which have similar embeddings as other words within that strand. Therefore it makes sense that all the blue points are contained within a single strand, as these words all share the sentiment of being offensive so would have a similar vector embedding.

The major benefit of FastText is that it can perform well on words that may not have been common in the initial training set by being able to break those words up and find the embedding for those smaller words. Additionally, it performed the training and testing steps of the process faster than other methods within Google Colab. The testing step was particularly significant as the model should be able to handle a high number of queries to it to allow it to moderate many posts.

The extent of the moderation design was represented within a work breakdown structure(Fig. 31).

Test Plan

ID	Req.	Test Description	Expected Output

1	M1	Test the moderation model with training data	The model should have a test accuracy of over 90% to prove that it is effective
2	M4	Provide a large input file of posts to the model for predict	The model should be able to classify 100,000 comments in under ten seconds
3	M3	Manually input test data to ensure the model has the expected results	The predicted label for each input should be the correct, at least 95% of the time

Table 9: Test Plan for Moderation Model Tool

Choice of Development Platform

The moderation tool was developed in a python notebook so that it could be easily ran in a browser and be ran using Google Colab to allow a developer to use the model and play around with it to see if it meets their needs without needing any resources on their end.

The tool makes use of the FastText library and the Datasets library to make development simpler. It would have been possible to create a NN for the FastText model and use that as the final model but for developers who are unfamiliar with machine learning and neural networks this could be overwhelming and turn them away from using this tool. Using FastText's library to create and train a model makes the whole process much more readable and understandable for someone without specialist knowledge.

Limitations

The most significant limitation of the design is it cannot produce an accurate embedding for novel words that are not within the training dataset. This means that it cannot correctly classify for new words that could be created. For example, if a new word came into circulation and was used as a slur or to insult people the model would not be able to learn this without including labelled examples within a new training dataset and retraining the model. This would have to be combated by maintaining the moderation model to ensure that new language is added to the training set.

Extensions

The main extensions to this model would be creating a method for the moderation model to perform well with words that are out of the vocabulary of the training dataset. This could be

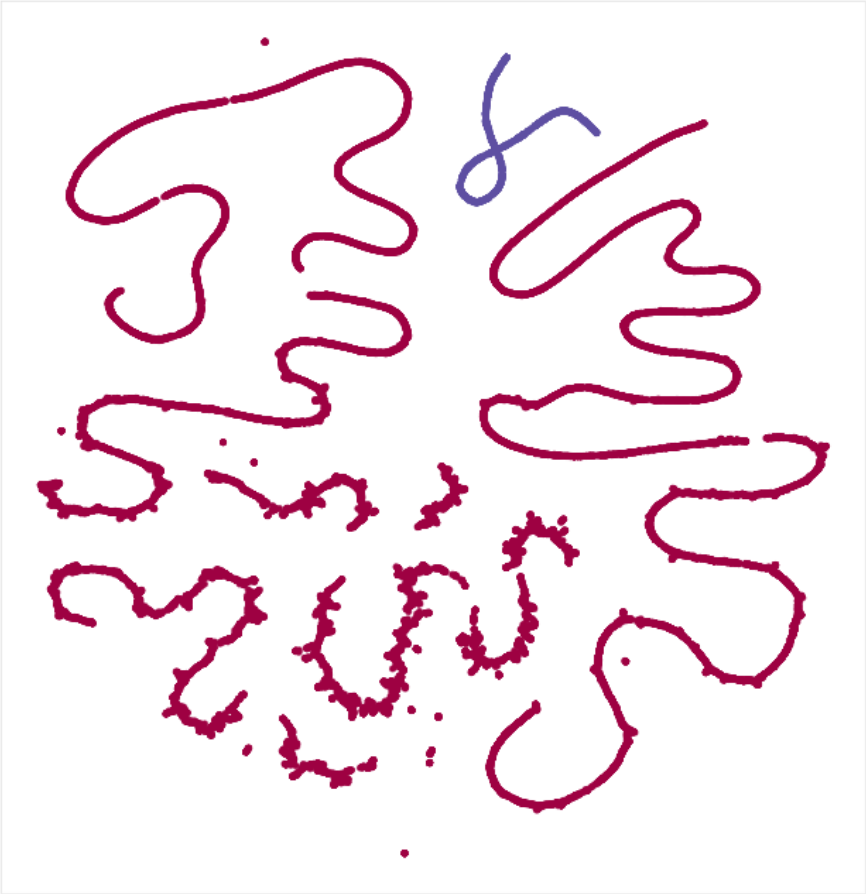


Figure 30: Visualisation of the text representation used for the moderation model

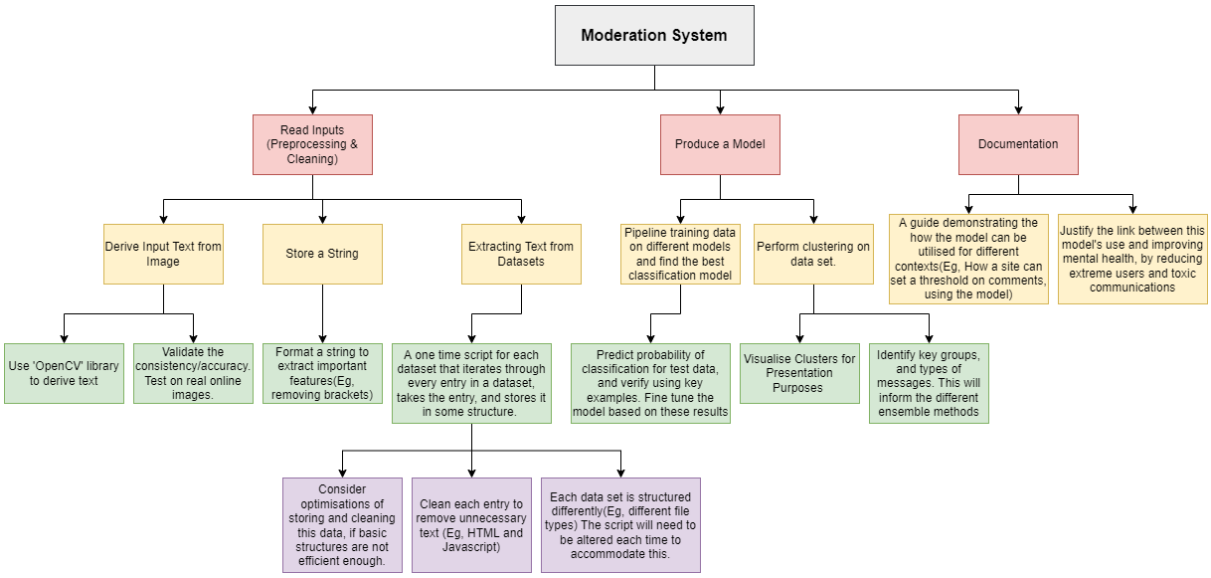


Figure 31: The work breakdown structure for the entire moderation system.

done by creating tokens for words that would be unacceptable and finding a method to map these words to an appropriate token then training a model using general tokens. The secondary extension would be creating and using a larger dataset to get even better results.

5.6 Toolkit Guide

Scope

The guide **must** explain all tools.
The guide **must** explain why the tools exist.
The guide **should** be clear and easily understandable to a developer.
The guide **should** allow a developer to implement the solutions on their own platform.

Design

The guide is structured into two main sections for the two main problems being addressed by the toolkit, reducing screen-time, and providing effective moderation. This is to make it clear which tools address which problem. Within these main sections are subsections for each tool, which provides all the information that a developer would need to be able to implement these solutions within their own platform.

Based on feedback duringtheteam's progress presentation about a lack of clarity of how all ofthetools fit together, we designed an architectural overview of the toolkit to visually explain each tool's use-case and where exactly they would be implemented within the context of a social media platform. This overview can be seen in Figure 33, with a corresponding key in Figure 32. This allows the guide to more intuitively explain the role of the tools to the developers.

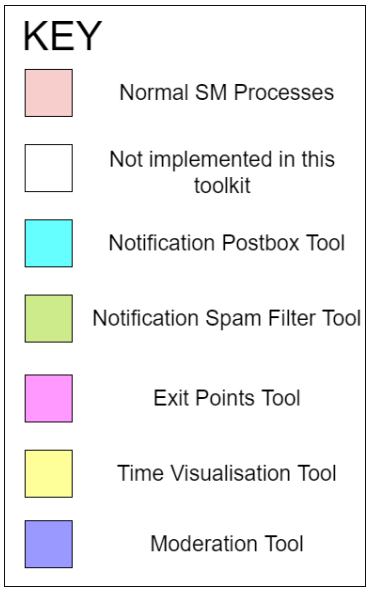


Figure 32: Key for Figure 33.

Each tool's section is made up of five parts. A glossary, a description of the use case, a technical overview, a link to a solution hosted on GitHub, and an explanation of any further considerations for that tool. The glossary provides an explanation of any key terms that are mentioned with the

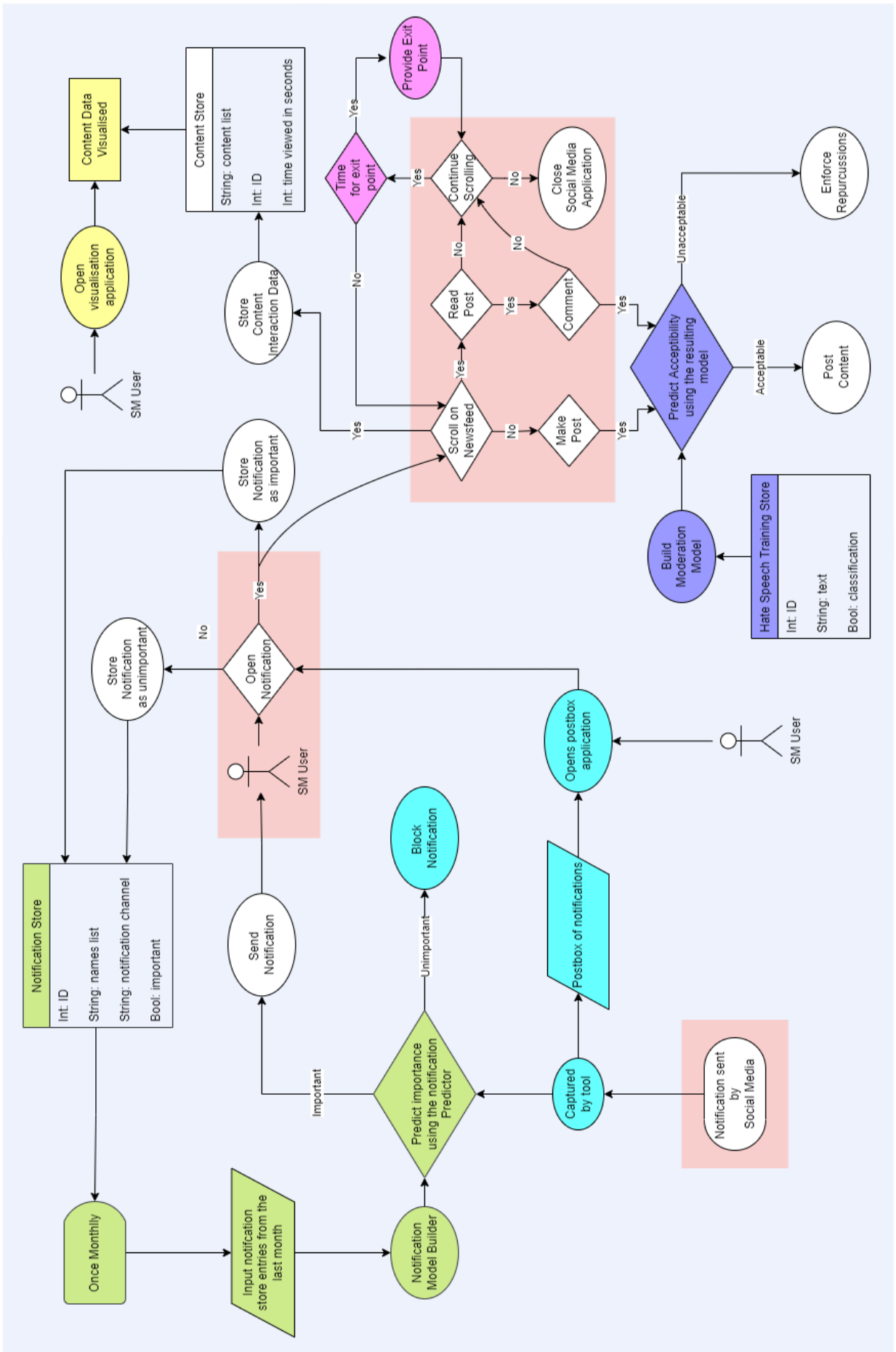


Figure 33: Architectural overview of the toolkit and how each of the tools fits into the life cycle of social media.

section ensuring that a reader won't be thrown of by any unfamiliar language. The use case description ensures that a reader is aware of how the relevant tool is intended to fit within a social media platform making the process easier for a developer to design the tool into their platform as they know where it is appropriate to use. The technical overview briefs a reader on what the major parts of the tool are and a how they would be implemented at a high level, they can then refer to solution and demonstration section to access the code linked by the guide to see an isolated example of the tool for them to use as a reference to develop their own. Finally the by discussing the further considerations of the tool the guide primes a developer on platform specific problems they may encounter with integrating the tool into their platform and how they can get around these. Additionally this section informs a reader of possible extensions to this tool they may wish to consider to try and implement for their own platforms to have an even more effective system.

The guide is a PDF which contains links and references to the code repository, so is intended to be interacted with on a computer. This allows for quick navigation around the guide as well as reference to code. As the code can be accessed through Google Colab this would allow a developer to fully access the toolkit within a browser making for a simple and easy way to interact with the tools and the guide at the same time.

6 Testing

6.1 Technical Testing

Notification Spam Filter

The testing for the spam filter was performed according to the test plan from the Design section. To test the execution times of the model builder and the predictor, the Python time library was used.[37] The start timestamps were taken after the cell for mounting Google Drive because that cell is optional depending on platform. The end timestamps were taken after any outputs had been written to file. The test inputs (referred to in the test description) are available in the Testing sub-directory of the spam filter tool. The results of the tests can be seen in Table 10.

Table 10: Testing for Notification Spam Filter.

ID	Req.	Test Description	Resulting Output
1	SF1	Train model with 10 notifications. Using 'Test 10.CSV'.	Took 2.003 seconds, Specificity=1.0, Sensitivity=1.0
2	SF3	Use 'Test 10.joblib' model to make a prediction on 'New Notification.CSV'.	Took 0.566 seconds
3	SF1	Train model with 100 notifications. Using 'Test 100.CSV'.	Took 3.549 seconds, Specificity=1.0, Sensitivity=1.0
4	SF3	Use 'Test 100.joblib' model to make a prediction on 'New Notification.CSV'.	Took 0.488 seconds
5	SF1	Train model with 1000 notifications. Using 'Test 1000.CSV'.	Took 4.075 seconds, Specificity=1.0, Sensitivity=1.0
6	SF3	Use 'Test 1000.joblib' model to make a prediction on 'New Notification.CSV'.	Took 0.668 seconds
7	SF1	Train model with 10000 notifications. Using 'Test 10000.CSV'.	Took 15.277 seconds, Specificity=1.0, Sensitivity=1.0
8	SF3	Use 'Test 10000.joblib' model to make a prediction on 'New Notification.CSV'.	Took 0.606 seconds
9	SF2	Train model with 100000 notifications. Using 'Test 100000.CSV'.	Took 829.659 seconds (over 13 minutes), Specificity=1, Sensitivity=1. Most of the time was spent performing the grid searches.

10	SF3	Use 'Test 100000.joblib' model to make a prediction on 'New Notification.CSV'.	Took 0.475 seconds
11	SF4	Train model with only important notifications, 'Test All Impnt.CSV'. Use 'Test All Impnt.joblib' with 'Impnt Notif.CSV'.	Error encountered. The tool needs an input with at least two classes.
12	SF4	Train model with only important notifications, 'Test All Impnt.CSV'. Use 'Test All Impnt.joblib' with 'Not Impnt Notif.CSV'.	Test not performed because error encountered using 'Test All Impnt.CSV'. The tool needs an input with at least two classes.
13	SF4	Train model with Daisy's notifications, 'Daisy Notifications.CSV'. This contains a redundant column with the whole notification message in, to check the tool does not error when given unusual input.	Took 3.248 seconds, Specificity=1.0, Sensitivity=0.6
14	SF3	Use 'Daisy Notifications.joblib' model to make a prediction on 'New Daisy Notification.CSV'.	Took 0.464 seconds
15	SF4	Train model with Lewis's notifications, 'Lewis Notifications.CSV'.	Took 1.884 seconds, Specificity=1.0, Sensitivity=1.0
16	SF3	Use 'Lewis Notifications.joblib' model to make a prediction on 'New Notification.CSV'.	Took 0.447 seconds
17	SF4	Train model with Patrick's notifications, 'Patrick Notifications.CSV'.	Took 4.923, Specificity=1.0, Sensitivity=1.0
18	SF3	Use 'Patrick Notifications.joblib' model to make a prediction on 'New Notification.CSV'.	Took 0.416 seconds
19	SF4	Train model with Matthew's notifications, 'Matthew Notifications.CSV'.	Took 3.227 seconds, Specificity=1.0, Sensitivity=1.0
20	SF3	Use 'Matthew Notifications.joblib' model to make a prediction on 'New Notification.CSV'.	Took 0.429 seconds

Building the model with 100,000 notifications (Test 9) was expected to take a long time because it is an extreme input case. However, it took ten minutes longer than was deemed reasonable

by the test plan. As a result the model builder part of the tool is decidedly not scalable up to such a high number of notifications. One possible solution is to truncate any exceedingly large input files to 10,000 notifications, since Test 7 managed to successfully and quickly train with 10,000 in a matter of seconds. Moreover, when recording the team's notifications, it was found that notification messages tend to repeat, as users interact again and again with their friends. So, to reduce input size, the pre-processing could involve restricting the data to only unique notifications, and then afterwards truncating to 10,000 notifications if needed.

Test 11 and 12 failed because the tool is built with the assumption that there are two classes. By only using one type in the data, the tool throws an error. This could be solved by only inputting data to the tool if a machine learning model is required. That is, in the case where all training data is one class, the platform could just predict all new notifications as that class. This would save on computation and memory, as well as avoid the error encountered with these tests.

Almost all of the training tests that succeeded scored perfectly in terms of metric performance; both sensitivity and specificity were 1.0. That is, the outputted models scored perfectly, but not necessarily all the models trained by the tool. As a reminder, the tool trains multiple models and then outputs only the best one. The models performed varyingly well, but there was almost always one of the models that performed perfectly. This is likely because of the relative simplicity of the data and the patterns. Due to time-constraints with the project, the tool could not be tested with notifications recorded over weeks or months; only one day. So, the data for the most part did not have inconsistencies because the user's opinion of importance did not have time to change over time. As a result, a perfect model could be produced.

One exception to the perfect models was Test 13. Test 13 used Daisy's notifications but with an extra column to test if the model could handle an unexpected input. The tool did not crash, and so passed this test. However, its sensitivity was 0.6, compared to the 1.0 of the other tests. This would not have been because of the extra column, since the column is essentially discarded by the model trainer at the start. The difference is possibly because Daisy's data did contain inconsistencies, unlike the other data. She uses social media for hours daily, and so her opinion of importance can vary as the day goes on. For example, Bob Bobbington's tags may not be important during the working day, but afterwards, they may be important for entertainment purposes. As a result, the notification importance now depends on a third factor; time of day.

Notification Postbox

The 'technical testing' of the notification postbox covers the automated testing carried out, covering the data layer and ensuring fulfilment of functional requirements PF1, PF6, PF8, PF9, and PF11. The other requirements are either guaranteed by the Android API or cannot easily be automatically tested e.g. device restarts. The exception is PF7 which, despite easily being testable, has no automated tests. This section will give an overview of the automated testing methods used and the test cases written, and then provide the test results and a consideration of the efficacy and coverage of the automated tests.

Broadly speaking, there are two types of automated tests that can be executed on an Android app; instrumented tests and unit tests. Unit tests separately test the functionality of objects away from the Android system, allowing code to be tested in isolation. Since there can be no easy use of Android API code this is mostly used to test the data layer (repositories, view models, etc) as they have this separation by design. Instrumented tests, on the other hand, are harder to implement and execute code on an Android emulator and thus may make use of the Android system, such as its message passing or API calls. All automated tests are implemented using JUnit4.

Typically, the data layer is the most tested as it contains the business logic and the UI layer rarely, if at all, tested as those tests are tricky to implement and are low value. The UI and user experience are tested through the user testing, which covers the functional requirements not covered by the automated testing – if arguably less rigorously.

Instrumentation Tests

There is only one group of code covered by the instrumentation tests and that is the utility functions used to construct a`StoredNotification` objects. This acts as partial cover for PF5, though the actual displaying of stored notifications is not covered. These tests must be instrumented as an Android context is required to query the package manager to extract application names and icon.

getAppName Tests

This test covers the conversion of package names into application names. The test is split into two functions: the first asserts that the result is never null and the second that the result is correct.

Input: the test input is three junk values ("blah", "", null) and three valid values ("com.android.chrome", "com.squadrant.postboxnotification", "com.android.apps.messaging").

Expected Output: The not null test case expects a non-null string to be returned for all inputs. The correctness check expects "(unknown)" for the junk inputs and "Chrome", "Notification Postbox", and "Messages" respectively for the valid inputs.

getTimeSent Tests

This function converts long values into the time that notifications were sent. The returned string is expected to be non null. Since the string is formatted to the users preferred time format (12 hour or 24 hour) the exact result cannot be tested so we check that the returned string is a sane length; obviously erroneous values should be caught by this test or if the function is poorly implemented.

Input: Three reasonable values are given (123456789, 1241, 765453) and three extreme values (Long.MIN_VALUE, Long.Max_VALUE, 0) as the expected input is an offset from Unix time.

Expected Output: All outputs should be between 2 and 10 characters long (two ensures the hour is displayed at least, and ten covers a long display format e.g. HH:MM:SS or HH:MM am).

getAppIcon Tests

This function should return an apps icon as a drawable object. This should never be null. It is not possible to assert that another apps icon is correct as we have no access to a reference file so the tests cover null checking and when that if the input does not have an associated icon a fallback icon is provided.

Input: the test input is three junk values ("blah", "", null) and three valid values ("com.android.chrome", "com.squadrant.postboxnotification", "com.android.apps.messaging").

Expected Output: All outputs should be non-null, additionally the junk values should return an icon identical to 'R.drawable.exclamation_mark' – the fallback icon.

Unit Tests

The unit tests cover the interception service, the view model, and the repository interface (not the local data store implementation class). For testing these components we primarily wish to ensure that appropriate values are forwarded to other classes, validating that the interface between these classes is valid. This is achieved using Mockito. Mockito is a mocking library for Junit, it allows the creation of mocked objects that replace the fully-functioned equivalents. By default it stubs all methods, though it can be used to instead return specified values to specific inputs and capture arguments to function calls on mocked objects. This allows linkage between components to be tested by mocking the other components and returning values as if the components that are not being tested are correctly functioning.

Interception Service Tests

There are several tests ensuring that the interception service is behaving correctly. When describing the behaviour of other components below that refers to the mocked behaviour. For example, 'added to the repository' means that the repository's add method is called.

The first test verifies that when interception is enabled that a notification from a package that should be intercepted is added to the repository.

The second test verifies that when interception is enabled that a notification from a package

that should not be intercepted is not added to the repository.

The third is that when interception is globally disabled even notifications from apps that should be blocked are not – the global switch has primacy.

The fourth, fifth, sixth, and seventh tests check that when a notification is cancelled it only removes the notification from the repository if all of the conditions are correct.

View Model Tests

These view model has three tests that verify correct behaviour. The first is that the LiveData is initially not null, provided that the repository provides a not null LiveData – in other words that the repository dependency is fulfilled.

The second is that the view models add notification method adds an identical stored notification object to the repository exactly once. This tests on stored notification key values: "arbitrary_key", the empty string, and "another_key".

The third is that the removal function for a single notification removes the specified notification, testing on a live data with four stored notifications and removing one of them and then resetting the live data and repeating.

Repository Tests

The repository tests verify that the repository, not the underlying data store class, operates correctly. This includes the maintenance of the `LiveData` variable's value and calls to the local data store.

The first test ensures that the repository is a singleton.

The second test verifies that when a new notification is added that the data store will write it to storage. And the third verifies that adding a new notification also adds the notification to the live data.

The fourth and fifth tests verify that when a notification is updated (a new notification with a key already present in the stored notifications) that the new notification replaces the old notification in both the write call to the local data store and the call to update the live data.

The sixth and seventh tests verify that removing a notification also triggers a write to the local data store without that notification and updates the live data correctly.

Limitation of Automated Tests

The automated tests are a good indicator that something has suddenly broken. However, they do not fully cover the code base nor are tests complete for the components they do cover. An example is the lack of tests for the empty inbox functionality which was overlooked when implementing tests. Additionally, the tests that are implemented do not cover sufficient cases, an example is that the behaviour of the view model when `removeAt` is called on an invalid index or when the repository must remove a notification that does not exist.

This is why the supplemental human testing is important. The automated tests give a sense that the code is okay and the human testing helps verify other behaviour is as expected. However, if future developers intended to expand the functionality of the app or revisit it, expanding the automated test cases , where possible, to be more critical would be an ideal first step.

Test Results

As Figure 34 and 35 shows, all tests passed, with the instrumentation tests being executed on three different emulated devices with three different SDK levels. The low runtime of the tests helps keep development rapid as correctness after changes can be quickly verified.

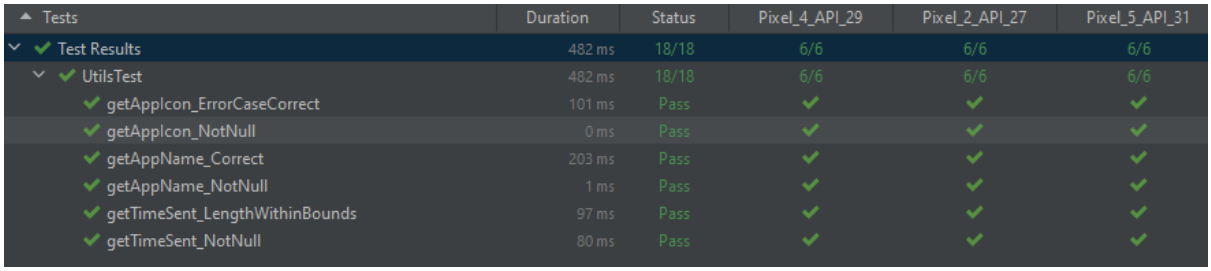


Figure 34: The instrumentation tests results

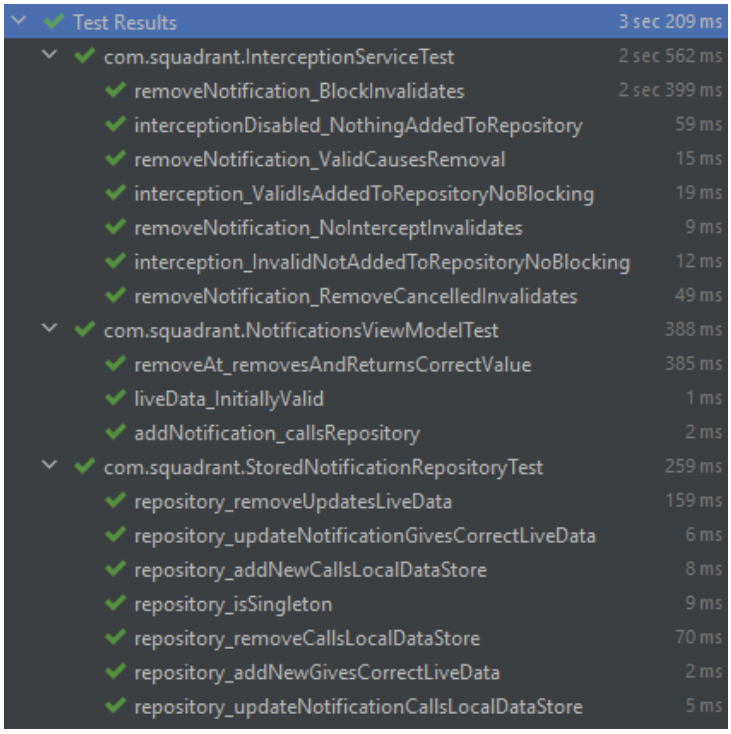


Figure 35: The unit tests results

Exit Points

The testing for the Exit points tool was performed according to the test plan set out in the Design section. It was tested within a python notebook. The results of the tests can be seen in table 12

ID	Req.	Test Description	Resulting Output
1	EP1	Test a range of input values and check the outputs are expected	All output values were as expected for the range of inputs provided
2	EP2	Test the functions using a range of input values and changing the parameters	All functions performed as expected with the different parameters
3	EP3	Test each function and compare their behaviour to ensure they provide different outputs	All functions had different behaviours as expected

Table 11: Test Results for Exit points Tool

The exit point tool was able to fulfill all of its requirements. It is a simple tool so there are not many points of failure for it, hence the small number of tests. The main functionality of the tool is to provide functions based on the time a user has spent on a platform so these are robust as long as a developer provides numerical input greater than one the functions should always be providing a common sense response to the developer.

More serious testing would have to be done within a full social media platform to ensure that the exit points are appearing correctly within a feed of content and that their frequency does not increase too fast so that it takes over the platform. However that sort of testing relies on having an implemented social media platform, producing a dummy platform to fully test out this tool was beyond the scope of this project as producing this dummy platform would have far more work than the benefit it would provide.

Time Transparency

Testing for the time transparency tool occurred after implementing the solution, to validate that the tool was complete to the standards of the requirements. Each test from the test plan was conducted, and the results are reported in Table 12.

Table 12: Test Results for Time Transparency Tool

ID	Req.	Test Description	Resulting Output
1	TT1	Verify that both content tags and time data is stored within a single data structure.	Content tag and timing data is stored within a single data structure called *timings*: a list of tuples with tuples in the form **[ContentTag, TimingNumber]**
2	TT1	Insert ”Gaming, 10”, ”Science, 20” and ”Cooking, 30” into the data structure and print the contents.	The contents of *timings* is **[[”Gaming”, 10], [”Science”, 20], [”Cooking”, 30]]** with no errors.
3	TT1	Insert 20 content tags into the data structure, each respectively named ”A”, ”AB” and so on until ”ABCDEFGHIJKLMNOPQRST”, and print the results.	The contents of *timings* is **[[”A”,1], [”AB”,1],...[”ABCDEFGHIJKLMN OPQRST”, 1]]** with no errors.
4	TT1	Insert 20 content tags into the data structure, each with respective time data in increments of 4320: ”0”, ”4320”, ”8640” and so on until ”86400”, and print the results.	The contents of *timings* is **[[”test1”, 0], [”test2”, 4320],...[”test20”, 86400]]** with no errors.
5	TT2	Initialise a data structure with 100 unique content tags with a timing of 1. Call the update function with identical content tags, with a timing of 2 as the parameters. Print the structure.	The contents of *timings* is **[[”test1”, 3], [”test2”, 3],...[”test100”, 3]]** with no errors.
6	TT2	Repeat test 5, and record how long the process takes during runtime.	The outputted runtime duration was 0.0072 seconds.
7	TT3	Initialise an empty data structure. Call the update function with 100 unique content tags, with a timing of 1 as the parameters. Print the structure.	The contents of *timings* is **[[”test1”, 1], [”test2”, 1],...[”test100”, 1]]** with no errors.
8	TT3	Repeat test 7, and record how long the process takes during runtime.	The outputted runtime duration was 0.0017 seconds.

9	TT4	Initialise a data structure with 100 unique content tags with unique timings, counting up from 1,2,3..100. Pass this data structure to the visualisation function.	A bar plot is produced with 10 bars: content tags *Test100 - Test90*. Each bar is slightly lower than the other, matching the *y* axis between 100-90.
10	TT4	Repeat test 9, and record how long the process takes during runtime.	The outputted runtime duration was 0.68 seconds.
11	TT5	Verify that at least 1 method of allowing a user to select a content tag is demonstrated.	A drop down box method is demonstrated and explained to the toolkit user, showing how the tags can be selected by a potential user via this drop down box.

For the transparency tool, every test produced a successful, expected output which validated the tool against the core requirements. This showed how this tool was representative of the original idea of being transparent with a user's data to enable a user's empowerment to make positive choices regarding their screen-time.

Moderation Model

The moderation model was tested following the test plan created for it in the design section. This was tested in a python notebook, with any timings being the reported time taken to run information, that is provided within Google Colab, this limits the precision of the timings to 1 second intervals. To run a test using 100,000 comments the model was queried with the same set of data 5 times to make up the 100,000 queries. The test set is made up of posts that the model had not seen during training.

ID	Req.	Test Description	Resulting Output
1	M1	Test the moderation model with training data, an effective model should have an accuracy greater than 90%	The model performed well with an accuracy of 92% with a test set of 33,000 entries.
2	M4	Provide a large input file of posts to the model for predict	The model was able to classify a file consisting of 100,000 posts in under a second. This means that the model is able to handle a large number of posts.
3	M3	Manually input test data to ensure the model has the expected results	The outputted labels for these manual inputs was what would be expected for both acceptable and unacceptable comments.

Table 13: Test Results for Moderation Model Tool

The moderation model passed all of its tests, performing much better than was expected. It was able to classify posts much quicker than was expected as well as achieve a very high accuracy while doing so. This means that the moderation model has been proven to be fast and effective, making it very suitable for a developer to use within their own social media platforms, to rely on to keep the majority of unacceptable posts filtered out of their platform automatically.

6.2 User Acceptance Testing

Notification Postbox

The purpose of user testing is to ensure that the app meets the non-functional requirements. There are that the app utilises modern UI design and that the app is easy to use for new users, that it guides them. Additionally, the user testing covers some of the functional requirements – automated testing is not ideal to exactly verify some of the behaviour that is contingent on Android provided functionality. This section will give an overview of two specific instances of user testing and the results generated from it. For analysis on how well each requirement was fulfilled based on the testing, read the evaluation section of the report.

A/B Testing

The first instance of user testing was a UI A/B test, where a representative group of end-users were shown two UI mockups and instructed to indicate which they preferred. This was used in two cases, the first relating to the preferred method of removing notifications and the second the settings page's app list entries.

There were two options explored for removing notifications, either a discard icon that was clickable or the swipe action to discard. When users were presented with this choice, most found that while it was more immediately obvious how to discard the notifications with the clickable icon the swipe action had less visual clutter on screen and aided readability. Many commented that once the action had been learnt there was no real difference between the ease of the two option. This suggested a general preference towards the swipe action as the drawback of a steeper learning curve could be removed with guidance text indicating a swipe action was needed.

The second was whether the app filter, for enabling interception for each app, should have an icon for each app. See Figure 36 to see the two options. Some younger users expressed a preference towards the more condensed view without the icons, however older users and some younger users preferred the icons as it made it cleared what each app was. Icons are the primary means of identifying apps on Android – you tap on an icon to launch the app and the name can be small text below that. This test determined that icons were to be used for the settings page.

Usability Test

A more general user experience test was conducted to verify that users had few issues with using the app. The users for this test had never seen the app before mimicking the expected knowledge of a new user. Below we describe the testing protocol used to ensure a fair test, as well as the results obtained.

The test should be conducted by presenting the user with the app running on a phone or em-

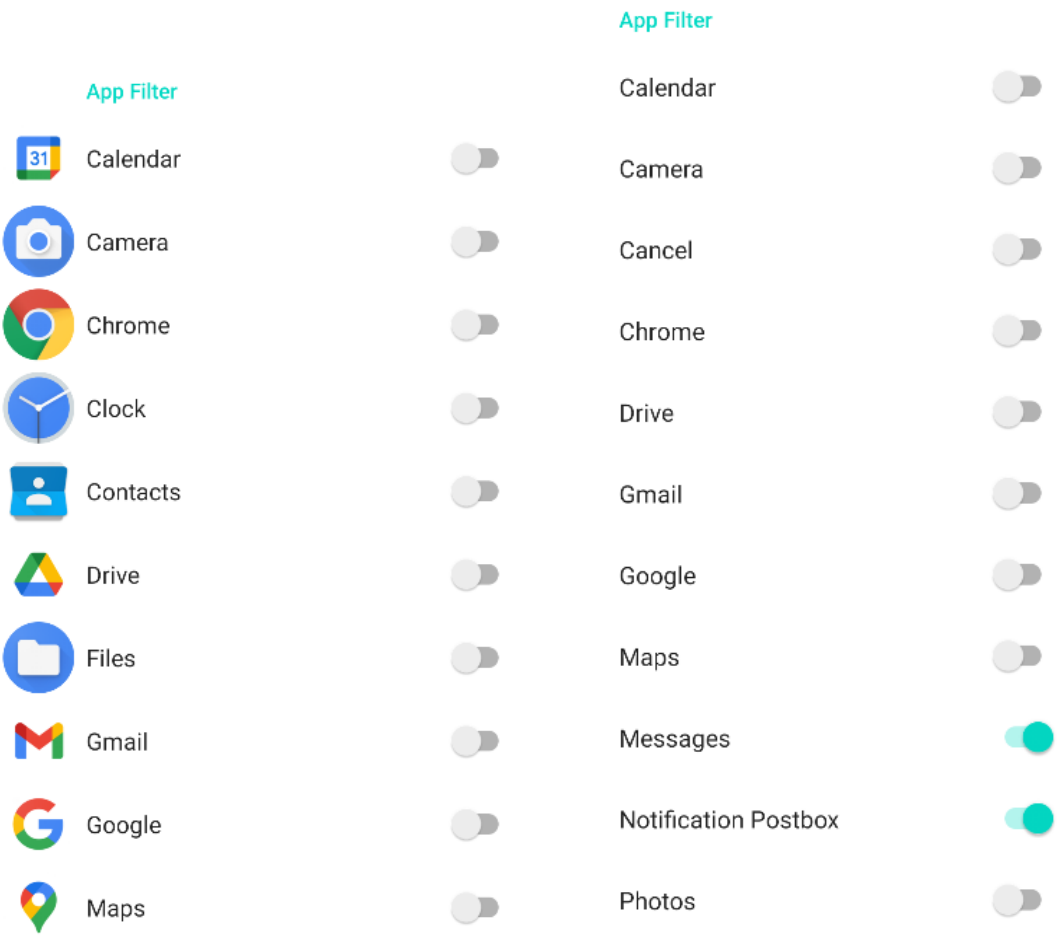


Figure 36: The settings's fragment A/B test

ulator, with the app in the state it is installed in – the service disabled without permission. The user should be told the name of the app and that they will be asked to carry out several operation using the app and to provide comments during the test if they wish. The observer should not provide information on the mechanics of the app to the user, though if the user is unsure as to what is being asked of them, they may rephrase and restate the question. Additionally, the tests require notifications to be generated for interception, since this is not functionality of the app that needs testing the observer may give specific instructions for this purpose, including how to generate notifications.

The user is asked to complete the following tasks in this order:

1. Enable interception
2. Enable capturing of notifications for this app, "Notifications Postbox"
3. Guide the user to generate two notifications using the debug fragment
4. Cancel a single notification
5. Use the debug fragment to have two notifications stored
6. Clear the inbox of all notifications
7. Disable app-wide interception

The expected user behaviour is to go to the settings fragment and toggle the "Enable interception" switch which launches the Android settings page to grant the interception service permission to run. Then return to the app. They should then scroll down the settings fragment to the switch specific to the app and toggle it on. At this point they are guided to create two notifications. They then should navigate to the inbox and swipe one of the notifications – if they empty the inbox instead inform them that they should only remove one notification and generate the notifications again. At this point the notifications should be recreated. Then they should navigate to the inbox and press the empty inbox button. Finally, they should navigate to the settings fragment and toggle the "Enable interception" switch to the off position.

The app was tested on three users, below is a summary of how they completed the tasks and any comments they made.

Step 1 All users managed this step with relative ease. One complaint was that it was a lot to accept as they were prompted by the app to accept the permission, which sends them to the Android settings page where they must enable the permission after another prompt and warning. There is not much that can be done to minimise this, informing the user in the app is the recommended behaviour so that the user knows why the permission is necessary and the Android settings page warning are unavoidable. This is, however, a one time cost and future toggling of the switch does not trigger permission requests, as the interception service retains the system level permission and is only disabled by the toggle.

Step 2 All of the users completed this step, though one user had difficulty understanding what was being asked. They instead enabled the blocking of system UI notifications rather than enabling interception for the specified app. After the instruction was restated they completed the task and commented that if they had been using the app themselves they would not have found it difficult; that the source of the confusion was how the question was worded.

Step 3 This is a debug step and not relevant for measuring app success.

Step 4 All of the users succeeded at clearing a single notification. Though one user did not read the in-app text stating that swiping is the appropriate interaction at first. After approximately 30 seconds they figured out the swipe interaction. All other users performed the intended interaction significantly quicker.

Step 5 This is a debug step and not relevant for measuring app success.

Step 6 All users easily carried out this step.

Step 7 All users easily carried out this step.

Additionally, one user requested dark mode due to personal preference. This was easily facilitated and no difficulties with the operation of the app were had as a result.

The majority of users easily used the full functionality of the app. This suggests that the app is easy to use for new users without guidance, though the number of users is small, so this is not as strong of a conclusion as desired. This ensures that the non-functional requirements were satisfied.

Tool Acceptance Testing

It was planned to perform general user acceptance testing on the guide as a whole. This was done by presenting testers with the entire finished guide to gather opinions from peers with experience in software development, about the efficacy of each tool specifically as to how they could be implemented using the guide instructions, and how useful they believe the tool is to solving the relevant mental health issue.

Testing was conducted by presenting the finished guide to testers using *Google Forms*, and asking them the questions digitally using this same platform. The results of this testing is summarized in Appendix C.

7 User Manual

The user manual for running the deliverables of this project is synonymous with the tool guide delivered by the project. This is included as Appendix A, and contains the complete information required to access and utilise each tool, as well as describing methods of running the code. The code is included within this guide via a link to the *Github* repository, where the code has been commented and organised to be as helpful as possible for a user to understand and follow the code.

8 Evaluation

The project's deliverables and their test results were evaluated to establish the project's outcomes. Considerations were made into analysing the results, and using this to reflect upon the project: specifically what went well and what did not. Possible extensions are also considered within this section, as well as considering the efficacy of the project's management methodology.

8.1 Notifications Spam Filter

Requirements Validation

The requirement **SF1 was successfully met** by the spam filter tool. This is evidenced by its relevant tests passing. This means that the spam filter tool is indeed scalable and efficient enough to be used by a platform on a regular basis.

However, **SF2 was not met**. The tool's model builder takes much too long to handle an extremely large set of training notifications (100,000). This does not indicate a massive failure though, because this requirement was mostly to cover the extreme input case, to check that the tool could handle an unusually large input. This tool did manage to produce a model. It just took 13 minutes rather than the expected maximum of 3 minutes. The model performs perfectly on much less than 100,000 notifications (as shown in the testing section). As a result, this failure can easily be bypassed by truncating the input to a smaller size (for example, 10,000 works well).

On all inputs, including the extremely large edge case of 100,000, the notification predictor was able to make a prediction on a new notification in less than 1 second, meaning that **SF3 was met** by the tool. The predictor has to be used every time there is a new notification, and so it was important that it could perform efficiently. Meeting this requirement was important for the real-world use-case of this tool.

The tool was tested with one day's worth of notifications from all four team members. The tool was able to produce well-performing models for each member, **meeting the requirement SF4**.

Extensions

The testing section found that a third factor may affect the importance of a notification; time of day. Therefore, more research could be done into the area of notifications, and the tool could be extended to accommodate this new possible feature. However, the current tool is scalable to 10,000 inputs from multiple users. So, the extension does not need to be done unless performance of the model begins to drop.

8.2 Notification Postbox

This section is in three parts. The first evaluates how successfully the original requirements for the tool have been met. The second the room for extensions to the app, and ideas for general improvement and implementation of the concepts. Finally, a reflection on the quality of the tool produced and some of the lessons learned during its development.

Requirements Validation

The first requirement, PF1, requires that the app intercepts notifications that meet user defined requirements. User defined was taken to mean coming from specified apps in the design section. This requirement was implemented correctly and can be verified through the automated tests as well as the user testing.

The second requirement, PF2, was validated through user testing throughout development. It was too integrated with Android system calls to validate fully automatically, but is fully functional in the final implementation. The third requirement is that the stored notifications persist between device and app restarts. This was tested throughout development and was fully implemented.

The fourth requirement is that notifications are intercepted even when the app is not open. Again, this was not subject to automated tests but was manually verified and is fully implemented.

The fifth, sixth, and seventh requirements were fully tested through user testing and have been fully implemented.

Requirements eight, nine, and ten were covered partially through user testing as well as manual verification throughout the project and have been fully implemented.

Requirement eleven has been verified through both the automated and user testing, as well as more varied cases throughout its development, it is fully implemented.

The first two non-functional requirements relate to user experience and UI design and were verified to be successfully implemented through the user testing. The experience of the users indicated that they had successfully been realised.

The final non-functional requirements specifies that the app is functional in both light and dark mode. This is the case and the UI for both these cases can be seen in Figure 37.

All of the requirements were met in the final implementation, though more rigorous tests for many of the requirements would be necessary to be confident that the implementation is 'bomb-proof'.

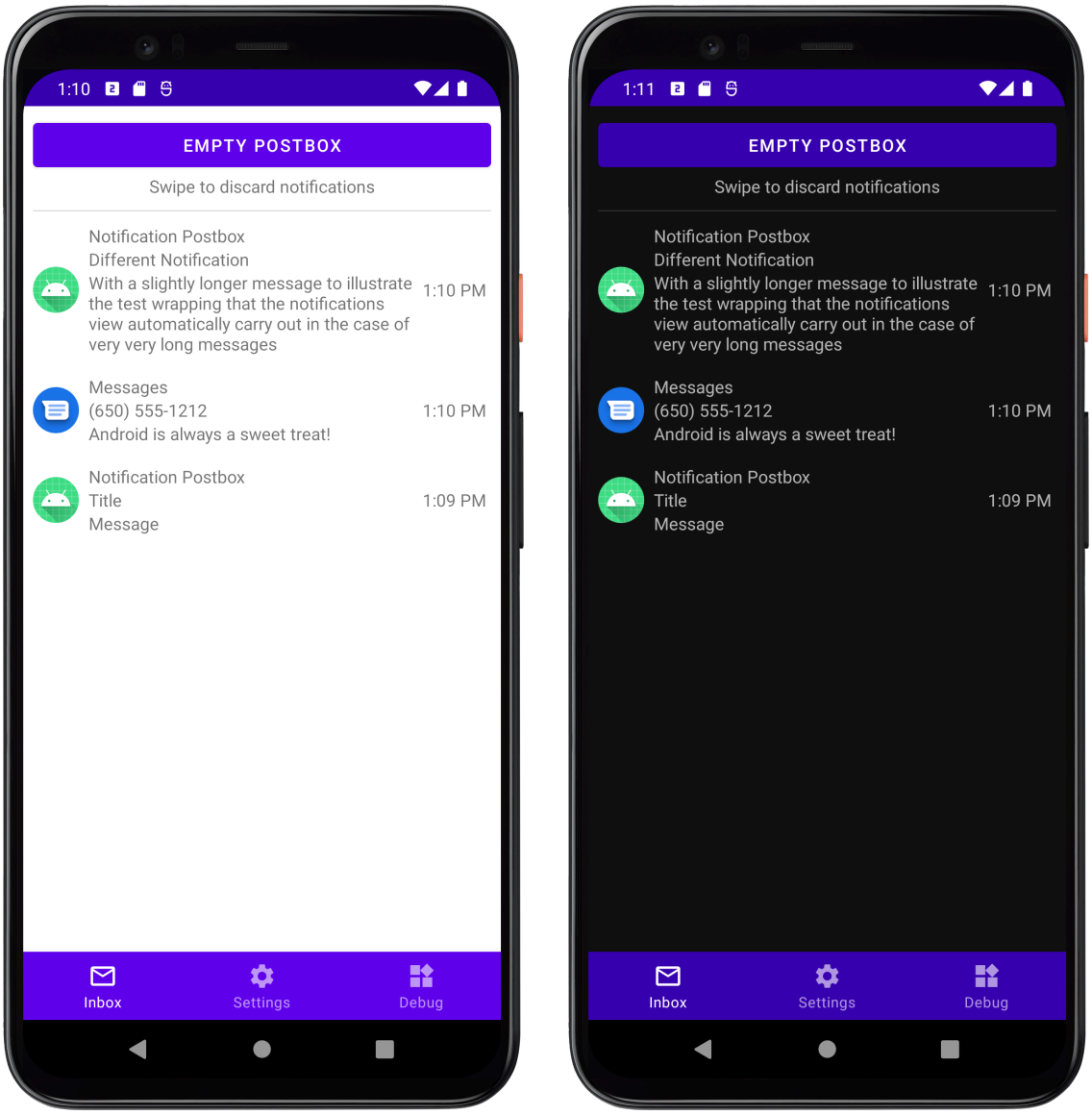


Figure 37: Intercepted notifications displayed in both light and dark mode.

Extensions

There is still room for future expansion of both the app and the ideas that it demonstrates. The first and most critical area for immediate improvement is to improve and expand the test cases as better coverage can certainly be achieved with a little more work; with more familiarity with the testing libraries and methodologies better tests can be constructed. On the same vein the dependency injection, where correctly implemented, is manual and haphazard. To have a more idiomatic codebase the intra-class dependencies should be resolved with a dependency injection library: Dagger and Hilt are purpose built for Android applications and would be very fit for purpose.

In terms of expanding the functionality of the app, greater granularity of control can be given to the user. At the moment interception is controlled globally and on an app-by-app basis. Each app may deliver many types of notifications on different notification channels, allowing the user to specify notifications interception on a channel-by-channel basis is more complex but could be very beneficial.

Additionally, blocking of system UI notifications is only a global toggle – the user could be given the means to control the specific behaviour of interception and blocking separately for each app, or even by each channel. This would allow, for example, Instagram to send direct message notifications to the system UI and have 'liked image' notifications be captured and blocked from the system UI. This would certainly add additional value without greatly increasing the scope of the app.

A third area of potential expansion is making the app function across multiple devices, or even platforms. Whilst an iOS app is not possible notification interception for desktop web browsers may remain feasible and some users may have multiple devices that produce notifications that they want corralled in one place. To implement this would require significant architectural changes in the data layer as a web server would need to be implemented and the repository must reliably access this. The UI would also need to be modified to indicate the device each notification is from as well as display loading messages when waiting for web request responses. The settings page and storage mechanism would also need reworking to better support the new behaviour.

For developers, the app functions as a good conceptual interface for notifications and, as Android 13 will not grant push notification permission to apps by default, having an in-app feed of notifications that can be pruned and managed more similarly to an inbox may have greater value. For developers the exact nature of their own apps will necessitate deviation from the design and possibly even architecture discussed and implemented here, but the concept has been shown to be feasible and, given longer term testing, may be of significant value.

Reflection

This tool is the proverbial ugly duckling of the project. Conceived earlier than the other tools it does not quite fit into the toolkit, having retained its user focus rather than recentring on the developer as other tools have. Nonetheless the tool shows value to both the project and to users in general; even developers may find some value in the concepts presented.

In terms of satisfaction with the produced software the UI and user experience design is very satisfactory. Use of standard libraries and techniques led to significant learning of how UI is typically implemented and why the architectural patterns associated with it are used. One early misstep was diving straight into development without a full reading of the Android documentation, this led to architectural woes that could have been avoided. It seemed unnecessary to fully implement the MVVM design pattern for such a simple app but the compounding complexity stalled development until this was reconsidered. Once the switch to the MVVM pattern was made further development was incredibly easy and the means to write meaningful and useful tests became apparent as the components became correctly decoupled.

That is not to say that all of the lessons were learnt in time. The cognitive load of understanding the needed design pattern and the integration with Android as well as the testing libraries, which the team had not used in any meaningful capacity before, meant that the resolution of dependencies was improper and not clean code. This is the next step for improvement for the development team: understanding the dependency injection pattern better in terms of both manual implementation, which was attempted and not fully realised; as well as automated implementation through Dagger or Hilt.

Despite the development difficulties and rough edges, the produced tool does function to specification and significant improvements in the team's understanding of how the Android ecosystem functions were made. The path to further professional development is clear and the lessons learnt are highly transferable to other UI systems (Windows Presentation Framework for desktop UI works similarly).

8.3 Exit Points

Requirements Validation

The requirement **EP1 was successfully met** by the exit point tool. It is able to provide the recommended frequency of exit points when it is queried. How effective the final tool is has a high dependency on how the exit points are designed, which has been left to developers to implement, so is unable to be tested. However, the functions provided to follow the expected behaviour that should allow the tool to be effective.

The requirement **EP2 was also successfully met**. The functions provided within the tool kit successfully allow for a developer to customise them freely to ensure that they can tune the per-

formance of the function to their exact needs. Similarly, the requirement **EP3 was successfully met** by providing a variety of functions which all had different rates of change over time. This type of differences allow a developer to select the function which scaling best applies to their platform, and then tune it to their exact needs.

Limitations

The main limitation of the tool is it does not provide examples of exit points for a developer to use as inspiration to create their own. However, it does provide examples from other industries. This was done as to make an appropriate example for each generic type of social media we could think of would have taken a long time, which would have taken away from the other more important tools. As well as these, even with a selection of example exit points they may still not be appropriate for a developer's social media platform which would then still require them to create their own implementations. Therefore this limitation of the tool was not catered for due to the lack of resources to achieve it.

Extensions

The main extension to this tool would be to provide the example exit points for a developer to either use directly or take inspiration from and develop their own. This would allow the tool to ensure it is correctly guiding a developer to create an effective exit point in addition to ensuring they are well spaced throughout the social media platform. Another extension would be for a user to be able to set a limit to their social media use by creating a hard limit. This would create a mandatory exit point that the platform enforces by timing a user out. This is only appropriate as a user set limit as each users' needs will be very different and a user would only accept such a system if they were able to have full control over it. if they desired to use it at all. This would allow user's who have a fairly heavy addiction to social media to better control their time spent on the platform when exit points are too soft a suggestion.

8.4 Time Transparency

Requirements Validation

Using the results of the technical testing on this tool, conclusions can be made as to the success of the tool: in reference to it meeting the original requirements.

The tests relating to requirements **TT1, TT2 and TT3 all passed**, showing that this tool was effective in demonstrating how a user's time data viewing content can be stored in a single scalable variable, thus the tool is able to compute the preliminary work required for this tool solution.

The **TT4 requirement tests succeeded**, meaning the tool was proven successful in being able to visualise a user's time data and effectively *be transparent* with a user's data.

Finally, the **TT5 requirement tests succeeded** also, meaning the tool was successful in providing a basic means of how the user can select a content tag to limit their exposure to, effectively *empowering* the user.

Through this testing, this tool has succeeded in applying the humane design principles to the screen-time reduction issue, to produce a solution which combats the core idea of addictive design, and by combating this issue, the resources have been provided to reduce the mental harm done by common features within social media platforms designed to be addictive.

Limitations

The tool being fully completed and documented within the guide meant the limitations of this tool could be considered.

It can be noted that strictly following the principles of transparency and empowerment limited the tool's efficacy, in that there is no certain enforcement of reducing a user's screen-time due to the tool's nature of solely empowering the *user* to make their own choices to limit their own screen-time. Hence, if a user is not aware of the unhealthy damage caused to them by extended lengths of screen-time, or they generally do not feel compelled to want to manage their screen-time, then this tool would be ineffective for these kinds of users. This tool being implemented within a platform is innately suited toward a user base that is active in maintaining their well-being, and hence may not suit a generalized user audience: essentially limiting this tool's efficacy. This does not damage this tool's overall success, as the tool is still an effective solution for users wishing to manage their screen-time habits, the tool is just limited by the audience it is useful for. To reduce this limitation's effect, the tool could be extended when implemented to help user's understand the risks for extended screen-time to help broaden the user base where this tool would be effective, as more users would feel the need to use this tool effectively.

Due to the project's limited resources of time and people-power, the tool's scope was limited. The scope had to be established to both maximize the demonstration and application of the technical tool, whilst minimizing the cost of time spent implementing this tool to ensure the project delegated its time resources evenly without skewing toward any one tool. Hence, this balance was struck between demonstrating the visualisation, collection of time data and the means of choosing which content tags to limit, but the scope specifically did not consider how content could actually be limited, or actually technically implementing the system into a larger, existing system. The scope did not investigate these sections due to the low reward of technical outcome compared to the time that would be required to achieve this implementation. Hence, the tool was innately limited in what could be demonstrated due to the project's time management, but overall the tool's success at demonstrating this idea did not suffer from these technical limitations. Given more time, this tool could be investigated further by implementing it within an

existing system and measuring its ability to integrate together effectively, rather than individual, slightly disconnected code cells.

Overall, some aspects of the tool were limited, however none of these limitations damaged the success of what the tool set out to achieve.

Extensions

The tool could be extended in numerous ways to further deal with the screen-time reduction issue.

A technical extension could be to deal with some extreme use cases of the tool, including erroneous time data. For example, if a user accidentally leaves their device idle viewing some content, the time data may become extremely skewed, and no longer represents the true habits of the user. Hence, an extension could be to mitigate this potential issue. A potential mitigation could be to have a timeout feature when viewing content, which would stop recording timings after a reasonable amount of time: which would be a primitive solution to dealing with an incorrect amount of content viewing.

An extension from a non-computational background could be to measure the efficacy of this tool against human usage. Whilst this project conducts a general enquiry into how users *feel* about how effective a tool is, a more extensive, sociological enquiry could be conducted to measure how effective the tool is when applied within a real environment. This could include a long term study into two groups of users with similar, consistent screen-times where one group would utilise the tool, whilst the other group would not. By measuring the two group's screen-times, prominent conclusions could be made about precisely how effective the tool is at reducing screen-time.

Additionally, a general extension for this tool would be to implement the idea within an existing system as intended by this project, using the toolkit guide demonstrations for guidance.

8.5 Moderation

Requirements Validation

The requirement **M1 was successfully met** by the moderation model. It was able to predict the correct label for a comment with a 92% accuracy which was accurate enough to fulfill the threshold set by the requirement for it to be considered effective.

The requirement **M2 was successfully met** by creating a software solution for create a tool for effective moderation. The model is able to be queried without any manual intervention by integrating it within the platform so that any new posts created by a user are passed to the model first. This allows the tool to operate automatically while also checking all comments and posts

posted to the platform.

The requirement **M3 was successfully met** by the dataset that the model was trained using. And has been verified by testing, with manual inputs, posts that would be deemed unacceptable due to targeting certain groups. Any clear biases would have allowed posts that should have been unacceptable to be posted but this did not occur. This is expected as the dataset was annotated by a large number of people so the chance of a biases being inherited is low.

The requirement **M4 was successfully met**. This is shown with it being able to handle a very large number of queries in less than a second. By providing the model with 100,000 posts to query at once the model was able to quickly classify all of these. This means that it is expected to be able to be suitable for a large social media platform as it is fast enough to handle an expected high throughput of posts and comments. Any platform which receives less than 100,000 new comments a seconds would be able to comfortably make use of this model.

Limitations

The main limitation of the moderation model is when language evolves over time. The model is unable to learn during run time so would not learn if a completely new word starts to be used on the platform is offensive or not. The model is able to be retrained and it was able to train in five seconds when trained initially. However the dataset would need to be maintained and updated to include labelled example posts which include any new words that need to be understood by the model. This is less important for new words which have a neutral or positive sentiment as could be safely ignored, however for any new slurs that become used on a platform this could allow a number of offensive posts to slip past the moderation model as it does not know how to treat these words. This however is an uncommon event.

Extensions

The main extension to the moderation model would be adapting it so it is able to learn how to deal with new words. This could be done by introducing two tokens that could be substitute for new words. A token to replace offensive words and a token to replace non-offensive words. The dataset would need to be adapted as well to contain examples of these tokens so the model can learn an embedding for the tokens. Then during runtime a secondary model can be used to classify any new words it has not seen before into two groups corresponding with the tokens based on the surrounding sentence the words appear within. Then these words can be substituted with tokens and passed to the moderation model, this would allow new words to be learnt without having to retrain the whole system, or update the dataset.

8.6 User Acceptance Testing Results

The results of the user acceptance testing were analyzed to gather thoughts as to how effective the guide was at being a resource for developing similar tools. The project aimed to achieve an average score of at least 7/10 per tool for how the tool's were presented, and this was achieved for every tool. This result evidenced how the guide had been reasonably effective in explaining the tools well enough so that they could be implemented by users effectively: allowing the resource to function as designed.

Additionally, this testing also covered how users felt about the potential for each tool in aiding users with their well-being, which was shown that every tool attained a score of at least 7/10 in these questions. This showed that users believed the tools would be useful if implemented correctly and gave further justification that the tools had been derived correctly and successfully. Furthermore, this testing did not intend to serve as the project's main success validator (this was primarily the technical tests and customer acceptance tests), but rather serve as a main evaluator for informing which tools could benefit from being extended.

In conclusion, these specific test results did not infer that any tool was identified as agreeably problematic, and the results showed only success, which highlighted that generally, the tools had been communicated and inferred effectively.

8.7 Customer Feedback

The final guide was sent to the customer to have them provide feedback on if they felt the toolkit met their expectations and they were happy with the result. They expressed that they were impressed with the scale of the guide, with the amount of tools there were and the functionality that this provided.

They also provided some feedback addressing extensions they would like to see, specifically regarding two of the tools, the notification spam filter tool and the exit points tool. The extension to the filter tool they would have liked to seen was to have a basic model for each user which gets updated over time automatically based on the notifications they receive. This would make the tool need less manual input, and therefore reduce the cost and complexity to maintaining this tool. The feedback on exit points was to include a hard limit to a users time on a platform. This would be set by the user to control the upper limit of their usage of social media platforms.

This feedback provides a clear direction for any extensions to the toolkit we made. As stated the introduction, this toolkit is expected to be able to extended by other developers so we were expecting to be able to extend it. Having clear feedback from the customer stating what they would like to see more of gives this extension a clear direction.

The feedback overall was very positive, showing that the customer was happy with what was produced from the project and were impressed by the scope of solutions provided. Due to this

feedback we feel confident in saying that the project was successful.

8.8 Project Management

There were some parts of the project which were over ambitious, such as initially planning two sprints within term 2 instead of one. As the team learned more about each other and how we work during term 1, we were able to refine scope to allow for a more realistic schedule and output. This prevented the team from being over-burdened by an unsustainable workload, which would have lead to not being able to finish planned parts of the project.

A main struggle for the team was staying motivated on the project when having to multi-task doing courseworks and doing project work at the same time. We were realistic about this as a team and knew that it would likely be an issue, so we embedded float time into the schedule to accommodate for any unexpected delays during development. In retrospect, it would have been better to have analysed the timetables and carved out dedicated days for working on the project, so that it would have time allocated to working on it. However, realistically, having a stricter schedule would likely have been ignored by the team, as the problem was that we did not see the project as a priority, when more urgent courseworks were due. This could have been improved by designing efforts to maintain excitement and motivation for the project. For example, having regular social sessions about the project as a whole and what we want to achieve as a team. Looking forward, the team now sees the importance of keeping up morale throughout a project, especially one as long as this has been. In future long-term projects, the team members intend to pay more focus to making sure everyone on the team is excited and cares about their work, in order to increase the chance of project success.

9 Conclusion

The project overall was successful because we achieved what we had set out to accomplish. We thoroughly and logically argued the need for the toolkit using extensive research into social media areas that cause mental health problems for users. We narrowed the scope down to developing a set of open-source tools for reducing screen-time and providing effective moderation, which were both proven by the research to be effective at improving the mental health of users.

In the goal of reducing screen-time, we found that reducing the number of notification triggers a user receives would help to reduce the amount of time they spend on social media. We implemented two tools in this area. One as a back-end tool (notification spam filter) that can be used to automate the reduction of notifications. Another as a front-end tool (notification postbox) that lets users toggle which apps they want to be notified by and stores the ones that they do not.

Another way of reducing screen-time, that was developed by the team, was to provide exit points during the endless scroll on social media. We successfully implemented back-end func-

tions for how to know when an exit point should be used, as they should not be too frequent or too sparse. These can be integrated with a social media platform to provide exit points at an optimal time.

The final way of tackling screen-time reduction, that we developed, was a proof-of-concept time visualisation tool, which can produce visualisations to a user about how much time they spend looking at different types of content on social media. Being transparent with a user about their interaction habits allows them to take control and reflect on how much time they spend online, in turn reducing how much time they spend scrolling.

In the goal of providing effective moderation, we developed an effective, open-source moderation tool, that can be used as a back-end filter to flag offensive content, preventing vulnerable users form being exposed to it.

All of these tools were successfully implemented by the team, as proven by the testing and evaluation sections of this report.

The toolkit is not exhaustive, nor was it intended to be, given the time-frame of this project. The tools produced are open-source and modular, which allows any developers to easily add new open-source tools which they develop in the goal of protecting the mental health of social media users. Developers are encouraged to extend the toolkit with their own tools, to help in the goal of there being an open-source resource for improving mental health on social media.

The project has successfully created a guide for the toolkit which explains the tools clearly, so that a developer can take them and use them to improve their own users' experiences and mental health. This report encourages developers to extend and integrate the tools with their own platforms.

10 References

[1] Amazon Rekognition. *Automate your image and video analysis with machine learning*. [Online] Available at: aws.amazon.com/rekognition/m. Accessed: 2022-03-20.

[2] Andrew K. Przybylski, Netta Weinstein. *A Large-Scale Test of the Goldilocks Hypothesis: Quantifying the Relations Between Digital-Screen Use and the Mental Well-Being of Adolescents*. [Online] Available at: https://doi.org/10.1177%2F0956797616678438. Accessed: 2022-04-23.

[3] Android. *What to test in Android*. [Online] Available at: https://developer.android.com/training/testing/fundamentals/what-to-test Accessed: 2022-05-02. 2021.

[4] Android. *Guide to app architecture*. [Online] Available at: https://developer.android.com/jetpack/guide Accessed: 2022-05-02. 2022.

[5] Android. *Digital Wellbeing New ways to find balance for you and your family.* [Online] Available at: https://www.android.com/intl/en_uk/digital-wellbeing/. Accessed: 2022-04-23.

[6] Android Developers. *Create and Manage Notification Channels*. [Online] Available at: https://developer.android.com/training/notify-user/channels. Accessed: 2022-04-25.

[7] Apple Support. *Manage notifications from News on Mac*. [Online] Available at: https://support.apple.com/en-gb/guide/news/iph82f485965/mac. Accessed: 2022-04-25.

[8] BeReal. *BeReal. Your friends for real.* [Online] Available at: https://play.google.com/store/apps/details?id=com.bereal.ft&hl=en_GB&gl=US. Accessed: 2022-04-23.

[9] Jud Brewer. *How to Break Up with Your Bad Habits*. [Online] Available at: https://hbr.org/2019/12/how-to-break-up-with-your-bad-habits. Accessed: 2022-04-23.

[10] Cheeken. *Sort a list of tuples by 2nd item (integer value)*. [Online] Available at: https://stackoverflow.com/questions/10695139/sort-a-list-of-tuples-by-2nd-item-integer-value. Accessed: 2022-04-23.

[11] Corsham Institute. *Written evidence submitted by Corsham Institute (SMH0147)*. Accessed: 2022-04-23. URL: http://data.parliament.uk/WrittenEvidence/CommitteeEvidence.svc/EvidenceDocument/Science%20and%20Technology/Impact%20of%20social%20media%20and%20screen-use%20on%20young%20people%E2%80%99s%20health/Written/81343.html.

[12] Emmanuel Gbenga Dada et al. "Machine learning for email spam filtering: review, approaches and open research problems". In: *Heliyon* 5.6 (2019), e01802. ISSN: 2405-8440. DOI: https://doi.org/10.1016/j.heliyon.2019.e01802. URL: https://www.sciencedirect.com/science/article/pii/S2405844018353404.

[13] Deyan G. *How Much Time Do People Spend on Social Media in 2022?* [Online] Available at: https://techjury.net/blog/time-spent-on-social-media/#gref. Accessed: 2022-04-23.

[14] ESRC International Centre for Lifecourse Studies in Society and Health. *Written evidence submitted by ESRC International Centre for Lifecourse Studies in Society and Health (ICLS) (SMH0128)*. [Online] Available at: http://data.parliament.uk/WrittenEvidence/CommitteeEvidence.svc/EvidenceDocument/Science%20and%20Technology/Impact%20of%20social%20media%20and%20screen-use%20on%20young%20people%E2%80%99s%20health/Written/81200.html. Accessed: 2022-04-25.

[15] Hoare, E., Milton, K., Foster, C., Allender, S. "The association between sedentary behaviour and risk of anxiety: a systematic review". In: *Int J Behav Nutr Phys Act* 13.108 (2016). ISSN: 2405-8440. DOI: https://doi.org/10.1186/s12966-016-0432-4.

[16] House of Commons Science and Technology Committee. *Impact of social media and screen-use on young people's health*. [Online] Available at: https://publications.parliament.uk/pa/cm201719/cmselect/cmsctech/822/822.pdf. Accessed: 2022-04-23.

[17] HT TECH. *How to cut down your screentime on YouTube*. [Online] Available at: https://tech.hindustantimes.com/how-to/how-to-cut-down-your-screen-time-on-youtube-71590517189361.html. Accessed: 2021-11-23.

[18] Ian Saunders. *CS352-15 Project Management for Computer Scientists*. [Online] Available at: https://warwick.ac.uk/fac/sci/dcs/teaching/modules/cs352/. Accessed: 2020-12-15.

[19] Jean M. Twenge, Thomas E. Joiner, Megan L. Rogers, Gabrielle N. Martin. *Increases in Depressive Symptoms, Suicide-Related Outcomes, and Suicide Rates Among U.S. Adolescents After 2010 and Links to Increased New Media Screen Time*. [Online] Available at: https://doi.org/10.1177/2167702617723376. Accessed: 2022-04-23.

[20] John Hunter, Darren Dale, Eric Firing, Michael Droettboom and the Matplotlib development team for 2002 - 2012 The Matplotlib development team for 2012 - 2022. *Matplotlib 3.5.1 documentation*. [Online] Available at: https://matplotlib.org/stable/. Accessed: 2022-04-23.

[21] Keipi, T. *Now you see me, now you don't: A study of the relationship between Internet anonymity and Finnish young people. Doctoral dissertation: University of Turku.* [Online] Available at: https://www.utupub.fi/bitstream/handle/10024/113050/

AnnalesB405KeipiDISS.pdf?sequence=2&isAllowed=y. Accessed: 2022-04-02. 2015.

[22] Chris J Kennedy et al. "Constructing interval variables via faceted Rasch measurement and multitask deep learning: a hate speech application". In: *arXiv preprint arXiv:2009.10277* (2020).

[23] Kira E. Riehm and Kenneth A. Feder and Kayla N. Tormohlen and Rosa M. Crum and Andrea S. Young and Kerry M. Green and Lauren R. Pacek and Lareina N. La Flair and Ramin Mojtabai. "Associations Between Time Spent Using Social Media and Internalizing and Externalizing Problems Among US Youth". In: *JAMA Psychiatry* 76.12 (2019). DOI:10.1001/jamapsychiatry.2019.2325, pp. 1266–1273.

[24] L. Lucero. *Safe spaces in online places: social media and LGBTQ youth*. Multicultural Education Review, vol. 9, no. 2, pp.117-128, 2017.

[25] Marwick, A. E., & Miller, R.W. *Online harassment, defamation, and hateful speech: A Primer of the legal landscape.* Fordham Center on Law and Information Policy Report. 2014.

[26] Meta. *Stories ads*. [Online] Available at: https://www.facebook.com/business/ads/stories-ad-format#. Accessed: 2022-04-23.

[27] Mobius Labs. *Superhuman Vision for every applicationg*. [Online] Available at: www.mobiuslabs.com. Accessed: 2022-03-20.

[28] Moderate Content. *Realtime Image Moderation API to protect your community*. [Online] Available at: moderatecontent.com. Accessed: 2022-03-20.

[29] N. Andalibi, P. Ozturk, A. Forte. *Sensitive Self–disclosures, Responses, and Social Support on Instagram: The Case of #Depression*. In Proc. 2017 ACM Conference on Computer Supported Cooperative Work and Social Computing, 2017, pp. 1486–1500.

[30] Overleaf. *Overleaf Documentation*. [Online] Available at: https://www.overleaf.com/learn. Accessed: 2022-04-05.

[31] Pardhan S, Parkin J, Trott M, Driscoll R. *Risks of digital screen time and recommendations for mitigating adverseoutcomes in children and adolescents*. Sch Health. 2022; DOI: 10.1111/josh.13170.

[32] Patti M. Valkenburg, Jochen Peter. *Social Consequences of the Internet for Adolescents: A Decade of Research*. [Online] Available at: https://doi.org/10.1111/j.1467-8721.2009.01595.x. Accessed: 2022-04-23.

[33] Pattr. *Create extraordinary customer experiences*. [Online] Available at: https://www.pattr.io/. Accessed: 2022-03-20.

[34] Preamble. *Middleware for AI Ethics and Safety*. [Online] Available at: https://www.preamble.com/. Accessed: 2022-03-20.

[35] Project Management Institute. *A guide to the project management body of knowledge*. 6th ed. Newtown Square, Pennsylvania, USA: Project Management Institute. 2017.

[36] Python Software Foundation. *Python 3.10.4 documentation*. [Online] Available at: https://docs.python.org/3/. Accessed: 2022-04-23.

[37] Python Software Foundation. *time — Time access and conversions*. [Online] Available at: https://docs.python.org/3/library/time.html. Accessed: 2022-04-29.

[38] Royal College of Paediatrics and Child Health. *Written evidence submitted by the Royal College of Paediatrics and Child Health (SMH0156)*. [Online] Available at: http://data.parliament.uk/writtenevidence/committeeevidence.svc/evidencedocument/science-and-technology-committee/impact-of-social-media-and-screenuse-on-young-peoples-health/written/82134.html. Accessed: 2022-04-25.

[39] S. L. Craig, L. McInroy. *You Can Form a Part of Yourself Online: The Influence of New Media on Identity Development and Coming Out for LGBTQ Youth*. Journal of Gay and Lesbian Mental Health, vol. 18, no. 1, pp. 95–109, 2014.

[40] Scikit Learn. *1.11. Ensemble methods*. [Online] Available at: https://scikit-learn.org/stable/modules/ensemble.html#categorical-support-gbdt. Accessed: 2022-04-25.

[41] Adam Scott. *The eight rules of good documentation*. [Online] Available at: https://www.oreilly.com/content/the-eight-rules-of-good-documentation/. Accessed: 2022-04-05.

[42] SELMA. *Hacking Online Hate: Building an Evidence Base for Educators*. [Online] Available at: https://hackinghate.eu/assets/documents/hacking-online-hate-research-report-1.pdf. Accessed: 2022-04-04.

[43] SELMA. *Hacking Online Hate: Insights from the SELMA Project*. [Online] Available at: https://hackinghate.eu/assets/documents/hacking-online-hate-insights-from-the-selma-project.pdf. Accessed: 2022-04-04.

[44] Teychenne, M., Costigan, S.A., & Parker, K. "The association between sedentary behaviour and risk of anxiety: a systematic review". In: *BMC Public Health* 15.513 (2015). ISSN: 2405-8440. DOI: https://doi.org/10.1186/s12889-015-1843-x. URL: https://bmcpublichealth.biomedcentral.com/articles/10.1186/s12889-015-1843-x.

[45] LLC The Business Professor Jason Gordon. *Content Tags Explained - What are Content Tags?* [Online] Available at: https://thebusinessprofessor.com/en_US/seo-social-media-direct-marketing/content-tags-definition. Accessed: 2022-04-23.

[46] Christina V, Karpagavalli S, and Suganya G. "A Study on Email Spam Filtering Techniques". In: *International Journal of Computer Applications* 12.1 (Dec. 2010). Published By Foundation of Computer Science, pp. 7–9.

[47] Waikato University. *Weka 3: Machine Learning Software in Java*. [Online] Available at: https://www.cs.waikato.ac.nz/ml/weka/. Accessed: 2022-04-21.

[48] Web Purify. *World-Class Image Moderation & More*. [Online] Available at: www.webpurify.com. Accessed: 2022-03-20.

[49] Jon Yablonski. *Empowering Design*. [Online] Available at: https://humanebydesign.com/principles/empowering/. Accessed: 2022-04-26.

[50] Jon Yablonski. *Humane by Design*. [Online] Available at: https://humanebydesign.com/. Accessed: 2022-04-26.

[51] Jon Yablonski. *Transparent Design*. [Online] Available at: https://humanebydesign.com/principles/transparent/. Accessed: 2022-04-26.

Appendix A

A Toolbox for Reducing Mental Health Issues Caused by Common Social Media Features

Contents

Introduction

This toolbox serves as a fundamental resource for any social media based software developer aiming to effectively consider the general mental health of their platform's user base.

With ever growing developments in social media as an industry (for example, the idealised concept of virtual worlds based on social connection, namely *Meta*[1]), the focus on encouraging digital social connections between users can sometimes be so prevalent that the negative, social repercussions of this encouragement can be ignored, namely mental health issues. As a software developer, these issues are rightfully difficult to accommodate for and mitigate. The issues can sometimes be intractable, such as sociological and psychological issues, where no algorithm or computation exists to solve them. Hence, this toolbox serves as a resource for software developers wishing to combat the negative effects their social media platform has on their user base's mental health.

To achieve this, the toolbox is structured around providing the reader with information about a number of thoroughly researched, adaptable tools for reducing or eliminating the damage done to a user's mental health by common social media features, specifically:

- **Reducing a User's Screen Time** - To combat the addictive design of social media platforms that evokes a sedentary, unhealthy lifestyle in their user base.
- **Improved Moderation** - To improve the existing method of moderating user-based content, providing a tool to reduce toxicity, bullying and the presence of hateful communities in social media platforms.

For each tool, the general steps of technical implementation are explained and detailed with the goal being that these instructions can be followed by the reader to adapt the tool for any social media platform or context. Additionally, code-based demonstrations, commented code and visualisations accompany the step-by-step implementation instructions where necessary, to further explain a tool's functionality and help explain its concept.

Each tool has also been fully implemented on at least one well suited platform to serve as a full technical reference. Each of these implementations can be found in the following Github repository https://github.com/Squadrant/Humane-social-media-guide. The directory structure can be seen in the image below. The reader is encouraged to use, adapt, and integrate the implementations into their own platforms.

[1] https://about.facebook.com/meta/

```
Mental-Health-Toolkit-for-Social-Media
├── Notification Postbox Tool
|   └──PostboxNotification                    (the Android Studio project directory)
|       └── app/src
|           ├── main
|           |   ├── java/com/squadrant        (java source files)
|           |   └── res                       (resources directory)
|           ├── test/java/com/squadrant       (unit tests)
|           └── androidTest/java/com/squadrant  (instrumented tests)
├── Spam Filter Tool
|   ├── Spam Filter Model Builder.ipynb
|   ├── Notification Predictor.ipynb
|   ├── Example Inputs and Outputs
|   |   ├── Example Notifications.csv
|   |   ├── Spam Filter Model.joblib
|   |   └── Notification Prediciton.csv
|   └── Testing
├── Exit Points Tool
|   └── Exit_Points.ipynb
├── Time Visualisation Tool
|   └── Time_Visualisation_Tool.ipynb
├── Moderation Tool
|   ├── moderation_model.ipynb
|   ├── Visualisation.ipynb
|   └── Image_to_Text.ipynb
└── Guide.pdf
```

As can be seen from the directory structure of the toolbox, most tools have been developed as Python notebooks (.ipynb files). This is because Python is easy to read, has a lot of supported libraries, and is very portable because of services like Google Colab. The simplest way to use these tools is to upload the Python notebook files into a Google Drive, and then open the files in Google Colab and run them according to the guide instructions for the relevant tool. The tools can be run in any Python notebook editor, e.g. Jupyter Notebook, though they were developed using Google Colab exclusively.

The one exception to the Python implementation style is the Notification Postbox Tool. This is a user facing feature and so was developed as an Android application. More details can be found in the Notification Postbox section of this guide.

Reducing a User's Screen Time

A common feature in social media platforms is their addictive design; they are developed in specific ways to keep the user engaged on their platform for as long as possible. From a business perspective this feature is justified, as a platform's revenue model is usually related to how long a user spends browsing a platform and interacting with content: such as advert revenue. However, these design choices can have a negative impact on the mental health of a regular user.

Addictive design choices encourage addictive interactions between a user and a social media platform which in turn encourages a sedentary lifestyle, as a user develops an unhealthy reliance on the addictive cycle provided by the platform. This sedentary lifestyle has links to poor mental health.[2] [3]

The tools provided for this issue aim to minimise the effects of addictive design, whilst still maintaining the benefits for a platform's business case.

Specifically, this toolbox provides the tools to combat these identified addictive design features:

- **Notifications as an Addictive Trigger** - For mobile based social media platforms, notifications can act as a means of luring users back into spending time on a platform when the user was not originally intending to spend time on the platform. For example, if a user receives a notification that a friend has uploaded a picture, this notification is not urgent but could feed the addictive loop by encouraging the user to use the platform and spend time browsing content.
  For this feature, the toolbox proposes a **Notification Post Box** and a **Notification Spam Filter**. These features are explained in detail in the corresponding sections of this guide.
- **Endless Content** - Many platforms deny users 'stopping points' whilst browsing platform content. When content is constantly fed to a user in an endless scroll, they struggle to find a reason to stop using the app. Consider the principle of inertia - users will keep scrolling unless triggered to stop. This design to keep users browsing content feeds into an addictive cycle.
  For this feature, the toolbox proposes the use of **Exit Points.**
- **Hiding a User's Interaction Time** - Platforms fail to be transparent with a user's browsing data, and fail to work with users to help them manage their own time spent browsing different types of content.
  For this feature, the toolbox proposes a means of **Time Visualisation and Control.**

[2] Hoare, E., Milton, K., Foster, C. et al. The associations between sedentary behaviour and mental health among adolescents: a systematic review. Int J Behav Nutr Phys Act 13, 108 (2016). https://doi.org/10.1186/s12966-016-0432-4

[3] Teychenne, M., Costigan, S.A. & Parker, K. The association between sedentary behaviour and risk of anxiety: a systematic review. BMC Public Health 15, 513 (2015). https://doi.org/10.1186/s12889-015-1843-x

Notification Postbox

Goal

Notifications are a very common feature in mobile applications. This is a proof of concept demonstrating an alternate method of handling notifications, with the secondary purpose as a tool for end-users to corral notifications from other apps. This tool is implemented for the Android operating system due to the ease of development on the platform and how widespread it is, though the design ideas are broadly applicable to all mobile applications.

Glossary

The Android operating system has several existing mechanisms for users to manage notifications from your application.

1. **Do Not Disturb** mode allows users to block all notifications from making sounds, though they appear in the system UI as normal unless otherwise specified. The user can also allow certain types of notifications to interrupt them (alarms, reminders, events, calls, and messages).

2. **Notification Permissions** for the app can be configured. At the broadest level this sets the maximum interruption an app is permitted to give the user. This can be done at the app level, or, since Android 8.0 (API level 26), for different types of notification.

For more information on notification specifics for Android consult the documentation page[4].

Use Case

Any app that uses notifications can benefit from considering how those notifications are received by users. By allowing users to tune notification settings and behaviours within the app itself you can offer finer control compared to the android system settings and better standardise across platforms. The android system settings are also difficult to find and so you offer better control to users.

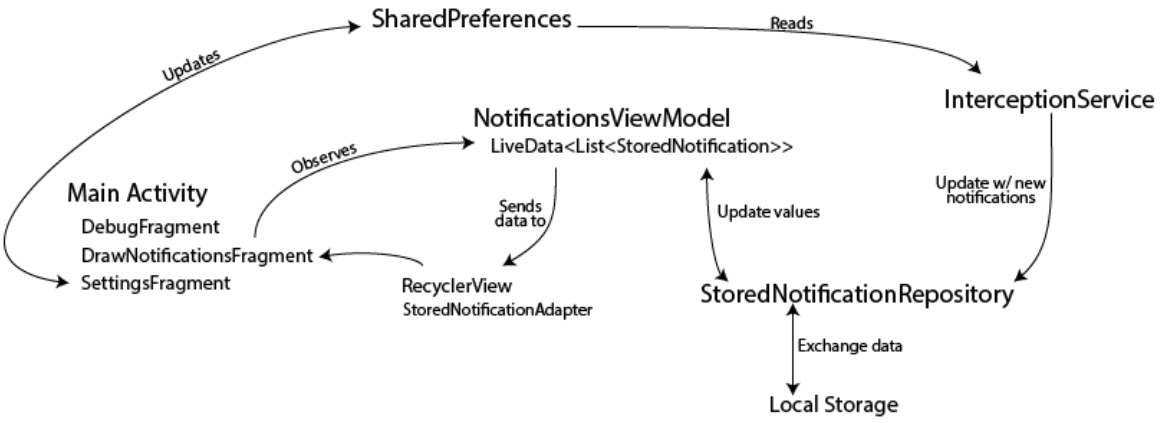
Whilst the exact techniques for this proof of concept may not be applicable to your app the principles of user control are universal and bear consideration.
The actual proof of concept app developed is aimed primarily towards users who are seeking to better control notifications than as a developer example because notification use is so specific to the platform, though some aspects may be helpful for developers.

Technical Overview

The app is implemented using the typical MVVM architecture (Model, View, View Model) which separates the UI, data, and data representation into separate classes. In android this is important for performance reasons and due to the lifecycle of UI elements being unpredictable and short - the operating system frequently recycles and resets them e.g. recreating the UI on phone orientation change. Apps with more complex behaviours may

[4] https://developer.android.com/guide/topics/ui/notifiers/notifications

require a more complex architecture[5] but the components of MVVM are widely used.



The UI is controlled by the MainActivity which hosts Fragments. The three fragments are navigated to by the user using a bottom navigation bar[6]. The debug fragment allows the generation of notifications for testing purposes. The settings fragment uses the AndroidX Preference library[7] to store user settings. The draw notification fragment unsurprisingly renders intercepted notifications.

Actual notifications are modelled using the StoredNotification class that stores a subset of android notification features. When implementing a similar system you will likely require additional fields for more complex actions (e.g. actions or images), but since the system notifications represented are not owned by the app it is not possible for those additional features to be extracted; a problem that you do not have for your own notifications.

Since notification data is stored to be used by the app, it is logical to have a class responsible for maintaining a single source of truth (SSOT)[8], this is the StoredNotificationRepository class. This class uses the local storage allocated to the app as its backend though this could easily be a remote or local database or both. The mechanism by which the repository stores the data is not relevant so long as it fulfils its contract as a SSOT.

So far the UI and data layers have been described, covering the model, view, and repository structure. The glue that links the two is the view model. The view model's lifespan is the same as the app meaning it persists over layout changes that cause UI elements to lose data. It maintains a LiveData object. LiveData objects are observable, that is the UI elements subscribe to the object and when the data changes (e.g. a new notification is added to the repository) a callback specified by each observer is executed. This callback is used to update the UI so that it matches the updated data. Therefore the view model interacts both with the UI (via DisplayNotificationsFragment) and with the repository (needs to ensure its data is concurrent with the SSOT).

The repository interactions are simple. The repository class has a live data that the view model gets a reference to, therefore when the repository processes any data update it

[5] https://developer.android.com/topic/architecture
[6] https://material.io/components/bottom-navigation/android
[7] https://developer.android.com/guide/topics/ui/settings
[8] https://en.wikipedia.org/wiki/Single_source_of_truth

can propagate those changes to the LiveData variables outwards - fulfilling its role as the single source of truth. The UI also uses the view model to propagate changes to the repository, for example when cancelling a notification. The view model simply calls the corresponding repository method. This keeps the modules as decoupled as possible - the UI does not know about the repository, only the view model.

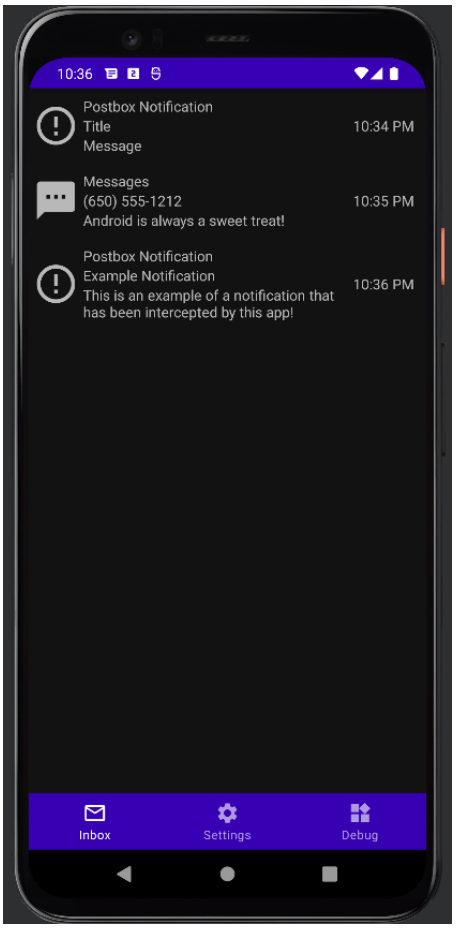
The view models interaction with the UI is a little more complex. The stored notifications are displayed in a recycler view[9], which is a container view for lots of data with each element having a view. The recycler view requires an adapter to convert each StoredNotification into a view for display. So when the LiveData updates, the callback creates a StoredNotificationAdapter. The adapter then, for each stored notification, inflates a layout and uses the notification to fill in the details (app title, notification content, notification time, etc). The callback also sets up swipe behaviour to delete notifications (via the view model) and to undo deletion through a snackbar[10].

The final core element of the app is the NotificationListenerService. This is not necessary for the displaying of notifications, so if you are implementing an alternative notification display system this is not pertinant, but is included for completeness. When a notification is intercepted it checks the preferences the user has specified (in the SettingsFragment) to see if it should first store the notification and then if it should suppress the notification from reaching the system UI. It also listens for notification cancellations as the user may also specify that if a notification is cancelled on the system UI but is also stored by the app then the app should delete its copy.

[9] https://developer.android.com/guide/topics/ui/layout/recyclerview
[10] https://developer.android.com/training/snackbar/showing

Demonstrations

An example of some intercepted notifications if shown to the right.



The app itself can be built and installed as any android application can, provided the device has at least SDK 27 (Android 8). The app was developed using Android Studio and so the source is best modified using Android Studio, though alternatives will work with slight adjustment.

The specific nature of how you need to present your notifications and the level of setup you require will vary heavily, though this example as well as the linked documentation should be beneficial.

For users of the app a bundle or APK can be built for installation or app store hosting, after which the app will prompt the user for requisite permissions (for the listener service). After that the app is fully functional and installed. Android Studio easily allows this from the provided source code.

Further Considerations

This particular task is more user facing than most of the tools here as its goal is not developer facing; while there may be some utility in considering how notifications should be handled the code developed is not likely to port to domain specific needs particularly well. However for users it potentially provides a useful tool for managing notifications and app interactions where existing measures are lacking or difficult to use.

Notification Spam Filter

Goal
This feature is a trained model that predicts whether a notification is important or unimportant to a user, similar to a spam filter used in an emailing system. The goal of this feature is to be a tool that can be used to stop the notification trigger in the habit cycle of overusing social media. This would be done by reducing the number of push-notifications a user receives, by blocking unimportant notifications.

Glossary
- Joblib File - joblib[11] is used by this tool to store the Python Machine Learning models produced by this tool as joblib files, to allow for the models to persist and be used without retraining the models from scratch each time a prediction is needed.
- Notification Channel - different types of notifications are grouped into channels, for example ‘Tags’ or ‘Comments’. Some users find certain types of channels more important than others. For this reason, the notification channel is used as a feature in this tool.

Use Case
This tool is a backend feature. An example use-case for how this tool could be used is as follows.
When a push-notification is sent out by a social media application, it would be intercepted and its details passed into the predictor model. Based on the notification channel and the names in the notification, the model predicts if it is important or not to the user.
- If it is deemed not important, the system can then block the notification from alerting the user.
- If it is deemed important, the system can allow the notification to be sent.

The model is to be trained and used on a per-user basis because notification importance is subjective to each user. The performance of a generalised model would be difficult to test since it could vary so much from user to user.

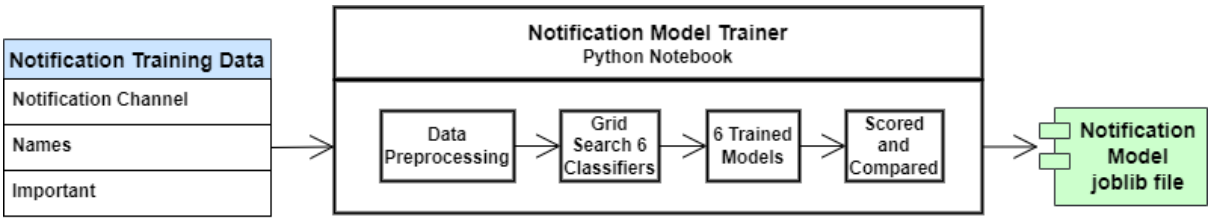
As will be explained later, the model will need to be re-trained regularly to ensure optimal performance.

Technical Overview
The predictor was implemented using Google Colab’s Python notebook. To load the relevant files into the notebooks from Google Drive, a cell is provided to mount a Google Drive directory, giving the notebook access to the files in that account. However, as already mentioned the code can be used in any Python notebook editor, e.g. Jupyter Notebook. The Google Drive mounting cell can simply be deleted in that case.

This predictor tool is split into two parts: training and using.

[11] https://joblib.readthedocs.io/en/latest/

Training the model



Spam Filter Model Builder.ipynb takes existing notification data and outputs the best model to a joblib file (see the image above for a visual breakdown). The notification training data must be a CSV file with a column for the notification channel, a column for the names involved with the notification (e.g. tagged in a post and by who), and a column for whether the user found this important or not. This importance could be measured by whether they opened a notification in the app. The notebook trains six different models and scores them using 'score = 2*Sensitivity + 3*Specificity'; the model with the highest score is outputted as a joblib file.

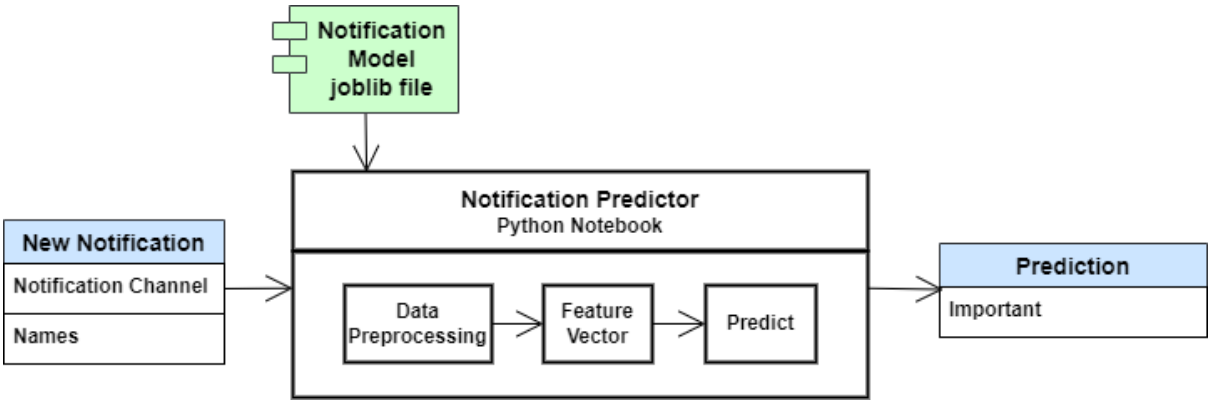
The two metrics, sensitivity and specificity, are unevenly weighted in the score calculation. Our customer felt that it was more important to block unimportant notifications (be specific) than to let important notifications pass through (be sensitive). This is because notifications are inherently not *vitally* important and so, improving mental health and perhaps missing a few interesting post notifications is preferred to sending the user every interesting post as well as some uninteresting ones that will needlessly trigger the user to scroll on social media. The goal of this tool's use is to only send notifications when it is important (i.e. only trigger the user when it is something they will likely find interesting).

Since each user would have a different model trained on their own data, it is impractical to manually check which model performs best for their data. For this reason, the notebook trains six classifiers (each with a grid search performed to find the best performing model parameters), and compares them using the score function described above. The six classifiers used in the notebook are:

1. sklearn.ensemble.HistGradientBoostingClassifier()
2. sklearn.neighbors.KNeighborsClassifier()
3. sklearn.tree.DecisionTreeClassifier()
4. sklearn.svm.SVC()
5. sklearn.naive_bayes.BernoulliNB()
6. sklearn.neural_network.MLPClassifier()

The best scoring model is outputted to a joblib file.

Using the model



Notification Predictor.ipynb loads in the joblib file's model and provides an interface to take a new notification and then outputs whether it is likely to be important or not (see the diagram above for a visual breakdown of how the model is used).

Guide

Training the model

What you need:

- *Spam Filter Model Builder.ipynb*
- CSV file with at least the three following columns, named exactly as follows:
    - 'Notification Channel' - taken straight from the notification.
    - 'Names' - taken from the notification separated by periods e.g. 'Forename Surname. Forename2 Surname2'. These names are who was involved in the notification e.g. *who* tagged you in a post.
    - 'Important' - 'T' if the notification is important, 'F' if the notification is not important.
    - An example CSV file, *Example Notifications.csv,* is included in the Example Inputs and Outputs directory. For ease, an additional example for the expected layout is shown in the image below.

1	Names	Notification Channel	Important
2	Afore Asur. Bfore Bsur	Comments	F
3	Bfore Bsur	Updates from friends	F
4		Reminders	T
5	Cfore Csur. Dfore Dsur. other people	More activity about you	T
6	Efore Esur. Cfore Csur	More activity about you	F
7	Cfore Csur	Tags	T
8	Ffore Fsur	Events	T

How to train the model:

1. If using Google Colab, run the first code cell. If not, ignore/delete it.
2. To load the training notifications CSV file, change the path in the second code cell to your training CSV file path.
3. Run the rest of the cells and the notebook will output a joblib file containing the best trained model.

Using the model

What you need:

- *Notification Predictor.ipynb*
- A joblib file containing the trained model outputted by *Spam Filter Model Builder.ipynb.* An example joblib file, *Spam Filter Model.joblib,* is included in the Example Inputs and Outputs directory.
- Exactly one notification to predict upon. The following fields can be loaded in as a CSV file or inputted manually:
    - ‘Notification Channel’ - can be taken straight from the notification.
    - ‘Names’ - taken from the notification and separated by periods: ‘Forename Surname. Forename2 Surname2’. ‘nan’ can be used to mean no names.

How to use:

1. If using Google Colab, run the first code cell. If not, delete it.
2. To load the trained model, change the path in the second code cell to your model’s joblib file path.
3. Manually change the notification data (period separated names and a channel name).
   Or
   Uncomment the relevant cell and change the path to your notification’s CSV file path.
4. Run the rest of the cells. The notebook will output a CSV file containing the notification prediction.

Maintaining the Model

If the new notification uses a channel that the model has not been trained on, or names which were not trained with, then those features are not added to the feature vector for prediction in *Notification Predictor.ipynb*. This is because the new feature vector for the prediction must match the format of the training feature vectors. The format of the training feature vectors is with the names and channel names being binary features of whether or not they are present. Visually, the training data is reformatted from

Row	Notification Channel	Names
1	Comments	Bob Bobbington. Charles Charlington
2	Tags	Emma Emmington. Bob Bobbington
3	Tags	Charles Charlington

to

Row	Bob Bobbington	Charles Charlington	Emma Emmington	Comments	Tags
1	1	1	0	1	0
2	1	0	1	0	1
3	0	1	0	0	1

with each category being a binary column. This was necessary due to the original columns being categoric.

As a result of this formatting, the model cannot make predictions with unseen categories (e.g. ‘Fred Fredington’ in this case), since ‘Fred Fredington’ would not be a column in the training feature vectors. The predictor simply ignores the presence of the new category(s), and makes a prediction based on the presence of the features from the training stage.

This limitation of the predictor is why the model should be updated regularly. For example, each month, store the data of whether a user opened a notification in-app or not (not opening it being synonymous with it not being important to the user). At the end of the month, re-train the model on the new data, and make predictions for incoming notifications based on the new model. This will help to ensure the model has an acceptable performance.

Further Considerations

The notification filtering feature developed in this project is a backend tool for predicting notification importance. The natural next steps are to link this to a mobile application that can block the android notifications predicted as unimportant. Some resources for this next step are provided in this section.

Notification Postbox

The notification postbox in this guide shows functionality for capturing and blocking android notifications.

Deploying Machine Learning Models In Android Apps Using Python
https://analyticsindiamag.com/deploying-machine-learning-models-in-android-apps-using-python/

Deploy a Python Machine Learning Model on your iPhone
https://towardsai.net/p/machine-learning/deploy-a-python-machine-learning-model-on-your-iphone

Exit Points

Goal

It is often hard for some users of social media to stop using a social media platform well after they had intended to use it. This can lead to an unhealthy relationship with social media, with the worse case being social media addiction. This unhealthy relationship is facilitated by the design of the social media platform. A constant stream of new content is provided to the user, and the user is constantly searching through this content to find something they enjoy. This creates a situation where a user continues to scroll through content for a much longer time period then they intended to, while gaining no additional satisfaction.

By breaking this stream of content up by using exit points we can disrupt the stream of content and give the user a clear opportunity to assess their use of the platform and if they have engaged with it as much as they wanted. If they have, they can break the content-seeking behaviour and then spend their time on other things. This helps encourage a healthier relationship with social media, one where the platform respects the users' time and does not try to place them in a skinner box to maximise time on the platform.

Use Case

This tool requires being able to keep track of the number of posts a user interacts with or goes past. It is best suited for social media platforms that otherwise would provide a constant feed of recommended posts that the user can stop scrolling and look at or ignore and continue scrolling to more content. It is intended to break up the feedback loop often present in social media streams to encourage users to not endlessly scroll.

It is most appropriate where a virtual skinner box is likely to exist, such as on platforms with short pieces of content that the user scrolls one by one through. This can easily become a skinner box situation for a user, as the user is unsure when the next video they really enjoy will come up in the feed so feel the need to keep scrolling for far longer than they originally planned to engage with the social media platform.

Technical Overview

What an exit point looks like will depend on the platform and design decisions made to best fit within the social media platform this feature is developed for.

Some example exit points are having a pop up asking the user if they want to keep scrolling, having a special buffer content which appears in the a users content feed, a user manually clicking a button to load more content when they have been recommended a set amount of content. There are many other possible ways to implement an exit point but the key part of an exit point is creating a situation where when an exit point occurs in a user's experience they can easily stop using the social media platform.

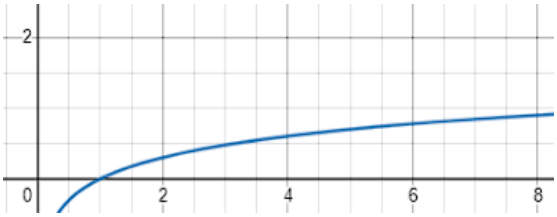
An important consideration with exit points is the frequency that they occur within a user's experience. Having exit points too often can drive users away as if they want to engage with the content on the platform then they have to get through an excessive amount of exit points to get to what they want to get to. Having too little exit points would mean that the users would have large amounts of time wasted, which exit points aim to solve.

To avoid these situations the right frequency of exit points is required. The simplest way is to decide an appropriate constant frequency so that after every set number of posts a user sees, there will be an exit point. The main problem with this implementation would be that a user will likely be more frustrated by exit points at the beginning of their usage of a platform

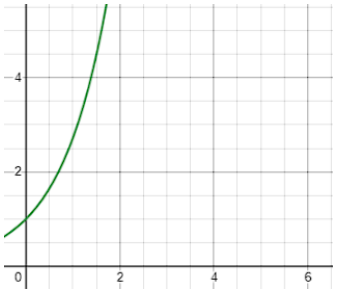
than later on, so tuning the frequency so that the user is not annoyed by exit points at the start of their usage would likely make the exit points too infrequent by the end of their usage. By making the frequency of exit points a function of the number of posts a user has seen in their current usage session, or the amount of time they have spent on the platform, exit points can be made to be more frequent the longer the user spends on social media.
A function which non linearly increases over time is preferred, ideally one which is bound. This is because with a linear function there will likely be a case when the frequency of exit points based on the function results in all the content becoming exit points instead.

Solutions/Demonstrations/Guide
Good functions to calculate the frequency of exit points would be :

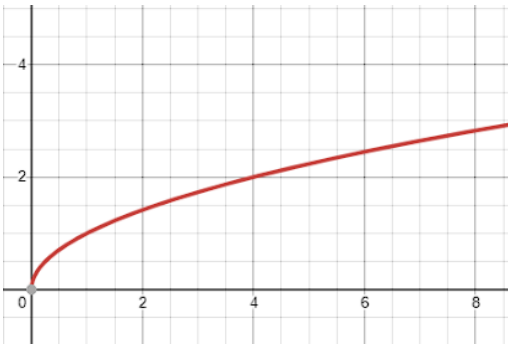
$freq = \alpha log(x) + \beta$



$freq = \alpha e^{x} + \beta$

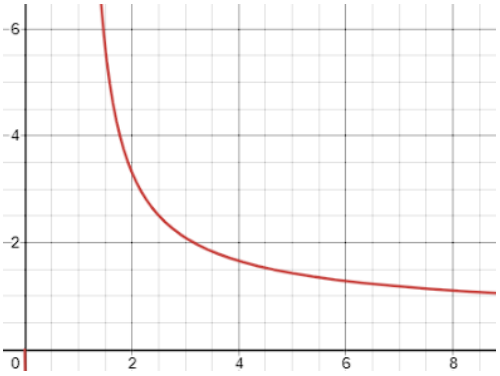


$freq = \alpha x^{1/2} + \beta$

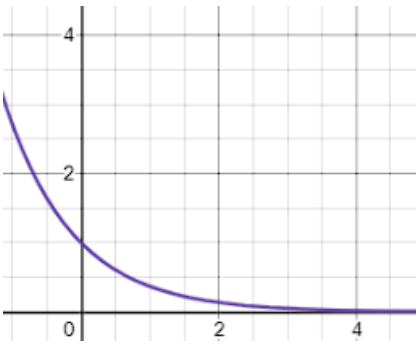


Good functions to calculate the period between exit points would be :

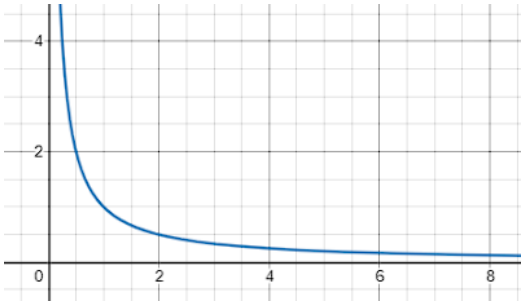
$period = \frac{1}{\alpha log(b)} + \beta$



$period = \frac{1}{e^{\alpha x}} + \beta$



$$period = \alpha\frac{1}{x} + \beta$$



(Images from desmos.com/calculator)

With the parameters α and β being tuneable depending on use case to get the function to behave appropriately for the social media platform it is being implemented for.
Additionally it is recommended to have a maximum frequency of exit point parameters so the final frequency of exit points would be given by $freq = max(f(x), max\ freq)$ with $f(x)$ being whatever function of time or number of posts seen is implemented.
These functions can then be used to determine how many exit points should appear over a set of 100 posts, the number of posts between exit points, the amount of time that should pass between exit points, or any other appropriate usage.
Each function changes at a different rate over time, so which one is the best to use will be platform dependent.
A python notebook containing these functions can be found in the GitHub repository, within the exit point tool folder.

Further Considerations
The type of exit point created is key to the success of the exit point but is very dependent on how content appears to a user within the social media platform. How easy the exit point is to ignore for a user will directly impact the effectiveness of the exit point. With less effective exit points it will likely be required to have a higher occurrence of exit points to compensate for how easily they can be skipped over absentmindedly.
For the exit point to work as intended it is required for a user to have some form of interaction with it to make them think that this would be a good place to end their session on the social media platform.

Time Visualisation and Control

Goal

The aim for this tool is to reduce a user's time spent browsing social media by being transparent with their time spent interacting with specific tagged content, and giving the option to limit their exposure to specific tagged content.

This will be achieved through identifying a simple technical pipeline for this process, and partially implementing the sections of this pipeline relevant to the project's scope.

Ultimately, this tool will reduce the amount of time spent on a platform but increase the quality of time spent, by allowing the user more control over the content they interact with. In turn, this will reduce the likelihood of a user developing a sedentary lifestyle dependent on social media, and in turn improving a user's mental health.

Glossary

- ***Content*** - Any media on a platform that a user can interact with, such as videos and images.
- ***Content Tags*** - A short category descriptor of a piece of content, describing what the content contains. A single piece of content may have multiple tags.

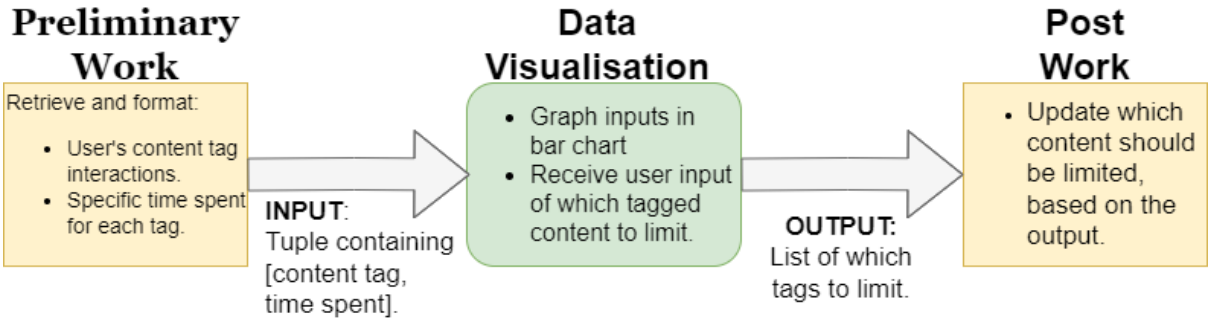
Use Case

This tool requires a platform where users interact with and browse tagged content of any kind, and a platform where users may wish to be aware of, and control, their time spent being exposed to types of tagged content.

The tool can be utilised by a platform wanting to respect their user's access to controlling their time spent interacting with their platform's content, and specifically to a platform wanting to shift a focus from increasing a user's quantity of interactions to quality of interactions.

Technical Overview

A simplified breakdown of what this tool aims to achieve on a technical level is presented as a data flow diagram:



This tool's solution assumes access to the following user data:

- The tags of the content the user interacts with.
- The time spent (in seconds) interacting with this content.

Content can be tagged in different ways. If a platform's content is uploaded by the user base, users may be able to tag their uploaded content manually. Alternatively, the platform itself may provide tagging for content manually, or automatically through feature detection algorithms and classification learning.

Using this data, the tool will communicate to the user their time spent interacting with each type of tagged content (not displaying content they do not interact with), and additionally giving the user the option to limit interactions with certain tagged data. This tool aims to give users more control in their browsing experience, by being transparent and helpful with communicating their browsing habits.

From the tool's output of being able to limit tagged content access, it is also assumed that the platform will be able to prevent a user's access to certain content. This implementation is generalised as an idea in this tool, and discussed further in the pseudocode section.

In summary, this tool will provide a technical implementation for handling time and content tag data, and outputting the visualisation of this data and a user's choice of which content tag to limit. The tool will only discuss pseudo solutions for gathering inputs and reacting to outputs, as on a technical level these implementations are beyond the project's scope.

Solutions/Demonstrations/Guide
To demonstrate this tool's potential implementation, different areas of the proposed solution are demonstrated and explained within the Github, in the Time Visualisation folder

Further Considerations
There are a few considerations for extending this tool's utility.

A major extension would be to decide the time scale of saving the time recordings, which would be specific to a platform's context. For example, an implementation of this tool could reset the recorded times per week or month, to be up to date with the user's interests. For example, would a platform want to maintain the viewing data from years prior to the visualisation, as this may not be relevant to the user anymore?

A further consideration is the idea of giving a user even more power in their viewing habits, specifically allowing them to not only choose which content to reduce their exposure to, but allow them to choose which content they wish to be exposed *more* to. This further gives transparency and honesty to the user, by allowing them to be in absolute control of the content they are recommended. This idea may infringe upon many fundamental design features within an existing social media platform, however, it is a consideration if a platform's context could benefit from this additional transparency.

Providing Effective Moderation

Moderation is a fundamental feature within any platform which encourages user-based content. The concept of moderation is the act of ensuring that user-based content is user friendly to a specific degree, and having an effective way to perform this moderation is crucial in ensuring that users of a social media platform are able to interact with each other in a healthy way.

Poor moderation of a platform can allow for toxic users to directly harass, bully and hate other users sharing the platform, and in some cases a total lack of moderation can allow for extreme platforms to exist which provide a safe haven for sharing harmful, extreme ideologies such as anti-LGBT and anti-semitic views. It can be imagined how individuals being exposed to this kind of extreme content can cause damage to their mental health, especially in the case of targeted bullying. Hence, by considering effective moderation this can help defend and protect users from being exposed to this type of harmful user-based content.

Moderation can be performed manually by humans, which is ideal as humans are generally reliable at labelling text as offensive, however, the human resources required to perform human moderation can be slow and is difficult to scale effectively with a large user base. Hence, an automatic moderation system may solve this issue which requires minimal human validation, which is the basis of this tool.

Model Construction

Goal

Human moderation is currently the gold standard for moderation, but this is often not a scalable solution for social media platforms due to the amount of new content that is being created constantly. As a result there must be an effective way to ensure that this new content is moderated in a scalable way and have posts blocked from being posted or flagged for manual investigation. Natural language processing techniques can be utilised to create a machine learning model that is able to filter out content that is deemed unacceptable for the platform ensuring that the general user of the social media platform is able to avoid seeing content that would be disturbing or unpleasant.

This tool delivers an adaptable implementation of constructing and visualising a generic moderation model, where the model can be adapted for any platform context of labelled comments, basically where the input dataset can be changed for any purpose.

Dataset

The dataset used to create the moderation model was made by the University of Berkeley and is freely available on Hugging Faces[12]. The dataset contains a large corpus of text posts which have been given a hate speech score based on factors such as violence, disrespect and dehumanisation within the post, scored by over 7000 annotators. The dataset contains 39565 posts labelled by 7912 annotators.

[12] Kennedy, Chris J and Bacon, Geoff and Sahn, Alexander and von Vacano, Claudia. *Constructing interval variables via faceted Rasch measurement and multitask deep learning: a hate speech application.* arXiv preprint arXiv:2009.10277. 2020.

This dataset is sufficiently large to produce a good moderation model. It is also a dataset created by a large number of people, this should reduce the chances of any model being trained on this model to inherent biases from creators of the dataset due to the large number of annotators used to build the dataset. This is crucial to ensure the moderation model itself does not allow hate speech against certain groups of people, due to an annotators' biases against that group.
This dataset is easy to use, being easy to download and use straight away with little overhead required, additionally the conversion from a hate speech score to a categorical label can easily be modified to change the sensitivity of the model or add intermediate labels such as a needs human attention label.
Other datasets could be used, and this could improve the performance of the model on a specific platform if the dataset is made up of posts from that platform. However, it is highly recommended to have a diverse team of annotators to try and eliminate any biases being inherited into the dataset.

Modelling

The moderation model is built by using the fasttext library[13] to learn word embeddings specific to the dataset it is given and then use these embeddings to create a model which tries to classify any given sentence/post it is given into an acceptable and an unacceptable category.
The fasttext model once trained can classify posts quickly and can be made to be easily stored on mobile devices, this allows this model to be used fairly robustly.
Fasttext was seen to perform better than alternative natural language processing methods to classify if a post would be seen as offensive or not using the same dataset. Additionally it is able to create word embedding for words outside its training vocabulary such as misspelt words.
Fasttext can be used to only create word embeddings, these embeddings can then be used within a custom model such as a neural network. Fasttext is able to create text representation that performs better than a typical word2vec representation, and performs better than other common text representation methods.
To use any dataset with the fasttext library the posts passed must in the form of a .txt file, with a new post on each line and the label appearing for each post within the same line with '__label__' appended to the front of the label to indicate to the library that that is a label. This may result in some need to edit and reformat a dataset to make it work with fasttext.

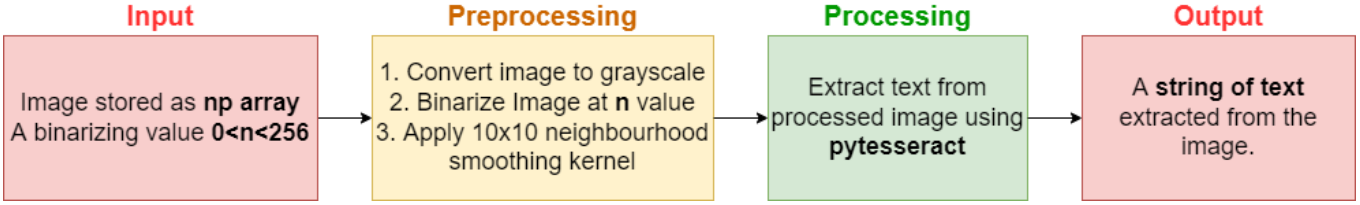
Image to Text

User based image sharing is a common feature of modern social media platforms, and hence is subject to the benefits of moderation. The text-based moderation model can be extended to accommodate for image media where text is present somewhere on the image, as Python libraries exist which can deduce text from a given image. Specifically, the *pytesseract* library[14] which is able to process text within a given image, which would be useful in this case for taking the text from an image and predicting its acceptability label to successfully predict if an image is appropriate or not.

---

[13] Joulin, Armand and Grave, Edouard and Bojanowski, Piotr and Mikolov, Tomas. *Bag of Tricks for Efficient Text Classification.* arXiv preprint arXiv:1607.01759. 2016.
[14] https://pypi.org/project/pytesseract/

The overview of this process is visualised below:



The *pytesseract* library's ability to deduce text is not immediately reliable, as some colours and backgrounds may make it difficult to immediately discern text. Hence, a stage of preprocessing the image using the *Pillow* library[15] must take place to filter out the 'non text elements' of an image, to make text easier to discern. On a technical level, this preprocessing includes:

1. Converting the image to grayscale, as colour information is redundant when trying to find text.
2. Binarizing the image, as the colour of text is likely to be the same or similar colour, so this step separates the text better.
3. Applying a mean smoothing kernel filter, which essentially just smoothens the edges of shapes, and makes text more 'text' like after binarization and erases unwanted noise.

These preprocessing steps, as well as the entire process from an image input to a text output is demonstrated with an example that can be found on the Github Repository within the Moderation tool folder

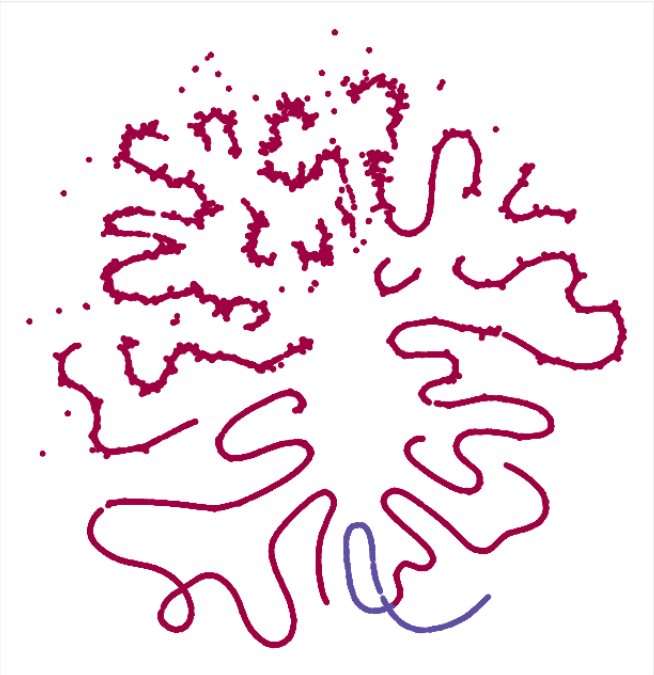
This image to text function is limited to images that may have sensitive text on them that requires moderating, and hence would be redundant against images that display graphic imagery. To solve this, an image moderation model could be produced similar to the text moderation model, through training on graphic and non-graphic labelled images.

Regarding portability, the *tesseract-ocr* engine[16] is available for usage on a variety of languages and platforms, and is hence not limited to this Python demonstration.

Visualisations

A visualisation of the words in the dataset has been made to give an understanding of how the model performs. It reduces word vectors with 300 dimensions down to a two dimensional space by making use of a technique called Uniform Manifold Approximation & Projection (UMAP[17]).

[15] https://pillow.readthedocs.io/en/stable/
[16] https://github.com/tesseract-ocr/
[17] McInnes, L., Healy, J., Saul, N. and Großberger, L., 2018. UMAP: Uniform Manifold Approximation and Projection. *Journal of Open Source Software*, 3(29), p.861.



In this image every red point is an acceptable word in the dataset and every blue word is unacceptable, based on the training of the model. You can see that all the blue words are all adjacent to one another showing that they have a similar vector embedding. Posts which contain these words are likely to be flagged as unacceptable while those made of only red words are acceptable. You can see in the strand of points containing the blue words it is connected to red words, the boundary between these is made up of words which could be unacceptable or acceptable depending on the context they are used within, so could be deemed unacceptable within the context of a post but by themselves are allowed.

The process of producing these interactable visualisations is demonstrated in the Github within the Moderation Tool folder

Further Considerations

There are many further extensions to consider from this moderation tool.

Firstly, there could be considerations into the order of words from the dataset as currently the tool uses the ‘bag of words’ approach, and does not consider their order. Training on the order of words would produce a much more complex model, though many more important features could be extracted from this to produce a more accurate model.

When applying this moderation model within a context, the repercussions of detecting that a user has sent an unacceptable message must be decided. For example, perhaps the message is deleted and the user that sent the message is warned or even educated as to the negative language they have used. These outcomes have not been technically considered within this tool as it is sensitive to the context the model is used within, however it would need to be considered when implementing this tool to render its usage effective.

The Python code can be edited to construct a model from any labelled data as long as it is in the correct format, and so if a different dataset is available it may be beneficial for both identifying platform specific negative language, and if the annotating is done to the standards

of the platform's tolerance, then the model can be more relevant to the goals of the platform, rather than a very generic hate speech level.

The image to text feature could be extended as its own personal project, as this tool was limited to just text based interactions. Potentially, a convolutional neural network model could be trained with *image* data to produce a similar model to this tool but for rendering image data unacceptable rather than text on an image.

Appendix B

Project Meeting Minutes

Monday 18th October 1pm - team meeting in person

- Chat about work that's been done so far
- Talk about project management methodology
    - Establish team roles
- Confirm-ish Project Scope
- Plan what to do next

Wednesday 20th October - meeting with Mike online ✓

- Inform Mike of what the project is/scope ✓
- Inform Mike of project management stuff maybe ✓
- Ask how many words is the specification? ✓
- Ask about what exactly should go in the spec. ✓
- Biweekly meetings ✓
- How specific do we have to be when describing our main features ✓
- LOG ON TABULA!!!!!, Each Individually

Meeting notes(20th):

It's fine that we can't define deliverables right now, for the specification
Spec is up to us mainly, needs to be able to inform brand new person about our project, including project management stuff. Just follow the 3rd year guidelines
Good document for the whole team to get on the same page

Risk management is important - plan, schedules, unavailability management.

Research beyond literature perhaps, using twitter data etc
Structure our research, very powerful.
Look at these sources - interrogate them by doing this and that, exclude certain sources, when we identify the relevant sources, we process the data in the following way. Such as content analysis,. Might struggle with a number of sources, need to cut it down appropriately.

Wed and thurs afternoon - evenings in person plan this over email.
Make sure api is not too ambitious, make sure it's focused on what we want to achieve - needs to be concrete. The API is v barebones no worries, Also though we may just not need it, depends on how independent we want each feature to be.

Data is a fair concern and we should definitely dedicate a sustainable time for that tbh. Ethics are gonna be a thing to consider, what can we use the data for, is it anonymous, GDPR vibes, what are the T&Cs, and so on. Type of data we can do the decision chart, for Text only, Text and Image, Image only, Video. Finally data source, So Kaggle or some Twitter API kinda thing.

Notes:
Main feature problems:

Managing screen time (Big One, maybe multiple features for this)
Content Moderation, helps people's mental health by reducing their exposure to harmful, offensive material

Difference in moderation, some methods employ amicability, some employ more democratic environments

Explore these levels of moderation

Hate scrolling
Identify bubbles of groups, semantic analysis on hate filled groups, then 'pop' these bubbles

Working on test data - limited data, different types of data eg twitter, facebook, tagged data
Kaggle or twitter for data sets

For decision of which social media to base this on, Ease of access of data should be a metric

Specification Outline:
Introduction to Social media, link to mental health problems, how an investigation into features can be helpful to solving this problem
An API skeleton where we will investigate, implement and demonstrate features that can reduce mental health problems caused by social media
Describe the parts we aim to have complete by the presentation
Schedule
Requirements definitely, and close ideas of deliverables.

Take a server from uni for database - Daisy in charge
Github setup

Meeting agenda, Team only, 25/10/2021
Legal social and ethics of project
Separating the long intro into an intro and a background research section
Stakeholders
Scheduling?
Resource section
Touch on Risk
Decide on team roles
Misc
Decide if we're all happy with the PM approach

- When we submit the draft (today) - email Mike and ask who should submit it (all of us?) -- emailed Sara about this already (M) Also tell Mike we've not done formatting to references yet :) How do we cite a module?
- Risk - have a few sources for datasets to mitigate risks
- Resources: need to split data into training and testing
- Schedule looks good - epic
- Legal social ethics needs work - gdpr, dotweets need the twitter handle with em (i dont think you can anonymise tweets).

- Stakeholders - in the resources section as people of interest
    - Social media - low interest
    - Mike Joy - How much input do we want from him - Unbiased viewer of our work.
    - User
    - Rob Procter
- Resources
    - Mention what we can use - not what we will use
    - E.g. experienced with python
    - Gives us **limitations** - we know what we are able to do
    - Describe the tools :)
- Risks:
    - Our devices may break (and dcs not open 0.0)
- Roles:
    - PM - Daisy
    - Rotate Scrum masters for each sprint - Keeps everyone aware and motivated
    - Code monkeys basically
- Conclude with where we are at now - and how we will measure success?
- Project title
- Microsoft academic is good for literature review source
- Contents page to be generated

Meeting in person - no Patrick

- Talk about what we know needs doing in this sprint.

Sprint backlog made (see Sprint 1 Initial Backlog document for the initial backlog and meeting agenda).
We didn't decide on a Scrum Master.

Meeting with Mike on Team 5/11/21

- Talk through our sprint plan with the kanban
- Decide on a scrum master :)
- Presentation Questions
    - Who's gonna be there? Second marker
    - In person? Presume online
    - When do we find out when?
    - Dont expect lots of software - they want proof that we're moving forward
    - Marker should be able to walk away going "yes i get what's going now"
    - Backup why each person is doing each task - e.g someone is good at machine learning so took a lead on the ML stuff
    - Circulating scrum master means that we all get to experience it and learn
    - Being reflective of what you've learnt is good

Meeting on Team 08/11/2021 - Matthew joins midway through call

- Don't do full write-up of motivation etc - as it may waste time rn
- Aim for this week - have motivation and existing solutions done for friday :)
- Think about objectives, and discuss on friday
- Read both documents before next meeting

PM notes: *Meeting was structured well. We spoke about what has been done, and let this start discussion on what needs to be done. We identified a bottleneck in the kanban board - we can't write objectives without motivation and existing solutions having been researched. Patrick stepped up as Scrum Master, and set a deadline for the current portion of work to be done, so that we can move past this bottleneck. Two people were assigned to each task. Daisy and Patrick on Motivation, Lewis and Matthew on Existing Solutions.*

Meeting on team 15/11/2021 - No Daisy
Talk on getting the WBS done for being able to start implementation
Existing solutions have some solid stuff that we should be able add
Going through existing solutions, android and apple have a few things we could make use of

- Postbox system
- Multiple User profiles for different engagement levels (Concentration), Soft cut off
- Allow user to pause recommendations of certain types of content at certain types, soft cut off method

Habit research

- Habits have a trigger
- Causes a behaviour
- Leads to a reward

Breaking any of these could lead to breaking the habit, the reward we don't want to fiddle with too much.
Triggers are more easily worked with, can disable non-essential notifications, storing in a postbox
Some incentive to not check postbox constantly
Change behaviour by staggering usage, then staggers the reward by limiting the amount of time they can use SM in one go without break
Type of solution
Gives voluntary tools to the user, effective when used, useless if the user doesn't use them or doesn't know about them. (Maybe more made for parents)
No choice solutions, everyone is impacted so gets the benefits of the management, but users who don't have a problem could be annoyed by being restricted by these.
What is better?

Video game solutions less concrete than expected
Humane design vs addictive design
Video game context, exit points and session length. Users stop playing when objectives are met or death as common exit points. How long does a designer want you to play,
Can apply to social media, gamification of social media
Perpetual games like MMOs very similar
Design exit points that are usable and work well with the idea of session length, stop a user scrolling infinitely, ties into Humane design principles.
See https://humanebydesign.com for concrete design templates

Creation of the objectives document to put some stuff down for that
Completion of said document with about 5 deliverables based on the object of reducing screen time
Lewis has picked up the WBS to make

Meeting Agenda week 8:

- Need clear objectives as I feel we are sometimes on different pages. E.g. the motivation document went into desktop vs mobile SM usage, but the deliverables don't touch that at all. If we don't wanna focus any efforts on desktop usage, then we need to make it clear in some objectives or somat.
- Need to divvy up tasks for/from the backlog
    - Matt - postbox
    - Daisy - spam filtering
    - Lewis - transparency
    - Patrick - toy social media

Meeting with Mike Week 8 Agenda:

- We're working on deliverables - explain them to him
- Explain what we plan to have ready for the presentation
- Ask him about his concerns

We need a customer…

Academic or Postgrad within Uni or something

Someone in Rob Procters Team, not Rob Himself => Postdoc?

Presentation: - show mockups of what we plan

- Show off our design and progress
- Show how well we manage as a team
- The process is the most important thing
- Need to give confidence that we're moving ahead well

- How sprints have helped us improve, learn what we're doing wrong
- Some kinda literature linked with our project management style and its suitability w/ our group
- Between sprints, before planning the next and in reflection of the past, be clear on what we are realistically expecting to do.
- Don't spend next term just researchin, faster turn around
- Next presentation as a draft

Meeting Tuesday Week 10

- Question: Do we want the classifier to prefer getting more important notifications labelled right, or more unimportant notifications labelled right? E.g. with emails, we prefer to let through occasional spam, than miss important emails. Is this the case here or…? With customer, we could let them answer this?

Meeting with Mike week 10

- Is there anything that shouldn't be in this progress presentation?
- What questions do you think we'd be asked?

Mike's comments

- Timing needs work
- Put less content on the slides
- How realistic are our tools - would this acc help or would people just continue using SM in the same way

Presentation notes:

- Background goes on for a while and can be hard to get what the point is. It's a bit wordy and covers stuff we maybe don't need in this presentation? Think we could just state a couple stats
- Addictive design v good
- Noone click the script during the presentation cause you name blocks off some of the text when we're talking
- Scheduling - we say we have freed up time by refining scope, but then say we have had to use the float time.
- Screenshots of the postbox?

20 minutes to get to the sprint talk
30 minutes to get the sprint review

- Sprint review might need speeding up?
- Don't introduce the next slide, just
- Project management - focus on what we're changing, not what scrumban is

13/01/2022

Merge sprint 2 and 3 into one (moderation?), based on the feedback we got. Focus more on more in depth, technical solutions, rather than breadth.

For sprint 1, Lewis and Patricks solution based on dummy social media (exit points and graphing user's time) can be more pseudo solutions - not worth it to employ a social media platform to demonstrate these somewhat simple features. Also, could not identify an open source, available software to use to suit this project, most were complicated, or not designed for testing intentions. Can do PDFs instead with a guide on pseudo implementations (inputs, outputs) for a software engineer, to implement into their own specific contexts.

Solution for exit points: a logarithmic function of time.

Solution for transparency, applying a python(??) script to graph dummy time and interaction data. Don't have to implement the entire collection technique for this data, not worth the scope.

Notification solutions (Matt and Daisy's), can be implemented. Need a data set to be collected over time, then a regression model made.

A pseudo solution PDF could include:

- Inputs and outputs (Inputs that a software engineer would require to implement our feature)
- Pseudo code/ pseudo solutions
- Actual solution examples (python code for graph data, etc)
- Justification as to why these solutions may reduce screen time/ work in general

A pseudo solution PDF serves as a basis to a library, the step before implementing these tools in a unified library.

Need to make clear our project outcomes, that we are not making a library - this is a next step of the project, we are right now:

- identifying which aspects of social media usage negatively affects mental health

- How these aspects could be fundamentally edited, or tools that could be implemented to reduce these aspect's impacts.
- Providing the technical basis of these implementables, in the form of a toolkit. **THE MAJOR CHANGE:** We will now consider which tools to focus on technically, and which to just design pseudocode solutions for, based on which would benefit the most from either way.

MODERATION:
Semantic analysis: toxicity, and negative engagements
Bubbles: broadening horizons whilst maintaining spaces. Target 'cultures' rather than politics. Tie semantic analysis to this to find what these topics are. Need a dataset, twitter maybe?
Context collapse: how do we maintain a context in data? Maybe through semantics. Perhaps identify groups. What can and can't be sent in these groups? Balance hostility, comradery, can this be done? How do cultures talk to each other? Context collapse could be a small pseudo, but AI exists that can handle semantics. There are BAD AI out there that are based on addictive design, but could these be used for humane design?

Can ASSUME we have bubble data for our functions, our semantic analysis. Use dictionaries? Develop in parallel. Can semantic analysis identify these bubbles, and which of these bubbles are sensitive, and not to be touched. Knowing these sensitive ones, can encourage people to join these bubbles in a safer space: but this could encourage a pipe line of bad stuff eg, linking people to terrorism.

FINAL THOUGHTS:
Could apply this project to the metaverse context, which tools could be used in a metaverse space?

18/04/2022
Next meeting on Wednesday

- Have requirements for your section written
- Have a test plan written at least, maybe even some testing done
- ~ Fill out as many sections as you can ~

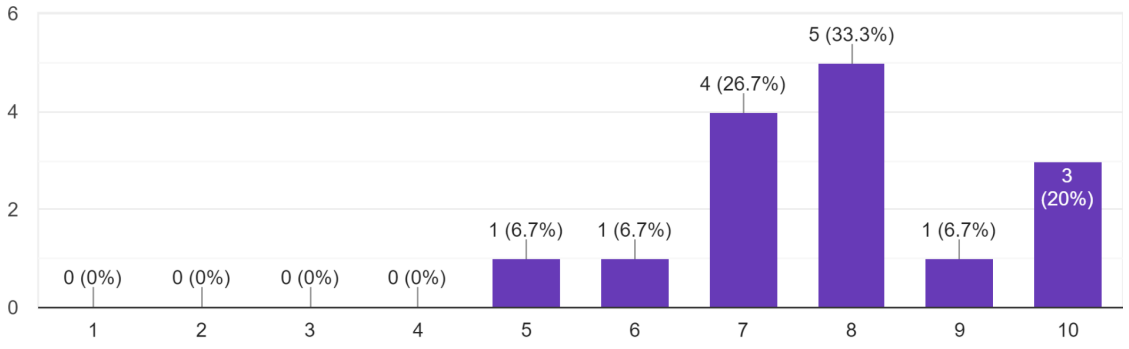
Appendix C

User Acceptance Results per Tool

Notification Postbox

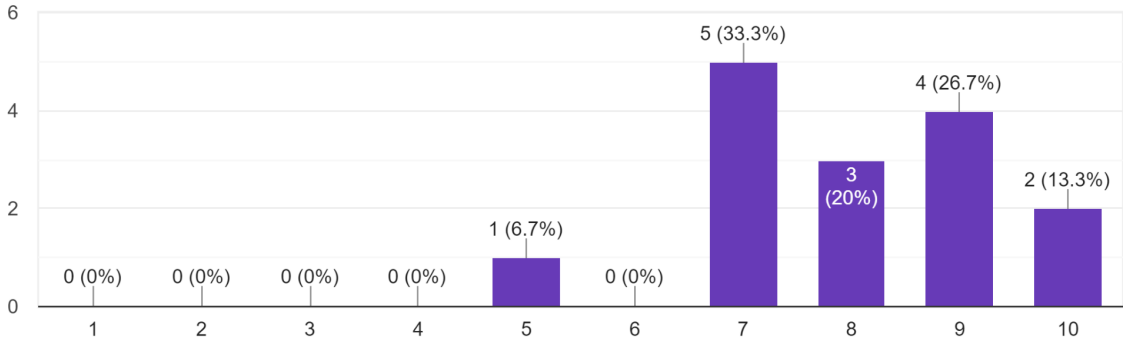
From your understanding, if the tool was implemented, how effective do you feel this tool would be at reducing a user's issues with mental health?

15 responses



From your understanding, if the tool was implemented, how effective do you feel this tool would be at reducing a user's issues with mental health?

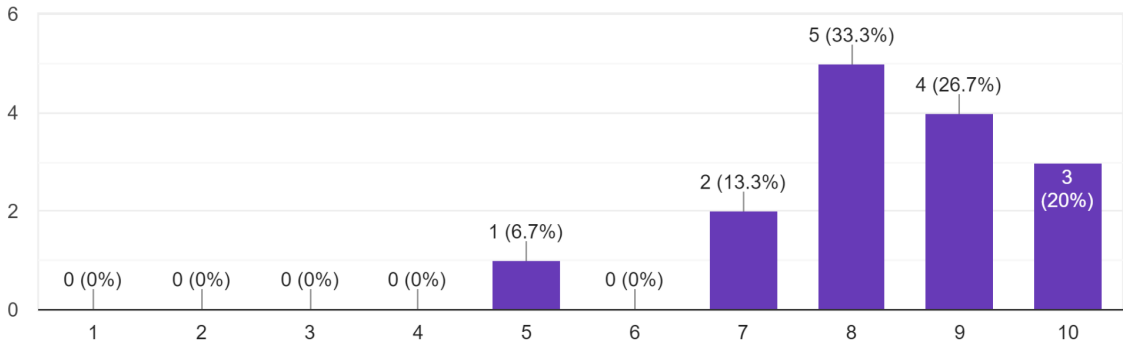
15 responses



Notification Spam Filter

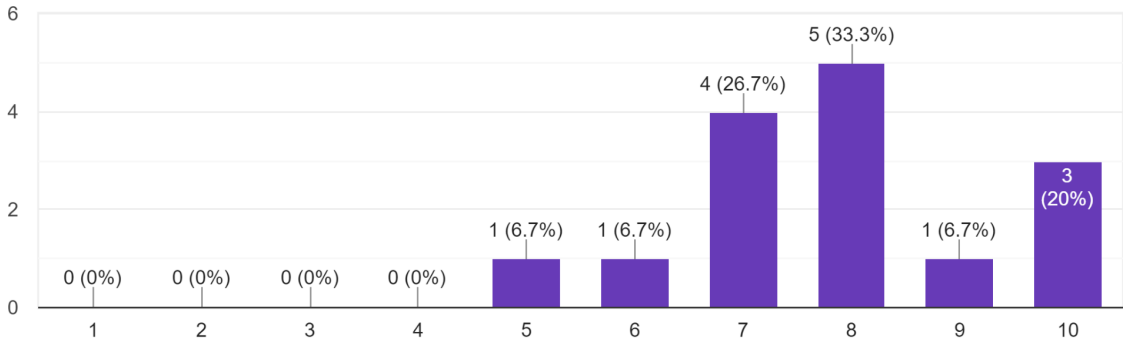
How confident do you feel that you could utilise this tool, using this guide.

15 responses



From your understanding, if the tool was implemented, how effective do you feel this tool would be at reducing a user's issues with mental health?

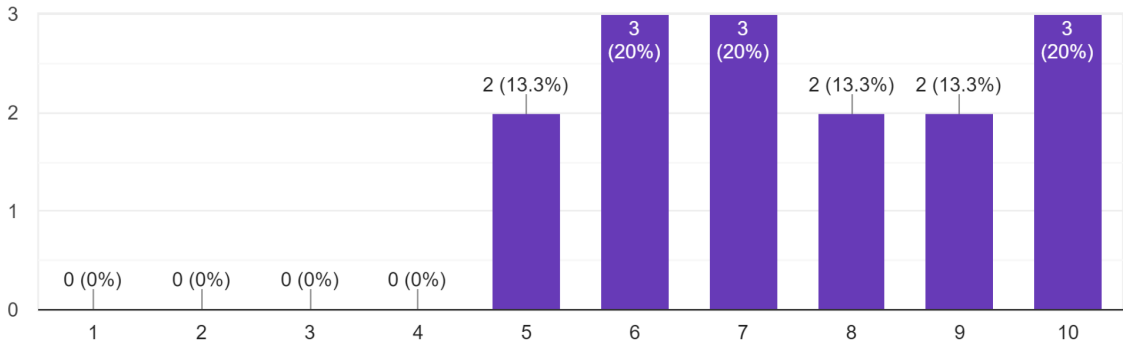
15 responses



Exit Points

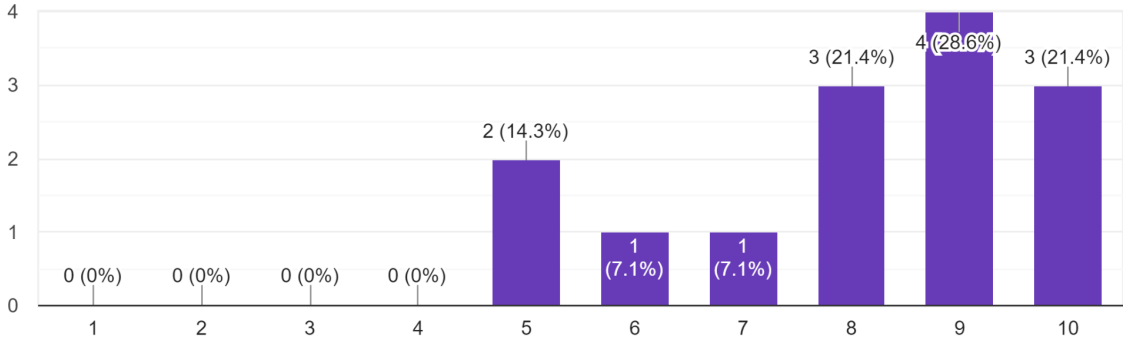
How confident do you feel that you could utilise this tool, using this guide.

15 responses



From your understanding, if the tool was implemented, how effective do you feel this tool would be at reducing a user's issues with mental health?

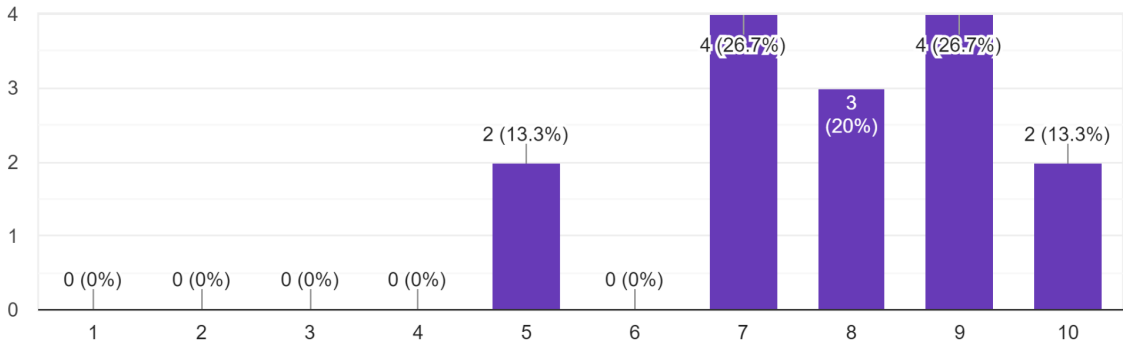
14 responses



Time Transparency

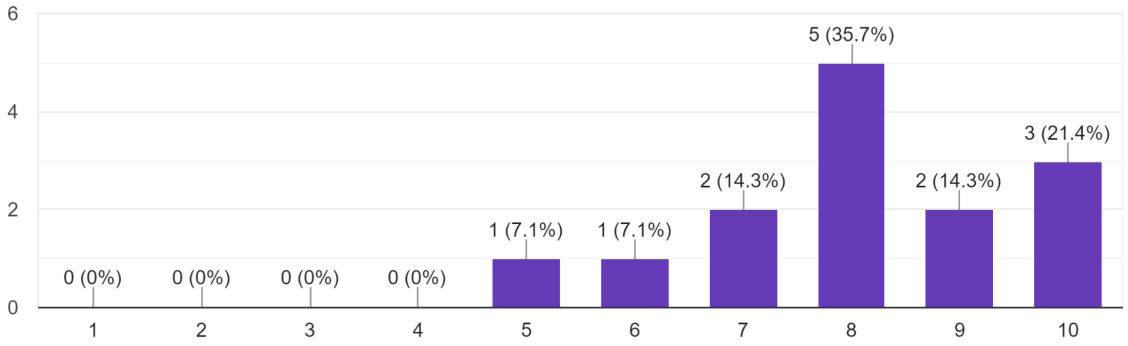
How confident do you feel that you could utilise this tool, using this guide.

15 responses



From your understanding, if the tool was implemented, how effective do you feel this tool would be at reducing a user's issues with mental health?

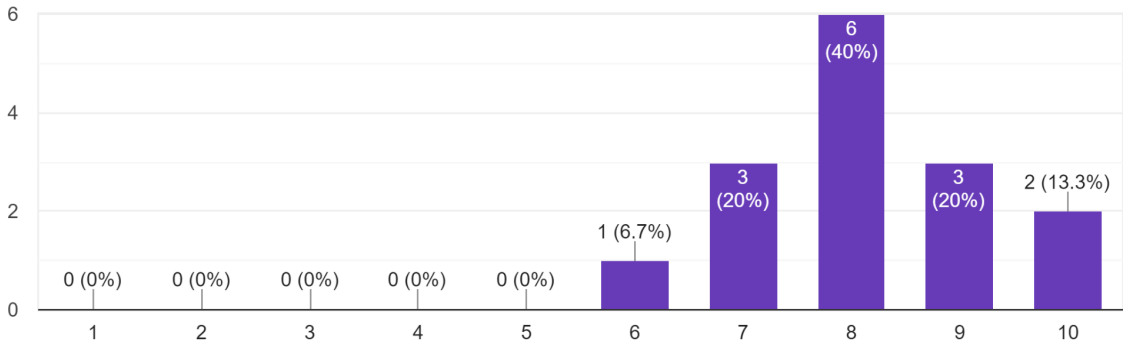
14 responses



Moderation

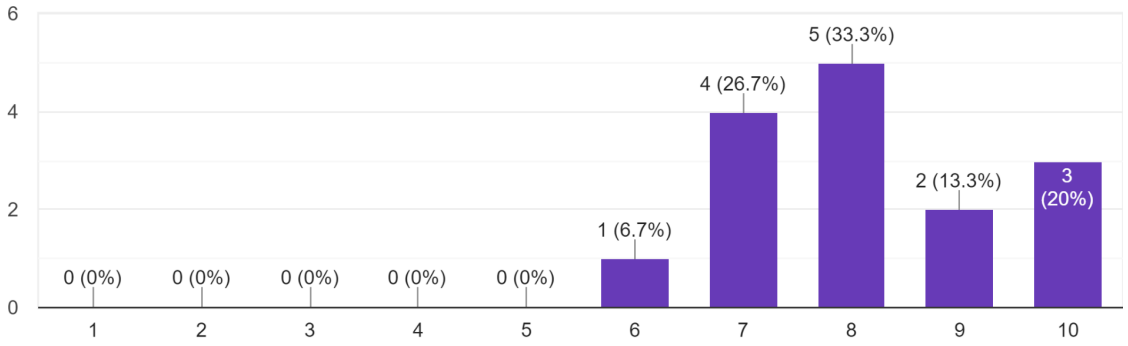
How confident do you feel that you could utilise this tool, using this guide.

15 responses



From your understanding, if the tool was implemented, how effective do you feel this tool would be at reducing a user's issues with mental health?

15 responses



Appendix D

Date	hours	hour minutes	Screentime Minu	Notification Recie	Num of Unlocks	Streaming Scree	Non streaming s	SM notifications
20/12/2021	5	47	347	283	144	0	347	188
21/12/2021	6	26	386	246	96	0	386	193
22/12/2021	8	39	519	280	108	0	519	169
23/12/2021	6	20	380	275	164	0	380	179
24/12/2021	4	57	297	315	110	0	297	245
25/12/2021	5	31	331	429	137	0	331	320
26/12/2021	5	13	313	221	164	0	313	184
27/12/2021	7	58	478	213	84	0	478	163
28/12/2021	9	57	597	251	84	192	405	159
29/12/2021	11	10	670	212	104	300	370	163
30/12/2021	5	59	359	291	134	0	359	174
31/12/2021	1	23	83	131	46	0	83	84
1/1/2022	4	28	268	204	98	0	268	147
2/1/2022	6	5	365	184	54	113	252	90
3/1/2022	5	45	345	233	91	0	345	177
4/1/2022	3	50	230	144	66	0	230	88
5/1/2022	3	32	212	196	73	0	212	106
6/1/2022	2	46	166	213	38	0	166	103
7/1/2022	5	5	305	239	67	0	305	173
8/1/2022	6	18	378	176	88	76	302	136
9/1/2022	6	24	384	305	117	0	384	211
10/1/2022	5	7	307	251	72	0	307	166
11/1/2022	2	48	168	145	69	0	168	86
12/1/2022	4	31	271	262	98	0	271	186