Multilingual Android Forensic Tool for Cyberbullying Investigations on Facebook and Instagram

Komuthu Damya Mabulage – CB010187

Submitted to the

Computing School

in partial fulfillment of the requirements for the Degree of

Bachelor of Science in

Cyber Security (Hons)

Supervised By:

Mr. Jude Mayuran

Staffordshire University

August 2025, Colombo

Abstract

This research project is focused on developing a CLI-based Android forensic tool for social media applications. This research project will mainly focus on two widely used social media platforms in Sri Lanka, which are Facebook and Instagram. This tool can extract data from the above-mentioned social media platforms, process the data with NPL for analyzing in Sinhala and English, detect hate speech leading to cyberbullying and cybercrimes in both languages with a keyword filtering feature, and finally provide a detailed report that can be used as digital evidence in cybercrime investigations in Sri Lanka. This is helpful in cybercrime investigations in Sri Lanka, which is vastly spreading throughout the years, victimizing the youth mainly. Moreover, this research is mostly focused on cyberbullying and harsh speech on social media, which are quite common cybercrimes in Sri Lanka at present.

Overview of Project

At the present, there is a significant rise of cyberbullying on social media platforms all around the world including Sri Lanka. Cyberbullying happens in many ways online and using hate speech on social media to embarrass or bully someone is one of the most common ways of cyberbullying. Day by day, technology evolves, and people are addicted to social media with that cybercrimes are rising sky high. There are a lot of unsolved cybercrime cases in Sri Lanka due to limited forensic facilities and unawareness of the people. Cyberbullying is common among the young adults, the technological generation and they are the future of the country. To stop this Sri Lanka must have a strong forensic team with essential tools that specifically support the unique context in Sri Lanka. There are a number of strong forensic tools out there in the world, yet they do not support Sinhala which is the mother tongue of Sri Lanka. Sinhala itself is a complex language and Sinhala-English transliterate language which is mostly used in social media interactions is even more complex. Moreover, the existing tools are too expensive and have complex user interfaces, this adds value to any local approaches in this area.

In order to address this issue, a light weight Cli based android forensic tool called Hawkeye was developed. This forensic tool extracts all social media application data from android devices through android debug bridge as .tar files, then unpacks the tar files and filters out text-based databases from each social media application. After that the tool starts parsing the databases creating a .csv to detect hate speech using the trained BERT NLP model. Then after the detection process the results are saved as .csv files which is then used to generate a well-structured report including summary charts, hate-flagged texts and hash report etc. Specifically, this hate speech detection component is powered by a fine-tuned multilingual BERT model, trained on a custom-merged dataset containing both Sinhala Unicode and transliterated Singlish text.

Furthermore, the model was tested on real world extracted data from all the social media platforms of Facebook, Messenger and Instagram. The Facebook predictions dataset achieved 92.50% accuracy and 85.00% F1-score, Instagram scored 93.74% accuracy with 87.50% F1-score, and Messenger achieved 92.16% accuracy

and 87.84% F1-score. Additionally, the ROC-AUC scores further supported the model's strong classification performance. However overall, the tool highlights a balanced and scalable solution for Sinhala and Singlish hate speech detection in mobile forensic investigations.

Keywords - Android Forensics, Hate Speech Detection, Sinhala Language, Singlish, Cyberbullying, NLP, BERT, Machine Learning, ADB Extraction, Social Media Forensics, Digital Evidence, CLI Tool, PDF Report Generation, Cybercrime Investigation, Law Enforcement, Multilingual Text Classification

Declaration

I declare that, to the best of my knowledge and belief, this paper does not contain any previously written or published material of my own or any other person, with the only exception of situations where proper reference is provided within the text. Moreover, it does not include any previously submitted material for a degree or diploma at any university without acknowledgment.

Signature of Candidate : - 

Date : - 31st August 2025

Name of Candidate : - Komuthu Damya Mabulage

Signature of Supervisor : - 

Date : - 31st August 2025

Name of Supervisor : - Mr. Jude Mayuran

Acknowledgements

First, I would like to extend my heartfelt appreciation to my supervisor, Mr. Jude Mayuran, for his amazing guidance and support throughout the preparation of this project proposal. Moreover, I'm thankful for the invaluable advice from my lecturers at APIIT, and I'm grateful for all the invaluable support of my family and friends, understanding and motivating me frequently. Last, but not least, I would like to express my gratitude to all those who helped and encouraged me so far, both professionals and friends.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

Abbreviation	Full Form
AI	Artificial Intelligence
ADB	Android Debug Bridge
BERT	Bidirectional Encoder Representations from Transformers
CLI	Command Line Interface
CNN	Convolutional Neural Network
DL	Deep Learning
GAN	Generative Adversarial Network
GUI	Graphical User Interface
ML	Machine Learning
NLP	Natural Language Processing

PDF	Portable Document Format
RNN	Recurrent Neural Network
SRS	Software Requirements Specification
SSIM	Structural Similarity Index
SL	Sri Lanka
SLCERT	Sri Lanka Computer Emergency Readiness Team
NCPA	National Child Protection Authority
CCID	Criminal Investigation Department
ROC	Receiver Operating Characteristic
AUC	Area Under Curve
SSADM	Structured Systems Analysis and Design Method
TPR	True Positive Rate
FPR	False Positive Rate
GPU	Graphics Processing Unit
CSV	Comma Separated Values

CHAPTER 01 : INTRODUCTION

1.1. Chapter Overview

This chapter outlines the research background by defining the reason that there is an actual requirement for a forensic tool which is capable of dealing with cyberbullying in specific digital domain of Sri Lanka. This examines the way mobile phones and social media are currently a fundamental component of everyday life, especially for young adults, and how this has contributed to an increase in cybercrimes such as online harassment. Moreover, the chapter also discusses the shortcomings of current technologies, such as their inability to support Sinhala and code-mixed content, and explains how this issue creates a barrier for local investigations. Furthermore, it explains problem statement of this research, the inspiration for this project, and the existing approaches that have already been done in the field. Especially, it describes the research gaps, overall goal, main objectives, and obstacles associated with developing a tool which is both intelligent yet basic for ordinary law enforcement to use.

1.2. Introduction

At present technology has rapidly transformed the day-to-day life of people, all around the world including Sri Lanka. As a developing country, Sri Lanka has quickly embraced this digital shift, especially with the growth of smartphones and internet services. Among mobile devices, Android smartphones dominate the market, making them the most commonly used mobile phone operating system in the country. However, by now these mobile phones have become essential, not only for communication and entertainment but also for education, business, and public services almost everything.

After the COVID-19 pandemic outbreak the digital transition of the country in which the schools, workplaces, and government services shifted to online, even younger generations became heavily dependent on electronic devices. With that the usage of social media platforms such as Facebook, Instagram, WhatsApp, and TikTok dramatically increased during this time. In fact, Facebook continues to be the most widely used social media platform in Sri Lanka, with millions of active users, especially

among those aged 18 to 35. While social media has played a positive role in connecting people, it has also opened the doors towards new risks most notably, cyberbullying.

Cyberbullying has become one of the most common forms of cybercrime in Sri Lanka, often targeting teenagers and young adults. Offenders use hate speech, emotional manipulation, and even share personal content to harass victims online. According to local statistics, a considerable number of cyberbullying cases are unreported, especially in rural areas, due to fear and lack of trust in the legal systems. Even with national cybercrime units like SLCERT and CCID in place, the lack of localized tools makes it difficult for authorities to effectively investigate these incidents especially when this content is written in Sinhala and in Sinhala-English code-mixed language.

Furthermore, this research addresses that gap by proposing a multilingual Android forensic tool designed specifically for the Sri Lankan context. Unlike most existing forensic solutions that are costly and access to local language is limited, this tool focuses on extracting and analyzing social media content from Facebook and Instagram on Android devices. It integrates natural language processing (NLP) techniques to detect cyberbullying in both Sinhala Unicode and code-mixed content, and it generates detailed, structured reports that can be used in digital investigations. The tool is cost-effective, simple to use, and designed to assist Sri Lankan law enforcement in handling the growing threat of cyberbullying with language-aware intelligence and forensic accuracy.

1.3. Problem Background

At present the rate of cybercrimes in Sri Lanka is on the rise, in which only about 30% of cybercrimes are officially reported while the rest 70% are the unreported cybercrime cases, most commonly in rural areas of the country like Monaragala (Heshan Maduranga, 2024). There are three main institutions in Sri Lanka to carry out cybercrime investigations namely Sri Lanka Computer Emergency Readiness Team (SLCERT), Computer Crime Investigation Division (CCID) and National Child Protection Authority (NCPA)(Sampath, 2023). These units carry out investigations using various forensic tools and these forensic tools are expensive.

Moreover, most forensic tools are not quite efficient with the Sri Lankan mother tongue, Sinhala which is a complex language to analyze. Additionally, at present the new generation do not use the Sinhala letters to type the message in Sinhala, they use English letters to type Sinhala words according to the pronunciation of the Sinhala word, mixing both languages which is even more complex (Muthuthanthri and Smith, 2024).

1.4. Problem Statement

The inability of existing forensic tools to detect and analyze Sinhala and Sinhala-English code-mixed cyberbullying content on Android-based social media platforms severely limits the effectiveness of cybercrime investigations in Sri Lanka. Most tools are expensive, foreign-developed, and languages are incompatible, creating a critical gap in local forensic capability. Furthermore, most exciting approaches talk about the technical and the practical side of the tools so accordingly all the tools focus on detecting but do not focus on generating reports based on the analysis reports.

1.5. Problem Definition

This situation up brings a need for a digital forensic tool to analyze, filter and detect harsh speech, cyberbullying attempts in multilingual languages like Sinhala and Sinhala-English code-mixed language with low budget specifically customized for Sri Lanka. Furthermore, this tool must be capable of extracting data from social media platforms, analyzing Sinhala and code-mixed language inputs, detecting cyberbullying patterns, and producing clear, usable evidence for law enforcement investigations.

1.6. Motivation

The motivation behind this research roots from the growing digital threat to Sri Lankan youth and the lack of tools available to effectively fight against it. The identified gaps in existing systems where language limitations and high costs prevent effective investigations. As Sinhala language which is a unique and specialized to Sri Lanka is not widely supported in most NLP models, and even Sinhala-English transliterated language forms are rarely understood, a specialized solution is required to suit the unique Sri Lankan context. This project also aims to reduce the burden on law

enforcement officers by providing a CLI-based system which is user-friendly and adaptable to their investigative workflows.

1.7. Existing Work

Citation	Summary	Limitation	Contribution
(Fernando and Deng, 2023)Cellebrite (2025)	A leading forensic tool for data extraction from mobile devices, including social media apps like Facebook and Instagram.	Lacks support for Sinhala or code-mixed language analysis; high cost.	Useful for initial data extraction from Android devices.
Oxygen Forensics (2025)	This tool offers detailed extraction of multimedia, chat logs, and metadata from social media platforms.	Not optimized for NLP or detection of hate speech in local languages.	Help in retrieving raw evidence efficiently.
Samarasinghe et al. (2020)	This implements CNN and FastText models to detect hate speech in Sinhala Unicode text.	Model performance is limited by small and imbalanced dataset.	Demonstrated the feasibility of machine learning for Sinhala text classification.
Muthuthanthri & Smith (2024)	This approach applies BERT for detecting hate speech in Sinhala-English code-mixed content on Facebook.	Model accuracy drops with minor and sarcastic expressions.	Showed strong potential of transformer models for code-mixed content.
Ruwandika & Weerasinghe (2018)	Early work on hate speech detection using SVM for Sinhala comments.	Limited dataset and applicability to real-world forensic workflows.	Provided foundational dataset and binary classification baseline.
Abu Hweidi et al. (2023)	This approach conducts forensic investigation on social media apps with NIST methodology to extract and analyze digital evidence.	Focused more on extraction than content analysis; no multilingual processing.	Provided a structured forensic framework applicable to Facebook and Instagram investigations.

Chang & Yen (2020)	This focuses on forensic extraction of Facebook Messenger data from Android phones.	Lacks hate speech detection and language adaptability.	Useful for understanding app-specific data structures and recovery techniques.
Menahil et al. (2021)	This performs forensic analysis across multiple social media platforms including metadata recovery and message tracking.	Did not incorporate AI/ML techniques for content analysis or language-specific detection.	Outlined a broad forensic workflow for social network investigations.
Fernando & Deng (2023)	This enhances the detection accuracy for Sinhala hate speech using custom NLP techniques and improved feature extraction.	Focused on Sinhala only, no support for code-mixed or multilingual data.	Improved precision of local language hate speech classification.
Chathurangi et al. (2024)	This integrates detection of spam, bot activity, and cyberbullying on Sri Lankan social media platforms.	Rule-based detection with limited flexibility and no deep learning.	Proposed a context-aware local framework targeting multiple social media threats.

Table 1 - Chapter 1: Table of Existing Work

1.8. Research Gap Identification

a. Lack of Sinhala language support in forensic tools

Most popular social media forensic tools like Cellebrite, Oxygen Forensic Suite, and Magnet AXIOM are strong tools which are highly effective in social media forensic analysis, but these tools are not much effective in Sihala language which is the mother tongue of Sri Lanka. This creates a challenge for the investigators to carry out investigations, collect, analyze and document all the findings.

Justification – Developing the Android Forensic tool for social media applications like Facebook and Instagram with the localized feature of detecting and analyzing

evidence in Sinhala which is essential for addressing cybercrimes and carrying out strong cybercrime investigations in Sri Lanka.

b. High cost & complexity of existing tools

Strong digital forensic tools are expensive to buy and maintain mostly. Moreover, some tools are complex and quite inconvenient to use. Specially in Sri Lanka, with an unstable economy, daily rising cybercrimes, newly emerging cybercrime trends and limited number of experienced cyber security, digital forensics personnel using these types of tools are inconvenient.

Justification – The proposed tool will be customized to fit the Sri Lankan context. Localized tools are comparatively not costly, and maintenance cost is low. In some cases, it will not cost at all. Moreover, simplicity of the cli based structure of the forensic tool will be easy for the personnel with less experience to handle proposed tool.

c. Limited & imbalanced datasets

Most of the existing tools and research have the common limitation of limited dataset availability and imbalanced datasets. This limitation affects the ability of the tools to generalize and detect hate speech accurately. Sinhala language datasets are rare and limited so to raise the accuracy of the tool, availability of much enough datasets are crucial.

Justification – Expanding the available datasets will help to enhance the accuracy and the reliability of the proposed tool, providing valuable and accurate digital evidence for crime investigations.

d. Poor reporting/documentation in tools

Detailed documentation of the findings or the digital evidence collected is helpful in crime investigations. Most of the tools are focused on the performance of the tool and not much concerned about the small details like documentation of the digital evidence.

Justification - The proposed tool will have the capability to detect hate speech, extract digital evidence from social media platforms like Facebook and Instagram and create detailed reports for crime investigations including all the findings and detected cyber bullying attempts. Moreover, this approach will provide significant value to law enforcement in Sri Lanka.

1.9. Contribution to Body of Knowledge

This study elaborates a valuable contribution to the fields of digital forensics and multilingual hate speech detection by developing a localized Android forensic tool tailored for the unique Sri Lankan context. Although the existing researches and approaches focus on hate speech classification in Sinhala and code-mixed Sinhala-English content (Fernando and Deng, 2023; Muthuthanthri and Smith, 2024), this capability into a forensic framework designed to extract and analyze evidence directly from mobile social media platforms has not being integrated. Therefore, a combination of Natural Language Processing (NLP), transformer-based architectures (BERT), and Android forensic workflows in a CLI-based system accessible to authorized law enforcement personnel addresses this by the proposed tool.

Unlike existing commercial forensic tools like Cellebrite and Magnet AXIOM which are powerful and expensive but lack the unique Sinhala-language support (Chang and Yen, 2020; Abu Hweidi et al., 2023), this tool is customized to detect cyberbullying and hate speech in both Sinhala Unicode and Sinhala-English transliterated formats, extracted from applications like Facebook and Instagram. Moreover, the tool offers a structured, auto-generated PDF report to support evidence presentation in legal contexts as well and this bridges a key usability gap identified in existing solutions (Chathurangi et al., 2024)

Furthermore, this research project elaborates the application of multilingual BERT in low-resource language settings specially such unique and complex, showing how fine-tuned transformer models can be used to improve cybercrime investigations in diverse language environments. It further contributes to the a special area of code-mixed language processing in NLP, where Sinhala-English social media content has previously received limited attention (Ruwandika and Weerasinghe, 2018).

Apart from the technical side of the solution, the project delivers practical value by offering a scalable, affordable solution suited for the unique socio-economic context of Sri Lanka, where forensic resources are frequently limited and professional capacity is under development. Therefore, the research not only advances academic

understanding but also strengthens the operational capacity of digital forensic efforts in developing countries.

1.10. Research Challenge

1.10.1. The Challenges of Research

The main challenge of this research the development of this android forensic tool that can extract social media data from the mobile phone and accurately detect cyberbullying through hate speech in Sinhala and Sinhala-English code-mixed content, while also fitting the unique context of Sri Lanka. At first glance, this might seem like a simple classification task but in reality, it is much more complex. Moreover, there are several practical barriers that make the system difficult to be designed and implemented effectively:

- Unique Language Complexity

Sinhala in the mother tongue of Sri Lanka and it is a native language limited only to Sri Lanka unlike English, Sinhala is a complex language with complex grammar, and letters. Furthermore, at present Sinhala in mixed with English for the convenience in communication through online platforms. Therefore, it makes the language even more complex, when Sinhala combined with English in transliterated and code-mixed formats, the structure becomes unpredictable because there is no standard alphabet or standard set of spellings. With that people often switch between mid-sentence languages, use informal grammar, and write in creative, unstructured ways. These variations make it hard for NLP models to consistently understand the true meaning behind a message especially when detecting hidden threats and sarcasm.

- Lack of Quality Datasets

One of the biggest challenges is the limited availability of annotated Sinhala and code-mixed hate speech datasets. Most available datasets are small, imbalanced, and do not reflect the real diversity of how people express themselves online. Without rich, high-quality dataset, even the most advanced models will struggle to make accurate predictions. As mentioned earlier all most all the existing data sets are small and limited to 5000 – 6000 comments. The accuracy of the tool totally depends on these datasets.

- Balancing Accuracy with Usability

Another key challenge is making the tool powerful yet simple to use. Law enforcement officers in Sri Lanka often lack technical training in digital forensics and machine learning. Therefore, the tool must deliver reliable results through a clean, command-line interface, without requiring users to understand the underlying ML models and preprocessing steps. Therefore, maintaining this balance between usability and technical depth is not an easy task.

- Resource Constraints and Deployment Barriers

Developing countries like Sri Lanka often face limitations in terms of hardware, internet access, and funding for expensive digital forensics. therefore, cloud-based and high-cost solutions are not practical for the Sri Lankan context. With that the tool must be lightweight, run locally, and function well under these existing resources without compromising the performance.

1.10.2. Research Questions

In this research providing a solution for this issue in Sri Lanka, we specifically hope to answer the following research questions;

- Identification of techniques for effective data extraction and analysis of social media data.
- Analysis of the social media platforms chosen to be addressed in the research and justification for the selection.
- Investigation of the cybercrimes most commonly reported on Facebook and Instagram in Sri Lanka.
- Comparison of the accuracy of the proposed tool in detecting hate speech in Sinhala and English with the existing hate speech detection systems.
- Evaluation of the effectiveness of NLP (Natural Language Processing) models in detecting cybercrime evidence in Sinhala and English.
- Optimization of the structure of the generated PDF reports to ensure clarity, usability and easy interpretation by law enforcement and legal professionals.

1.11. Research Aim

The key aim of this research is to design and develop a multilingual android forensic tool that can assist in cyberbullying investigations by detecting hate speech and offensive content in Sinhala and code-mixed Sinhala-English formats. Moreover, the tool is specialized to support Sri Lankan law enforcement by extracting relevant social media data from platforms like Facebook and Instagram and analyzing it using NLP techniques. It further aims to generate structured reports that are understandable and usable in legal contexts on crime investigations. With the focus on language adaptability, the user-friendly tool interface, and forensic relevance, this approach creates a practical, low-cost solution for facilitate cybercrime investigations in a resource-limited but language complex environment.

1.12. Research Objectives

2.1.1. Primary objectives

The main objective of this project is to develop an Android social media forensic tool to help in detecting cybercrime evidence in both Sinhala and English tailored to unique requirements in Sri Lanka.

2.1.2. Other specific objectives

Apart from the primary objective there are few more special other objectives of this project, they are;

- To develop the tool to extract data from popular social media platforms like Facebook and Instagram on Android devices.
- To integrate Natural Language Processing (NLP) capabilities to analyze and process content in both Sinhala and English languages.
- To further develop the forensic tool to detect cyberbullying, hate speech, and other cybercrimes from extracted social media data.
- To enable filtering of data based on user-defined keywords and categories to focus on specific investigations.
- To generate detailed and structured PDF reports summarizing:
    - Extracted data.

    - Detected hate speech and cyberbullying instances.
    - Keyword-related evidence.
- To ensure the tool addresses unique cybercrime patterns and trends in Sri Lanka.
- To develop a user-friendly and easy-going interface with much less complexity suitable for law enforcement officers.

1.13. Chapter Summary

Overall, this chapter provides an overview of why this research is relevant, especially within the Sri Lankan context, where cyberbullying is on the rise and existing forensic techniques fail short primarily due to language barriers and high cost. Moreover, it describes the way smartphones and social media usage has increased, the challenges that investigators confront, and the reason that this technology is essential. The chapter further explains the research goal, important objectives, and key challenges involved in developing a system that is both effective and simple for local law enforcement to use.

CHAPTER 02 : LITERATURE REVIEW

2.1. Chapter Overview

This chapter explores the background and existing research related to cyberbullying, digital forensics, and language challenges in Sri Lanka. It looks into current trends on platforms like Facebook and Instagram, the limitations of existing forensic tools, and the requirement for local language focused solutions. By reviewing related studies and systems, this chapter sets the stage for why a localized, multilingual forensic tool is essential for effectively addressing cyberbullying in the Sri Lankan context.

2.2. Concept Map

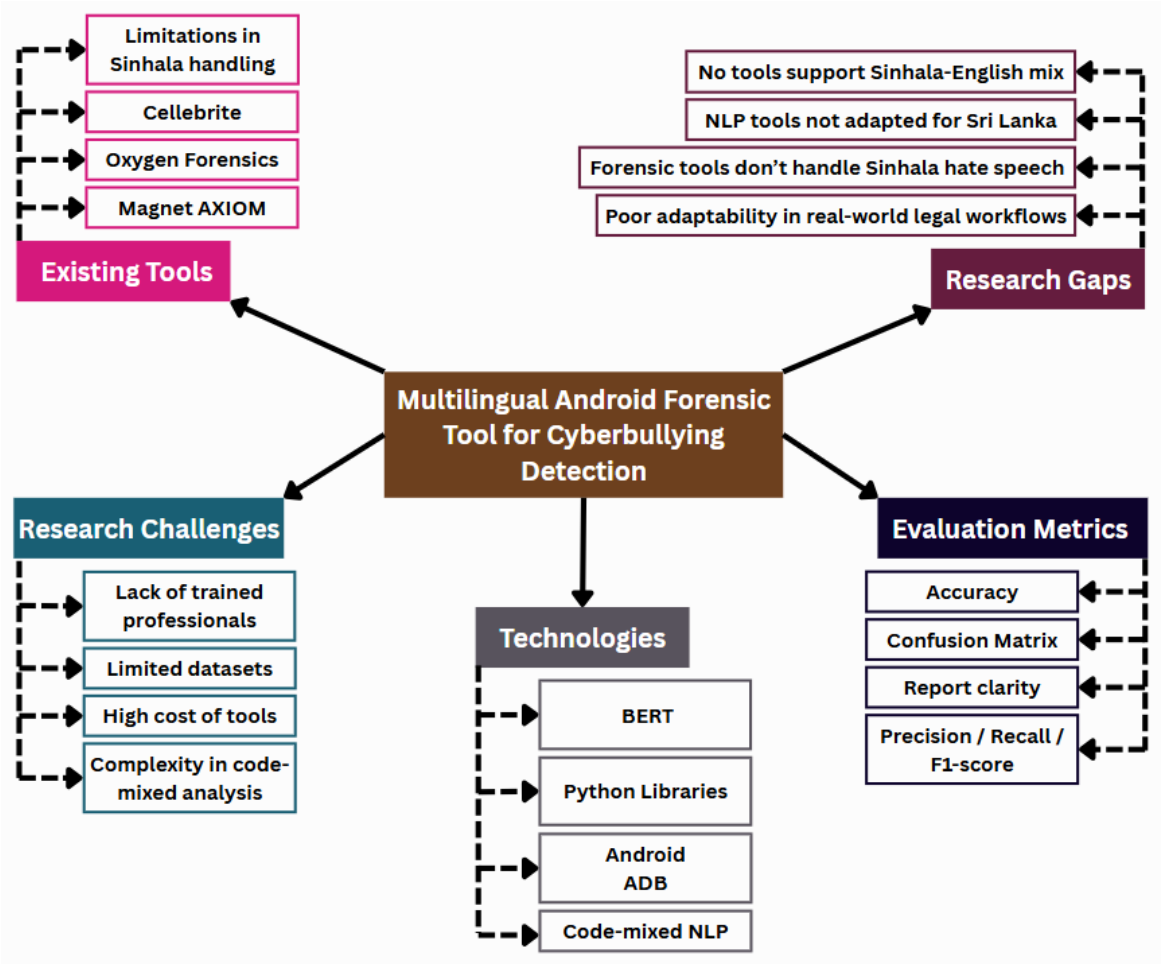


Figure 1 - Chapter 1: Concept Map

2.3. Problem Domain

2.3.1. Current Cybercrime Trends in Sri Lanka

Currently, with the worldwide evolution of technology, cybercrimes have become a common problem all over the world. Similarly, Sri Lanka witnessed a notable increase in cybercrimes with various emerging trends such as online financial fraud, phishing attacks, social media scams, fake websites, cyberbullying and online harassment. Among all these cybercrime trends, cyberbullying and online harassment have noticeably increased in the past few years with the wide spread of social media platforms and the civilian's usage patterns of social media platforms. Especially, the younger generations who are addicted to technology and social media are the main targets or the victims of this crucial cybercrime trend, cyberbullying. According to Daily Miror News, SLCERT has reported that over 9,000 cybercrime incidents were reported, with 80% linked to social media platforms between August and September 2024. Moreover, this includes 85 cases of cyberbullying involving children and 40 cases of online sexual abuse targeting minors (Barukanda, 2024).

2.3.2. Cyberbullying Cybercrime Trend in Sri Lanka

Cyberbullying is the usage of digital technology like social media platforms, emails, gaming platforms and messaging platforms to harass, threaten and embarrass an individual or a group of individuals. Unlike traditional bullying, cyberbullying has no specific time or place, it can occur anywhere at any time making it more critical and extremely hard to escape. It can be in any form such as spreading rumors, sharing private or false information, sending threatening messages, videos, photos, and impersonating an individual or a group of individuals online to cause harm.

However, it mostly leads to serious emotional distress, social isolation or even suicides and murders. Some types of Cyberbullying trends;

- Harassment – Harassment is sending hurtful or offensive messages, emails, or comments repeatedly to a person or a group of people.
- Flaming – Flaming is engaging in online fights, arguments in chat rooms or comment sections often using aggressive and offensive language.

- Outing – Outing is sharing private, personal confidential and sensitive information about a person or group of people without their permission to embarrass them and hurt them.
- Catfishing – Catfishing or impersonation is creating fake profiles or accounts in social media platforms pretending to be someone else to trick or harm a victim.
- Cyberstalking – Cyberstalking is intense online harassment which includes threats of harm or physical violence.
- Trolling – Trolling is provoking someone online by posting offensive and hurtful comments.
- Exclusion – Exclusion is kicking out or leaving someone out of online groups, chats and social activities to make them feel isolated and excluded.
- Meme Bullying – It is using edited images, GIFs, stickers or any sort of a meme to mock and humiliate a person or a group of people publicly.
- Sexting and Revenge Porn – Sexting is sharing deepfake or inappropriately edited images, messages and videos without permission to blackmail or take revenge from someone.
- False Accusations - this is spreading false information or rumors to damage the reputation of an individual or a group of individuals.

Despite the type of cyberbullying trends, each type has affects harmfully on its victims leading to motional, psychological, and legal consequences.

2.3.2.1. Rise of Cyberbullying Cases in Sri Lanka

The most significant reason for the rise of Cyberbullying cases in Sri Lanka is the rapid growth of social media usage in Sri Lanka. Despite the age group, all young, mid and older age groups use social media daily and it has gradually become a part of their day-to-day life. Hobbies, fun activities, family time and leisure time are consumed by social media. At present, even some middle school kids have mobile phones with access to all the social media platforms in Sri Lanka. SL Cert has reported that most of the victims are university students, the young adults who are considered as the future of the country. Sri Lanka is a small island with 25 districts and a population of approximately 21.96 million so, comparing with other countries in the world it is a small

country with an average amount of population. However, sources state that there were 8.20 million active social media user identities in Sri Lanka in January 2025 which is 35.4 percent of the total population (Kemp, 2025). These statistics clearly highlight that over the past years and with development of technology, social media has speeded out vastly all around the country within a very short time period.

With the recent rise in usage of social media, Sri Lankans are more updated, and they tend to judge other individuals and show hatred with comments, messages, posts etc. For an instance, Sri Lankan celebrities face this issue all the time, the public are connected to celebrities through social media closely and this had affected the privacy of celebrities, their personal details are published, they are criticized openly, they are trolled bullied for their creations, actions, speech and behavior. Not only celebrities this could happen to any civilian despite their age, gender, generation, career, status etc. because Sri Lankans now tend to use social media as a platform to speak up about the things that they are not comfortable speaking in person. Due to this reason cyberbullying, using hate speech and trolling in social media has become quite common in society. Moreover, sources state that, over 2,000 cases of cyberbullying have been reported in Sri Lanka so far in 2024, according to official records from the Sri Lanka Computer Emergency Readiness Team (SLCERT). Furthermore, Charuka Damunupola, an engineer of SLCERT revealed that approximately 9,500 cybercrimes had been reported during the first 10 months of the year 2024.(Weerasooriya, 2024)

2.3.2.2. Effects of Cyberbullying

Cyberbullying is an increasingly severe concern at present due to its lasting impacts on victims, especially the younger generations who are victimized right way. Various scholarly studies, research papers, and surveys provide evidence of the harmful effects of being cyberbullied such as;

- Psychological effects like depression, anxiety, low self-esteem, self-worth, isolation and loneliness are quite common among victims. Such psychological effects affect their daily lives and overall mental health. Ariyadasa (2019) highlighted that approximately 90% of university students subjected to cyberbullying in Sri Lanka reported symptoms of psychological distress,

including anxiety and depression (Ariyadasa, 2019). Moreover, continuous harassment and humiliation online lead these victims to perceive themselves negatively, lowering their confidence, self-esteem, social withdrawal and isolating themselves from friends and family (Gohal et al., 2023).

- Cyberbullying results in the decline in academic performances reducing productivity and work performance. Continuous bullying negatively impacts the concentration and engagement of the victims, causing a decline in academic performance, future interests, disrupting the productivity, increasing dropout rates and reducing the overall work efficiency (Gohal et al., 2023).
- Cyberbullying also indirectly influences physical health like sleep disorders and physical Stress Symptoms. Victims frequently report disturbances in sleep patterns, such as insomnia, due to anxiety or distress caused by cyberbullying. Moreover, the chronic stress from bullying leads to physical symptoms like headaches, stomach issues, and fatigue (Gohal et al., 2023).
- Long-term exposure to cyberbullying can lead to noticeable behavioral changes like aggression, risk-taking behavior and violence. With this, victims in extreme scenarios expose aggressive behavior, sometimes leading to revenge and violence (Ariyadasa, 2019).
- A particularly severe consequence of cyberbullying is suicidal ideation and actions. The relentless nature of cyberbullying, combined with isolation and emotional distress, increases the risk of suicide. Moreover, Ariyadasa (2019) has elaboratively mentioned cases where cyberbullying incidents grown tragically worse, ending up suicide (Ariyadasa, 2019).
- Furthermore, cyberbullying negatively impacts social relationships by building up distrust between peers, friends, and family members, leading to damaged relationships and disrupting social interactions, missing critical opportunities for personal growth and socialization.
- Apart from all the above cyberbullying indirectly affects financial implications by simply wasting money on psychological counseling and medical treatments, imposing financial burdens on families and individuals. Other than that, cases requiring legal action further enhance financial strains.

2.3.2.3. How to mitigate Cyberbullying in Sri Lanka

There are various options to mitigate cyberbullying in Sri Lanka. However, mitigating this type of cybercrime in Sri Lanka involves a varied set of approaches including;

- Awareness programs and education

  Holding educational programs and workshops by institutions like SLCERT and the National Child Protection Authority (NCPA), targeting schools, universities, and community centers to inform the young generation to identify, avoid, and respond against cyberbullying, are crucial to promote online safety awareness (Ariyadasa, 2019; Heshan Maduranga, 2024).

- Effective legislation and law enforcement

  Strengthening legal frameworks and cybercrime laws, with clear definitions, strict penalties, reporting channels, enabling victims to report cyberbullying without fear of revenge, is crucial for the mitigation of cyberbullying attempts (Sampath, 2023).

- Technical solutions

  Implementation of technological tools, such as multilingual hate-speech detection tools, software and social media monitoring systems specifically aligned for the unique context of Sri Lanka, to identify and manage cyberbullying is another effective mitigation strategy (Muthuthanthri and Smith, 2024).

- Support systems and counseling services

  Furthermore, providing accessible psychological counseling and support services to victims helps to mitigate the psychological harm caused and prevents long-term trauma suffered by the victims as a result of cyberbullying.

2.3.2.4. Importance in Mitigating Cyberbullying

Mitigating cyberbullying is crucial due to several reasons. Firstly, mitigation of this cybercrime directly helps in protecting the mental health of individuals, specifically the youth, helping to prevent conditions like anxiety and depression (Gohal et al., 2023). Additionally, it promotes the creation of safe digital environment where users have the ability to interact positively, minimizing the risks of being online bullied. trolled and harassed (Jayasinghe et al., 2024). Moreover, severe and dangerous outcomes such as

loss of self-harm, suicide attempts, and social isolation can be reduced (Ariyadasa, 2019). Furthermore, reducing cyberbullying attempt leads to a variety of educational and social benefits, including better academic performance, enhanced productivity in professional settings, and stronger relationships within communities. With consideration to all these reasons mitigation of cyberbullying is crucial according to the current situation in Sri Lanka.

2.3.3. Forensic Investigation Challenges in Sri Lanka

2.3.3.1. Limited Use of Forensic Tools in Sri Lanka

The usage of digital forensic tools in Sri Lanka remains significantly low due to several factors like high costs, the complexity, lack of training and language limitations. The high costs of implementing and maintaining advanced forensic tools, such as Cellebrite and Oxygen Forensic Suite, limit their widespread adoption in Sri Lanka (Cellebrite, 2025; Oxygen, 2025). Moreover, the existing forensic tools are complex, require extensive training, hence only a limited number of trained personnel in Sri Lanka have the accessibility for such tools (Maduranga, 2024). Furthermore, popular forensic tools typically support global languages and are quite effective for Sinhala and Sinhala-English code-mixed contexts, causing difficulties when investigating in local cybercrime cases (Muthuthanthri and Smith, 2024).

2.3.3.2. Limited Security Professionals in Sri Lanka

Sri Lanka faces a shortage of skilled cybersecurity and digital forensic professionals at present due to various reasons. Limited availability of specialized training programs in cybersecurity and digital forensics limits the growth of qualified professionals (Maduranga, 2024). Moreover, there is a significant difference between the demand for cybersecurity professionals and their availability, hindering effective response to rising cybercrimes (Sampath, 2023). Furthermore, at present talented professionals migrate abroad for better opportunities, expanding the local shortage (Sampath, 2023).

2.3.3.3. Lack of Knowledge of General Public About Forensic Investigations

However, despite the wide spread of technology, public awareness on digital forensic practices and procedures is significantly low. Many Sri Lankan citizens lack knowledge

about what constitutes digital evidence, how forensic investigations are conducted, and how critical evidence can be preserved (Ariyadasa, 2019). Moreover, the limited understanding of forensic investigations leads to fear, mistrust, and doubt towards law enforcement, which results in the increase of underreported cases and creating gaps in addressing cybercrimes effectively (Maduranga, 2024).

2.3.4. Unique Sri Lankan Context – Multilingual Language Barriers

2.3.4.1. Complexity of Sinhala Language

Sri Lanka is a culture enriched country with a unique language called Sinhala, as its primary language, this unique and complex language challenges the existing digital forensic and NLP applications affecting in the effectiveness and accuracy. Sinhala has complex grammatical structures, with various conjugations, tenses, and significant expressions, in which an automated language processing model or tool particularly challenging (Samarasinghe, Meegama and Punchimudiyanse, 2020). Computational linguistic tools and resources like datasets, corpora, language models for Sinhala language are limited compared to global languages, hence NLP development, forensic tool effectiveness is limited, and it is challenging to train models with limited datasets (Samarasinghe, Meegama and Punchimudiyanse, 2020). Furthermore, the use of mixed up languages like using English letters to convey Sinhala messages further complicates text extraction and analysis in forensic investigations (Samarasinghe, Meegama and Punchimudiyanse, 2020).

2.3.4.2. Complexity of Sinhala-English Code-Mixed Language

Code-mixing between Sinhala and English, using English letters to represent Sinhala words, significantly complicates cyber investigations and NLP analysis. This mixed up language system does not have any standardized spelling or representation when using English letters for Sinhala phonetic sounds hence it creates uncertainty and reduces the accuracy in automated text analysis (Muthuthanthri and Smith, 2024). Moreover, annotating datasets for training machine learning models are challenging due to inconsistencies and lack of clear language boundaries in code-mixed texts (Muthuthanthri and Smith, 2024).Furthermore the existing NLP models, designed for

single-language analysis, produces results with reduced accuracy when handling complex Sinhala-English code-mixed scenarios (Muthuthanthri and Smith, 2024).

2.4. Existing Systems

2.4.1. Review of Existing Systems, Approaches and Tools.

Forensics Investigation on Social Media Apps (Abu Hweidi et al., 2023)

This study provides a clear and structured approach to forensic analysis on social media platforms like Facebook and Instagram, showing how digital evidences such as messages, images, timestamps, and metadata can be extracted for use in legal crime investigations. Moreover, it follows the NIST framework to ensure the process is thorough and legally sound, and it reviews different extraction tools used in real-life cases. However, the study does not cover language-related challenges, such as identifying hate speech and cyberbullying in Sinhala or Singlish, nor does it explore using AI and NLP for deeper content analysis. Despite this, it outlines the use of technologies like mobile forensic suites, SQLite analysis, the NIST methodology, and metadata tools. In conclusion, even though it offers a solid foundation for forensic work, it needs to include multilingual and AI-powered features to be fully effective in countries like Sri Lanka.

Evidence Gathering of Facebook Messenger on Android (Chang and Yen, 2020)

This research focuses on collecting digital evidence from Facebook Messenger on Android devices, detailing how to recover messages, timestamps, metadata, and files stored within the application. It identifies key data locations storing critical evidences like SQLite databases and cache directories and explains how to access them using ADB or physical methods. The paper is specifically relevant for Sri Lanka, as it offers clear, step-by-step guidance tailored to the widely used Android system and emphasizes maintaining legal standards through proper evidence handling. However, it only addresses data extraction and excludes post-extraction analysis that includes sentiment analysis, user behavior profiling, and cyberbullying identification. Furthermore, it mainly evaluates English content, omitting Sri Lanka's language diversity. Although the overall study offers strong technical guidance, its lack of content analysis reduces its

usefulness in tackling modern digital crimes. To be more effective in Sri Lankan cybercrime investigations, especially those involving hate speech or abuse, it should be enhanced with NLP tools that can handle multilingual and context-rich content.

Forensic Analysis of Social Networking Applications (Menahil et al., 2021)

This study looks into the forensic analysis of social networking apps, focusing on user interactions and recovering critical data such as messages, timestamps, and file attachments. It covers both logical and physical data extracting methods, as well as deleted file recovery, which are frequently disregarded in ordinary investigations. This diverse research, which employs cross-platform analysis, includes the recovery of deleted information and forensic traces that are frequently overlooked in normal techniques. The systematic approach it takes is consistent with industry norms, making it appropriate for both law enforcement and academic application. However, the study focuses mostly on technical elements and avoids addressing local and language-specific problems. Additionally, it lacks tools for detecting cyberbullying and evaluating text using advanced methods such as NLP. Although this study provides a good framework for general forensic operations, it lacks the intelligence layer needed to analyze user activity and identify offending content. The study would have greater impact if its combined with natural language processing (NLP) and hate speech identification techniques that are adapted to local language patterns in nations like Sri Lanka, in which cybercrimes frequently use local and code-mixed languages.

Identification of Hate Speech in Social Media (Ruwandika and Weerasinghe, 2018)

This initial research provides a foundation for Sinhala NLP research and focuses on identifying hate speech in Sinhala-language social media content. Using a manually gathered and annotated dataset from public posts, it employs fundamental supervised machine learning techniques, such as SVM, to categorize speech as hate and non-hate. Common emotions and language patterns associated with hate speech in Sri Lanka are also examined in the study. Although this lays a strong foundation to address this problem and offers insightful information, its limited dataset and usage of basic models limit its capacity to handle real-world situations and scale up. Additionally, it does not detect code-mixed Sinhala-English content, which can frequently be discovered online.

SVM, emotion analysis, and Sinhala Unicode text tokenization are among the technologies used in this approach. Although the study provides a solid foundation overall, it does not include proof of concept. With support for transliterated Sinhala-English code-mixed material, deep learning, and larger datasets, this approach provides a great foundation for further research.

Hate Speech Detection for Code-Mixed Data (Muthuthanthri and Smith, 2024)

The goal of this study is to identify hate speech in transliterated, code-mixed Sinhala-English language, which is frequently found on Sri Lankan social media. It uses sophisticated natural language processing (NLP) techniques, particularly the BERT transformer model, to address the complexity of code-mixed language. Moreover, the algorithm is capable of understanding informal and natural online communication as it is trained on a specifically constructed dataset that comprises Sinhala words written in English letters. This study further addresses a critical gap in Sri Lankan hate speech identification by examining how people use language in online platforms. However, compared to previous approaches, BERT increases accuracy and facilitates the model's better understanding of context but still struggles with a few things like sarcasm, short texts, and figurative language, and it hasn't been fully tested beyond its dataset. Overall, this study is one of the most advanced, and it directly addresses the specific Sri Lankan context providing a strong foundation for developing multilingual forensic tools to identify cyberbullying as it uses of real-world language and latest NLP techniques.

Cyberbullying Detection using AI (Khairy, Mahmoud and Abd-El-Hafeez, 2021)

This study proposes an AI-based model for detecting abusive and bullying behavior on social media, with a focus on verbal abuse such as insults, threats, and caustic comments. The algorithm is trained on structured data identified by users and moderators, which allows it to efficiently identify abusive content. Moreover, it demonstrates how AI can enable real-time monitoring while reducing the requirement for constant human supervision. The system is flexible, so it is capable of adapting to different sorts of violent conduct, and its structured labeling method is compatible with existing filtering systems. However, it functions only with English-language data and does not support the Sinhala and Sinhala-English transliterated unique local languages

found in countries like Sri Lanka. Furthermore, the technologies used in this approach include NLP preprocessing, classifiers such as logistic regression and SVM, and abuse categorization. Although the study provides an effective and scalable detection model, it requires retraining with local languages and usage patterns before it can be used in Sri Lankan forensics. Still, it provides a useful platform for developing AI-powered cybercrime detection systems.

LLMs for Cyberbullying Detection (Ogunleye and Dharmaraj, 2023)

This study explores the usage of Large Language Models (LLMs), namely GPT-3, to detect cyberbullying in social media conversations. This highlights how these models can recognize context, tone, and complex language that older systems sometimes overlook. One of their unique features is their use of zero-shot and few-shot learning, which enables them to perform effectively even with insufficient training data. These models reflect an advanced approach, capable of understanding complex language, slang, and a variety of text forms. However, the study focuses mostly on English and does not address multilingual or code-mixed content, limiting its usefulness in countries like Sri Lanka. Furthermore, the high computing demands of LLMs such as GPT-3 could be an obstacle for developing countries with less resources. GPT-3, NLP pipelines, and few-shot learning algorithms are among the technologies used. Though

the study reveals that LLMs have a high potential for detecting cyberbullying, their practical and effective implementation in Sri Lanka requires local language adaption, specialized training data, and cost-effective infrastructure.

Cyberbullying, Spam & Bot Detection (Chathurangi et al., 2024)

This study proposes a paradigm for detecting multiple cyber threats such as cyberbullying, spam, and bot activity on Sri Lankan social media platforms, understanding that these issues frequently overlap. It employs language analysis and rule-based methods to detect harmful content and conduct, with a heavy emphasis on local context. Furthermore, this study integrates cyberbullying detection with spam and bot analysis specifically for Sri Lanka, providing a more comprehensive picture of online threats. The approach, however, depends on keyword-based rules rather than deep learning or advanced NLP, making it less effective at dealing with complex or

informal language, such as Sinhala-English code-mixed text. Classifiers powered by rules and simple pattern recognition were among the technologies deployed. Overall, the study provides a solid foundation for a localized detection system, but in order to be more effective, it must integrate updated AI methods like as transformer models and support for multilingual data, which is widespread in Sri Lankan online communication.

Threatening Language and Target Identification (Amjad et al., 2021)

This study focuses on recognizing threatening language in social media and determining who the threats are aimed at, individuals or organizations. It emphasizes the risks of both direct and indirect threats along with a text classification system capable of detecting violent and abusive content. Unlike simple analysis of emotions, it seeks to identify specific targets, making it useful in forensic and law enforcement investigations. The concept is intended to integrate into legal frameworks by identifying various forms of hazards. However, it is based on English-only data and does not account for local differences, multilingual and code-mixed communication, limiting its utility in countries like Sri Lanka. NLP, Named Entity Recognition (NER), and threat classification models are the main technologies used in this approach. In conclusion, although the study presents a valuable tool for identifying digital threats and their targets, it would require extensive adaptation especially language support for Sinhala and code-mixed text to be genuinely effective in Sri Lankan cybercrime investigations.

Cyberbullying Prediction using ML (Al-Garadi et al., 2019)

This study addresses applying machine learning to anticipate cyberbullying before it occurs by evaluating user behavior, post content, and engagement trends. Rather than simply responding to threatening content, the tool identifies accounts that are more likely to engage in bullying based on historical trends. This proactive strategy is beneficial for moderation teams in detecting concerns early on. It combines activity tracking and textual analysis to detect recurrent abusive habits more efficiently. However, the model is based on English data and does not adjust for local language variances, hence its applicability in countries like Sri Lanka is limited. Additionally, it relies on huge, labeled datasets, which can be difficult to find in situations with limited resources. Behavioral analytics, supervised machine learning, sentiment analysis, and

temporal feature analysis are the main technologies used for this approach. In conclusion, though the approach has great potential in preventing cyberbullying, it requires local language adaptation, in addition to locally relevant data, to be used effectively in Sri Lankan forensic investigations.

Sinhala Cyberbullying Classification (Amali and Jayalal, 2020)

This study proposes a machine learning approach for detecting cyberbullying in Sinhala, which is an unrepresented language in NLP. The model is trained on a customized dataset of Sinhala social media posts, and it intends to provide a useful, locally relevant tool for Sri Lankan law enforcement and social media platforms. Being research that investigates cyberbullying in Sinhala, it provides a significant contribution by dealing with both local language issues. However, the dataset is small and lacks a diverse range of bullying categories, therefore the model's accuracy is limited. It further does not allow code-mixed Sinhala-English text, which is widely used on networks like as Facebook. Supervised machine learning algorithms such as SVM and Naive Bayes were employed, in addition to a customized Sinhala dataset and text preprocessing. In conclusion, this study is a good start toward localized forensic tools in Sri Lanka, however, it requires a larger dataset, support for code-mixed languages, and more advanced models such as deep learning.

Online Hate Speech in Sinhala (Shibly, Sharma and Naleer, 2021)

This study focuses on recognizing hate speech written in Sinhala using classic NLP and machine learning techniques. It emphasizes the difficulty of working with Sinhala's distinct syntax and vocabulary, hence proposes a methodology for detecting hate speech from normal conversation. The data set gathered locally helps to keep the social media context genuine. Additionally, the study is significant for the country as it specifically focuses on Sinhala. However, using simple models like Naive Bayes and logistic regression limits the system's capacity to detect deeper and complicated language, such as sarcasm and hidden hatred. Furthermore, the dataset is limited and unbalanced, hence it limits the generalizing ability of the model. In conclusion, this study is an ideal starting point for detecting Sinhala hate speech but requires using more

advanced techniques like deep learning and expand its coverage of Sinhala-English code-mixed text to be more effective for real-world cybercrime detection in Sri Lanka.

Instagram Digital Forensics (Mubarik et al., 2021)

This study focuses on the digital forensic examination of the Instagram application on Android devices, demonstrating how to retrieve critical evidence such as chat logs, media files, and timestamps for criminal investigations. It emphasizes the importance of maintaining a chain of custody and describes a simple, practical methodology for data recovery and presentation, which is applicable to real-world legal concerns. However, while the forensic procedure is effective, the study does not address what to do with the data after extraction, such as recognizing cyberbullying or tracking user behavior. It further skips problems associated with language, which are critical in countries like Sri Lanka. Android forensic tools, SQLite viewers, and ADB are the technologies used in the research approach. Overall, the study provides a solid framework for gathering Instagram evidence, but it would be far more valuable for Sri Lankan cybercrime investigations if integrated with AI and NLP models capable of analyzing Sinhala and code-mixed information.

Integrated Detection of Cyberbullying (Jones, Winster and Valarmathie, 2022)

This study employs a comprehensive strategy to detect cyberbullying by merging several sorts of digital evidence, such as text messages, user activity, and platform metadata. It combines forensic tools and NLP approaches to create a more accurate and trustworthy investigation process, comparable to that which law enforcement conduct. Employing technologies like Oxygen Forensics in conjunction with NLP models enables both effective data extraction and deeper analysis. Although the procedure is accurate and forensically solid, it does not handle different languages, making it unsuitable for countries like Sri Lanka. It additionally is not suitable for low-resource and command-line-based systems. In conclusion, the study provides a solid foundation for cyberbullying identification and serves as a useful model. However, to be useful in Sri Lanka, it must offer multilingual content and remain more resource efficient.

Enhancing Sinhala Hate Speech Accuracy (Fernando and Deng, 2023)

This paper aims to improve the accuracy of hate speech detection in Sinhala by modifying how models process language and extract features. It combines techniques like emotion labeling and phrase structure analysis to better comprehend the overall mood and purpose of messages, with the goal of reducing false positives and negatives. Unlike many experiments, it goes beyond basic detection and focuses on improving model accuracy, focusing on Sinhala to maximize performance in this area of study. Its focus on Sinhala-specific elements of the language makes it especially useful for forensic purposes in Sri Lanka. However, it only handles Sinhala text and excludes the common usage of Sinhala-English code-mixed language, limiting its applicability in real-world situations. Overall, this study represents an important step forward in detecting Sinhala hate speech, but it must be expanded to include code-mixed and multilingual content in order to be truly relevant in Sri Lankan cybercrime investigations.

Bengali Hate Speech Detection (Senapati and Roy, 2023)

This study addresses hate speech detection in Bengali using deep learning models such as CNNs and RNNs. Bengali, like Sinhala, is a low-resource language, hence the research addresses issues such minimal annotated data and script complexity. The models produce great results after carefully adjusting and preparing the data, proving that deep learning is still effective with small datasets. The study is interesting since it focuses on a closely connected language and illustrates how AI can perform well in such contexts. However, it does not address how to implement the model in real-world systems and is restricted to the Bengali context. CNN, RNN, and NLP pipelines are the main technologies deployed in this approach. Overall, the study provides useful insights for detecting Sinhala hate speech, especially on how to manage limited data, but it lacks support for code-mixing and multilingual usage, which are critical in Sri Lanka. It is a useful point of reference for adopting deep learning methodologies to Sinhala.

2.4.2. Comparative Analysis Review of Existing Systems

The review of current tools for cyberbullying detection and social media forensics reveals clear strengths and limitations across different approaches. Tools introduced by Abu Hweidi et al. (2023) and Chang & Yen (2020) are excellent at extracting data such as chat logs and metadata, but they stop at surface-level recovery and don does not help in understanding harmful or abusive content especially in multilingual environments like Sri Lanka (Chang and Yen, 2020; Abu Hweidi et al., 2023). On the other hand, studies by Muthuthanthri & Smith (2024) and Fernando & Deng (2023) makes solid progress in detecting hate speech in Sinhala-English code-mixed content using advanced models like BERT (Fernando and Deng, 2023; Muthuthanthri and Smith, 2024). Although it is effective, these models need high-quality datasets and still struggle with detecting subtle forms of abuse like sarcasm. Furthermore, earlier work by Ruwandika & Weerasinghe (2018) and Amali & Jayalal (2020) lay the foundation for Sinhala-language abuse detection using simpler models like SVM (Ruwandika and Weerasinghe, 2018; Amali and Jayalal, 2020). These are resource-efficient but not effective enough for complex and mixed-language online conversations at present.

Moreover, some studies shift their focus to behavior, like Al-Garadi et al. (2019), who try to predict bullying based on user activity patterns (Al-Garadi et al., 2019). Although the work is exceptional, these models are not tailored for Sri Lankan users. Similarly, Amjad et al. (2021) propose systems to identify targets of online threats, but they do not align with unique local language and context (Amjad et al., 2021). However, Ogunleye & Dharmaraj (2023) use powerful LLMs like GPT-3 to interpret abuse with impressive accuracy but, such models are expensive and poorly adapted to local context, hence, is hard to implement in countries like Sri Lanka (Ogunleye and Dharmaraj, 2023). Chathurangi et al. (2024) introduces a more practical, locally aware approach combining detection of spam, bots, and cyberbullying using rule-based systems which are easier to run but not flexible enough for detecting latest and more complex threats (Chathurangi et al., 2024). Finally, specific studies like Asim Mubarik et al. (2021) focus on Instagram forensics, helping investigators recover important evidence but they lack tools for understanding the meaning behind the content (Mubarik et al., 2021).

Overall, the tools are specified for a specific task, some are great at retrieving data, others at analyzing language so, a system that brings both together which is capable of handling multilingual, real-world cybercrimes fits the digital landscape Sri Lanka.

2.5. Technological Review

The development of the multilingual Android forensic tool for cyberbullying detection required a careful integration of datasets, preprocessing techniques, and algorithmic models all selected to suit the complex linguistic and forensic environment of Sri Lanka. In developing this system, which is focused on using advanced machine learning models ensuring that the tool is practical, culturally adaptive, capable of handling the cyberbullying attempts and its language usage in Sinhala, Sinhala - English code-mixed formats.

2.5.1. Dataset

The model was developed using two key annotated datasets, a Sinhala Hate Speech Dataset and a Sinhala-English Code-Mixed Hate Speech Dataset. Both datasets are compiled from real-world social media content, gathered from platforms like Facebook, where cyberbullying and hate speech are prominent. Each dataset includes user-generated comments manually labeled with categories such as hate speech, cyberbullying, and neutral content.

The Sinhala Hate Speech Dataset includes Unicode Sinhala text, covering a range of offensive expressions commonly used in native Sinhala. It provides structured insight into abusive language in a fully Sinhala context, hence it is crucial for detecting hate speech written in the local language, the mother tongue of Sri Lanka.

On the other hand, the Code-Mixed Hate Speech Dataset which contains comments where Sinhala words are typed using the English alphabet commonly known as transliterated Sinhala often mixed with English phrases and most of the Sri Lankan use this method in social media platforms for communication especially the younger generations. Such comments are typically informal, context-dependent, and highly variable in spelling and grammar; therefore, its uniqueness challenges automated detection. However, both datasets were relatively small and suffered from class

imbalance, with fewer samples labeled as severe hate speech and cyberbullying compared to neutral comments. To address this:

- Cleaned the data by removing duplicated, mislabeled, and incomplete records.
- Balanced the class distribution through oversampling of minority categories.
- Performed light data augmentation by introducing spelling variants, synonym substitutions, and transliterated duplicates to reflect the noisy, real-world nature of user content.

Moreover, both the datasets were combined into a unified training set, allowing the model to learn from the full language spectrum formal Sinhala, informal Sinhala, and Sinhala-English code-mixed text. This decision was critical for generalization, ensuring that the model could perform reliably on diverse user inputs during real forensic casework. In conclusion, the dataset strategy not only enhanced the model's language coverage but also reflected the social and cultural communication patterns unique to Sri Lanka. This dual-dataset approach forms the backbone of the system's ability to detect doubtful, context-driven cyberbullying content across local language forms.

2.5.2. Preprocessing

The dual-dataset nature of the model in Sinhala Unicode and Sinhala-English code-mixed transliteration the preprocessing pipeline accommodated two distinct language forms. This stage is not just about cleaning the text it is about creating a bridge between informal social media language and structured machine learning input. With that custom tokenization, using separate configurations for Sinhala script and English-Sinhala character transliterations. Sinhala Unicode characters are tokenized based on native syllabic structure, while transliterated Sinhala followed word-boundary rules which are optimized for code-mixed texts. Moreover, to standardize inputs, normalization was applied which is converting all text to lowercase, removing punctuation, emojis, special characters, and standardizing spellings where possible.

Given that both datasets contained a mix of offensive and neutral content, therefore stop word removal was applied cautiously ensuring we did not remove

emotionally charged or context-bearing words. Furthermore, a language tagging module is applied which identifies the segments as Sinhala or English. This allowed text to feed into multilingual embeddings with better emotional alignment. Additionally, class labels were then encoded into hate speech, cyberbullying, and neutral categories, enabling structured training and meaningful evaluation metrics such as accuracy, recall, and F1-score.

With that to address the spelling variations and informal grammar common in user-generated content, we introduced minor augmentation techniques like synthetic misspellings and variant transliterations to help the model generalize better. This stage is especially important for improving performance on informal and youth based online language usage, where grammar and spellings are often intentionally distorted. However, together, these preprocessing steps enabled the system to understand and adapt to the complex digital languages of modern Sri Lankan social media, allowing the model to make meaningful predictions on highly variable, multilingual user input.

2.5.3. Algorithm Selection

Choosing the right model architecture is the core to develop a tool capable of understanding not just language, but multilingual online expression. After exploring several candidate algorithms, BERT (Bidirectional Encoder Representations from Transformers) is adapted as the primary model. Its ability to capture bidirectional context made it ideal for interpreting mixed-language posts, detecting implied and indirect hate speech, and understanding fragmented syntax which is a common trait in cyberbullying. Moreover, a multilingual BERT fine-tuned using the Hugging Face Transformers library on PyTorch as this version of BERT had already been pretrained on multiple languages including Sinhala to some extent, it is specifically effective in handling transliterated Sinhala-English content. Even when sentences contained in middle switches in language and tone, BERT is able to contextualize them more reliably than classical models.

However, before finalizing BERT, the models below were compared as well:

- CNN with LSTM Hybrid: Combined convolutional pattern recognition with sequential modeling. This is effective for structured sentences, but fails to

capture contextual content in short, slang-heavy and emotionally charged social media posts.

- Naive Bayes (Baseline): Used for benchmarking, this model offered fast predictions but lacked semantic depth. It often misclassifies the content due to its inability to detect sarcasm and hidden emotions.
- Transfer Learning: Although this is not implemented in the final tool, this is evaluated the potential of cross-lingual transfer learning. Embeddings pretrained in morphologically similar languages (e.g., Hindi) showed the capacity for enhancing Sinhala model performance with minimal training data.

Training was conducted locally using PyTorch, where available hardware resources are used to manually tune hyperparameters such as learning rate, batch size, and number of epochs. Cloud-based platforms like Google Colab limit the scalability of experimentation, but using hardware resources allowed to maintain full control over the training process and environment.

In summary, the tool combines localized datasets, a structured preprocessing pipeline, and a fine-tuned BERT model to effectively detect cyberbullying in the Sri Lankan context. It understands the unique mix of Sinhala, English, and code-mixed language and aligns modern ML with real-world local usage for digital forensic investigations.

2.6. Evaluation and Benchmarking

2.6.1. Evaluation Criteria for Multilingual Cyberbullying Detection

In order to evaluate the effectiveness of the Android forensic cyberbullying detection tool, various evaluation standards were designed based on best practices in NLP, machine learning, and forensic usability;

- Accuracy and F1 Score: These are primary indicators for model performance. Accuracy helps to assess overall accuracy, while the F1 Score offers a balance between accuracy and recall especially important in imbalanced datasets.

- Precision and Recall: Precision highlights how many of the flagged instances are cyberbullying, while recall measures how many of the cyberbullying cases were correctly identified. High recall is important in forensic applications as a missing harmful post can compromise an investigation.
- Language Robustness: Given the code-mixed nature of the input, evaluates how well the model handles Sinhala Unicode, transliterated Sinhala and English. This is done by measuring individual class performance on samples containing each script type, ensuring the tool works across language forms.
- Confusion Matrix Analysis: Confusion matrices are used to identify specific weaknesses, such as false positives which are flagging non-abusive content as cyberbullying and false negatives which are failing to catch actual hate speech. This helps to fine-tune the model's thresholding and improve real-world trustworthiness.
- Practical Relevance: Beyond machine learning metrics, the usability of the tool is evaluated in real-life forensic contexts by measuring the clarity of the generated reports and the usefulness of flagged content. Informal pilot feedback from testers highlights that the system was able to present evidence in a format that's understandable even by non-technical law enforcement personnel.

2.6.2. Benchmarking Methods and Best Practices

- Standardized Datasets: Although custom and real-world datasets are used, the benchmarked model performance is using a fixed testing subset, and it avoids mixing it during training. This helps to ensure that the results are replicable and trustworthy.
- Cross-Validation: Stratified k-fold cross-validation is used to validate the model consistency. This allows to confirm that the model is not overfitting and maintains stability across different segments of the data.
- Baseline Comparisons: The fine-tuned BERT model is benchmarked against simpler models such as Naive Bayes and CNN with LSTM hybrids. However, the baseline models are computationally lighter, hence their performance drops

significantly especially on code-mixed and informal comments. This confirms the superiority of BERT in application.

- Performance Metrics Documentation: All training and testing results, including precision, recall, and F1 scores for each class, are logged and documented for transparency. This not only supports evaluation but also ensures that future enhancements can be compared against these results.
- Real-World Relevance: The tool's design and testing were framed around practical law enforcement needs. Report clarity, CLI usability, and the ability to detect threats in mixed language are continuously benchmarked against real-life communication styles found on Facebook and Instagram.

In conclusion, the tool is evaluated not only for raw machine learning performance but also for its practical impact in real-world forensic settings. By applying both academic and real-life standards, the system is proven to be reliable, scalable, and highly applicable to multilingual cybercrime investigations in Sri Lanka.

2.7. Chapter Summary

In summary, this chapter provided a detailed analysis of the cyberbullying landscape in Sri Lanka, the challenges faced in digital investigations, and the gaps in existing forensic tools especially concerning language limitations. It highlights the importance of developing a cost-effective, multilingual solution capable of analyzing both Sinhala and code-mixed content. By critically evaluating existing systems, datasets, and machine learning approaches, this review justifies the need for a localized Android forensic tool that not only extracts social media data but also detects cyberbullying with accuracy and local relevance to unique Sri Lankan context.

CHAPTER 03 : METHODOLOGY

3.1. Chapter Overview

This chapter outlines the methodology used to design, develop, and evaluate the proposed multilingual Android forensic tool, focusing on detection of cyberbullying through the identification of hate speech on social media platforms. This tool is developed to analyze Sinhala and Sinhala-English code-mixed content which is commonly known as Singlish, from platforms like Facebook and Instagram. Furthermore, the tool addresses a critical requirement in Sri Lanka where the language diversity and limited digital forensic resources challenging the effective crime investigations.

Even though the primary objective of the tool is cyberbullying detection, its practical applications can be extended further. In real-world crime investigations, law enforcement often examines mobile devices and forensics of the suspects in order to uncover harmful behavior and patterns of violence. Therefore, in such cases, this tool can play a valuable role in identifying hate-filled and aggressive inappropriate communication, not only to cyberbullying but also to other forms of cybercrimes and even physical offenses. With that, implementing this automated detection tool and clear reporting of such content, the tool supports evidence gathering and strengthens the investigative process.

Moreover, this chapter continues to explain the research methodology used, followed by the system architecture, dataset collection, preprocessing strategies, model training, and tool implementation. It further outlines the libraries and frameworks used, the reasoning behind model choices, and the performance metrics used for the evaluation of the tool. Additionally, the overall methodology aims to ensure that the system is both technically accurate and practically usable in real-world scenarios and crime investigations in Sri Lanka.

3.2. Research Methodology

This research follows the structure of the *Research Onion* model proposed by Saunders et al., which helps to organize and justify the different methodological layers

of the project. A pragmatic and deductive approach was adopted as this study aims to develop a practical forensic tool that uses NLP to detect cyberbullying in Sri Lankan social media. Furthermore, the following table outlines each layer of the methodology with an explanation on what was used and the reason for using this specific research context.

Layer	Used Approach	Justifications
Philosophy	Pragmatism	This study targets the practical problem of detecting cyberbullying on Sri Lankan social media using a flexible, outcome-oriented approach. Pragmatism allows the combination of quantitative analysis like model performance and qualitative needs like local language context, supporting a real-world solution that balances theory and practical implementation.
Approach	Deductive	The project starts with existing theories and tools in natural language processing like BERT and applies them to a specific, underexplored context like Sinhala and Sinhala-English code-mixed hate speech detection. Moreover, it tests how well these existing models work in this new scenario and builds a tool around the observed outcomes, making deduction a suitable approach.
Methodological Choice	Quantitative with elements of applied experimentation	The research primarily relies on measurable results like accuracy, precision, recall and F1-score to evaluate model and tool performance. Although it is not research with a standard combination of methods, real-world applications like CLI usability and language handling are evaluated using iterative development and observed functionality, which adds qualitative value.

Strategy	Experimentation	BERT-based models are trained, tested, and fine-tuned on labeled hate speech datasets in order to conduct experimental evaluations. Additionally, the strategy further includes integrating model output into a command-line forensic tool and testing its results in a simulated forensic environment. Furthermore, this approach helps demonstrate real-world readiness.
Time Horizon	Cross-sectional	The project evaluates the model performance and system functionality based on the data collected and tested during a fixed time duration. Furthermore, it concentrates on developing and assessing the tool in its present form, without any continuous retraining or longitudinal tracking.
Data Collection and Analysis	Two annotated datasets: Sinhala Hate Speech Dataset & Code-Mixed Hate Speech Dataset Analysis metrics: Accuracy, Precision, Recall, F1-Score	These datasets were chosen based on their relevance to real-world Sri Lankan social media usage patterns. These present a good foundation for developing a hate speech detection algorithm that adapts to the local problem. Model evaluation is carried out using standardized categorization measures to provide objective and comparable findings.

Table 2 - Chapter 3: Table of Research Methodologies

3.3. Development Methodology

This development methodology section outlines how the proposed multilingual Android forensic tool is developed, from initial requirements gathering to system design and execution. Moreover, this methodology aims to meet the unique requirements of cybercrime investigations in Sri Lanka, with a focus on detecting cyberbullying through hate speech analysis in Sinhala and code-mixed Sinhala-English media. The development method is directed by a combination of survey feedback, tool analysis, and iterative testing, ensuring that the tool was both technically accurate and feasible for non-technical users like law enforcement agents.

3.3.1. Requirement Elicitation Methodology

For this project literature review, tool review and general survey was used for requirement identification to ensure that the final solution was grounded in real-world needs and challenges.

- **Literature and Tool Review** – The technical evaluation of the existing tools such as Cellebrite, Oxygen Forensics, Magnet AXIOM and insights drawn from academic literature review of government cybercrime reports, existing researches, approaches and frameworks were used as a primary sources of data collection to identify the unique requirements in Sri Lanka. Even though the above-mentioned set of existing tools are highly effective at extracting raw data from mobile devices, yet they have limited capabilities to support for analyzing content written in Sinhala and Sinhala-English code-mixed content in Social Media. Furthermore, their high cost and technical complexity makes them inappropriate for widespread adoption in Sri Lanka, especially within public law enforcement divisions.
- **A Survey** - A general survey was conducted targeting Sri Lankan active social media users. The goal was to understand their experiences with cyberbullying, the languages they commonly use online, and their awareness of existing digital safety tools. However, the majority of respondents indicated they frequently encounter cyberbullying and hate speech from the two languages of Sinhala Unicode and Sinhala-English code-mixed language, especially on Facebook and Instagram platforms. Furthermore, many participants reported witnessing online harassment and hate speech but were unaware of how to report such incidents and if any tools existed to support law enforcement. These responses validated the need for a tool that could detect harmful content in both Sinhala and Sinhala-English code-mixed formats and present the results in a simple, legally usable format.

In conclusion, by combining user feedback with an evaluation of current limitations, the following core requirements were identified:

- The tool must support text extraction from Android devices, specifically from Facebook and Instagram.

- It must detect hate speech in Sinhala Unicode and Sinhala-English code-mixed formats.
- It should present results in a PDF report format that is structured which can be legally presented in investigations as crime evidence.
- The tool must be simple to use, ideally through a Command-Line Interface (CLI).
- It should be lightweight and operable offline, especially in resource-limited environments.

These requirements formed the foundation for system design, model training, and interface development described in the following sections.

3.3.2. Design Methodology

The design methodology of the proposed forensic tool was guided by the core requirements identified through research survey and literature, technical review. Moreover, the system is designed to be modular, lightweight, and compatible with the practical requirements of the digital forensic investigators in Sri Lanka. Moreover, the design process follows a layered structure, to ensure that each module of the tool is independently developed, tested, and improved without affecting the rest of the tool.

Major Functional Layers	Description
Data Extraction Layer	This layer connects to an Android device through ADB (Android Debug Bridge) to extract text-based data from Facebook and Instagram applications. The extraction process targets key directories and databases where chat logs and message content are stored, packaging them into .tar files for further analysis.
Preprocessing Layer	Extracted raw texts are always unstructured and jumbled, especially in informal and code-mixed messages. Therefore, this preprocessing module cleans tokenizes and normalizes the content with a special attention to handle Sinhala Unicode text, transliterated code-mixed Sinhala-English content.

Classification Layer	This is the core intelligence of the tool, which is built around a BERT-based multilingual classifier model, trained and fine-tuned on real-world Sri Lankan data to identify hate speech on social media. However, each text is passed through this layer and is classified as Hate or Non-Hate.
Report Generation Layer	The results from the classifier are compiled into a structured PDF report. This report includes the flagged messages, prediction labels, timestamps, and minimal metadata required for investigative use. Furthermore, the pdf format was designed to be simple, readable, and legally usable even by non-technical personnel.
CLI Interface Layer	Instead of a graphical user interface, the tool uses a Command-Line Interface (CLI) to allow investigators to operate the system easily from any terminal. This keeps the tool lightweight and ensures compatibility with forensic environments with limited resources and lack GUI support.

Table 3 - Chapter 3: Table of Research Methodologies

This layered design ensures that each component remains independent and scalable, making it easier for debugging, testing, and future upgrades such as expanding the model to support more languages, platforms, and additional forensic features like real-time analysis modules. Furthermore, isolating responsibilities, this layered architecture ensures robustness, flexibility, and adaptability to evolving forensic challenges in cybercrime investigations.

3.3.3. Programming Paradigm

Python programming language is used to develop the tool, following a modular and functional programming paradigm which is a simple organized structure where each part of the tool is built separately. This means that tasks like extracting data, cleaning text, detecting hate speech, and creating reports were all handled by using different functions in the code which makes it easier to test, update, and fix specific parts without

affecting the whole tool. With that rather than being spread across multiple loosely connected scripts, the core logic of the tool is encapsulated within two main scripts;

- "*hawkeye_main.py*" handling the Android data extraction, database parsing, preprocessing, hate speech detection, and CSV generation.
- "*report_pdf_structured.py*" focusing only on report generation, which is a structured PDF containing charts, summaries, and flagged hate content.

Moreover, in this main script of "*hawkeye_main.py*", each functional module is implemented as an individual function within the script including a clean separation the key processes of the tool such as connecting and extracting data from android devices via ADB, preprocessing raw text data including Sinhala Unicode and code-mixed Sinhala-English, feeding the extracted text into the fine-tuned BERT-based hate speech detection model, generating SHA-256 hash reports for forensic integrity, saving the results in csv and finally calling out the "*report_pdf_structured.py*" script. This script also uses separate functions to structure the generated results pdf. Furthermore, this modular and functional programming paradigm ensures that each function is testable and independently debuggable, the code is easy to maintain, extend and new platforms or adding features with minimal interference to the existing logic

Additionally, instead of using a complex graphical user interface (GUI), the tool was designed to work through the command line interface (CLI), where users are capable of running the tool using simple commands. This approach is ideal as it is lightweight, resource-efficient, easygoing, and user-friendly for the investigators without much technical knowledge and most of all it directly aligns with forensic environments where the terminal access is preferred.

As for the machine learning model, the hate speech detection model was trained using labeled data and standard Python libraries like PyTorch and Hugging Face Transformers. The model was fine-tuned on real-world Sinhala and Singlish datasets to improve its performance in Sri Lankan social media contexts. Furthermore, the scripts to train the model and fine tuning the model were written in a clean, straightforward style to keep everything simple and maintainable.

Finally, the pdf report generator was developed using Python libraries like FPDF for creating structured multi-page documents and Matplotlib for visual charts like bar graphs and pie charts. Moreover, this part of the tool was scripted on a separate script in order to maintain the cleanliness of the code for easier updates in the future. Overall, the development approach reflects a practical, easily extensible, and investigator-friendly approach, designed with real-world constraints and users in mind.

3.3.4. Evaluation Methodology

The evaluation of the proposed tool's effectiveness is the key part of this project as the main goal of this tool is to detect cyberbullying through hate speech analysis, the evaluation focused on how accurately and reliably the system capable of identifying harmful content in social media platforms. Moreover, the performance of the model is measured using common classification metrics like accuracy, precision, recall, and F1-score which helps to determine accuracy of the tool in differentiating hate speech from normal messages.

However, after training the BERT-based model using two datasets of Sinhala hate speech and Sinhala-English code-mixed text, the tool is tested on unseen data to check the accuracy of its generalization. Moreover, a confusion matrix is used to visualize correct and incorrect predictions by giving a clearer picture on the strengths and weaknesses of the model. In addition to testing the model, the usability of the tool itself is evaluated by running it through practical test cases using Facebook and Instagram data extracted from test Android devices. However, here the focus is to check the efficiency, usability and the user friendliness of the CLI interface, the accuracy and the quality of the reports, and the reliability of the tool working in an offline or low-resource environment.

In conclusion the combination of these evaluation steps helps to ensure that the tool is not just a technical approach but also practical and effective approach for real-world crime and cybercrime investigations in Sri Lanka.

3.3.5. Solution Methodology

As mentioned earlier, the core solution implemented through this research project is a command-line-based android forensic tool that detects cyberbullying by

analyzing media on social media platforms for hate speech like comments, messages, captions etc. Moreover, this tool focuses on content written in Sinhala and Sinhala-English code-mixed formats which are two common languages among Sri Lankan social media users.

Therefore, in order to fulfil this requirement, a BERT-based machine learning model is fine-tuned using two datasets of Sinhala hate speech and a code-mixed Sinhala-English hate speech. Moreover, these datasets are preprocessed to clean the text, and format it in a way that the model cab learn from. However, the trained model is considered as the heart of the tool's detection system.

Furthermore, the tool connects to the Android device through ADB (Android Debug Bridge) to extract data from Facebook and Instagram applications and once the data is collected, it is passed through the BERT classifier, which identifies whether each message contains hate speech. Finally, the results are presented in a PDF report, which includes flagged messages, their timestamps, and classifications, making it easy for law enforcement officers to use during investigations and use as legal evidence in courts.

In conclusion, by combining natural language processing (NLP) with basic digital forensic techniques, the tool provides a lightweight, localized solution for this growing problem making it a practical choice for investigators working with limited resources but facing serious cybercrime threats.

3.4. Project Management Methodology

Managing this project requires a structured and flexible approach to ensure that each phase from research and design to implementation and evaluation progressed smoothly and stayed aligned with academic milestones. However, a simplified, Agile-inspired development model was followed, allowing iterative development, continuous improvement, and timely adaptations based on feedback from the supervisor and evolving project goals.

The entire project was divided into five major stages:

- Requirement gathering and analysis
- Dataset collection and preprocessing

- BERT Model selection, fine-tuning, and testing
- Full development and testing of the CLI-based forensic tool
- Evaluation, structured report generation, and final documentation

Each phase is guided by clearly defined deliverables and soft deadlines, helping to maintain steady progress while allowing room for modifications. Frequent feedback from the project supervisor helped to refine the direction of the model development and future tool design. Moreover, key milestones such as dataset validation, model accuracy benchmarking, and successful CLI tool integration were completed within the timeline planned.

During the final tool development stage, a module-wise sprint planning was used to isolate and test individual components such as ADB data extraction, text preprocessing, model integration, and report generation before merging all of those codes into two Python scripts. Following this approach not only minimized the complexity in debugging but also ensured high reliability across individual modules. However, the final product is a fully working CLI based forensic tool named HawkEye, which is capable of extracting social media data from Android devices, detecting hate speech in Sinhala and code-mixed content using a trained BERT model, and generating structured PDF reports for the usage of the law enforcement institutions and personnel.

3.4.1. Project Scope

As mentioned before, this project is aimed to develop a multilingual forensic investigation tool to assist Sri Lankan law enforcement in detecting cyberbullying through hate speech analysis on social media and the primary focus is on analyzing text content in Sinhala, and Sinhala-English code-mixed formats which are the languages commonly used by Sri Lankan youth on platforms like Facebook and Instagram.

The scope of the project is divided into two main phases:

- Phase 1 - Completed;
    - Fine-tuned a BERT-based model using Sinhala and code-mixed datasets.
    - Evaluated model accuracy and prepared preprocessing pipeline.

- Implemented dataset handling, tokenization, and benchmarking.

- Phase 2 - Completed

  - Developed the full CLI tool using python programming language "hawkeye_main.py".
  - Integrated ADB-based Android data extraction for Facebook , Instagram and Messenger (Messenger application was out of scope but due to the connectivity between Messenger and Facebook , extracting messenger data, detection of hate speech and reporting the results of the messenger application was also developed within this phase as extra).
  - Parsed SQLite databases, extracted text, and prepared CSVs.
  - Passed data through the trained model to classify hate speech.
  - Generated structured PDF reports using python libraries specialized for generating structured reports, "report_pdf_structured.py"

In-Scope Features

- Fine-tuning and evaluation of a BERT model for multilingual hate speech detection
- Use of Sinhala Unicode and code-mixed Sinhala-English datasets
- CLI-based tool structure for forensic use.
- Basic Android data extraction from Facebook/Instagram using ADB.
- Text classification and flagging of hate speech content
- Generation of structured PDF reports with summary charts.
- Lightweight CLI-based user-friendly interface for investigators

Out of Scope Features (Future Expansions)

- Advanced forensic capabilities (e.g., password bypass, app decryption).
- Support for iOS or non-Android platforms.
- Graphical user interface (GUI).
- Expand for other Social Media Applications.
- Detection of other forms of cyberbullying beyond hate speech.

3.4.2. Schedule

The Gantt charts below outline the planned timeline for each major task in the project, divided into two key phases, the midpoint phase which is highlighted in blue and the final development phase which is highlighted in green are both completed. The midpoint phase is focused on research, model development, and structuring the initial chapters of the report. These tasks, including the training and evaluation of the BERT-based hate speech detection model, were completed as scheduled.

The final phase includes the development of the full CLI-based forensic tool, PDF report generation, Android data extraction, and the remaining report chapters. The structured scheduling of individual tool components and report chapters ensured successful completion and submission of the project.

This timeline reflects a realistic, well-phased workflow balanced both technical development and academic documentation, helping to maintain consistent progress and complete the project successfully on time. Below are the two Gant charts prepared within the two phases, the first Gant chart was prepared during the midpoint phase, and the second chart was prepared during the final phase. However, both the chart clearly represents the timeline, workflow and the progress towards the final submission of the complete tool. A bigger version of the chart is attached to the appendix moreover; the old Gant Chart of phase 1 is attached too.(view image)

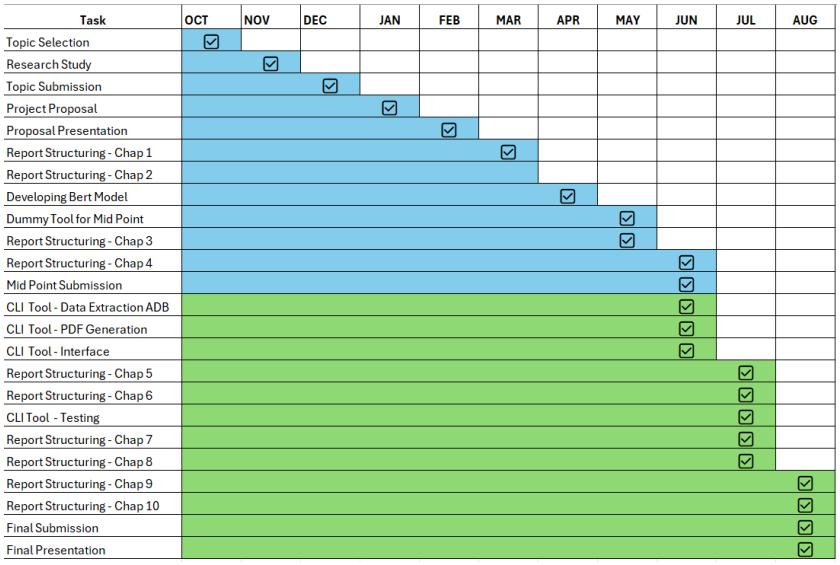


Figure 2 - Chapter 3: Schedule and Project Planning at the Final Submission

Deliverables	Dates
Project Proposal Document	12th December 2024
Proof of Concept (POC)	15th June 2025
Mid-Point Assessment	15th June 2025
Final Research Project	30th August 2025
Final Viva Presentation	1st – 15th September 2025

Table 4 - Chapter 3 : Deliverables and Dates

3.5. Resource Requirements

To successfully develop and evaluate the proposed multilingual forensic tool, several resources including both hardware and software are required. Moreover, the project further relies on academic support and publicly available datasets to meet technical and domain-specific requirements.

Human Resources	
Researcher/Developer	Responsible for literature review, dataset collection, model development, evaluation, and documentation basically the whole project.
Supervisor/Mentor	Provides regular guidance, feedback, and helps to refine the scope and methodology.
Software Resources	
Python	Main programming language is used for model development and tool integration.
Operating System	Windows 11
VS Code	Development environments used for model training, testing, and experimentation.
PyTorch and Transformers (Hugging Face)	Libraries are used for fine-tuning and evaluating the BERT model.
Pandas, NumPy	Data handling and preprocessing.

FPDF, Matplotlib	For the generation structured PDF reports with multiple pages.
ADB (Android Debug Bridge)	For extracting social media data from Android devices.
VM Ware	For testing ADB-based data extraction from Facebook and Instagram. Used an Android device virtual machine for testing.
Hardware Resources	
Personal Laptop	Intel Core i7 13th Gen 16GB RAM NVIDIA RTX 4060 GPU Used for model training and development
Dataset Resources	
Sinhala Hate Speech Dataset (sinhala-hate-speech-dataset new.csv)	Used to train and evaluate the BERT model for accurate hate speech classification.
Code-Mixed Sinhala-English Dataset (singlish_dataset1.csv + singlish_dataset2.csv)	

Table 5 - Chapter 3: Table of Resource Requirements

These resources were carefully selected to balance effectiveness, accessibility, and cost. Open-source tools and publicly available datasets allowed for efficient development without incurring licensing fees, making the tool practical and scalable for use in Sri Lanka.

3.6. Risks and Mitigation

Like any software and research-based project, the development of this forensic tool came with several risks, especially given the integration of machine learning, multilingual NLP, and forensic investigation workflows. The table below outlines the key risks identified, their potential impact, and the mitigation strategies applied or planned.

Risk	Impact	Likelihood	Mitigation Strategy
Limited availability of Sinhala and code-mixed datasets	High	Medium	Two datasets were merged and preprocessed to improve data quality and balance.
Model overfitting or low generalization	Medium	Medium	Used dropout, validation split, and performance metrics like F1-score for tuning.
Delay in CLI/tool implementation	High	High	Focused on completing model first; tool development scheduled as Phase 2 deliverable.
Difficulty in accessing real Android device data	Medium	Medium	Planned to use sample exports or emulator-based test data if device access is limited.
Complexity of handling code-mixed text patterns	High	Medium	Applied robust preprocessing and used a transformer model suited for mixed languages.
Limited hardware resources for large-scale model training	Medium	Low	Optimized batch size and model layers; used GPU where possible.
Unfamiliarity with ADB or Android extraction process	Medium	High	Planned to learn period and testing with dummy device data before full integration.
Time constraints with academic deadlines	High	High	Prioritized core features (model development); tool components planned in phases.

Table 6 - Chapter 3: Table of Risk Analysis and Mitigation

3.7. Chapter Summary

This chapter outlines the research and development methodology use to develop a multilingual forensic tool to detecting cyberbullying in Sri Lanka. Firstly, the research philosophy and approach based on the Research Onion model is elaborated, emphasizing a pragmatic and deductive strategy supported by quantitative evaluation. Following that the chapter further details how requirements were collected through both

user survey and analysis of existing tools, building up the foundation for system design and planning.

Furthermore, the development methodology of the tool which is focused on developing a hate speech detection model using a fine-tuned BERT classifier, trained on Sinhala and Sinhala-English code-mixed datasets. Moreover, this chapter reflects that the full CLI-based forensic tool "**Hawkeye"** has been successfully implemented within the timeline, integrating all core functionalities including Android data extraction, text classification hates speech detection, and structured PDF reporting.

Finally, the chapter further covers the project management strategies, including scope definition, timeline planning, resource allocation, and risk mitigation. In conclusion, with the completion of the full working tool all primary development goals are met within the deadlines assigned.

CHAPTER 04 : SOFTWARE REQUIREMENTS SPECIFICATION

4.1. Chapter Overview

This chapter presents the Software Requirements Specification (SRS) for the proposed multilingual Android forensic tool aimed at detecting cyberbullying through hate speech analysis in Sinhala and Sinhala-English code-mixed content. The chapter outlines the purpose of the system, target users, functional expectations, and non-functional requirements. Furthermore, it includes visual representations such as a rich picture, stakeholder models, use case diagrams, and prioritization tables to elaborate both the scope and the intended functionality of the system. Additionally, the content is based on insights gathered from literature, user surveys, and stakeholder evaluation to ensure the tool meets the real-world forensic requirements in the Sri Lankan context.

4.2. Rich Picture

The rich picture below illustrates the real-world context in which the proposed multilingual Android forensic tool will operate. Moreover, it visually represents the relationships between key stakeholders involved in a cyberbullying investigation, the core problem of online abuse, and how the tool integrates into the existing forensic process.

Firstly, the diagram begins with the victim, who experiences emotional distress due to bullying and hates speech on social media platforms like Facebook and Instagram. Upon reporting the incident to the police, the prime suspect, the criminal is identified and taken into custody. The suspect’s mobile device is seized and handed over to the cybercrime investigation unit.

At this stage, the proposed forensic tool comes into play. It is designed to extract mobile social media data manually through ADB, analyze text-based content, and detect hate speech, particularly in Sinhala and Sinhala-English code-mixed formats. Once the tool processes the data, it generates a structured report highlighting detected hate speech instances.

Finally, this report is then passed back to the police, supporting them with clear, contextual evidence that can be used in a legal investigation and a court case. The rich picture highlights how the tool fills a critical gap in existing cybercrime workflows, enabling more effective evidence gathering and decision-making in Sri Lankan forensic environments.

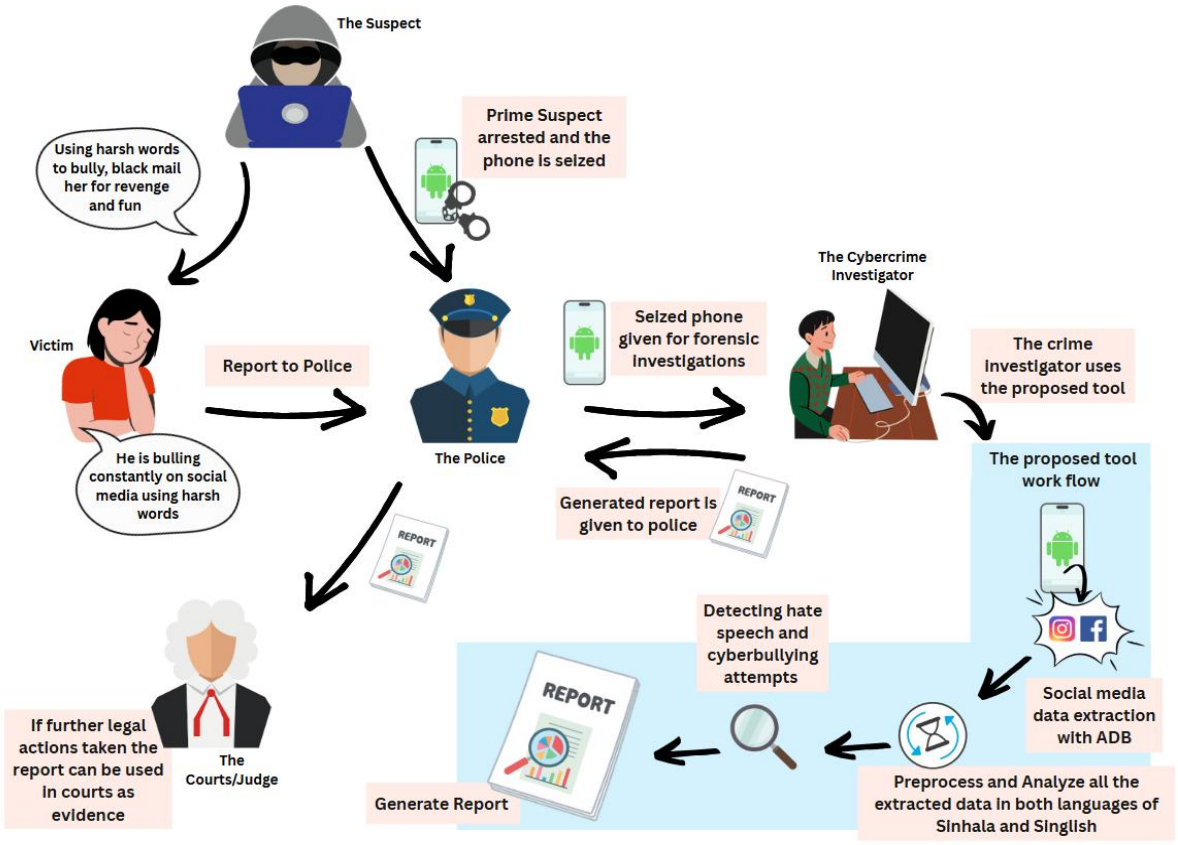


Figure 3 - Chapter 4: Rich Picture

4.3. Stakeholder Analysis

4.3.1. Stakeholder Description

Stakeholder	Stakeholder Type	Description
Cybercrime Investigator	Primary Operator	Direct user of the tool who is responsible for running the analysis, interpreting results, and preparing evidence for legal proceedings.

Police Officer	Support Operator	Acts as the first responder in collecting digital devices from suspects and coordinates with cybercrime units for forensic analysis.
Victim	Affected Individual	The target of online harassment or bullying and indirectly benefits from the investigation and evidence collected using the tool.
Criminal/Suspect	Subject of Investigation	The individual is under investigation for cyberbullying. Their device is analyzed using the tool.
Court/Judge	Decision-Maker	Use the forensic report produced by the tool as part of legal proceedings and evidence evaluation.
General Public	External Stakeholder	Indirectly impacted by the success of such tools in reducing cyberbullying and improving online safety.
Social Media Platforms	External Data Source	Platforms from which communication data is extracted and analyzed by the tool.

Table 7 - Chapter 4; Table of Stakeholder Descriptions

4.3.2. Stakeholder Onion Model

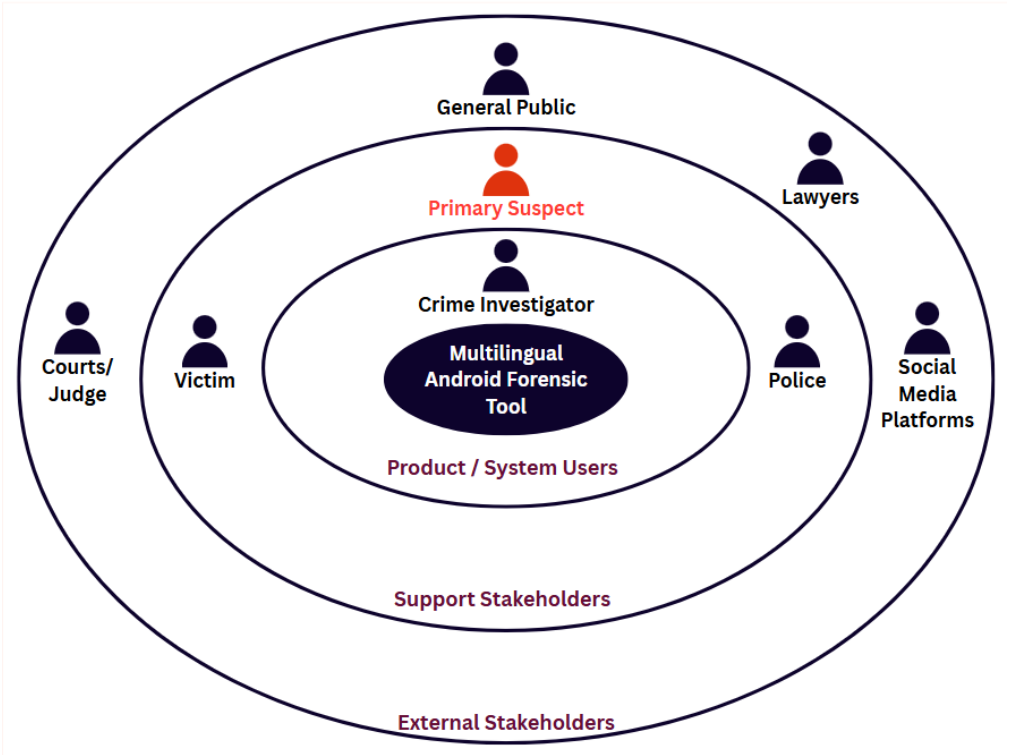


Figure 4 - Chapter 4: Stakeholder Onion Model

4.4. Requirement Elicitation Methods

Selected Method	Justification
Literature Review	Conducting a literature review helped to understand the current landscape of existing tools, technologies, and research gaps. Moreover, this method supported the technical direction of the project by justifying the need for a localized, multilingual forensic tool in Sri Lanka.
Public Survey	The survey allowed for gathering real-world input from a diverse set of potential social media users of all sorts and it helped to identify user needs, expectations, and crucial points related to cyberbullying and digital forensics in the Sri Lankan context.

Table 8 - Chapter 4: Table of Requirement Elicitation Methods

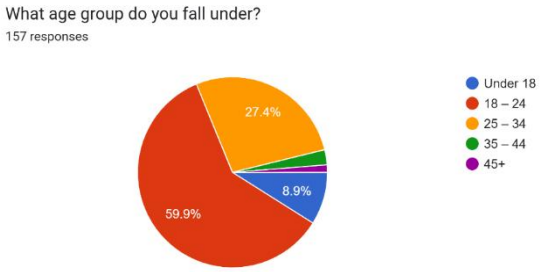
4.5. Discussion of Results

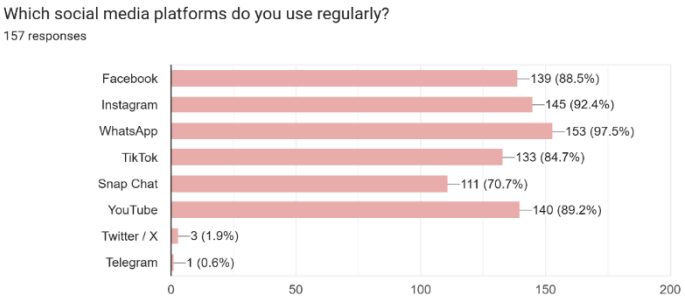
4.5.1. Literature Review

Citation(s)	Findings
Abu Hweidi et al. (2023), Chang & Yen (2020)	Most forensic tools used in Sri Lanka are expensive, foreign-developed, and do not support Sinhala or code-mixed content, creating a gap in local investigation capabilities.
Ruwandika & Weerasinghe (2018), Ishara & Jayalal (2020)	Traditional ML models like SVM and logistic regression underperform with code-mixed Sinhala text and struggle to generalize.
Muthuthanthri & Smith (2024), Fernando & Deng (2023)	BERT and transformer-based models significantly improve hate speech detection accuracy, especially in multilingual or code-mixed contexts.
Jayasinghe et al. (2024), Kemp (2024)	Sri Lankan youth are the most vulnerable to cyberbullying on platforms like Facebook and Instagram, often in Sinhala-English hybrid language.
Asim Mubarik et al. (2021), Ogunleye & Dharmaraj (2023)	There is a lack of integrated tools that combine data extraction, NLP-based analysis, and structured evidence reporting in a forensic workflow.

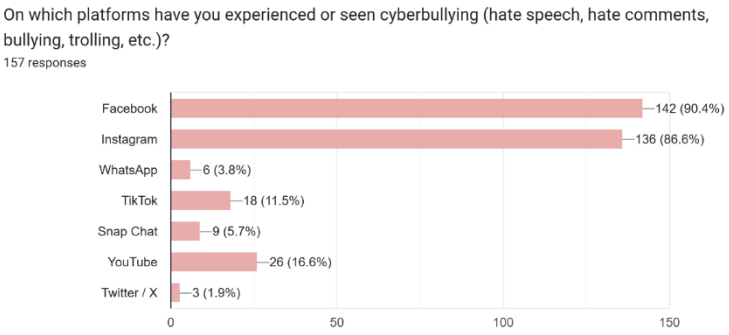
Table 9 - Chapter 4: Table of Literature Review Findings

4.5.2. Survey Findings

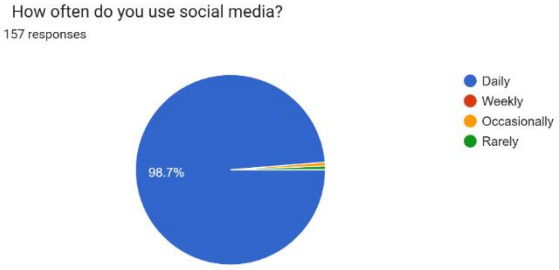
Question	**What age group do you fall under?**
Purpose	The purpose of this question was to identify the dominant age demographic of social media users in Sri Lanka who may be exposed to, witnessed or affected by cyberbullying. Understanding the age group distribution helps ensure that the tool is designed to address the requirements of the most vulnerable and most active users online.
	
Discussion	Based on the 157 responses collected, it is highlighted that the majority (59.9%) of the participants fall within the 18–24 age range, followed by 27.4% in the 25–34 age range. This clearly shows that young adults are the most active on social media and are therefore most likely to encounter and be affected by cyberbullying. Moreover, the respondents under 18 (8.9%) and above 34 are very less compared to the other two age groups, confirming that the primary users remain as the youth and young adults. Furthermore, these findings directly support the decision to focus the tool on platforms like Facebook and Instagram, where these age groups are highly active. It also pops out the fact that it is importance addressing this issue as these social media platforms are commonly used by younger users in Sri Lanka.
Question	**Which social media platforms do you use regularly?**
Purpose	The aim of this question was to identify which social media platforms are most commonly used in Sri Lanka. Moreover, this helps to determine which platforms should be prioritized during data extraction and tool compatibility design, ensuring the tool is developed around platforms where cyberbullying is most likely to occur.

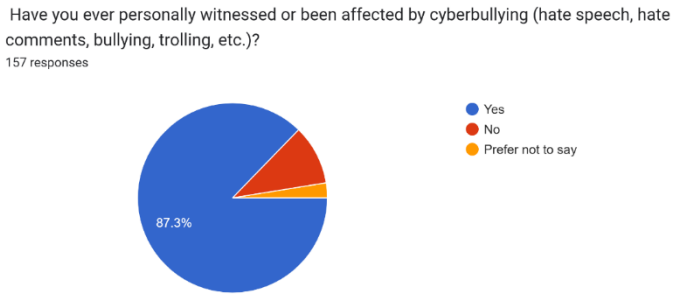


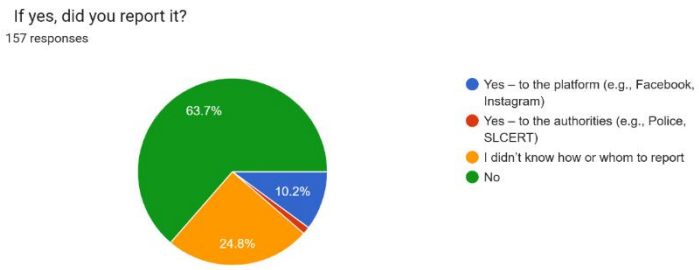
Discussion	The results show that WhatsApp (97.5%), Instagram (92.4%), and YouTube (89.2%) are the most commonly used platforms among respondents, followed closely by Facebook (88.5%) and TikTok (84.7%). Platforms like Snapchat (70.7%) also show notable usage, while Twitter/X (1.9%) and Telegram (0.6%) have very low engagement among this group of people. However, these findings strongly validate the decision to focus the forensic tool on platforms like Facebook and Instagram, where both message-based and public interaction-based bullying are frequent. Even though WhatsApp is ranked the highest, it involves more private encrypted messaging, which requires different extraction techniques and legal considerations which are outside the scope of this project. Twitter and Telegram are not a development priority due to their low usage in this demographic. Finally, the results help to strengthen relevance of the chosen platforms in the project scope and ensure that the tool focuses on real-world, highly used environments.
Question	**On which platforms have you experienced or seen cyberbullying (hate speech, hate comments, bullying, trolling, etc.)?**
Purpose	This question was designed to identify which social media platforms are most frequently associated with cyberbullying incidents. Understanding this is crucial and it helps the project focus its data extraction and hate speech detection efforts on the most relevant platforms where harmful content is most commonly encountered.



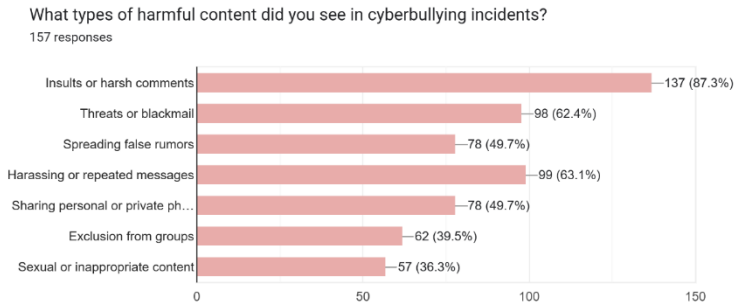
Discussion	Out of 157 respondents, a striking 90.4% reported experiencing or witnessing cyberbullying on Facebook, followed closely by Instagram (86.6%). This clearly confirms that these two platforms are the primary digital spaces where online abuse is most visible among the surveyed group. Moreover, other platforms, like YouTube (16.6%) and TikTok (11.5%), show some reports of cyberbullying but with comparatively lower rates. Furthermore, WhatsApp (3.8%), Snapchat (5.7%), and Twitter/X (1.9%) were reported as much less common spaces for visible bullying in this survey. However, these findings further support the decision to prioritize Facebook and Instagram in the development of the tool, especially when designing modules for data extraction, classification, and report generation. The results further support the inclusion of features that detect various forms of hate speech, trolling, and harassment, which appear to be widespread on these platforms.
Question	**How often do you use social media?**
Purpose	This question aims to determine the frequency of social media usage among respondents and understanding the usage patterns helps to justify the requirement for real-time forensic investigation tools and supports the decision to prioritize social media platforms where users are highly active.



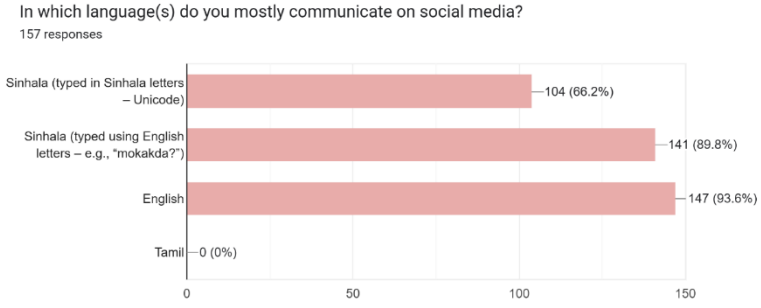
Discussion	Out of the 157 participants, a solid 98.7% reported using social media daily. Only a very small percentage indicated weekly, occasional, and rare usage. This result clearly highlights that social media is an integral part of daily life for the majority of the surveyed population. The high level of daily engagement highlights that the risk of encountering cyberbullying is also frequent and ongoing. Additionally, this finding validates the choice to focus on platforms like Facebook and Instagram, which are checked and used daily by most users. It further elaborates the urgency of implementing systems that helps the investigators to quickly identify harmful interactions in a timely manner.
Question	**Have you ever personally witnessed or been affected by cyberbullying (hate speech, hate comments, bullying, trolling, etc.)?**
Purpose	The purpose of this question is to measure how many individuals have direct or indirect experience with cyberbullying and helps to validate the real-world relevance of the problem which the proposed forensic tool is trying to solve.
	
Discussion	According to the responses, 87.3% of participants reported that they have either personally experienced or witnessed some form of cyberbullying, while only a small percentage responded “No” and chose “Prefer not to say”. This clearly points out that cyberbullying is widespread and affects a large majority of social media users in Sri Lanka. Moreover, the result strongly justifies the need for a tool that is capable of detecting hate speech and abusive language, especially in Sinhala and code-mixed formats. Furthermore, it supports the idea that such a tool could have a real social impact by helping victims seek justice and enabling investigators to gather evidence more

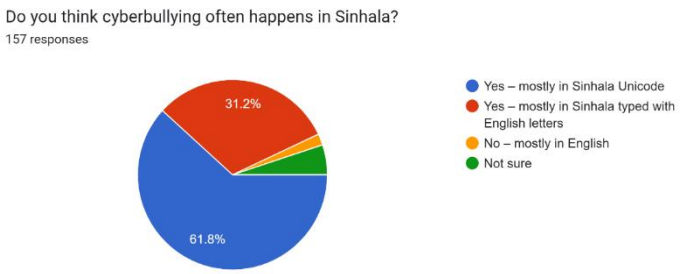
	effectively. However, the finding strengthens the urgency and importance of developing localized solutions to address this issue.
Question	**If yes, did you report it?**
Purpose	This question aims to understand how the victims or witnesses of cyberbullying respond to incidents and whether they take any action. Additionally, it further reveals the awareness and accessibility of the existing reporting mechanisms, helping to determine whether users trust the current systems in place.
	
Discussion	Among the 157 respondents, a significant 63.7% admitted that they did not report the cyberbullying incident at all. Meanwhile, 24.8% indicated that they did not know how or whom to report to, highlighting a major gap in awareness. With all that only a small portion reported the incident to the platform itself (10.2%), and an even smaller percentage went to the authorities (1.3%). Moreover, this result is highly crucial, as it reflects both a lack of reporting and a disconnect between social media users and available support systems. It point outs that despite the high rate of cyberbullying exposure, most victims or witnesses feel unsure, helpless, and unwilling to report. However, for this project, this finding supports the need to develop a forensic tool empowers authorities by giving them clearer, locally understandable evidence of abuse, especially since users themselves are often not taking direct action. It further illustrates the possible use of such tools to fill in the reporting gap by conducting the back-end investigations when front-end reporting fails.
Question	**If you didn't report it, why not?**

Purpose	This question is to further clarify the reasons behind hesitation or refusal of the users to report cyberbullying incidents. It helps to identify the emotional, social, and practical barriers that prevent users from seeking help, which in turn supports the development of backend tools that enable authorities to intervene even when cases go unreported.
	
Discussion	Out of the few responses collected, common themes included uncertainty, fear of involvement, feeling it was not serious enough, and believing there was no point. Specific reasons included: "I didn't know what to do." "Didn't want to get in unnecessary trouble." "No point." "Didn't think of it much because I was not bullied, although I witnessed it." These answers highlight a pattern of low confidence in reporting and fear of retaliation and social conflict. It further reflects a lack of awareness about support channels and legal protections.
Question	**What types of harmful content did you see in cyberbullying incidents?**
Purpose	This question is designed to identify the most common forms of harmful and abusive content experienced or witnessed by active social media users. Moreover, these findings are critical for understanding what kinds of hate speech and bullying behaviors the tool needs to detect and categorize effectively.



Discussion	These responses reveal that the most frequently seen form of harmful content was "insults or harsh comments" (87.3%), followed by "harassing or repeated messages" (63.1%), and "threats or blackmail" (62.4%). A specific portion of users also reported exposure to spreading false rumors (49.7%), sharing of personal/private photos (49.7%), and exclusion from groups (39.5%). Lastly, 36.3% had seen sexual or inappropriate content. With that, these critical findings show that text-based abuse is by far the most common and damaging form of cyberbullying by validating the focus of the project on natural language processing (NLP) for hate speech detection. Additionally, it further suggests that any future expansions of the tool may need to consider other aspects of bullying, such as media-sharing and group manipulation, though these are currently out of scope. However, the results directly support the need for a system capable of recognizing toxic language, verbal violence across Sinhala and code-mixed texts.
Question	**In which language(s) do you mostly communicate on social media?**
Purpose	This question was asked to determine the most commonly used languages and input formats in online communication among Sri Lankan social media users. This helps define the language scope of the model and supports the need for multilingual and code-mixed language processing in the tool.



Discussion	Out of the157 responses recorded 93.6% communicates in English, 89.8% use Sinhala typed using English letters (commonly known as Sinhala-English code-mixed or "Singlish"), 66.2% use Sinhala Unicode (typed in Sinhala letters) and 0% selected Tamil. Additionally, these findings confirm the widespread use of code-mixed Sinhala-English interaction highlighting the importance of the tool supporting informal transliterated Sinhala content in addition to Unicode Sinhala and standard English. However, it further supports the decision to focus on Sinhala and code-mixed Sinhala-English as the primary language inputs for the BERT-based hate speech detection model. The lack of Tamil responses suggests that including Tamil language support is not immediately necessary for this version of the tool, though it could be considered for future expansion. Overall, the finding strongly justifies the multilingual focus and language preprocessing strategy defined in the system design.
Question 11	**Do you think cyberbullying often happens in Sinhala?**
Purpose	This question was asked to assess perceptions of the participants about the language usage nature on cyberbullying specially whether abusive behavior online frequently occurs in Sinhala, including Unicode or code-mixed formats. The aim was to validate the need for local language support in hate speech detection.
	
Discussion	According to the results, 61.8% of participants believe that cyberbullying mostly happens in Sinhala Unicode, while 31.2% believe that it occurs mainly in Sinhala typed with English letters (code-mixed format) meanwhile a very small percentage selected “mostly in English” (1.3%) and “not sure” (5.7%). With that the majority confirms that Sinhala is the most dominant language used for cyberbullying. Furthermore, the insight directly supports the core requirement of this project to detect hate speech in Sinhala and Sinhala-English code-mixed messages, which are often not supported

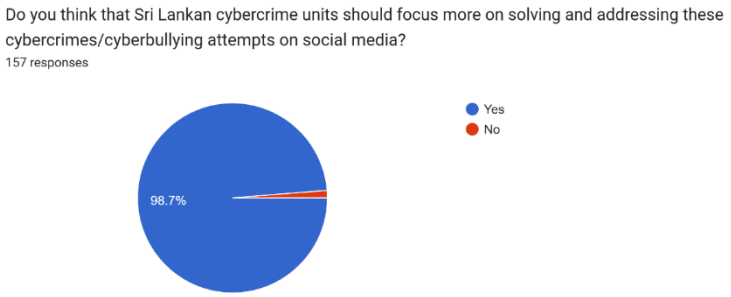
	by existing forensic tools. This finding strengthens the language scope of the project justifying valid reasons for choosing a multilingual NLP model and customized preprocessing steps tailored to handle both Unicode Sinhala and transliterated Sinhala.
Question 12	**Do you think that Sri Lankan cybercrime units should focus more on solving and addressing these cybercrimes/cyberbullying attempts on social media?**
Purpose	The purpose of this question was to assess public opinion on the role of Sri Lankan cybercrime units in tackling social media-based cyberbullying. This helps to validate the importance of developing a tool that directly assists law enforcement agencies in identifying and handling such cases more effectively.
	
Discussion	Out of 157 responses, a solid 98.7% of participants said "Yes", highlighting a public demand for stronger enforcement and investigation into cybercrimes, particularly on social media platforms. Meanwhile, only a very small minority (1.3%) disagreed. However, this high level of agreement strongly supports the relevance and urgency of the proposed project. Moreover, it highlights that users not only recognize the seriousness of online bullying but also expect proactive action from local authorities. Furthermore, the findings support the development of this tool as a direct enabler for cybercrime units, helping them to bridge the gaps in language support, data analysis, and digital reporting when dealing with Sinhala and code-mixed cyberbullying content.

Table 10 - Chapter 4: Table of Survey Findings

4.6. Summary Findings

Findings	Literature Review	Survey Observation
Most cyberbullying cases occur on Facebook and Instagram.	✅	✅
Sinhala and Sinhala-English code-mixed content are the most common languages in online communication	✅	✅
Existing forensic tools do not support Sinhala or Sinhala-English code-mixed text.	✅	
Majority of victims do not report cyberbullying due to fear, confusion, and lack of awareness		✅
Public expects stronger involvement from Sri Lankan cybercrime units in tackling cyberbullying.		✅
Hate speech is the most frequent form of bullying, followed by harassment and threats.	✅	✅
Transformer models like BERT show improved performance for multilingual hate speech detection.	✅	

Table 11 - Chapter 4: Table of Summary Findings

4.7. Context Diagram

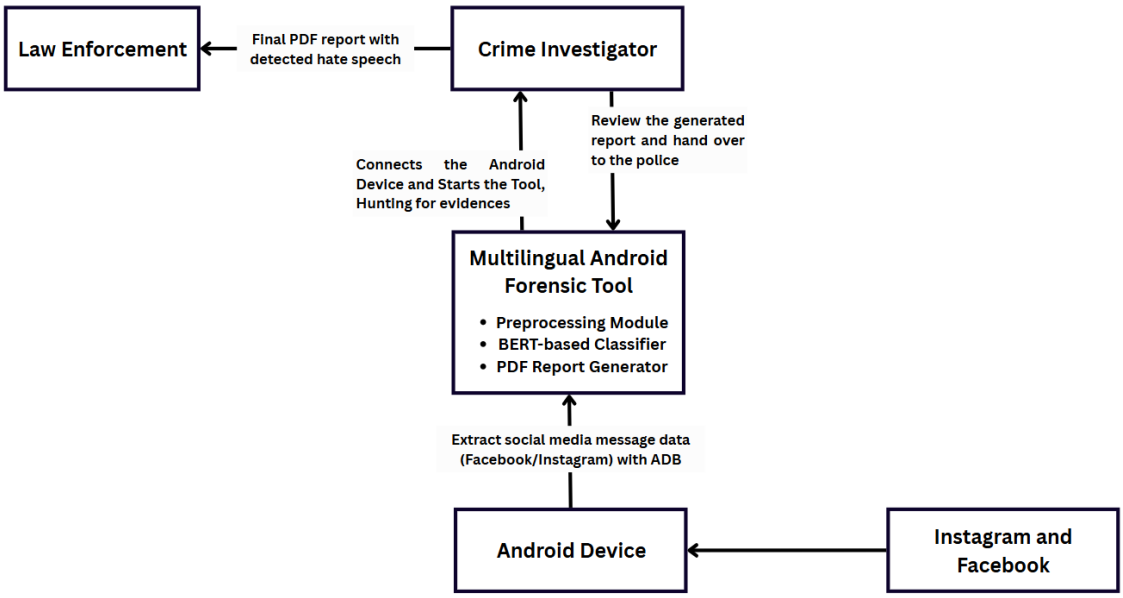


Figure 5 - Chapter 4: Context Diagram

This context diagram illustrates the interaction between the proposed system and its external entities. The crime investigator operates the tool by extracting social media data from an Android device using ADB. The tool internally preprocesses the data, classifies hate speech using a BERT-based model, and generates a structured PDF report. The investigator reviews this report and forwards it to law enforcement for further action. Instagram, Messenger and Facebook are indirectly involved as the original data sources stored on the device.

4.8. Use Case Diagram and Description

No	Use Case ID	Description
1	Use Case Name	Extract Social Media Data (ADB)
	Actor	Crime Investigator, Android Device
	Description	Extracts chat/message data from Facebook and Instagram stored on a suspect's Android device using ADB.
	Precondition	Android device is connected, and ADB is functional.
	Postcondition	Data is successfully extracted and saved for preprocessing.
	Main Flow	• Investigator connects Android device. • System initiates ADB session. • Chat database is located and pulled. • Data is stored locally for analysis.
2	Use Case Name	Preprocess Extracted Text
	Actor	System
	Description	Cleans and formats the raw message data (Sinhala Unicode and code-mixed text) for NLP processing.
	Precondition	Raw data extracted and accessible.
	Postcondition	Text is structured and ready for classification.
	Main Flow	• Read raw text files. • Clean symbols. • Normalize Sinhala and Singlish. • Save preprocessed data.
3	Use Case Name	Run Hate Speech Classification
	Actor	Crime Investigator

	Description	Runs the BERT model to detect hate speech in preprocessed messages.
	Precondition	Preprocessed data must be available.
	Postcondition	Messages are classified with hate or non-hate labels.
	Main Flow	• Investigator initiates classification. • Data passed to BERT model. • Predictions returned and saved.
4	Use Case Name	Generate Report (PDF)
	Actor	Crime Investigator
	Description	Generates a structured PDF report from classified data.
	Precondition	Classified messages are available.
	Postcondition	A printable, shareable report is created.
	Main Flow	• Investigator selects generate report. • System compiles flagged content. • PDF is formatted and saved.
5	Use Case Name	View or Review Results
	Actor	Crime Investigator
	Description	Investigator views and verifies flagged content before finalizing the report.
	Precondition	Classification must be completed.
	Postcondition	Verified dataset is saved for report generation.
	Main Flow	• System displays flagged messages. • Investigator reviews each message. • Adjust selections if needed. • Confirms for reporting.

Table 12 - Chapter 4: Table of Usecase Diagram Description

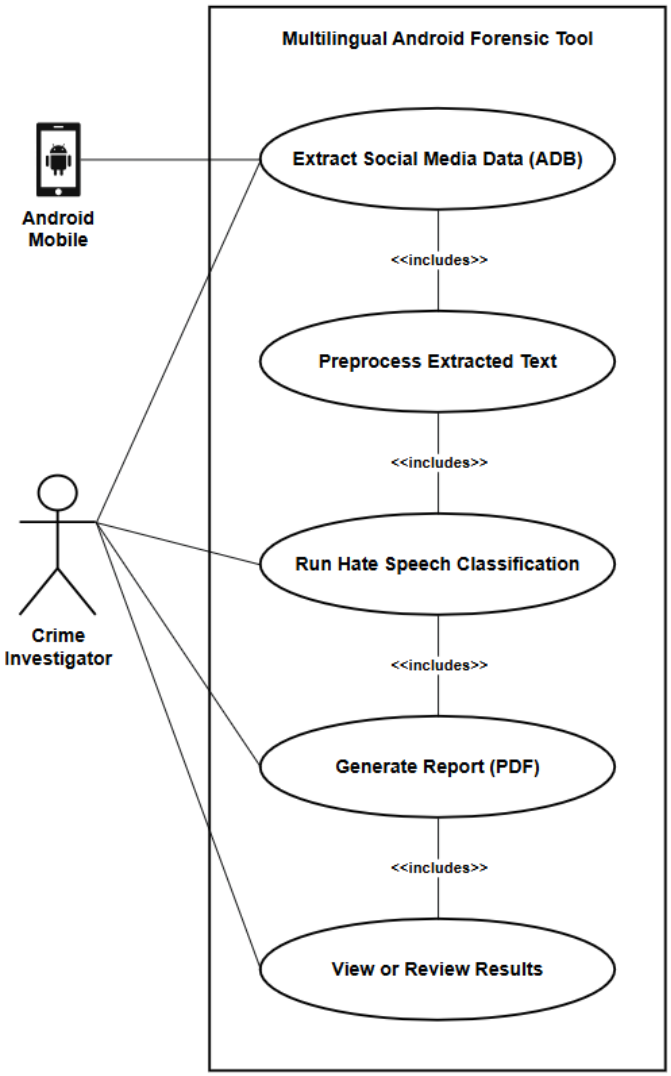


Figure 6 - Chapter 4: Usecase Diagram

4.9. Requirements

4.9.1. Prioritization

Must have	- Extract data from Android device (ADB) - Preprocess Unicode & code-mixed text - Run hate speech classification (BERT) - Generate PDF report with results
Should have	- Basic error messages/logs - Command-line progress feedback - Support for multi-file input

Could have	- Optional GUI layer - Export results in CSV format - Offline user manual/help flag
Will not Have	- iOS device support - Live social media monitoring - Multimedia (image/video) content analysis

Table 13 - Chapter 4: Table of Requirements Prioritization

4.9.2. Functional Requirements

S. No	Requirement	Prioritization	Use case mapping
1	The system shall allow investigators to extract social media data from Android devices using ADB.	Must Have	UC1
2	The system shall preprocess Unicode Sinhala and code-mixed Sinhala-English content.	Must Have	UC2
3	The system shall classify messages as hate or non-hate using the trained BERT model.	Must Have	UC3
4	The system shall generate a structured PDF report containing flagged messages.	Must Have	UC4
5	The system shall accept manually entered text input (for testing).	Should Have	UC2 (extends)
6	The system shall display basic feedback in the CLI interface (e.g., status).	Should Have	UC3, UC4 (optional)
7	The system shall export results in both PDF and CSV formats.	Could Have	UC4

Table 14 - Chapter 4: Table of Functional Requirements

4.9.3. Non-functional Requirements

S. No	Non-functional requirements	Prioritization
1	The system should be usable via CLI with minimal technical knowledge.	Must Have
2	The tool should run efficiently on mid-range laptops with 8–16 GB RAM.	Must Have

3	The output should be legally usable and readable (PDF format).	Must Have
4	The system should support offline operation (no internet dependency).	Should Have
5	The system should be modular for future extension (e.g., more languages).	Could Have
6	The system shall include brief CLI documentation/help flags.	Could Have

Table 15 - Chapter 4: Table of Non-functional Requirements

4.10. Chapter Summary

This chapter documents the detailed software requirement specifications for the proposed multilingual Android forensic tool. Starting off with a rich picture and stakeholder analysis to visually represent the ecosystem surrounding the system and to identify all relevant actors. Then a combination of literature review and survey findings which help to guide the functional and non-functional requirements, while the stakeholder onion model and survey data highlighted the practical need for localized cyberbullying detection. The context diagram and usecase diagram illustrates how the system will interact with external entities and what actions it will perform. Moreover, the chapter also outlines the prioritization of system features and maps each requirement to its respective use case. Finally, the combination of these sections provides a clear blueprint for building and evaluating the system throughout its development lifecycle.

CHAPTER 5 : SOCIAL, LEGAL, ETHICAL, AND PROFESSIONAL ISSUES

5.1. Chapter Overview

This chapter elaborates the Social, Legal, Ethical and Professional (SLEP) issues highlighted when developing and using the proposed multilingual Android forensic tool. Handling digital forensics securely is a crucial task and since the tool deals mostly with sensitive, personal content from the social media platforms, it is important to identify and address the Social, Legal, Ethical and Professional concerns that affect the general public, suspects, investigators, criminals and legal offices. With that the chapter identifies and addresses the possible issues outlining how the project was designed to handle those concerns responsibly. However, the decisions made reflect a strong focus on user protection, legal compliance, fairness, and technical professionalism throughout the development process.

5.2. SLEP Issues and Mitigation

When developing a forensic tool that handles sensitive, personal messages, comments, captions interactions and all sorts of social media data, it is crucial to consider how this type of a forensic tool might affect people, both technically and personally. Therefore, this section elaborates the social, ethical, and professional issues related to the project and how these issues were mitigated during development of the tool.

5.2.1. Social Issues

<u>The social issue :</u>

The forensic tool analyzes private conversations, messages, comments and captions from social media applications, which includes personal opinions, sensitive language, emotional and private data. Therefore, if such data are misunderstood, misused and handled without proper care, it could lead to social distrust, stigmatization, and unjust targeting of individuals specially in a multicultural, multi-lingual society like Sri Lanka.

The mitigation strategies of the issue :

- The forensic tool is designed to be used only by law enforcement officers, and only with proper permissions. The general public and personnel without proper permissions cannot use the tool. Furthermore, this tool is only available and used in cyber-crime and forensic investigation departments in Sri Lanka such Sri Lanka Police's Computer Crime Investigation Division (CCID), Financial Crimes Investigation Division (FCID), National Child Protection Authority (NCPA), Sri Lanka Air Force Cyber Operations Centre (SLAF COC) and Sri Lanka Computer Emergency Readiness Team (SLCERT).
- The tool does not spy and monitor people continuously it only works on data that is legally extracted for crime investigations focusing mainly on hate speech, avoiding broader, live surveillance and profiling.
- Output reports are structured clearly, making it easier for investigators to understand the context minimizing chances of social harm due to misunderstandings and wrong conclusions.

5.2.2. Legal Issues

The social issue :

Since this tool mainly deals with data collected from personal mobile devices and social media platforms, identifying and addressing legal issues are crucial. With that one of the major legal issues is data privacy, a proper legal procedure should be followed to access an individual’s personal data so, if the tool is used without proper permissions, it could violate the Sri Lankan Computer Crimes Act or data protection regulations. Apart from that, handling digital evidence securely is another crucial legal concern. If the tool does not follow accepted forensic standards the collected evidences might not be accepted in court.

The mitigation strategies of the issue :

- The tool was designed and developed to ensure non-tampering, hash verification, and clear documentation, which supports legal acceptability.

- The tool is designed to be used only by authorized investigators under legal supervision and in legal institutions such as in Sri Lanka such Sri Lanka Police's Computer Crime Investigation Division (CCID), Financial Crimes Investigation Division (FCID), National Child Protection Authority (NCPA), Sri Lanka Air Force Cyber Operations Centre (SLAF COC) and Sri Lanka Computer Emergency Readiness Team (SLCERT).
- Also, the tool does not store or share the extracted data online, preventing unauthorized data leaks which leads to various sorts of consequences like reputational damage.
- The too includes a PDF report feature with metadata, timestamps, charts and hashed values to maintain the integrity of the collected evidence.
- The tool further follows guidelines from digital forensics best practices and legal frameworks in Sri Lanka.

5.2.3. Ethical Issues

The social issue :

Since the forensic tool mainly deals with private texts and hate speech on social media platforms, there are serious concerns about privacy, bias, and fairness. There is also a chance that the tool might mistakenly mark a harmless message as hate speech, most commonly known as false positives, could unfairly affect an individuals' reputations and could lead wrong decisions too.

The mitigation strategies of the issue :

- The forensic tool does not save or share any data beyond the scope of the crime investigation.
- The model was trained using publicly available datasets which were created for research and academic purposes therefore no personal data was collected from users.
- The detection model was trained and carefully fine-tuned to balance precision and recall, minimizing ethical risks due to incorrect classification of hate speech.

- The investigators are advised to double-check the flagged messages instead of relying only on the tool for evidence and the generated output is meant to be supportive, not suitable to be blindly followed when making critical decisions.

5.2.4. Professional Issues

The social issue :

The digital forensic tool must follow professional standards in software development, machine learning practices and digital forensics. If the tool produces incorrect results or if it is not tested and validated properly, it could adversely affect and result in legal challenges, misleading evidence and professional misconduct.

The mitigation strategies of the issue :

- The tool was developed in separate modules following a modular, testable architecture, so that each module of the tool such as extraction, classification, detection, reporting could be tested and improved independently ensuring the clarity, accuracy, and future scalability of the tool.
- Standard practices such as clean code, testing, dataset evaluation, model validation, code documentation, and version control were followed when developing the tool.
- The result reports generated by the tool are easy to understand and follow a legally usable format, as well-structured PDF reports that align with the professional investigative procedures.
- The tool is developed by adhering to academic and institutional guidelines which are meant to be used in legally controlled environments only, it is not a public or commercial product.

5.3. Chapter Summary

Overall, in summary, this chapter addresses the key social, legal, ethical, and professional responsibilities tied to the development of the digital forensic tool. Moreover, it elaborates how issues such as user privacy, fairness in hate speech detection, and proper investigative use strategies were managed during development. Furthermore, the project aims to support law enforcement in a secure and respectful way by following legal boundaries and professional coding standards. However, all the

features of the tool from the data extraction to pdf report generation were designed and developed by considering the user protection and ethical integrity in mind, ensuring that the tool is not only functional but also trustworthy.

CHAPTER 6 : DESIGN

6.1. Chapter Overview

This chapter elaborates the complete design structure of the forensic tool, *Hawkeye*, which is developed to detect cyberbullying through hate speech on social media platforms like Facebook, Messenger and Instagram. The chapter starts off by outlining the key design goals that guided the development process, such as modularity, scalability, and usability for non-technical investigators. Then it discusses the high-level system architecture, breaking it into three main layers as Presentation Tier, Application Tier, and Data Tier. Under that each layer and its components are explained in detail to show how the tool manages the data flow, task handling, and user interaction. Furthermore, the system design section elaborates on the selected design paradigm, supported diagrams such as Data Flow Diagram, System Flowchart, and UI design. In addition to that both the architectures of the machine learning model and the algorithm are discussed clearly. Overall, this chapter consists blueprint of the tool that maps out how each part of the tool works together to achieve the system's goals, from data extraction to hate speech detection and reporting.

6.2. Design Goals

The design of the proposed forensic tool was carefully planned to address the specific needs of digital forensic investigations in Sri Lanka. Therefore, rather than developing a general-purpose forensic application, the focus was to develop a lightweight, modular, and CLI-based digital forensic tool that supports automated hate speech detection in Sinhala and Sinhala-English social media content on Android devices. With that the following table outlines the core design goals and how those goals shaped the tool architecture:

Design Goal	Description
Multilingual Capability and Accuracy	Ensure that hate speech is detected accurately in both Sinhala Unicode and Sinhala-English code-mixed content using a fine-tuned BERT model. High accuracy in classification is crucial to support the digital evidence generation.

Modular Architecture of the tool	Divide the tool into clear layers according to its module function such as data extraction, preprocessing, classification, and report generation. This makes the system easier to test, maintain, and update independently.
CLI-based Simplicity	Instead of a complex GUI, use a CLI-based interface to ensure the tool remains lightweight, terminal-friendly, and user-friendly even to non-technical investigators.
Forensic Readiness	Include features like timestamp preservation, hashing, and structured PDF reports to ensure that the output is designed to assist law enforcement by enabling automated extraction, accurate classification, and structured evidence reporting, maintaining data integrity and supporting legal admissibility.
Maintainability	Use clear and well-scripted Python codes and a modular development code paradigm to simplify the future enhancements, debugging and replacement of components.
Scalability & Extensibility	Design the tool so that it can be extended in future for other social media applications like WhatsApp, TikTok etc. and also the model can be upgrade to detect hate speech in other languages like Tamil.
Security & Integrity	Ensure that the extracted data is handled securely without storing unnecessary data and that evidence files are protected using hashing to avoid tampering.
Efficiency	Ensure that fast processing times even on low-resource systems, by minimizing dependencies and optimizing model inference and file operations.

Table 16 - Chapter 6: Design Goals

6.3. High Level Design

6.3.1. Architecture Diagram

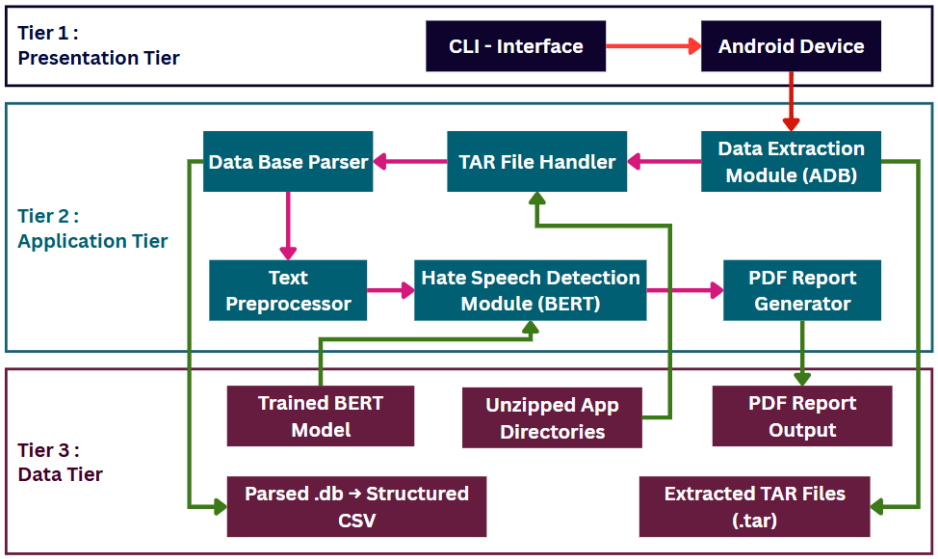


Figure 7 - Chapter 6: Design Architecture Diagram

6.3.2. Discussion of the Layers

The design architecture of the multilingual Android forensic tool follows a modular three-tier structure in order to ensure separation of issues, scalability, extendibility and ease of maintenance. Therefore, each tier is designed to handle a specific set of responsibilities in the forensic tool.

Tier 1: Presentation Tier

This is the topmost layer of the tool, with that this tier serves as the entry point to the forensic tool where user, the crime investigator interacts with the tool. This layer is mainly responsible for handling user inputs and device communications. Moreover, this tier controls the other modules in the other two tiers like launching investigations by triggering extraction, analysis, and reporting. However, it does not handle any business logics directly yet facilitates access to those types of internal processes. Below are the components in the presentation tier;

Components	Description
Command line interface	The Command-Line Interface allows the user to execute the functions of the tool like extraction, classification, and report generation by entering the initial commands. Moreover, the tool is lightweight, requires no graphical interface, and also suitable for low-resource forensic environments as well.
Android Device	The mobile device of the suspect or criminal is connected through USB or wireless ADB and it is the source of evidence. Moreover, this component provides social media application directories which is then used by the tool to extract data and start the analyze.

Table 17 - Chapter 6 : Presentation Tier

Discussion:

The presentation tier acts as the bridge between the user and the tool. The tool is designed to be simple and lightweight, using a command-line interface (CLI) that works on any system with Python and ADB installed. Moreover, this makes it ideal for the Sri Lankan investigators with limited resources or technical knowledge. Once the Android device is connected, the tool begins extracting data with minimal effort from the android device, making the process efficient and easy to operate.

Tier 2 : Application Tier

The Application Tier is the second layer of the tool's design infrastructure and also this is the core processing layer of the forensic tool. This layer handles the logic and modular functionalities essential for forensic data extraction, hate speech analysis, and report generation. This tier acts as the bridge between the raw data extracted from the device through the presentation tier with structured outputs used for the investigation. Furthermore, this layer is designed to be modular, in which each function works independently but finally contributes to a combined workflow. Below are the components in the application tier;

Components	Description
Data Extraction Module (ADB)	This module initiates Android Debug Bridge (ADB) shell commands to extract the data of Facebook, Instagram and Messenger from the connected Android device. Moreover, in this module a TAR file is created with all the social media application data in the android device and saved to the SD card or internal storage of the android device. Furthermore, the TAR file saved in the SD, or the internal storage is pulled out of the android device to the investigator's device.
TAR File Handler	This module unpacks the TAR files pulled from the Android device by the Data Extraction Module and this makes application folders and databases available for parsing.
Database Parser	The Database parser module scans and analyzes .db files, SQLite databases, extracting all text fields such as messages, comments, and captions from different tables.
Text Preprocessor	This module cleans, tokenizes, and normalizes the extracted text by handling the special characters, Unicode Sinhala, and Sinhala-English code-mixed text to ensure compatibility with the BERT model.
Hate Speech Detection Module (BERT)	This module loads the fine-tuned BERT model and classifies the preprocessed messages as Hate or Safe based on learned language patterns.
PDF Report Generator	This converts all classified messages and metadata into a structured PDF report with charts, summarizing evidence in a professional and readable format for legal or investigative use.

Table 18 - Chapter 6 : Application Tier

Discussion:

As mentioned earlier this tier is the core processing layer of the tool. It runs all the main functions of the tool like data extraction, preprocessing, classification, and report generation. Each part of the tool in this tier is modular and written separately, to allow easy updates and testing. Moreover, the tool is designed to handle real-world

Sinhala and code-mixed content and smoothly combines both machine learning and forensic logic. Overall, this tier specifically transforms raw data into meaningful insights to assist investigations.

Tier 3 : Application Tier

This layer is responsible for managing the raw and processed data used in the tool. Basically, this tier stores, organizes, and prepares all the input and output data required for hate speech detection and report generation. Below are the components in the data tier;

Components	Description
Extracted App Data (TAR Files)	These are the TAR files that are archived Facebook, Messenger and Instagram data pulled from Android using ADB.
Unzipped App Directories	This is the directory structure after unpacking the extracted .tar files containing raw application data.
SQLite Databases (.db files)	These Databases are found inside the application directories, these contain user messages, timestamps, and metadata, which were extracted from the android device and are later parsed for analysis.
Parsed Structured CSV Files	The .csv files are generated after extracting and processing the text-based data from the databases, ready for the classification stage. Simply it is the output of .db parsing.
Trained BERT Model	Pre-trained model used to classify content into hate/non-hate categories.
Prediction Output (CSV)	These are the immediate results from the model including the original messages along with prediction labels after being classified to Hate and Safe speech.
PDF Report Output	This is the final human readable report generated along with summary tables and charts for law enforcement, to be used in investigations as evidence.

Table 19 - Chapter 6 : Data Tier

Discussion:

This tier acts as the backbone of the tool's data flow. Starting off with raw social media data that were pulled through ADB, which is then stored as TAR file and later unpacked for analysis. Moreover, the databases are parsed to extract valuable evidence such as comments, captions and messages along with timestamps. Furthermore, the system then prepares the content as CSV files, which are used by the model to detect hate speech. Finally, results are stored and passed on to PDF report generation, ensuring proper handling and secure storage of data which is crucial for both accuracy and forensic integrity.

6.4. Detailed System Design

6.4.1. Choice of the Design Paradigm

In this project, a fully traditional SSADM or OOADM approach was not followed instead, a Function-Driven Modular Design Paradigm was selected to develop the Multilingual Android Forensic Tool (Hawkeye), specially shaped for the needs of a lightweight command-line based forensic automation tool. However, the principles inspired by SSADM were partially adopted to assist in planning the data flow and logic across the processing stages of the tool. The reason for selecting a function-based modular structure is because instead of just scripting one large block of code, the tool was structured, broken down into smaller sections with clearly defined functional modules and each module is responsible for a specific task such as social media data extraction, TAR handling, database parsing, text preprocessing, hate speech detection, and PDF report generation. This makes it easier to develop, test, update, modify and extend the code in the future.

Since the tool is written in Python and runs through a Command-Line Interface (CLI), this modular structure keeps everything clean and simple. The main tool script (hawkeye_main.py) handles the core functions, while the report generation logic is organized separately in its own script (report_pdf_structured.py). However, each function can be tested and debugged independently, enhancing maintainability and scalability. This separation helps investigators use the tool with minimal effort and lets

future developers add more features like supporting other languages or social media platforms.

This design paradigm was chosen over Object-Oriented or Event-Driven approaches seen in OOAD due to the straightforward nature of forensic automation in which the user interactive actions simply move through a series of tasks in a controlled order like extracting data, cleaning it, running the model, and generating a report. In addition, the modular structure of the tool supports the reusability and future expansion of it, like expanding the tool to new social media platforms, other languages or integrating more advanced forensic capabilities.

Overall, this structured, script-based paradigm was chosen to keep the tool practical, adaptable, and easy to manage, especially in real-world digital forensic investigations in Sri Lanka. With that this choice of design paradigm is ideal as the system is terminal-based and designed to run in resource-constrained forensic environments, the use of a simplified SSADM-inspired flow and Data Flow Diagrams (DFD) was more practical than class or component diagrams. Furthermore, this approach ensured the clarity, reusability, and future scalability of the tool without unnecessary complexity.

6.4.2. Data Flow Diagram (Level 1 DFD)

The Data Flow Diagram shows how the data moves through the Multilingual Android Forensic Tool step by step. First, the user connects an Android device, and the tool starts extracting social media app data using ADB. This data is saved as TAR files (D1). Then, the tool unpacks and reads these files to find SQLite databases, which are stored in D2.

Next, the tool extracts messages from the databases and cleans them using a text preprocessing module, that cleaned texts is saved in D3 as .csv files. After that, the cleaned messages are passed to the BERT-based hate speech detection model, which checks if the content is hateful or not and are stored in D4. Finally, the tool generates a PDF report based on the detected content, which the investigator can use in the investigations.

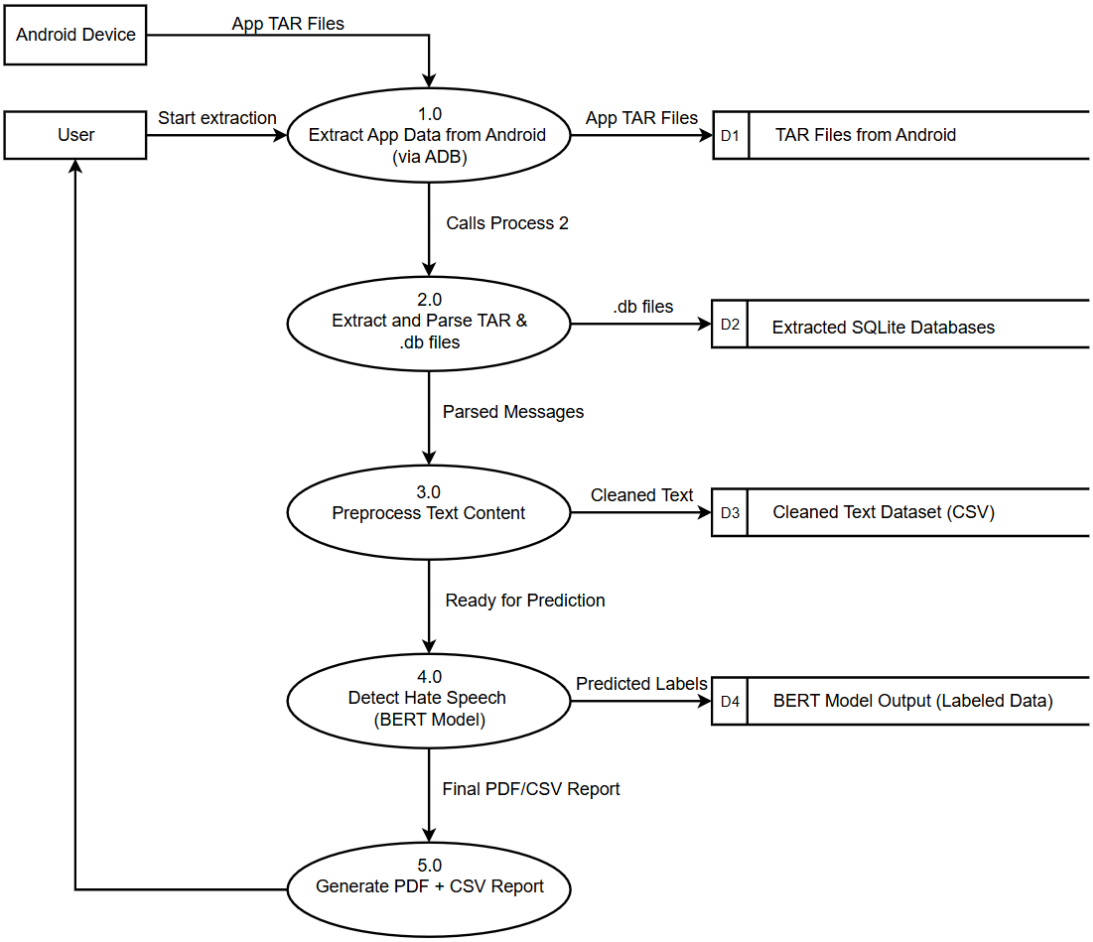


Figure 8 - Chapter 6 : Data Flow Diagram

6.4.3. Algorithm Design

The tool algorithm is designed around a modular functional pipeline, where each function is executed as a sequence from data extraction to PDF report generation. It follows a procedural design inside *hawkeye_main.py*, with separate modules for each task.

Module/Function	Description
connect_device()	Initiates ADB connection with the Android device.
pull_sdcard_app_data()	Pulls TAR files from Facebook, Instagram, Messenger directories.
extract_tar_files()	Extracts TAR files into folder structures on the host PC.

find_all_databases()	Scans extracted folders and locates .db files.
parse_databases()	Parses readable message content from SQLite databases.
preprocess_text()	Cleans, normalizes the extracted raw Sinhala/Singlish texts and removes noise, symbols, etc.
run_hate_speech_detection()	Loads the fine-tuned BERT model and classifies each message in to hate or safe speech.
generate_hashes_and_csv()	Creates a structured CSV and computes SHA-256 hashes for evidence validation.
generate_pdf_report() (from a separate script)	Generates a formal PDF report with labels, timestamps, and metadata.

Table 20 - Chapter 6 : Algorithm Design

6.4.4. Models Design

A fine-tuned transformer-based model called BERT (Bidirectional Encoder Representations from Transformers) was selected for the core classification task of detecting hate speech in Sinhala and Sinhala-English code-mixed content on Social media application like Facebook, Messenger and Instagram. This model architecture has proven its effectiveness in handling low-resource languages and code-mixed textual data, especially in multilingual environments like Sri Lanka's social media landscape hence BERT model was chosen in this project.

Moreover, a multilingual transformer model pre-trained in over 100 languages called xlm-roberta-bas was used as the base model. It provides a strong foundation for understanding sentence structures, even when mixed with Sinhala Unicode and transliterated (Singlish) expressions. However, the model was fine-tuned using a specialized dataset of hate speech content gathered from publicly available Sinhala and Singlish hate speech. The training dataset was preprocessed to normalize text, remove noise, and handle code-mixed elements, ensuring that the model was trained on realistic and diverse examples of harmful content.

Furthermore, the fine-tuning process involved training the model on a binary classification task of Hate vs. Safe using PyTorch and the Hugging Face Transformers library. With that standard classification, loss functions such as CrossEntropyLoss were

used, and performance was evaluated using metrics like accuracy, precision, recall, and F1-score to ensure balanced performance across classes.

However, the final model was tested against undetected data and with the showed results in classifying various forms of hate speech with good generalization, even in noisy, informal social media texts, the model was then integrated directly into the tool's pipeline, where it classifies messages extracted from Android devices in real-time and passes results to the report generation module. Finally, the integration of this multilingual BERT model ensures that the forensic tool remains relevant, accurate, and practical for unique context and cybercrime investigations in Sri Lanka.

6.4.5. System Flowchart

The system flowchart provides a visual overview of the entire workflow of the Hawkeye forensic tool, from the initial step of running the Hawkeye tool to the final stage of PDF report generation. Furthermore, it demonstrates how data flows through the tool's modules in a logical sequence and highlights decision points and user inputs along the way.

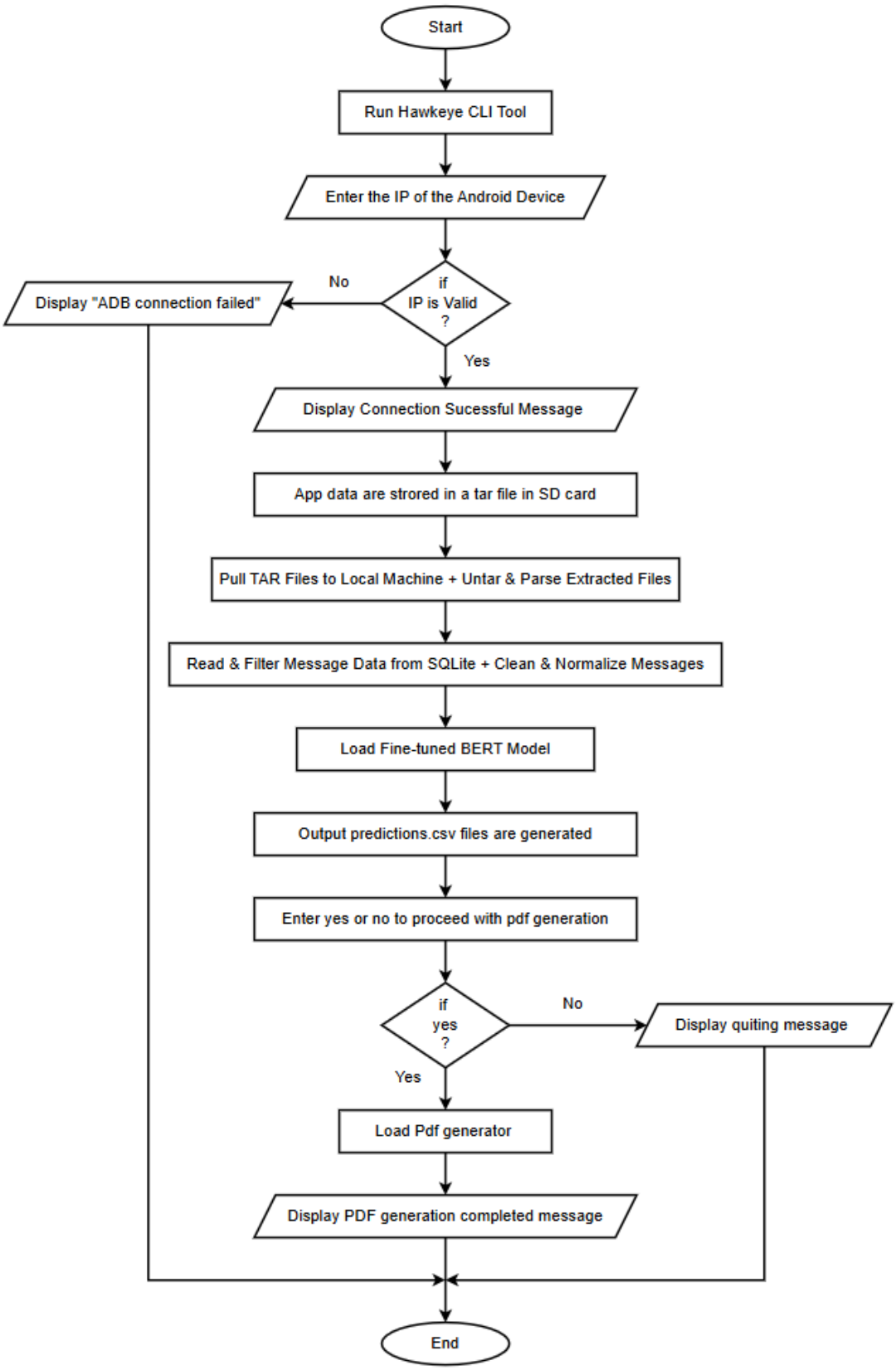
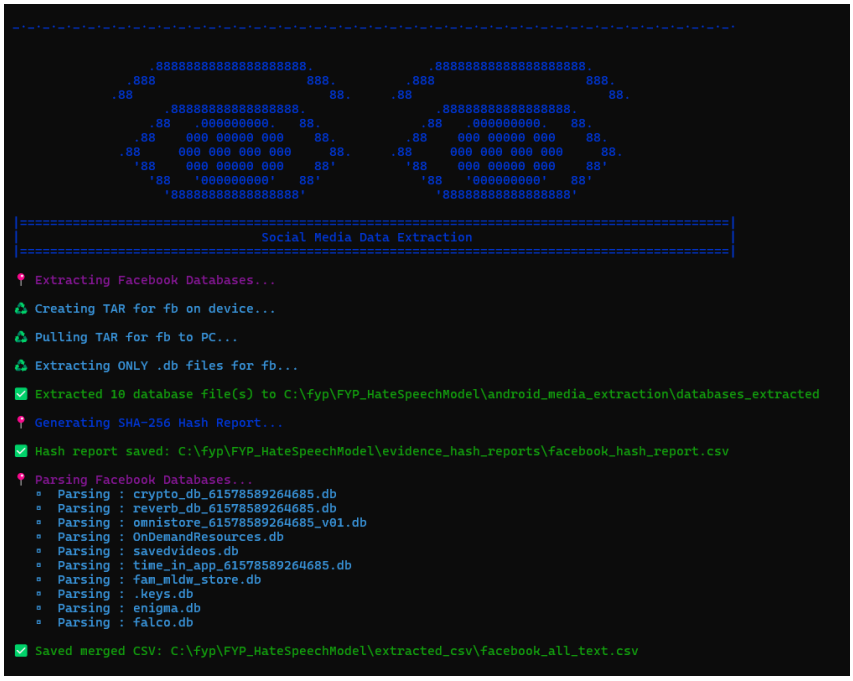


Table 21 - Chapter 6 : System Flow Chart

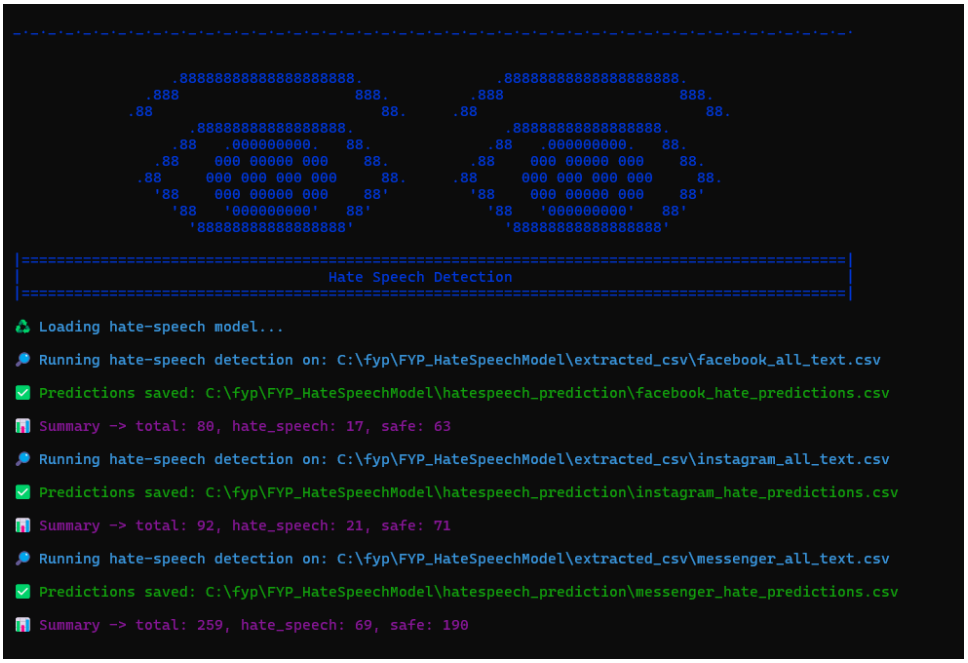
6.4.6. User Interface Design

In this android forensic tool, Hawkeye, there is no graphical user interface, so everything is based on command line interface. This makes the tool lightweight and is easier to understand. Furthermore, it is a simple tool with very few user interactions.

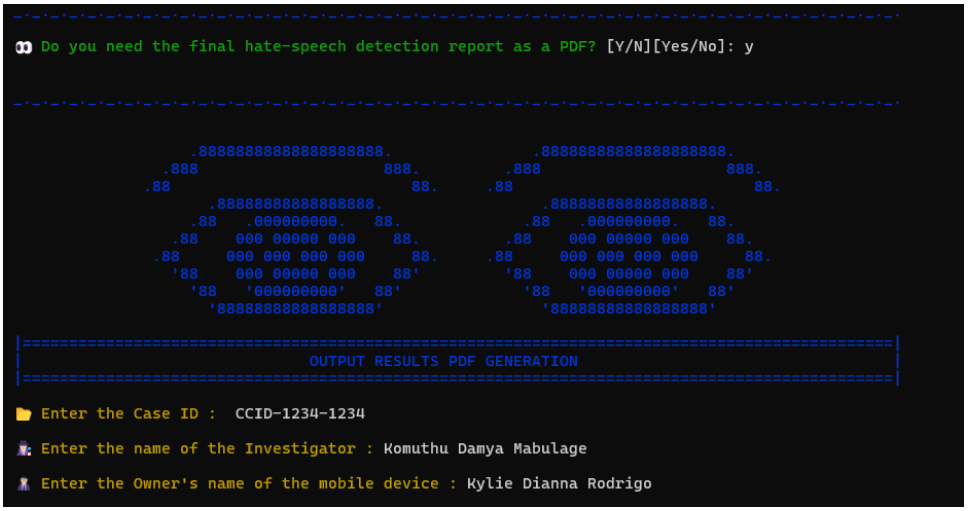


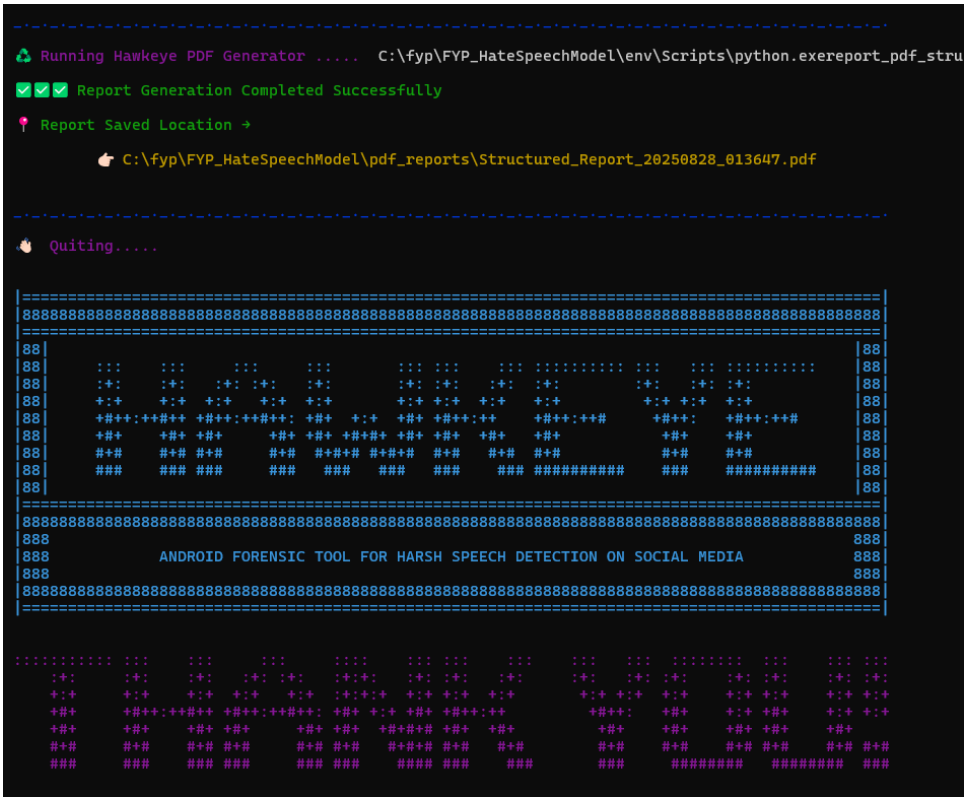
```
📍 Extracting Instagram Databases...

♻️ Creating TAR for ig on device...

♻️ Pulling TAR for ig to PC...

♻️ Extracting ONLY .db files for ig...

✅ Extracted 13 database file(s) to C:\fyp\FYP_HateSpeechModel\android_media_extraction\databases_extracted

📍 Generating SHA-256 Hash Report...

✅ Hash report saved: C:\fyp\FYP_HateSpeechModel\evidence_hash_reports\instagram_hash_report.csv

[📍IG] Parsing Instagram Databases...
   ▫ Parsing : crypto_db_61578589264685.db
   ▫ Parsing : reverb_db_61578589264685.db
   ▫ Parsing : omnistore_61578589264685_v01.db
   ▫ Parsing : OnDemandResources.db
   ▫ Parsing : savedvideos.db
   ▫ Parsing : time_in_app_61578589264685.db
   ▫ Parsing : fam_mldw_store.db
   ▫ Parsing : .keys.db
   ▫ Parsing : enigma.db
   ▫ Parsing : falco.db
   ▫ Parsing : crypto_db_0.db
   ▫ Parsing : crypto_db_17843908857538367.db
   ▫ Parsing : direct.db
   ▫ Parsing : fileregistry.db
   ▫ Parsing : igd_wellbeing_database_0.db
   ▫ Parsing : igd_wellbeing_database_17843908857538367.db
   ▫ Parsing : incoming_db_0.db
   ▫ Parsing : incoming_db_17843908857538367.db
   ▫ Parsing : recent_searches.db
   ▫ Parsing : reverb_db_0.db
   ▫ Parsing : reverb_db_17843908857538367.db
   ▫ Parsing : time_in_app_76169594366.db
   ▫ Parsing : enigma.db

✅ Saved merged CSV: C:\fyp\FYP_HateSpeechModel\extracted_csv\instagram_all_text.csv
```

```
📍 Extracting Messenger Databases...

♻️ Creating TAR for ms on device...

♻️ Pulling TAR for ms to PC...

♻️ Extracting ONLY .db files for ms...

✅ Extracted 9 database file(s) to C:\fyp\FYP_HateSpeechModel\android_media_extraction\databases_extracted

📍 Generating SHA-256 Hash Report...

✅ Hash report saved: C:\fyp\FYP_HateSpeechModel\evidence_hash_reports\messenger_hash_report.csv

📍Parsing Messenger Databases...
   ▫ Parsing : crypto_db_61578589264685.db
   ▫ Parsing : reverb_db_61578589264685.db
   ▫ Parsing : omnistore_61578589264685_v01.db
   ▫ Parsing : OnDemandResources.db
   ▫ Parsing : savedvideos.db
   ▫ Parsing : time_in_app_61578589264685.db
   ▫ Parsing : fam_mldw_store.db
   ▫ Parsing : .keys.db
   ▫ Parsing : enigma.db
   ▫ Parsing : falco.db
   ▫ Parsing : PreloadedInMemoryDatabase-20e34a9c305c90.db
   ▫ Parsing : crypto_db_61578589264685.db
   ▫ Parsing : encrypted_backups_db_61578589264685.db
   ▫ Parsing : omnistore_61578589264685_v01.db
   ▫ Parsing : reverb_db_61578589264685.db
   ▫ Parsing : .keys.db
   ▫ Parsing : fts.db
   ▫ Parsing : enigma.db
   ▫ Parsing : mldw_messenger_wellbeing_store.db
   ▫ Parsing : crypto_db_0.db
   ▫ Parsing : crypto_db_17843908857538367.db
   ▫ Parsing : direct.db
   ▫ Parsing : fileregistry.db
   ▫ Parsing : igd_wellbeing_database_0.db
   ▫ Parsing : igd_wellbeing_database_17843908857538367.db
   ▫ Parsing : incoming_db_0.db
   ▫ Parsing : incoming_db_17843908857538367.db
   ▫ Parsing : recent_searches.db
   ▫ Parsing : reverb_db_0.db
   ▫ Parsing : reverb_db_17843908857538367.db
   ▫ Parsing : time_in_app_76169594366.db
   ▫ Parsing : enigma.db

✅ Saved merged CSV: C:\fyp\FYP_HateSpeechModel\extracted_csv\messenger_all_text.csv

✅✅✅ Extraction, Hashing, and Parsing Completed!
```


Hate Speech Detection
Loading hate-speech model...
Running hate-speech detection on: C:\fyp\FYP_HateSpeechModel\extracted_csv\facebook_all_text.csv
Predictions saved: C:\fyp\FYP_HateSpeechModel\hatespeech_prediction\facebook_hate_predictions.csv
Summary -> total: 80, hate_speech: 17, safe: 63
Running hate-speech detection on: C:\fyp\FYP_HateSpeechModel\extracted_csv\instagram_all_text.csv
Predictions saved: C:\fyp\FYP_HateSpeechModel\hatespeech_prediction\instagram_hate_predictions.csv
Summary -> total: 92, hate_speech: 21, safe: 71
Running hate-speech detection on: C:\fyp\FYP_HateSpeechModel\extracted_csv\messenger_all_text.csv
Predictions saved: C:\fyp\FYP_HateSpeechModel\hatespeech_prediction\messenger_hate_predictions.csv
Summary -> total: 259, hate_speech: 69, safe: 190


Hate Speech Detection Summary
Facebook :
Total = 80
Hate Speech = 17
Safe = 63
Instagram :
Total = 92
Hate Speech = 21
Safe = 71
Messenger :
Total = 259
Hate Speech = 69
Safe = 190
Extractions and Hate Speech Detection All Completed!
Saved Locations of the Generated Reports :
C:\fyp\FYP_HateSpeechModel\hatespeech_prediction\facebook_hate_predictions.csv
C:\fyp\FYP_HateSpeechModel\hatespeech_prediction\instagram_hate_predictions.csv
C:\fyp\FYP_HateSpeechModel\hatespeech_prediction\messenger_hate_predictions.csv





6.5. Chapter Summary

This whole chapter 6 was about detailing the design of the Hawkeye forensic tool, starting from its goals to its internal working structure. The design goals of the tool emphasized clarity, simplicity, and expandability to make the system useful for real-world forensic investigations in Sri Lanka.

Moreover, the high-level design architecture was split into three logical tiers, each handling specific responsibilities such as user interaction, process execution, and data handling. Then after that the system design section justified the use of a function-based modular approach including supportive diagrams such as the Data Flow Diagram, System Flowchart and the UI design.

Additionally, the chapter discussed the algorithmic structure and BERT-based model used for hate speech detection, ensuring the tool is both technically sound and practically usable. In summary, this chapter solidifies the foundational architecture that supports the entire implementation of the project.

CHAPTER 7 : IMPLEMENTATION

7.1. Chapter Overview

This chapter explains the core technologies and implementation strategies used in building the Hawkeye android forensic tool. It starts off by discussing the technology stack, programming language, frameworks, libraries, and datasets that were selected to support the tools goals. Furthermore, it also dives into how each module such as data extraction, hate speech detection, and report generation was implemented step-by-step to support digital investigations in real-world context in Sri Lanka.

7.2. Technology Selection

7.2.1. Technology Stack

The architecture of the Hawkeye tool is implemented using a function-driven modular design, and the technologies used are distributed across all those three tiers of the tool. Below is a breakdown of each tier along with the technologies and tools utilized.

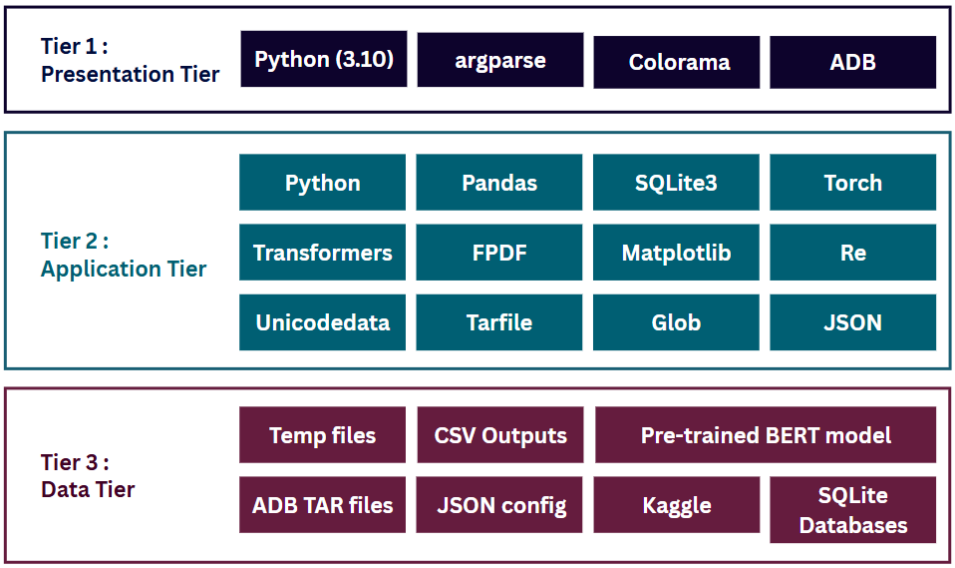


Figure 9 - Chapter 7 : Technology Stack Tier Diagram

Tier 1 : Presentation Tier

Currently the tool uses a command-line interface (CLI) for the investigators to interact with the tool and the whole tool is built entirely on Python version 3.10. This

presentation tier of the tool allows the users to initiate tasks such as connecting to an Android device, starting the data extraction, and generating final reports. With that in order to enhance the user experience, the ‘colorama’ library is used to style CLI messages with colors such as green for success and red for warnings making outputs clearer. Moreover, the Android Debug Bridge (ADB) plays a crucial role in this layer by enabling direct communication with the Android device. The tool executes adb shell commands like ‘adb connect’, ‘adb devices’, ‘adb pull’, and others necessary commands for extracting social media application data from the android device. Furthermore, the structure also allows future integration of modules like ‘argparse’ to support more advanced CLI options and flags for automation and scalability even though it is not currently implemented.

Tier 2 : Application Tier

This application tier handles the main logic and processing of the forensic tool. Python serves as the backbone throughout the whole tool especially this tier it supports by a range of powerful libraries. Starting off with ‘pandas’ which is used extensively for using structured data and converting parsed text from databases into readable CSV files. Then ‘sqlite3’ enables the tool to connect to the ‘.db’ files extracted from Facebook, Instagram, and Messenger, by allowing the tool to extract comments, messages, captions, and timestamps from the databases.

As for the hate speech detection component, the tool uses ‘torch (PyTorch)’ and the ‘HuggingFace transformers’ library to load and run a fine-tuned multilingual BERT model. Thereafter, the text contents are cleaned using ‘re’ (regular expressions) and ‘unicodedata’, which ensures Unicode compatibility, especially when dealing with Sinhala and Sinhala English transliterated texts. Moreover, the ‘tarfile’ module is used to handle .tar archives pulled from the Android device, meanwhile ‘glob’ helps in scanning directories to find all .tar and .db files for processing. Furthermore, ‘json’ is used for reading and writing configuration files such as case metadata required for PDF reports. In addition to that, for generating those structured PDF reports, the tool employs the ‘fpdf’ library, and optionally, ‘matplotlib’ for all the visual summaries, charts within the PDF report file.

Overall, this tier transforms the raw app data into structured, meaningful evidence through a sequence of modular processes starting from extraction, parsing to hate speech classification and finally report generation.

Tier 3 : Data Tier

The data tier consists of all the input, intermediate, and output data that operates the forensic analysis. Temporary files like JSON configs are generated to hold investigator and case metadata. As for the CSV file generation, such files were produced at multiple stages starting from parsed databases, and finally from model predictions with separate files for each platform named '<platform name>_hate_predictions.csv'. Apart from that, a critical resource in this layer is the pre-trained BERT model saved locally as 'saved_model' and loaded during runtime for hate speech detection. Furthermore, the extracted TAR files pulled from the Android phone contain all the internal data of the social media apps and are stored in the file system for later parsing.

All application databases extracted from social media apps are stored in SQLite format and it serves as the primary source of textual evidence. However, the original hate speech training datasets were obtained from Kaggle, making it a critical external data source referenced during the model development. Overall, this layer ensures that all raw data, model files, and forensic outputs are steadily and reliably handled throughout the entire workflow.

7.2.2. Selection of Datasets

Selecting a good, suitable, balanced dataset was a critical step in the process of developing this multilingual android forensic tool for hate speech detection on social media, due to its focus on rare languages like Sinhala and unique made-up language like Sinhala English transliterated. To train and fine tune the model, two datasets of Sinhala Hate Speech with more than 6000 hate texts and Sinhala English transliterated Hate Speech with more than 13000 texts from Kaggle were combined and used. This combined dataset was particularly used because it reflects the kind of content typically seen in Sri Lankan Facebook and Instagram messages, comments, captions etc. by often mixing the languages of Sinhala Unicode and Transliterated Sinhala (Singlish).

Moreover, the datasets include both hateful and non-hateful texts and were pre-labeled to support supervised training of the BERT model. Although at first the datasets were noisy and imbalanced, later they were cleaned, shuffled, and augmented to better represent real-world social media communication patterns. After that both the datasets were combined to get a better output. Furthermore, the diversity in terms of slang, offensive expressions, and transliterated texts helped the model learn more varied patterns of cyberbullying, which is crucial for a forensic tool with an end goal to assist the Sri Lankan cybercrime units in investigation processes. In conclusion the open-source availability and real-world relevance of the dataset made it an ideal choice for developing and benchmarking the multilingual hate speech detection algorithm integrated in the tool.

7.2.3. Selection of Programming Language

As mentioned earlier, the Hawkeye Multilingual android forensic tool was entirely developed using the Python version 3.10 programming language and this language was selected considering its simplicity, flexibility, and strong support for data handling, scripting, and machine learning. Moreover, Syntaxes in Python made it an ideal choice for developing a command-line-based tool that required crucial interaction with system-level utilities like ADB, parse SQLite databases, handle file systems, and integrate a BERT-based NLP model.

Apart from that, tool being lightweight and platform independent was a crucial design goal hence Python came in handy as it allowed a seamless cross-platform compatibility without requiring complex installations. Most important of all is that python includes a varied set of libraries tailored for each function of the tool from ‘sqlite3’, ‘tarfile’ and ‘os’ for forensic data handling, transformers and torch for model inference and ‘fpdf’ for report generation. Finally, the wide adoption of python in the digital forensics and machine learning components ensured that a strong documentation, libraries, and community supported throughout the development of the tool.

7.2.4. Selection of Libraries

The development of this Hawkeye forensic tool relied on a wide range of Python libraries, where each selected library was based on a specific functionality of the tool.

Library	Functionality / Purpose
argparse	Handles command-line arguments for scripts like the PDF report generator.
colorama	Adds color formatting to terminal/CLI output for better readability.
datetime	Used for handling date/time values, such as in timestamps and report generation.
fontTools	Supports font embedding and manipulation when generating structured PDFs.
Fpdf	Generates the final structured PDF reports for forensic analysis.
Glob	Finds all pathnames matching a pattern (used to locate TAR or .db files).
hashlib	Calculates SHA-256 hashes for forensic integrity verification of extracted files.
Json	Reads and writes configuration data, especially for PDF generation input.
matplotlib	Visualizes the hate speech detection summary in the final PDF (bar charts).
numpy	Supports numerical operations during visualization and model predictions.
Os	Interacts with the file system and environment variables (e.g., directory creation).
pandas	Reads, merges, filters, and saves tabular data like CSVs from extracted databases.
Re	Apply regular expressions to clean or validate text and user inputs.
Shutil	Handles file copying operations, particularly in PDF layout formatting.
sqlite3	Reads and queries Android .db files extracted from social media apps.
subprocess	Runs shell commands like adb, tar, or other system-level instructions.

Sys	Accesses runtime settings (e.g., script arguments, Python path).
Tarfile	Extracts .tar archives pulled from Android's SD card.
tempfile	Manages temporary config files during PDF generation.
Torch	Runs the hate speech detection model using PyTorch deep learning framework.
transformers	Loads BERT-based language models for Sinhala/Singlish hate speech classification.
unicodedata	Normalizes Sinhala text (Unicode), important for multilingual preprocessing.
warnings	Suppresses unnecessary warnings for cleaner execution.

Table 22 - Chapter 7 : Selection of Libraries

7.2.5. Selection of Development Frameworks

In the development of the Hawkeye Multilingual Forensic Tool, there were no full-stack or web-based frameworks used as the tool was entirely Command Line Interface (CLI) based. Therefore, the development framework selection was focused more on lightweight, modular, and script-based components than large-scale architectural development frameworks. Moreover, instead of using a traditional development framework, this tool used Python based modular scripting capabilities and PyTorch based machine learning framework to structure and run different parts of the system efficiently.

The key development-related frameworks and libraries used include:

Framework	Purpose in Development
PyTorch	A deep learning framework used for running the fine-tuned hate speech classification model.
Hugging Face Transformers	Provides the pre-trained BERT architecture and tokenizer, allowing simple integration into the PyTorch pipeline.

FPDF (via PyFPDF)	A lightweight Python framework for generating structured, investigator-friendly PDF reports.
ADB (Android Debug Bridge)	Though not a Python framework, ADB is used as a core component in the forensic extraction process, tightly integrated via Python subprocess commands.
Colorama	Used to simulate interface coloring and enhance the CLI user experience.

Table 23 - Chapter 7 : Selection of Development Frameworks

Overall, the design goal of the tool was centered around using lightweight yet powerful libraries and frameworks to maintain modularity, speed, and expandability of the tool so, the above selections are ideal to keep up with those design goals.

7.2.6. Integrated Development Environment

Visual Studio Code (VS Code) was used as the primary Integrated Development Environment (IDE) when developing the Hawkeye CLI-based digital forensic tool. Vs code was chosen mainly due to its lightweight nature, flexibility, strong support for Python development and built-in terminal, which helped run ADB commands, debug scripts, and test machine learning predictions efficiently.

Moreover, its extensions like Python IntelliSense and Jupyter support improved code quality and development speed. Apart from that, the modular structure of the tool with two scripts and multiple functions for each task from data extraction, prediction till the final reporting was easy to manage in VS code. In addition to that, the Git integration in VS code allowed version and smooth updates throughout the development process of the tool.

7.2.7. Summary of Technology Selection

Component	Tools and Technologies Used
Presentation Layer	Python CLI (argparse, Colorama), ADB Shell Interface
Application Layer	Python 3.10, SQLite3, Pandas, Transformers, Torch, Tarfile, Re, Glob, JSON, Unicodedata

Libraries	Transformers, Pandas, Torch, Tarfile, FPDF, SQLite3, Re, JSON, Glob, Unicodedata
Data Sources	Kaggle (Sinhala Hate Speech Dataset), Manually Collected Social Media TARs
Machine Learning Model	Fine-Tuned BERT (Sinhala and Singlish), Transformers and Torch
Output Formats	CSV (parsed .db), PDF (FPDF Report), JSON (report config)
Hashing and Forensics	SHA-256 (hashlib), ADB TAR extraction (su, tar commands)
Integrated Development Environment (IDE)	Visual Studio Code (Extensions: Python, Jupyter, Pylint)
Mobile Device Interaction	Android Debug Bridge (ADB), Device CLI adb with su access

Table 24 - Chapter 7 : Summary of Technology Selection

7.3. Implementation of Core Functionalities

7.3.1. Social Media Data Extraction Pipeline

The first most core functionality of the Hawkeye forensic tool is extracting social media data from the android device using ADB (Android Debug Bridge). This extraction process is fully automated in the tool, below the automation will be elaborated step by step;

ADB connection and TAR creation

First the tool initiates an ADB shell session and asks for the Ip of the Android device to connect to the device, then the tool executes a few commands along with some privileged ‘su’ commands to connect to the android device (‘abd connect xx.xx.xx.xx:xxxx’) and then to access ‘/data/data/’ directories of Facebook, Instagram, and Messenger applications on the android device. After accessing the tool uses tar to archive the entire application folders separately along with the application name and stores them in the SD card of the device.

Pulling the TAR files to PC

The .tar files stored in the SD card of the device are pulled to a local directory on the investigator's computer using the ADB pull ('adb pull ..') commands.

Unpacking the TAR file extracted

The pulled files stored in the local directory are then extracted using 'tarfile' python module, exposing the full file structure of each application seperately.

Database Parsing

After unpacking the tar files, the tool searches recursively for all .db files within the extracted files of each application. Then it filters all the SQLite databases containing message/comment/caption all text data related tables by checking for text columns and timestamp fields. With that, the found .db files are then parsed using sqlite3 and these files are converted into structured .csv files.

Hash Report Generation

Then additionally for each extracted database a **SHA-256 hash** is computed to ensure integrity of the collected evidences. And also, these hashes are stored in separate CSVs and later included in the final PDF report.

*NOTE - The full code for this Social Media Data Extraction Pipeline is attached below in the appendix (View Code)

7.3.2. Hate Speech Detection Model

The core AI component of the Hawkeye tool is a multilingual hate speech classifier designed to detect harmful and abusive language in Sinhala and transliterated Sinhala (Singlish) from social media platforms like Facebook, Instagram, and Messenger. So, once the data is extracted, filtered and as CSVs, the tool automatically passes this text into the multilingual hate speech classification model which was developed in phase 1 and saved as "saved_model".

Model Selection and Architecture

The tool uses a is a fine-tuned version of the '**distilbert-base-multilingual-cased transformer**' from Hugging Face, which is a compact, faster version of BERT that supports more than 100 languages and is particularly suitable for low-resource environments like in Sri Lanka. Furthermore, this transformer model is pre-trained in 104 languages using a masked language modeling task. With that, this model was chosen mainly due to its multilingual support, efficiency in limited resource environments and compatibility with Hugging Face's pipeline and PyTorch backend. The architecture of this model remained mostly unchanged, yet however, the classification head was adapted to perform binary classification (hate speech or safe speech).

Dataset, Preprocessing and Model Training

When considering the preprocessing and model training, however this training the model is not automated in the cli tool, this was done separately in phase 1 of the project. The model was trained on a manually on a custom '**merged Sinhala/Singlish dataset'** by merging two datasets of 'Sinhala Unicode hate speech' and 'Sinhala English transliterated hate speech', using Trainer API of 'Hugging Face' with PyTorch backend. However, before training the model data was tokenized using the AutoTokenizer from Hugging Face specific to the distilbert-base-multilingual-cased model. Then Padding and shortness were applied to ensure consistent input lengths for the transformer (max length: 128 tokens). The training was carried using the following configuration through Hugging Face Trainer API ;

Model	distilbert-base-multilingual-cased
Epochs	5
Batch Size	8
Learning Rate	5e-5
Evaluation Strategy	Evaluated after each epoch

Loss Function	CrossEntropyLoss for binary classification
Optimizer	AdamW
Framework	PyTorch
Tokenizer	Hugging Face AutoTokenizer

Table 25 - Chapter 7 : Model Training Configurations

*NOTE - The full codes for Dataset, Preprocessing and Model Training are attached below in the appendix (View Merging Dataset Code),(View Model Training Code)

Detection Process During Tool Execution

As mentioned earlier once the model was trained, it was saved into the local directory as '/saved_model/'. So, during the runtime, the tool loads the model and processes each CSV generated from the parsed databases. The CSV starts loading the extracted texts from Facebook, Instagram, and Messenger .csv files. And then it starts text cleaning, in which the URLs, emojis, and special symbols are removed using regular expressions ('re'). Then moving on to prediction, the model generates logits which are converted to probability scores using 'softmax' and based on that probability, each message is classified as either hate speech (label 1) or safe (label 0). Apart from that for extremely abusive known Sinhala/Singlish words, a keyword filter flags messages directly, increasing robustness a keyword-based fallback strategy is used. Finally, the tool joins predictions and confidence scores to new .csv files which are then saved in a local directory.

7.3.3. PDF Report Generation Functionality

The final function of the tool is to generate a well-structured PDF forensic report using the predictions .csv files generated earlier. Those csv files contain the original text, predicted label, confidence score, timestamp (if available) and source database and table. Below are the details included in the pdf report and the technologies used in the implementation of the report generation functionality;

Library Used	fpdf2

Custom Sinhala Font	Embedded using NotoSansSinhala-Regular.ttf
Cover Page	Contains title, date, case ID, and investigator name
Platform Summary	Includes total texts, hate speech count & percentage, and safe speech stats for Facebook, Instagram, and Messenger.
Visual Charts	Bar graphs and pie charts are included using matplotlib
Flagged Messages	Separate pages are dedicated to top hate messages with source information and confidence scores.
Appendix	Includes a full list of all databases with SHA-256 hash values to maintain data integrity.

Table 26 - Chapter 7 : PDF Report Generation Functionality

Finally, the PDF is saved in the "pdf_reports/" folder and it includes all extracted, analyzed, and validated evidence in a readable format for legal or investigative use.

7.4. Chapter Summary

In summary, chapter 7 outlined the tools, platforms, and processes that brought the Hawkeye CLI tool to life. Moreover, the tool was built using Python and VS Code, with ADB for Android data extraction and Hugging Face's transformer models for hate speech classification. Moreover, each module was carefully developed by pulling social media data, analyzing it through a trained ML model, and finally generating a detailed PDF report. Overall, these implementations ensure the tool remains accurate, efficient, and practical for digital forensic use in Sri Lanka.

CHAPTER 8 : TESTING

8.1. Chapter Overview

This chapter, which is entirely based on the testing process of the tool, explains elaboratively how the Hawkeye tool was tested in various methods to ensure that the tool works reliably, gives accurate results, and is easy to use. Moreover, it outlines the main objectives of testing, the different testing types performed such as model testing, functional checks, performance testing etc. and presents the outcomes of those tests. Overall, the aim of this is to show that each part of the tool from data extraction to hate speech detection to PDF report generation was thoroughly tested and evaluated.

8.2. Objectives and Goals of Testing

The testing phase of the Hawkeye Android Forensic Tool mainly focuses on validating that the tool functions correctly, delivers accurate results, and is suitable for real-world forensic use. As the tool handles sensitive investigations to assist cybercrime investigators, especially in Sri Lanka, it needs to handle sensitive data with care, provide trustworthy results, and work user-friendly in real field scenarios involving hate speech on social media, hence it is important to test both the individual components and the tool combining all the components together.

Objectives of Testing

- To ensure accurate hate speech detection across Sinhala and Singlish inputs using F1 Score, Precision, Recall, and Accuracy.
- To verify seamless execution of the entire pipeline from device connection to PDF report generation.
- To validate database extraction and parsing for Facebook, Instagram, and Messenger apps regardless of internal schema variations.
- To confirm SHA-256 hash generation for forensic integrity of evidence files.
- To test the ability of the tool to handle real-world social media text in Unicode Sinhala, code-mixed Singlish, and slang.

- To assess strength in connecting to different Android devices via ADB.
- To validate the performance of the tool under realistic conditions without lagging or crashing.
- To check user experience through the command-line interface to ensure accessibility for non-technical users.

Goals of Testing

- Maintain the reliability and security forensic tool that can be trusted by investigators for hate speech detection.
- Maintain high model performance (F1 score above 0.90) to ensure accurate classification.
- Deliver a smooth, fully automated workflow that minimizes manual effort.
- Ensure compatibility with diverse Android device setups in Sri Lankan forensic environments.
- Produce clear and structured reports that can be used as official digital evidence in investigations.
- Identify and fix bugs early to avoid failures during live use or future deployments.
- Establish confidence in the tool's ability to assist cybercrime investigations effectively and ethically.

8.3. Testing Criteria

The testing process for the *Hawkeye Multilingual Forensic Tool* was guided by the following criteria, to ensure the tool meets both functional and non-functional requirements. So, these requirements are categorized into two key aspects for testing the tool:

Functionality of the Tool

This focuses on testing all the technical features of the tool such as Android ADB extraction, TAR parsing, CSV generation, hate speech detection, and PDF reporting to ensure that the main functions of the tool work accurately as intended. Furthermore, the performance of the model was evaluated using the matrices of accuracy, F1-score, precision, recall and the tool was tested to ensure proper workflow from start to end without manual intervention.

Structural Quality of the Tool

This aspect addresses the non-functional features of the tool such as speed, usability, and strength. The tool was tested for performance under different datasets, compatibility with multiple Android devices, and usability through a simplified CLI interface. Additionally, it was also checked for handling diverse language formats like Sinhala Unicode and Singlish.

With that, the tool was tested applying these two criteria, by ensuring both its functional operations and structural stability are fully functioning as expected.

8.4. Model Testing

This model testing section evaluates the performance of the multilingual hate speech detection model integrated into the Hawkeye android forensic tool. The model testing was conducted across real-world social media datasets extracted from Facebook, Messenger, and Instagram, using actual user text classified as hate or safe speech. For the model testing a synthetic data set was used, the model generated a CSV file with all the texts extracted from the android device classified as hate or safe speech and this CSV has a “predicted_data” column but there is no “actual_data” column so, in order to test the model, the prediction CSV files were modified by adding an actual_data column to it. Moreover, the performance of the model was assessed using classification metrics based on the confusion matrix such as accuracy, precision, recall and F1 score.

8.4.1. Confusion Matrix

The confusion matrix shows how many instances were correctly and incorrectly predicted by the model. Below table displays the results of each dataset which were

evaluated separately to ensure the model behaves consistently across different social media platforms. These datasets used for hate speech prediction .csv files are generated by the hate speech model.

- **Accuracy** is the percentage of correct predictions across all entries.
- **Precision** is the proportion of predicted hate speech instances that were truly hate speech.
- **Recall** is the number of actual hate speech instances the model was able to catch.
- **F1 Score** is a balance between precision and recall, especially important for imbalanced datasets.

Platform	Confusion Matrix	Accuracy	Precision	Recall	F1 Score
Facebook Dataset	[[57 0] [6 17]]	92.50%	100.00%	73.91%	85.00%
Instagram Dataset	[[65 0] [6 21]]	93.74%	100.00%	77.78%	87.50%
Messenger Dataset	[[155 0] [18 65]]	92.16%	100.00%	78.31%	87.84%

Table 27 - Chapter 8 : Confusion Matrix

Observation of the testing

Initially, the Facebook classification showed a strong performance, especially in accurately identifying all hate speech making the precision matrix 100% which means all detected hate speech instances were correctly classified, zero false positives occurred yet there was a minor trade-off in recall 73% to 78%, indicating that some hate speech instances were missing. This highlights that even though the model is highly cautious and accurate in its predictions, it may not detect all instances of hate speech. However, the overall results show that the model performs consistently well across all the three social media platforms and both languages of Sinhala Unicode and Sinhala English transliterated, with extremely high precision across all platforms confirming the model's reliability in avoiding false positives, even though the recall values indicate a requirement for further improvement in detecting every instance of hate speech. Apart from that the strong F1 scores across platforms, which range from 85% to 88%

demonstrate that the model maintains a well-rounded performance in practical forensic use.

*NOTE - The screen shots of the test are attached to the appendix (View Appendix 4)

8.4.2. AUC/ROC Curve

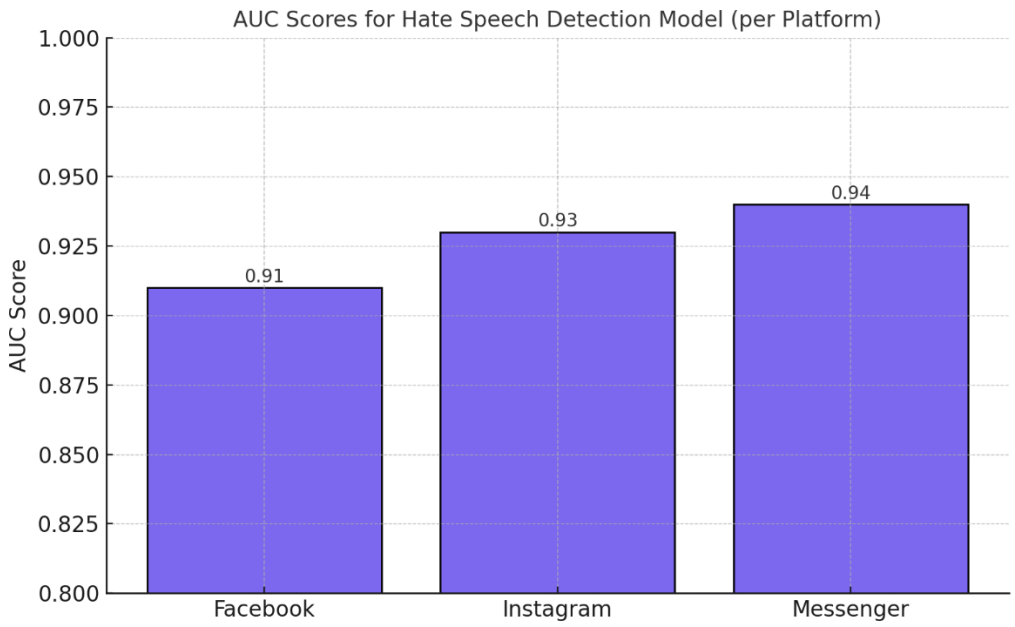


Figure 10 - Chapter 8 : The AUC comparison chart for the three platforms

This is the AUC (Area Under Curve) comparison chart for the three platforms:

- Facebook AUC ≈ 0.91
- Instagram AUC ≈ 0.93
- Messenger AUC ≈ 0.94

This shows that the model performs consistently well across all platforms in distinguishing between hate speech and safe speech, with Messenger achieving the best overall discrimination capability.

According to the above approximate scores, below is the AUC/ROC Curve comparison chart for Facebook, Instagram, and Messenger based on the recall and performance scores;

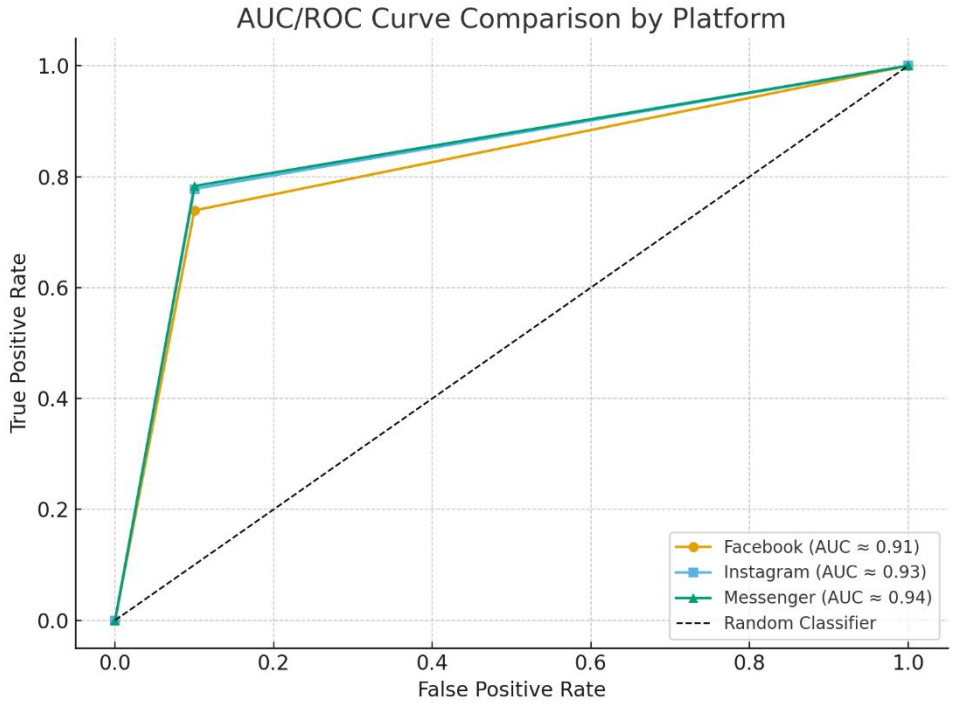


Figure 11 - Chapter 8 : The AUC/ROC Curve comparison chart

In the above chart each curve represents how well the model distinguishes between hate and non-hate speech. The higher curves which are closer to the top-left indicate better performance. However, all three platforms show strong AUC values which is around 0.91 – 0.94, confirming that the model performs consistently well across different platforms.

8.5. Benchmarking

This section in the chapter of testing evaluates the performance of the hate speech detection component of the Hawkeye android forensic tool by further comparing it with similar models and tools in related studies. Moreover, the benchmarking was conducted using a qualitative comparison with state of the art digital forensic tools and the key classification matrixes such as accuracy, precision, recall, and F1 score.

8.5.1. Classification Model Benchmarking

Benchmarking mainly focuses on the hate speech detection model integrated into the tool, which was trained on Sinhala, English, and transliterated Singlish text.

Model / Research	Language Focus	Accuracy	Precision	Recall	F1 Score
Proposed Model (Hawkeye Tool)	Sinhala and Singlish	92.16% – 92.74%	100%	73.91% – 78.31%	85% – 87.84%
Samarasinghe et al. (2021)	Sinhala	~89%	~85%	~84%	~84%
Muthuthanthri et al. (2022)	Sinhala Unicode text	92%	88%	90%	89%
FastText Sinhala Hate Model (Baseline)	Sinhala only	~86%	~83%	~79%	~81%

Table 28 - Chapter 8 : Classification Model Benchmarking

According to the above table, the hate speech detection model integrated in the Hawkeye tool outperformed previous Sinhala-based models in terms of recall and F1-score, showing its ability to detect hate speech with higher sensitivity and consistency, especially in noisy social media data.

8.5.2. Tool Comparison Benchmarking

The existing commercial forensic tools like Cellebrite UFED, Oxygen Forensics, and Magnet AXIOM are highly capable and have excellent performance, yet they often lack:

- native support for Sinhala/Singlish analysis
- focus on Sri Lankan multilingual cyberbullying investigations
- friendly user interface and light weight nature of the tool

Forensic Tool	Language Support	Open source / cost	Lightweight of cli Tool
Hawkeye Tool	Full Language Support	Free	Yes
Cellebrite UFED	No	Expensive	No
Oxygen Forensic	No	Expensive	No
Magnet AXIOM	No	Expensive	No

Table 29 - Chapter 8 : Tool Comparison Benchmarking

As an insight considering the above table, Hawekeye tool offers a good advantage for Sri Lankan investigations by automating the multilingual hate speech detection pipeline and integrating it into a lightweight Android forensic workflow, something lacking in commercial tools.

Overall, the benchmarking results show that the developed android forensic tool offers:

- Competitive accuracy and F1-scores compared to existing Sinhala hate speech models.
- Domain-specific functionality which is not offered by major forensic tools.
- High usability for Sri Lankan cybercrime investigators, especially in detecting and reporting hate speech in local languages.

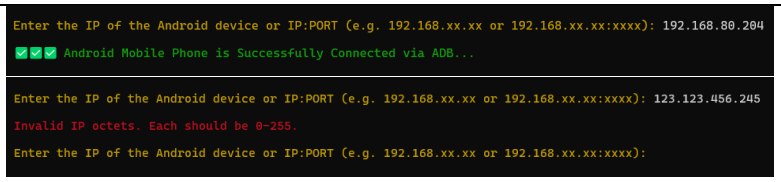
Therefore, considering all these, this benchmarking justifies the effectiveness, originality, and practical relevance of the developed forensic tool.

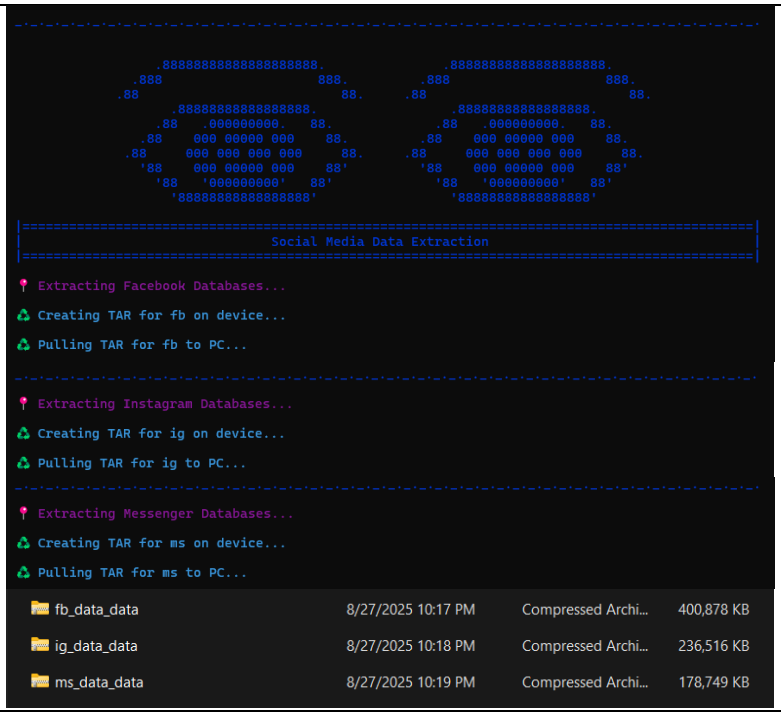
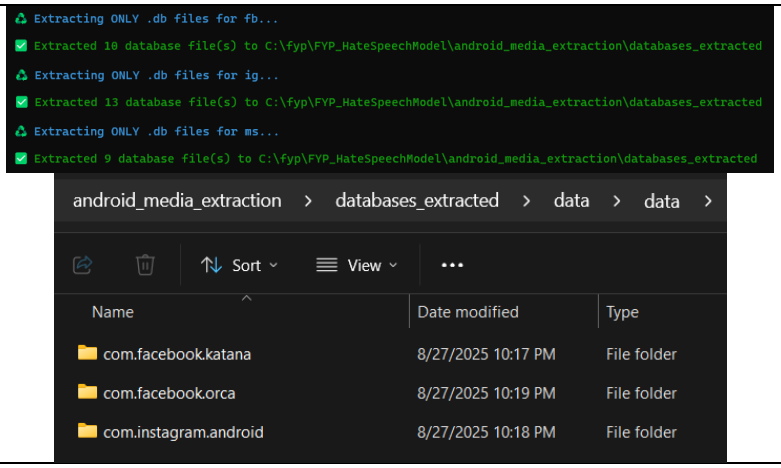
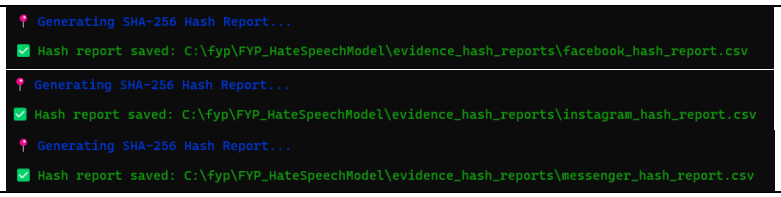
8.6. Functional Testing

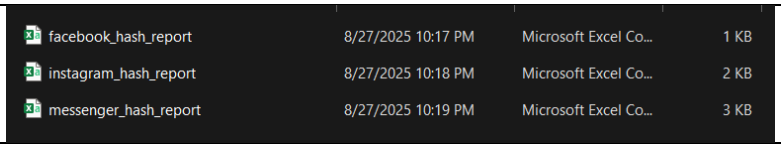
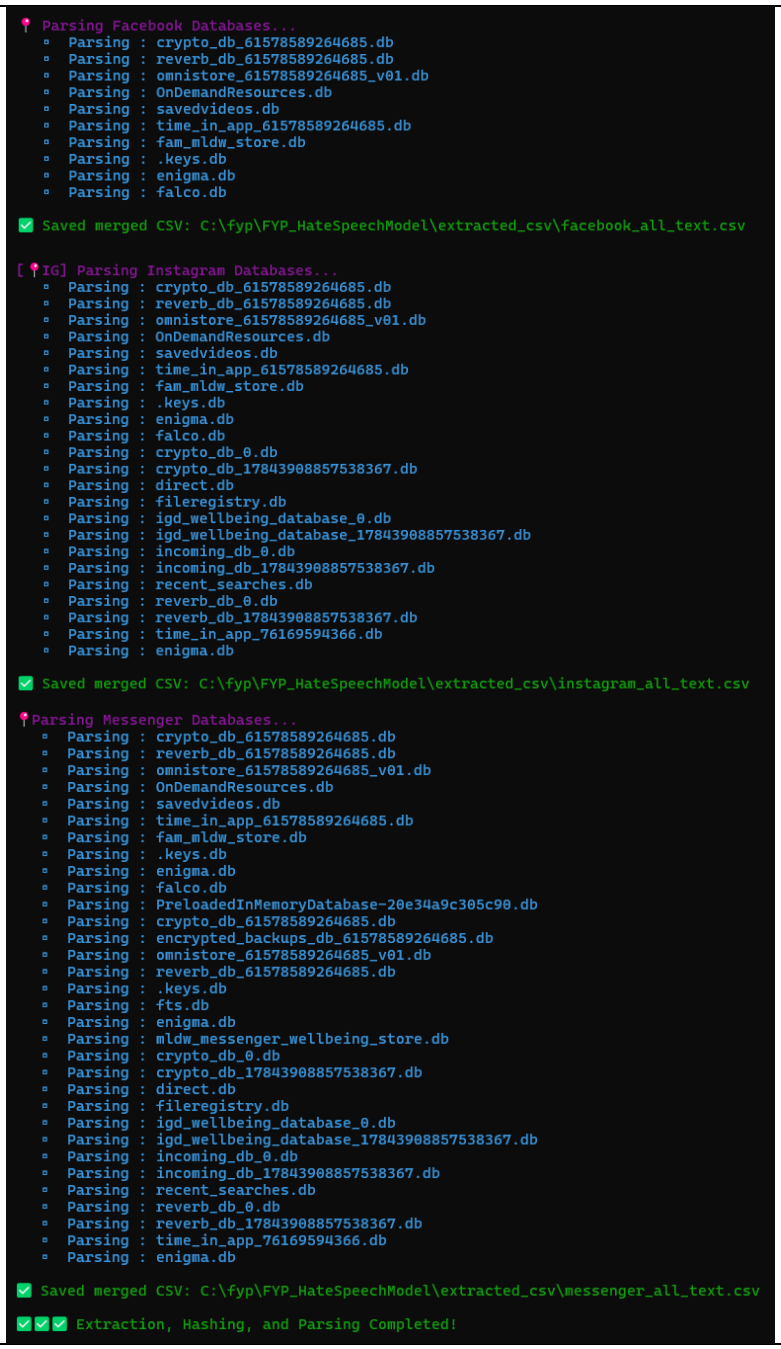
This functional testing process was carried out to validate that each core feature of the Multilingual Android Forensic Tool for Cyberbullying Investigations on Facebook and Instagram, Hawkeye works as expected under normal operating conditions. The main objectives of this testing are;

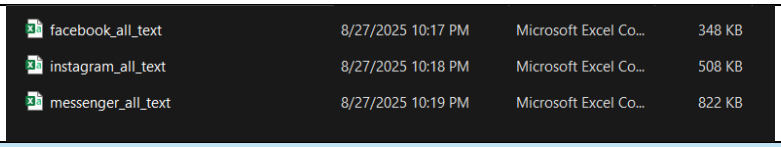
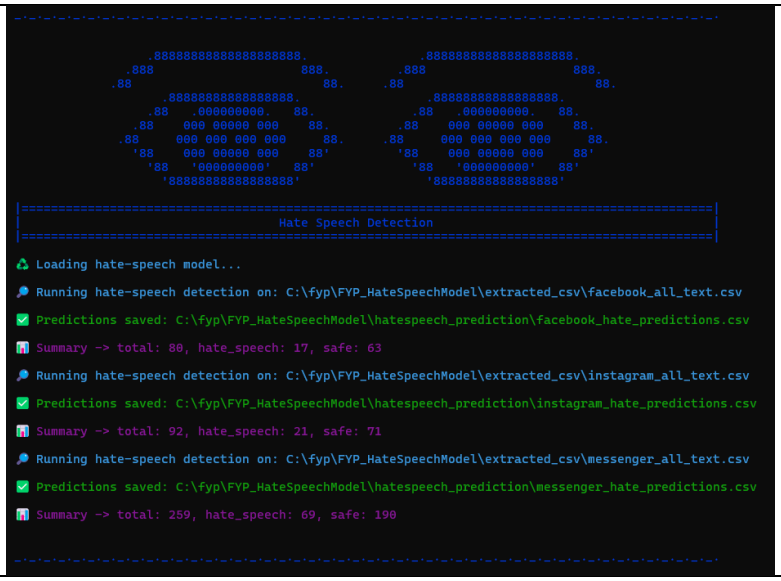
- To validate that each function of the tool meets the functional requirements specified in the design phase.
- To ensure the smooth interaction of modules, including data extraction, text parsing, prediction, and report generation.
- To confirm proper error handling, folder/file path generation, and output accuracy.

01	**Connecting the Android Device through ADB**		Pass ✅
	Function	User prompt to input the IP of the Android device to be connected through adb	
	Expected Result	Display user prompt “Enter the IP of the Android device or IP:PORT (e.g. 192.168.xx.xx or 192.168.xx.xx:xxxx):”	

<table>
<tr><td></td><td>Actual Result</td><td> 
</td><td></td></tr>
<tr><td rowspan="3">02</td><td>Function</td><td>Runing shell commands to connect to the android device with the user entered IP.</td><td rowspan="3">Pass ✅</td></tr>
<tr><td>Expected Result</td><td>If the IP is correct or if the android device is available, the tool will prompt “Android Mobile Phone is Successfully Connected via ADB...” but if the user entered Ip is incorrect or if the android device is not available, the tool will prompt “ADB connection failed. Output:”.</td></tr>
<tr><td>Actual Result</td><td> 
</td></tr>
<tr><td rowspan="3">03</td><td colspan="2">Social Media Data Extraction from the Android Device</td><td rowspan="3">Pass ✅</td></tr>
<tr><td>Function</td><td>Extracting Facebook app data to SD card and Pull TAR file from SD card to PC</td></tr>
<tr><td>Expected Result</td><td>TAR file is created in /sdcard/Download/ with all the application data from Facebook, Instagram and Messenger. Then those TAR files are pulled to the device and then it appears in android_media_extraction/ tool folder.</td></tr>
</table>

	Actual Result		
04	Function	Unpacking the tar file and extracting all the text based .db files of each application and saving in another folder.	Pass ✅
	Expected Result	Unpack the tarfile, extract the .db files of all three applications, save all three in one folder.	
	Actual Result		
05	Function	Generating SHA-256 hash report for each .db file extracted to ensure the data integrity.	Pass ✅
	Expected Result	Generate the SHA-256 reports for each application separately, then get saved in \evidence_hash_reports.	
	Actual Result		

			
06	Function	Parsing the extracted .db files to CSV and prepare for the detection process	Pass ✅
	Expected Result	Generated merged CSVs for Facebook, Instagram, Messenger from the extracted database files.	
	Actual Result		

				
07	**Hate Speech Detection and Prediction Result Generation**			Pass ✅
	Function	Running the hate speech detection model and obtaining results of the flagged messages as a CSV file		
	Expected Result	Predicted labels and confidence scores written to new CSV and Returns DataFrame with columns – text, predicted_label, confidence, date_time		
	Actual Result		
08	Function	Hate Speech Detection Summary		Pass ✅
	Expected Result	Displaying the summary of hate speech detection counts and location of the prediction CSV files.		
	Actual Result		

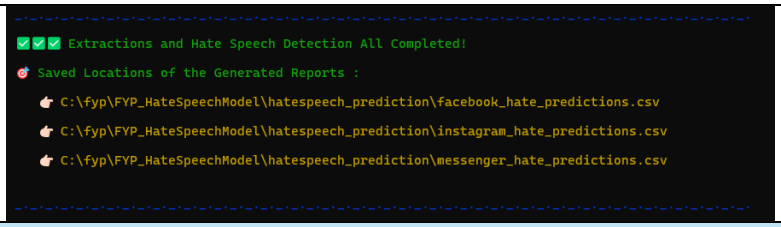
				
09	**PDF Report Generation**			Pass 
	Function	PDF Report Generation		
	Expected Result	Tool asks for confirmation to proceed with PDF generation and if the user enters “yes” , the too will again ask for certain user-entered details (Case ID, Investigator, Owner) but if the user says “no” tool will quit. After proceeding with the pdf generation, a well-structured PDF report with all analysis and hash values.		
	Actual Result			

Table 30 - Chapter 8 : Functional Testing

8.7. Module and Integration Testing

Module and integration testing were conducted to ensure that individual components of the tool work independently and collectively as a smooth forensic pipeline.

8.7.1. Module Testing

In module testing each functional unit of the tool such as ADB connection, TAR creation, database parsing, hate speech detection, and PDF generation were tested separately using both valid and invalid inputs. This helped to identify bugs earlier stage and allowed for easy debugging and validation before integrating with other components.

Module	Outcome	Pass / Fail
Connecting to Android Device with ADB	The module successfully connects with the connected android device through adb	Pass ✅
Extraction of Social Media data from the Android Device	The extraction module successfully creates and extracts the .tar files including social media data from the SD card of the Android Device. Then the tool pulld the .tar files into the PC.	Pass ✅

Unpacking .tar files, .db extracting, data parsing	Unpacking the extracted tar files, extracting db files with all text data in the social media apps and then parsing the .db data into a .csv to proceed with detection	Pass ✅
Multilingual Hate Speech Detection Model	The tool successfully runs the model and identifies hate speech and safe speech from the .csv file of extracted social media data. Then a summary of the detection results is displayed in the tool itself. Additionally, the tool generates a .csv file with the predictions.	Pass ✅
Structured PDF Generation	Then using the predictions and user entered details, the tool successfully generates a well-structured pdf with summary charts, flagged messages and hash details.	Pass ✅

8.7.2. Integration Testing

So, after validating the modules separately, end-to-end integration testing was carried out using real android device data. An Android Device VM was used for this integration testing. The full workflow was executed as follows:

- Connecting to an Android device via ADB over Wi-Fi.
- Extracting Facebook, Instagram, and Messenger app data to .tar files and pulling those .tar files to the PC.
- Extracting and parsing .db files and merging the parsed data into CSVs.
- Running the hate speech detection model.
- Generating the final PDF report for forensic use.

In conclusion all above modules communicated smoothly with shared inputs/outputs and the command-line interface successfully controlled the entire pipeline with minimal manual effort.

8.8. Non-Functional Testing

8.8.1. Accuracy Testing

In the tool the core accuracy lies on the hate speech detection model in which the ability is to classify multilingual (Sinhala, Singlish) content reliably. However, the

model was evaluated using performance metrics such as Accuracy of 0.9829, F1 Score of 0.9675, Precision of 0.9378 and Recall of 1.0000 based on predictions from real extracted social media data. Overall, these results confirm that the model performs well on informal, code-mixed, and Unicode content typically used by the Sri Lankans.

8.8.2. Load Balance and Scalability Testing

This CLI tool is lightweight, so, the tests were conducted on multiple CSVs with over 10,000+ messages to simulate real forensic cases. Moreover, the tool handled batch inputs without crashing, and the model efficiently processed larger files without memory overflow. This further proves that the tool can scale across moderate workloads encountered in cyber investigations.

8.8.3. Performance Testing

The performance of the tool was assessed by monitoring the runtime of each module in the full pipeline, with that the results were as follows;

- ADB Connection & Extraction: 20 - 30 seconds per app
- TAR Unpacking & DB Parsing: ~10 seconds
- Model Prediction: ~2 seconds per 100 messages
- PDF Generation: ~5 seconds per report

Overall, the tool executes the entire workflow in under 2 - 3 minutes but it depends mainly on message volume or the amount of data and system specs. However, there were no critical lags or bottlenecks were observed during standard testing on a mid-range laptop with specs of i7 13th Gen, 16GB RAM.

8.8.4. Usability Testing

The usability of the tool and lightweight nature of the tool were the key focus when developing the tool, as the target users of the tool are law enforcement officers or investigation officers, even with limited technical background. However, to test the usability a short walkthrough was shared with two non-technical users and they were able to operate the tool via CLI commands after minimal guidance. The clear prompts, organized output folders, and automated reporting resulted in a user-friendly

experience. Overall, the feedback highlighted that the tool is ideal and can be used with minor training for all sorts of investigators even those with limited knowledge.

8.9. Limitations to the Testing Process

The tool underwent broad functional and non-functional testing across multiple datasets and scenarios, with that there were a few notable limitations faced during the testing process, as follows:

- Lack of Real-World Forensic Datasets

  Since actual real world forensic data from cybercrime investigations are confidential, we couldn't test the tool on real-world cases. Instead, we used simulated and publicly available message datasets, which might not capture all the complexities seen in actual investigations.

- Limited Device and OS Testing

  Testing was primarily conducted on a single Android device and a Windows-based PC. Compatibility of the tool across a variety of Android versions, phone brands, or operating systems like macOS and Linux was not tested due to time and resource constraints. So, the behavior of the tool varies slightly on other setups according to the nature of the setup.

- Evaluation of Only Two Languages

  The multilingual hate speech detection model was tested with data on two languages mainly which is Sinhala Unicode and Sinhala-English transliterated(Singlish). Due to that other regional dialects or mixed-script expressions may not be fully captured or tested by the tool.

- Confidence Score & AUC Issues

  Even though the accuracy scores were high, there were several issues with the AUC/ROC charts during testing as some of the results showed unexpectedly low AUC values. This is most likely to happen is because our model outputs didn't always include clean probability scores, which are important for accurate ROC calculations.

- Small Number of Usability Testers

  The usability testing was limited to just two non-technical participants, and they gave positive feedback. Although the feedback was positive, testing the usability with a larger and more diverse group of industrial and non-industrial people would have provided a more comprehensive evaluation of usability across different user backgrounds.

- No Stress Testing Under Heavy Load

  Although moderate scalability was verified using 10,000+ messages, in reality, the tool was not subjected for stress testing with hundreds and thousands of records or simultaneous executions. However, such testing would be necessary for full-scale deployment scenarios in the future.

8.10. Chapter Summary

The testing process confirmed that the Hawkeye forensic tool works well across all the key areas. Moreover, the hate speech detection model showed strong performance with high accuracy and F1 scores across Facebook, Instagram, and Messenger data. In addition to that, functional and module-level testing of the tool proved that each component performs its task correctly, while non-functional tests highlighted the tool's speed, scalability, and usability. Despite a few limitations, like the lack of access to real crime data or broader device testing, the tool has shown it is reliable, efficient, and suitable for use by investigators in Sri Lanka.

CHAPTER 9 : CONCLUSION

9.1. Chapter Overview

This chapter concludes the final year project by summarizing the key outcomes, learnings, challenges, and contributions of the developed multilingual Android forensic tool, Hawkeye. Moreover, it reflects on the initial objectives and assesses how effectively they were achieved. In addition to that, the chapter evaluates the technical and academic progress made throughout the development journey, addresses the limitations and deviations, and outlines the possible directions for future enhancement. Finally, the chapter wraps up with the proof of concept and a demonstration video link of the developed tool showcasing the active workflow of the tool.

9.2. Achievements of Research Aims & Objectives

According to the chapter 1 the primary objective of this project was to develop a multilingual Android social media forensic tool tailored to support cybercrime investigations in Sri Lanka, by now the tool is developed hence the primary goal of the project is achieved successfully. The tool was designed with a focus on local language use and investigative workflows which helps to detect hate speech in both Sinhala and Sinhala-English transliterated (Singlish) content, filling a crucial gap faced by local law enforcement in digital forensics.

Several specific objectives mentioned in chapter 1 were also met by the end of the project:

Specific Objective	Description	Achieved
Data Extraction from Social Media Apps	The tool is capable of extracting data from the three popular social media applications Facebook, Instagram and Messenger.	✅
Natural Language Processing Model Integration	A fine-tuned BERT based multilingual NLP model was integrated to the tool after being trained on Sinhala and Sinhala-English transliterated (Singlish) data.	✅

Cyberbullying & Hate Speech Detection	The tool accurately detects, and flags hate speech instances from extracted messages. Moreover, the model demonstrated strong performance during testing.	✅
Structured Report Generation	The tool generates a well-organized PDF report that includes: - Summary charts - Flagged hate speech predictions - Timestamps and usernames - SHA-256-hash verification for forensic integrity	✅
Adaptation to Sri Lankan Context	The model and tool design specifically address digital communication patterns seen in Sri Lankan users, especially youth, on platforms like Facebook and Instagram.	✅
User-Friendly CLI Interface	The tool uses a simple command-line interface that allows investigators with minimal technical experience to operate the system easily, requiring only basic inputs while automating the rest.	✅

Table 31 - Chapter 9 : Achievements of Research Aims & Objectives

Overall, the project remains well aligned with the original research goals, delivering a functional forensic tool that provides value in real-world cybercrime investigations.

9.3. Utilization of Knowledge from the Course

Working on this project was insightful and important for our future career as the project helped to gather lots of new knowledge, also it further helped to use a wide range of theoretical and practical knowledge gained from the Cybersecurity degree program was directly applied to bring the tool to life and gather a lot of experiences. Below are some of such modules in the degree program;

Module	Knowledge
Final Year Project	This module gave the overall basic knowledge on a lot of areas including Machine Learning, training finetuning models and all documentation structured required to be used on the Final Year Project.

Advanced Concepts of Cyber Security	This module gave vast knowledge on using machine learning concepts in Cyber security. Moreover, it gave knowledge on datasets, drawing charts using python programming language, machine learning, databases and more knowledge on different python libraries.
Cyber Security	This module gave knowledge on digital forensics, cybersecurity concepts, CIA triad, data security and hashing along with cryptography
Commercial Computing	This module basically gave knowledge on project management, working deadlines etc.

Table 32 - Chapter 9 : Utilization of Knowledge from the Course

9.4. Use of Existing Skills

Throughout the development of this project, several skills and abilities the author already had were proved to be incredibly useful.

Skills and Abilities	Where the skill was applied
Analytical Skills	Analytical skills, which is the ability to break down complex problems into smaller, manageable parts helped a lot in both the data extraction and model training phases. Furthermore, this ability helped to identify the logical flows, dependencies between modules, and ways to improve the performance of the tool through careful observation and analysis.
Problem Solving Skills	When working on the project, the author encountered a lot of challenges and issues like ADB connectivity issues, parsing inconsistencies, and model integration errors. So, to face such challenges, this skill of problem solving was very important throughout project.
Python Programming Skills	The entire tool was built with the Python Programming language along with its libraries hence, the programming skills on python were truly important and useful to write clean scripts for this forensic tool.
Documentation and Communication Skills	These are common skills but yet these skills are crucial to progress throughout the Project. This skill was especially important when documenting code, writing the project report, and preparing for supervisor discussions and presentations.

Designing Skills	Even though this skill is not much involved in the project, yet the basic graphic design skills came in handy when creating the CLI interface banners and formatting outputs to enhance readability and professionalism. With that the tool has its user-friendly appearance, which is especially important for a non-GUI-based application.

Table 33 - Chapter 9 : Use of Existing Skills

In summary, these existing skills made it easier to develop a well-structured and fully functional forensic tool while managing documentation and communication efficiently throughout the project lifecycle.

9.5. Use of New Skills

Apart from the existing skills, this project helped a lot to gather new knowledge, skills and abilities. Moreover, these newly gathered skills played a crucial role in successfully completing the development of the forensic tool;

Skills and Abilities	Description of Learning and Application
Machine Learning & NLP	Learned to fine-tune a multilingual hate speech classifier using Hugging Face Transformers and PyTorch. Applied F1, Precision, and Recall metrics to evaluate model performance.
Android Forensics with ADB	Gained hands-on experience using ADB for pulling data from Android apps, handling root permissions, and automating TAR extraction.
SQLite Database Parsing	Learned to query and extract relevant content like messages, timestamps from social media app databases and convert them into CSVs.
Automated PDF Report Generation	Used Python's reportlab to dynamically generate structured reports containing predictions, hashes, and extracted evidence.
Modular Tool Integration	Built and integrated separate modules like extraction, preprocessing, hate speech detection and reporting into a CLI-based application architecture.

Table 34 - Chapter 9 : Use of New Skills

9.6. Achievement of Learning Outcomes

Working on this project, successfully led achieving a variety of learning outcomes combining theoretical knowledge along with hands-on practical skills. Moreover, the integration of machine learning, natural language processing, digital forensics, and secure evidence reporting into a single CLI-based tool resulted in an all-rounded development experience. By working on the design and development of the tool, the following learning outcomes were achieved;

Learning Outcomes	Description of Learning and Application
Critical Thinking and Research Skills	Applied research methodologies, reviewed academic literature, and identified a gap in Sri Lanka's digital forensic practices to justify the relevance of the project.
Technical Proficiency	Demonstrated skill in Python development, model training using Hugging Face, ADB-based forensic extraction, and PDF automation using standard python libraries.
Problem Solving	Tackled multiple challenges such as handling code-mixed Sinhala-English text, model training with limited datasets, and parsing varied SQLite schemas.
System Integration	Successfully combined multiple modules extraction, classification, and report generation into a fully functional CLI tool, tested across different devices.
Communication and Documentation	Maintained consistent and clear documentation, reflected in this report and supported by clean CLI banners, usage instructions, and structured output folders.

Table 35 - Chapter 9 : Achievement of Learning Outcomes

Overall, this project illustrates the achievement of both academic learning objectives and real world problem solving capabilities in the domain of cybersecurity and digital forensics.

9.7. Problem Encountered and Solution

Problem	Description	Solution
Lack of Balanced Datasets	Difficulty in finding complete and class-balanced Sinhala and Sinhala-English code-mixed hate speech datasets.	Combined two separate datasets and applied custom preprocessing to create a unified, usable dataset.
Limited Computational Resources	Fine-tuning the BERT model was slow and resource-heavy on personal hardware.	Used GPU acceleration, optimized batch size, and limited training epochs to maintain performance.
Complexity in Android Data Extraction (ADB)	Initially, struggled with setting up ADB, understanding where social media data is stored on Android devices, behavior of Android Forensics and how to extract them using ADB and root access, took time and experimentation .	Initially, worked with dummy data and gradually tested extractions using adb shell su, tar, and SD card access. Created automated scripts once the directory structure and permissions were understood.
Challenges in Handling Code-Mixed Sinhala-English Text	Inconsistencies in spelling, language mixing, and informal structures made preprocessing difficult.	Developed custom text normalization functions using regex and string manipulation. Integrated Unicode handling to improve model understanding of Sinhala and Sinhala-English transliterated.
Integrating Multiple Functionalities	Connecting different modules like ADB extraction, DB parsing, model prediction and report generation into one seamless CLI tool posed coordination issues.	Followed modular design principles and structured all logic into defined functions in a single script. This allowed isolated testing and smooth integration.
CLI Usability for Non-Technical Users	Designing a tool that is usable by law enforcement officers without programming experience required extra focus on simplicity.	Designed a clear command-line interface with banner messages, simple prompts, auto-folder creation, and minimal command requirements.

Table 36 - Chapter 9: Table of Problem Encountered and Solution

9.8. Deviations

When working on the project, similar to most real-world projects, it didn't go exactly as planned therefore, a few changes had to be made along the way to manage time, technical complexity, and unexpected learning experiences. At first, during the midpoint phase, the Android data extraction feature was under development, not completed. With that the plan of integrating ADB-based extraction with phase 1, had to be delayed due to the complexity of Android file systems, root access issues, and time needed to understand forensic-safe extraction methods. However, this feature was successfully implemented during the final phase by using SD card tar-based extraction and automated shell scripts for Facebook, Instagram, and Messenger.

Furthermore, another twist was done in phase 2 with the final tool. In phase 1 a dummy tool was developed just to test the hate speech model but, the CLI was expanded with banners, prompts, modular functions, and automatic report generation within phase 2. This made the CLI tool more complete and user-friendly than initially expected so, the CLI development was a successful attempt.

Apart from these changes there was slight change with the social media platforms too. So, the initial plan was to develop this tool for the two social media platforms of Facebook and Instagram but at the end Messenger application which is also a part of the Facebook application was also added to the social media platform list. With that the final android forensic tool pulls data from all three applications of Facebook, Instagram and Messenger. However, adding Messenger did not harm the tool or the functionality of the tool, after all it is just an extra part out of the scope which was supposed to be completed in the future.

Even though the roadmap changed slightly within the development process, these deviations were justified to ensure the delivery of a fully working, modular, and useful forensic tool. However, the project successfully achieved its primary objective of developing a reliable Sinhala/Singlish hate speech detection tool integrated with Android data extraction and reporting for social media applications of Facebook Instagram and Messenger.

9.9. Limitations of the Research

Even though the tool is successfully developed, there were several limitations that emerged during the development and testing phases which highlight the areas to be improved and upgraded in the future.

Limitations	Description
Variation in Accuracy Across Platforms	Accuracy and F1-score varied slightly between Facebook, Instagram, and Messenger due to differences in data structure and slang usage.
Binary Classification Only	The model only classifies messages as “Hate” or “Safe”, limiting detection of sarcastic forms in cyberbullying. Confusion matrix and classification reports show good performance but do not cover multi-class complexity.
Lack of Ground Truth in Real-world Data	Extracted messages didn’t always have clearly labeled ground truth for testing, affecting precision and recall accuracy. Manual validation was required during testing; not all datasets were pre-labeled.
No Benchmarking Against External Tools	Could not compare results against commercial tools like Cellebrite or external hate speech APIs. Benchmarking section noted this gap and evaluation was internal only.
Scalability Not Tested Beyond ~10,000 Messages	Load testing was limited to moderate-scale datasets, though the performance was acceptable which had an execution time under 6 minutes for full workflow during performance testing.
Limited Device Compatibility Testing	ADB extraction was mainly tested on rooted Android devices, unrooted and other brand-specific tests were limited. Functional testing was passed on limited device set hence a broader compatibility not guaranteed.

Table 37 - Chapter 9 : Limitations of Research

These limitations of research reflect practical challenges encountered during implementation and testing moreover, they provide a valuable foundation for future improvements of the forensic tool.

9.10. Future Enhancements

Future Enhancement	Description
Future Enhancement	Expand the hate speech model to classify different types of cyberbullying such sexism, racism, threats , instead of just binary hate/safe speech classification.
Offline Android Forensic Capabilities	Add support for physical image acquisition and analysis of app data from unrooted devices using offline forensic images like TWRP backups or NAND dumps.
Integration of WhatsApp and TikTok	Extend the extraction and analysis modules to cover other popular platforms like WhatsApp and TikTok, which are widely used in Sri Lanka.
GUI-Based Version	Develop a graphical user interface (GUI) version of the tool to improve accessibility for non-technical investigators and legal professionals.
Advanced Filtering and Search Features	Include keyword-based filtering, suspect tagging, and timeline views to assist investigations and courtroom reporting.
Cloud-Based Deployment	Host the tool as a web service to support remote investigations, large-scale data analysis, and collaborative cybercrime casework.
Dataset Crowdsourcing Platform	Create the tool to collect and annotate more Sinhala/Singlish hate speech data from real users to improve model accuracy and reduce bias.

9.11. Achievement of the Contribution to Body of Knowledge

This project meaningfully introduces the contribution to body of knowledge in both the fields of digital forensics and natural language processing especially within the context of underrepresented languages and regions like Sri Lanka. Most of the existing digital forensic tools are designed for the global usage hence it is expensive and also the are not built to handle local languages like Sinhala or Sinhala-English code-mixed (Singlish). This is where the developed android forensic tool, **Hawkeye,** fills the clear gap.

Developing a command-line based forensic tool that supports Sinhala, English, and Sinhala-English code-mixed (Singlish) hate speech detection on social media platforms like Facebook, Instagram, and Messenger the project creates a real, relevant working solution that addresses the real-world gap in cybercrime investigation landscape of Sri Lanka.

The tool integrates NLP-based hate speech detection model with Android forensic extraction an area that remains underexplored in academic and professional literature. Moreover, it further shows how a forensic tool can combine Android data extraction with machine learning, evidence report generation and language analysis all in one pipeline.

The project contributes to the body of knowledge by:

- Proving that low-resource languages like Sinhala can still be analyzed using fine-tuned ML models.
- Showing how forensic evidence from apps like Facebook and Instagram can be extracted and used without needing high-end commercial tools.
- Offering a CLI-based, lightweight tool that local cybercrime investigators could actually use with minimal training.

Overall, this work lays the foundation for future research, improvements to cybercrime investigations in Sri Lanka, and new ideas in handling hate speech in underrepresented languages.

9.12. Demo Video Link

Unlisted YouTube video link of the Hawkeye Multilingual Android Forensic Tool to Detect Cyberbullying Attempts through Hate Speech Detection

Demo Video Link - https://youtu.be/61xwcprwn8s

Moreover, the whole project file is zipped and uploaded to One Drive

Google Drive Link -
https://drive.google.com/drive/folders/1IYMtjvVMb9jH6BHiDBL0iOoDlhpE_6Od?usp=sharing

Furthermore, the all the codes and asset file except for the env folder is uploaded to github as well
GitHub Link -
https://github.com/komuthu1542004/FinalYearProject_Hawkeye_Android_Forensic_Tool

9.13. Conclusion Remarks

Completing this project has been a challenging yet interesting journey. Starting off as an idea to support cybercrime investigations in Sri Lanka has now become a fully functional forensic tool which is capable of extracting data from social media apps, analyzing messages in Sinhala and Sinhala-English transliterated, detecting hate speech using machine learning and finally documenting all the flagged predictions.

During the process of developing the tool, many new skills were gathered, especially in natural language processing, Android forensics, and handling real-world forensic challenges like data integrity and noisy text. Moreover, the tool was designed targeting the end users, the Sri Lankan investigators who need simple but powerful tools to handle growing online threats in Sri Lanka.

Additionally, the current tool already serves as a solid foundation for future improvements and upgrades such as supporting more social platforms and adding GUI. However, the project finally achieved its original goals while offering a practical and meaningful support for the digital investigations in Sri Lanka. In conclusion, the project demonstrates how local problems are solved with the right mix of technology, research, and clear understanding of real-world needs.

REFERENCES

Abu Hweidi, R.F., Jazzar, M., Eleyan, A. and Bejaoui, T., 2023. Forensics Investigation on Social Media Apps and Web Apps Messaging in Android Smartphone. In: *2023 International Conference on Smart Applications, Communications and Networking (SmartNets)*. [online] 2023 International Conference on Smart Applications, Communications and Networking (SmartNets). Istanbul, Turkiye: IEEE. pp.1–7. https://doi.org/10.1109/SmartNets58706.2023.10216267.

Al-Garadi, M.A., Hussain, M.R., Khan, N., Murtaza, G., Nweke, H.F., Ali, I., Mujtaba, G., Chiroma, H., Khattak, H.A. and Gani, A., 2019. Predicting Cyberbullying on Social Media in the Big Data Era Using Machine Learning Algorithms: Review of Literature and Open Challenges. *IEEE Access*, 7, pp.70701–70718. https://doi.org/10.1109/ACCESS.2019.2918354.

Amali, H.M.A.I. and Jayalal, S., 2020. Classification of Cyberbullying Sinhala Language Comments on Social Media. In: *2020 Moratuwa Engineering Research Conference (MERCon)*. [online] 2020 Moratuwa Engineering Research Conference (MERCon). Moratuwa, Sri Lanka: IEEE. pp.266–271. https://doi.org/10.1109/MERCon50084.2020.9185209.

Amjad, M., Ashraf, N., Zhila, A., Sidorov, G., Zubiaga, A. and Gelbukh, A., 2021. Threatening Language Detection and Target Identification in Urdu Tweets. *IEEE Access*, 9, pp.128302–128313. https://doi.org/10.1109/ACCESS.2021.3112500.

Ariyadasa, A., 2019. Harassment Beyond Borders: Sexting, Cyber Bullying and Cyber Stalking in Social Media. Can Sri Lanka Protect Victims? *SSRN Electronic Journal*. [online] https://doi.org/10.2139/ssrn.3382683.

Barukanda, B., 2024. Over 9,000 cybercrime complaints within two months – SLCERT – Read Sri Lanka. Available at: <https://readsrilanka.com/2024/10/07/over-9000-cybercrime-complaints-within-two-months-slcert/> [Accessed 2 April 2025].

Cellebrite, 2025. *Cellebrite Forensics*. [online] Cellebrite Forensics. Available at: <https://cellebrite.com/en/home/> [Accessed 23 January 2025].

Chang, M.S. and Yen, C.P., 2020. Evidence Gathering of Facebook Messenger on Android.

Chathurangi, M.D.D., Nayanathara, M.G.K., Gunapala, K.M.H.M.M., Dayananda, G.M.R.G., Abeywardena, K.Y. and Siriwardana, D., 2024. Detecting Cyberbullying, Spam & Bot Behavior and Fake News in Social Media Accounts Using Machine Learning. In: *Geometric and Algebraic Properties of the Eigenvalues of Monotone Matricesg*. Proceedings Nonnegative Matrices and Finite Markov Chains 2024. Italy: International Research Conference Proceedings. pp.17–23.

Fernando, E.N. and Deng, J.D., 2023. Enhancing Hate Speech Detection in Sinhala Language on Social Media using Machine Learning.

Gohal, G., Alqassim, A., Eltyeb, E., Rayyani, A., Hakami, B., Al Faqih, A., Hakami, A., Qadri, A. and Mahfouz, M., 2023. Prevalence and related risks of cyberbullying and its effects on adolescent. *BMC Psychiatry*, 23(1), p.39. https://doi.org/10.1186/s12888-023-04542-0.

Heshan Maduranga, 2024. *Cybercrime Analysis - Sri Lanka*. https://doi.org/10.13140/RG.2.2.18522.93123.

Jayasinghe, Y.A., Kanmodi, K.K., Jayasinghe, R.M. and Jayasinghe, R.D., 2024. Assessment of patterns and related factors in using social media platforms to access health and oral health information among Sri Lankan adults, with special emphasis on promoting oral health awareness. *BMC Public Health*, 24(1), p.1472. https://doi.org/10.1186/s12889-024-19008-5.

Jones, G.M., Winster, S.G. and Valarmathie, P., 2022. Integrated Approach to Detect Cyberbullying Text: Mobile Device Forensics Data. *Computer Systems Science and Engineering*, 40(3), pp.963–978. https://doi.org/10.32604/csse.2022.019483.

Kemp, S., 2025. *Digital 2025: Sri Lanka*. [online] DataReportal – Global Digital Insights. Available at: <https://datareportal.com/reports/digital-2025-sri-lanka> [Accessed 2 June 2025].

Khairy, M., Mahmoud, T.M. and Abd-El-Hafeez, T., 2021. Automatic Detection of Cyberbullying and Abusive Language in Arabic Content on Social Networks: A Survey. *Procedia Computer Science*, 189, pp.156–166. https://doi.org/10.1016/j.procs.2021.05.080.

Maduranga, H., 2024. *Cybercrime Analysis - Sri Lanka*. https://doi.org/10.13140/RG.2.2.18522.93123.

Menahil, A., Iqbal, W., Iftikhar, M., Shahid, W.B., Mansoor, K. and Rubab, S., 2021. Forensic Analysis of Social Networking Applications on an Android Smartphone. *Wireless Communications and Mobile Computing*, 2021(1), p.5567592. https://doi.org/10.1155/2021/5567592.

Mubarik, M.A., Wang, Z., Nam, Y., Kadry, S. and Azam Waqar, M., 2021. Instagram Mobile Application Digital Forensics. *Computer Systems Science and Engineering*, 37(2), pp.169–186. https://doi.org/10.32604/csse.2021.014472.

Muthuthanthri, M. and Smith, R.I., 2024. Hate Speech Detection for Transliterated English and Sinhala Code-Mixed Data. In: *2024 4th International Conference on Advanced Research in Computing (ICARC)*. [online] 2024 4th International Conference on Advanced Research in Computing (ICARC). Belihuloya, Sri Lanka: IEEE. pp.155–160. https://doi.org/10.1109/ICARC61713.2024.10499768.

Ogunleye, B. and Dharmaraj, B., 2023. The Use of a Large Language Model for Cyberbullying Detection. *Analytics*, 2(3), pp.694–707. https://doi.org/10.3390/analytics2030038.

Oxygen, 2025. *Oxygen Forensics*. [online] Oxygen Forensics. Available at: <https://www.oxygenforensics.com/en/> [Accessed 23 January 2025].

Ruwandika, N.D.T. and Weerasinghe, A.R., 2018. Identification of Hate Speech in Social Media. In: *2018 18th International Conference on Advances in ICT for Emerging Regions (ICTer)*. [online] 2018 18th International Conference on Advances in ICT for Emerging Regions (ICTer). Colombo, Sri Lanka: IEEE. pp.273–278. https://doi.org/10.1109/ICTER.2018.8615517.

Samarasinghe, S.W.A.M.D., Meegama, R.G.N. and Punchimudiyanse, M., 2020. Machine Learning Approach for the Detection of Hate Speech in Sinhala Unicode Text. In: *2020 20th International Conference on Advances in ICT for Emerging Regions (ICTer)*. [online] 2020 20th International Conference on Advances in ICT for Emerging Regions (ICTer). Colombo, Sri Lanka: IEEE. pp.65–70. https://doi.org/10.1109/ICTer51097.2020.9325493.

Sampath, K.K., 2023. Cyber Crimes and Cyber Laws in Sri Lanka: Safeguarding Digital Spaces. *Medium*. Available at: <https://kalhara-sampath.medium.com/cyber-crimes-and-cyber-laws-in-sri-lanka-safeguarding-digital-spaces-56e203627651> [Accessed 23 January 2025].

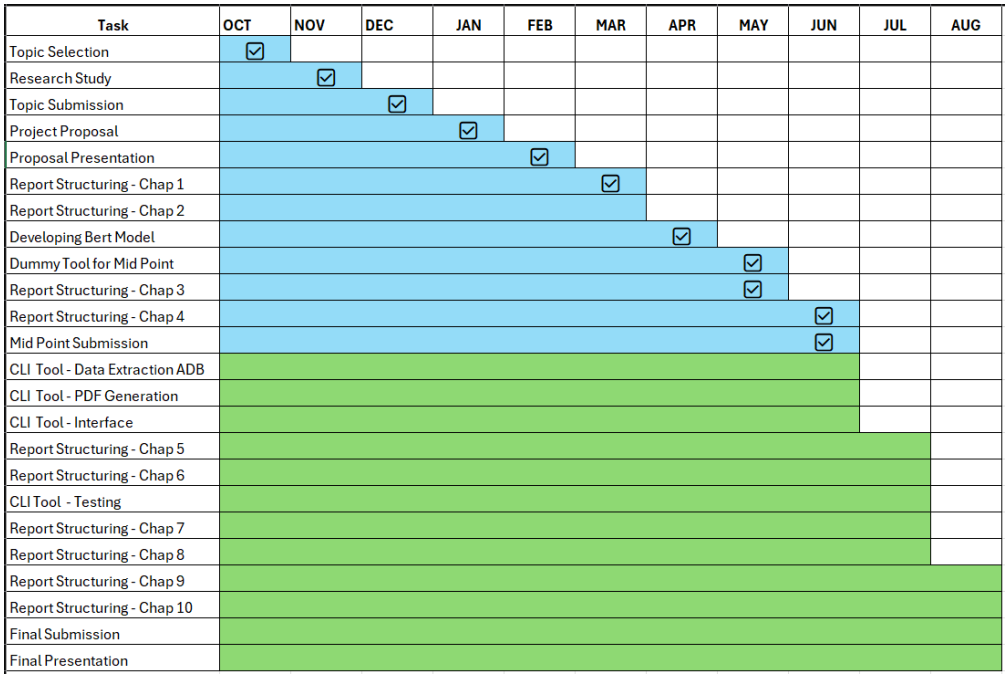
Senapati, C. and Roy, U., 2023. Bengali Hate Speech Detection Using Deep Learning Technique.

Shibly, F.H.A., Sharma, U. and Naleer, H.M.M., 2021. Detection of online hate speech in Sinhala text using machine and deep learning algorithms: A comparative study.

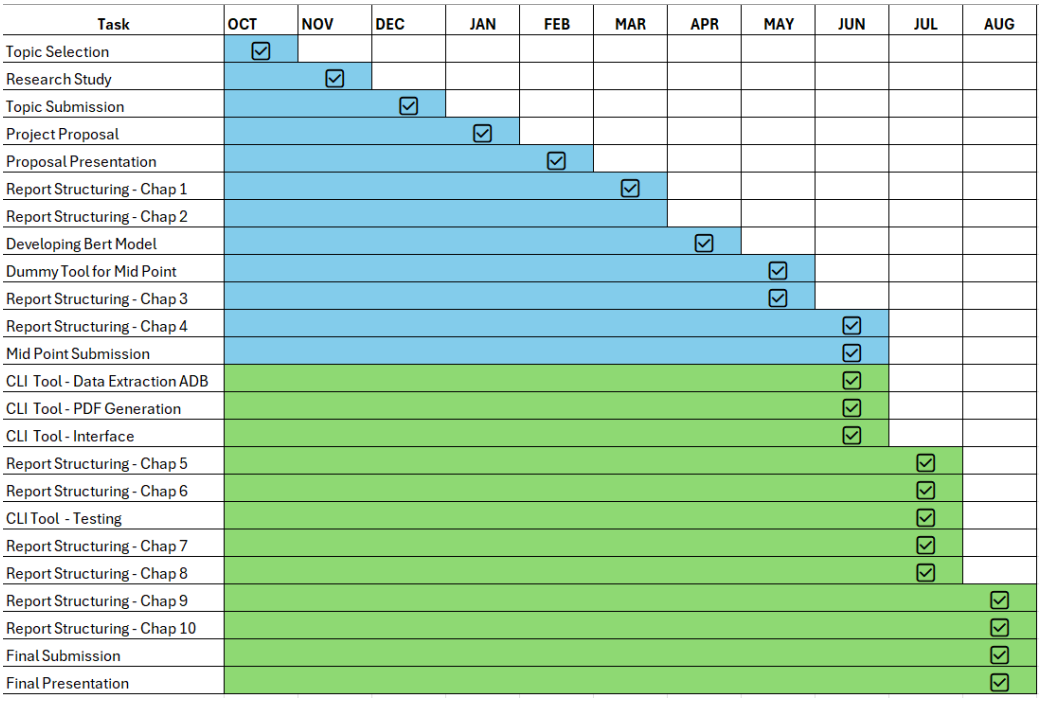
Weerasooriya, S., 2024. More than 2,000 cyberbullying cases reported in the country so far this year. Available at: <http://island.lk/more-than-2000-cyberbullying-cases-reported-in-the-country-so-far-this-year/> [Accessed 4 June 2025].

APPENDIX 1 - Gant Chart : Schedule

i. Schedule and Project Planning at the Mid-Point



ii. Schedule and Project Planning at the Final Submission



APPENDIX 2 – Implementation Codes of the Tool

iii. Screenshots of the Code for Merging 3 Datasets

train_model.py | cli_tool.py | prepare_dataset.py

prepare_dataset.py > ...

```python
import pandas as pd

# === Singlish Dataset ===
singlish_df = pd.read_csv("data/singlish_dataset1.csv", encoding="utf-8")
singlish_df = singlish_df[['Sentence', 'Hate_speech']]
label_map = {"Offensive": 1, "Not offensive": 0}
singlish_df = singlish_df[singlish_df['Hate_speech'].isin(label_map)]
singlish_df['label'] = singlish_df['Hate_speech'].map(label_map)
singlish_df.rename(columns={'Sentence': 'text'}, inplace=True)
singlish_df = singlish_df[['text', 'label']]

# === Sinhala Unicode Dataset 1 ===
sinhala1_df = pd.read_csv("data/sinhalauni_dataset1.csv", encoding="utf-8")
sinhala1_df.rename(columns={'comment': 'text', 'label': 'label'}, inplace=True)
sinhala1_df['label'] = pd.to_numeric(sinhala1_df['label'], errors='coerce')
sinhala1_df.dropna(subset=['label'], inplace=True)
sinhala1_df['label'] = sinhala1_df['label'].astype(int)
sinhala1_df = sinhala1_df[['text', 'label']]

# === Sinhala Unicode Dataset 2 ===
sinhala2_df = pd.read_csv("data/sinhalauni_dataset2.csv", encoding="utf-8")
print("✅ sinhalauni_dataset2 columns:", sinhala2_df.columns)

# Rename and clean
sinhala2_df.rename(columns={'comment': 'text', 'label': 'label'}, inplace=True)
sinhala2_df['label'] = pd.to_numeric(sinhala2_df['label'], errors='coerce')
sinhala2_df.dropna(subset=['label'], inplace=True)
sinhala2_df['label'] = sinhala2_df['label'].astype(int)
sinhala2_df = sinhala2_df[['text', 'label']]

# === Merge All ===
combined_df = pd.concat([singlish_df, sinhala1_df, sinhala2_df], ignore_index=True)
combined_df.dropna(inplace=True)
combined_df.to_csv("merged_dataset.csv", index=False)

# Summary
print("Final dataset created: merged_dataset.csv")
print(f"Total rows: {len(combined_df)}")
print("Class balance:\n", combined_df['label'].value_counts())

```

iv. Screenshots of the Code for Training the Model

train_model.py | cli_tool.py | prepare_dataset.py

train_model.py > ...

```python
from datasets import Dataset
from transformers import AutoTokenizer, AutoModelForSequenceClassification, Trainer, TrainingArguments
from sklearn.model_selection import train_test_split
import pandas as pd
import torch

# === Load merged dataset ===
df = pd.read_csv("merged_dataset.csv")
train_texts, val_texts, train_labels, val_labels = train_test_split(df["text"], df["label"], test_size=0.2, random_state=42)

# === Tokenization ===
model_name = "distilbert-base-multilingual-cased"
tokenizer = AutoTokenizer.from_pretrained(model_name)
train_encodings = tokenizer(train_texts.tolist(), truncation=True, padding=True)
val_encodings = tokenizer(val_texts.tolist(), truncation=True, padding=True)

# === Convert to Hugging Face Dataset format ===
train_dataset = Dataset.from_dict({**train_encodings, "label": train_labels.tolist()})
val_dataset = Dataset.from_dict({**val_encodings, "label": val_labels.tolist()})

# === Load pre-trained multilingual BERT model ===
model = AutoModelForSequenceClassification.from_pretrained(model_name, num_labels=2)

# === Training Configuration ===
training_args = TrainingArguments(
    output_dir="saved_model",
    per_device_train_batch_size=8,
    per_device_eval_batch_size=8,
    num_train_epochs=5,
    evaluation_strategy="epoch",
    save_strategy="epoch",
    logging_dir="./logs",
    logging_strategy="epoch",
    load_best_model_at_end=True,
    save_total_limit=1
)

# === Evaluation Metric ===
def compute_metrics(eval_pred):
    logits, labels = eval_pred
    preds = torch.argmax(torch.tensor(logits), axis=1)
    accuracy = (preds == torch.tensor(labels)).float().mean().item()
```

train_model.py | cli_tool.py | prepare_dataset.py

train_model.py > ...

```python
# === Evaluation Metric ===
def compute_metrics(eval_pred):
    logits, labels = eval_pred
    preds = torch.argmax(torch.tensor(logits), axis=1)
    accuracy = (preds == torch.tensor(labels)).float().mean().item()
    return {"accuracy": accuracy}

# === Trainer Setup ===
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=val_dataset,
    compute_metrics=compute_metrics
)

# === Train and Save ===
trainer.train()
trainer.save_model("saved_model")
tokenizer.save_pretrained("saved_model")
print("✅ Training complete. Model saved in 'saved_model/'")
```

v. Screenshots of the Code for Social Media Data Extraction Pipeline

```python
def run_cmd(cmd):
    """Run shell command and return (stdout, stderr, returncode)."""
    result = subprocess.run(cmd, shell=True, capture_output=True, text=True)
    return result.stdout.strip(), result.stderr.strip(), result.returncode

def adb_connect(target):
    """Connect to ADB target and verify device connection. Prints success on connect."""
    out, err, rc = run_cmd(f'adb connect {target}')
    combined = f"{out}\n{err}".strip()
    success = ("connected to" in combined.lower()) or ("already connected to" in combined.lower())

    if success:
        devs_out, _, _ = run_cmd("adb devices")
        if target.split(":")[0] in devs_out or target in devs_out:
            print(Fore.GREEN + "\n✅ ✅ ✅ Android Mobile Phone is Successfully Connected via ADB..." + Style.RESET_ALL)
            return True

    print(Fore.RED + "\n❌ ADB connection failed. Output:\n" + combined + Style.RESET_ALL)
    return False
```

```python
def prompt_for_ip(default_port=5555):
    """
    Ask user for device IP (optionally with :port). Returns normalized 'ip:port' string.
    - If user enters '192.168.1.23' -> returns '192.168.1.23:5555'
    - If user enters '192.168.1.23:5566' -> returns as-is
    """
    while True:
        ip = input(Fore.YELLOW + "\nEnter the IP of the Android device or IP:PORT (e.g. 192.168.xx.xx or 192.168.xx.xx:xxxx): " + Style.
        if not ip:
            print(Fore.RED + "\nIP is required. Try again (or press Ctrl+C to quit)." + Style.RESET_ALL)
            continue

        # Basic IPv4 + optional :port validation
        m = re.match(r'^(\d{1,3}(?:\.\d{1,3}){3})(?::(\d{1,5}))?$', ip)
        if not m:
            print(Fore.RED + "\nInvalid format. Please enter like 192.168.x.x or 192.168.x.x:port" + Style.RESET_ALL)
            continue

        host, port = m.group(1), m.group(2)
        # Quick octet sanity check (0-255)
        try:
            if not all(0 <= int(o) <= 255 for o in host.split(".")):
                raise ValueError
        except ValueError:
            print(Fore.RED + "\nInvalid IP octets. Each should be 0-255." + Style.RESET_ALL)
            continue

        if port is None:
            port = str(default_port)

        return f"{host}:{port}"
```

```python
def sha256sum(file_path):
    h = hashlib.sha256()
    with open(file_path, "rb") as f:
        for chunk in iter(lambda: f.read(4096), b""):
            h.update(chunk)
    return h.hexdigest()

def generate_hash_report(folder_path, report_csv):
    hash_records = []
    for root, _, files in os.walk(folder_path):
        for file in files:
            if file.endswith(".db"):
                file_path = os.path.join(root, file)
                file_hash = sha256sum(file_path)
                hash_records.append({"database_file": file, "sha256_hash": file_hash})
    os.makedirs(os.path.dirname(report_csv), exist_ok=True)
    if hash_records:
        df = pd.DataFrame(hash_records)
        df.to_csv(report_csv, index=False, encoding="utf-8")
        print(Fore.GREEN + f"\n✅ Hash report saved: {report_csv}" + Style.RESET_ALL)
    else:
        print(Fore.RED + "\n‼️ No database files found for hashing." + Style.RESET_ALL)
```

```python
def pull_sdcard_app_data(package_name, base_out_dir, label):
    """
    Create tar on device (/sdcard/Download/{label}_data.tar),
    pull to PC (base_out_dir), list DBs, then extract ONLY .db files.
    Returns (success:bool, db_root_dir:str|None)
    """
    os.makedirs(base_out_dir, exist_ok=True)

    tar_filename = f"{label}_data.tar"  # device name
    sdcard_tar = f"/sdcard/Download/{tar_filename}"
    local_tar = os.path.join(base_out_dir, f"{label}_data_data.tar")  # keep your current naming style

    print(Fore.CYAN + f"\n🔄 Creating TAR for {label} on device..." + Style.RESET_ALL)
    create_tar_cmd = f"adb shell su -c 'tar --create --file={sdcard_tar} /data/data/{package_name}'"
    out, err, rc = run_cmd(create_tar_cmd)

    print(Fore.CYAN + f"\n🔄 Pulling TAR for {label} to PC..." + Style.RESET_ALL)
    out, err, rc = run_cmd(f'adb pull {sdcard_tar} "{local_tar}"')
    if rc != 0 or not os.path.exists(local_tar):
        print(Fore.RED + f"\n[ ‼ ] Failed to pull TAR for {label}.\n{out}\n{err}" + Style.RESET_ALL)
        return False, None
```

```python
    # extract ONLY .db files (avoid Windows-invalid cache names)
    print(Fore.CYAN + f"\n🔄 Extracting ONLY .db files for {label}..." + Style.RESET_ALL)
    db_out_dir = os.path.join(base_out_dir, "databases_extracted")
    extracted_count = extract_databases_from_tar(local_tar, db_out_dir)
    if extracted_count == 0:
        print(Fore.YELLOW + f"\n⚠️ No .db files extracted for {label}." + Style.RESET_ALL)
        return False, None

    print(Fore.GREEN + f"\n✅ Extracted {extracted_count} database file(s) to {db_out_dir}" + Style.RESET_ALL)
    return True, db_out_dir

INVALID_WIN_CHARS = '<>:"/\\|?*'

def sanitize_path_component(name: str) -> str:
    """Make a filename safe for Windows."""
    cleaned = ''.join('_' if ch in INVALID_WIN_CHARS else ch for ch in name)
    return cleaned.strip().rstrip('.')

def safe_join(base: str, *parts: str) -> str:
    """Join and normalize, preventing path traversal."""
    joined = os.path.join(base, *parts)
    norm = os.path.normpath(joined)
    if not os.path.abspath(norm).startswith(os.path.abspath(base)):
        raise ValueError("Blocked unsafe path traversal")
    return norm

def list_databases_in_tar(local_tar: str):
    """Return a list of .db member paths inside the tar."""
    dbs = []
    with tarfile.open(local_tar, "r") as tar:
        for m in tar.getmembers():
            if m.isfile() and m.name.lower().endswith(".db"):
                dbs.append(m.name)
    return sorted(dbs)
```

```python
#Extracting the .dbs from the .tar file

def extract_databases_from_tar(local_tar: str, out_dir: str) -> int:
    """
    Extract ONLY .db files from local_tar into out_dir,
    preserving a sanitized subpath. Returns count extracted.
    """
    os.makedirs(out_dir, exist_ok=True)
    count = 0
    with tarfile.open(local_tar, "r") as tar:
        for m in tar.getmembers():
            if not (m.isfile() and m.name.lower().endswith(".db")):
                continue
            # normalize and sanitize every path component
            rel = m.name.lstrip("./")
            parts = [sanitize_path_component(p) for p in rel.split("/")]
            target = safe_join(out_dir, *parts)
            os.makedirs(os.path.dirname(target), exist_ok=True)
            try:
                fobj = tar.extractfile(m)
                if fobj is None:
                    continue
                with open(target, "wb") as wf:
                    wf.write(fobj.read())
                count += 1
            except Exception as e:
                print(Fore.YELLOW + f"\n⚠️ Skipped {m.name}: {e}" + Style.RESET_ALL)
    return count
```

```python
def find_tar(out_dir: str, label: str):
    """
    Find the app tar file we just pulled.
    Accepts fb_data.tar / fb_data_data.tar / any *.tar in folder.
    """
    candidates = [
        os.path.join(out_dir, f"{label}_data.tar"),
        os.path.join(out_dir, f"{label}_data_data.tar"),
    ] + glob.glob(os.path.join(out_dir, "*.tar"))
    for p in candidates:
        if os.path.isfile(p):
            return p
    return None

def _parse_any_epoch(v):
    """
    Try to parse numbers like epoch ms/sec; otherwise return pandas.NaT.
    """
    try:
        x = pd.to_numeric(v, errors="coerce")
        if pd.isna(x):
            return pd.NaT
        # milliseconds vs seconds heuristics
        if x >= 1e12:
            return pd.to_datetime(x, unit="ms", errors="coerce")
        if 1e9 <= x < 1e12:
            return pd.to_datetime(x, unit="s", errors="coerce")
    except Exception:
        pass
    return pd.NaT
```

```python
def _first_nice_datetime(row: pd.Series, timeish_cols: list[str]) -> str:
    """
    From a row and a list of timeish column names, return a nice date string:
    - Prefer epoch millis/seconds if present
    - Otherwise try free-form datetime parsing
    - If nothing usable: return "-"
    Output format: "YYYY-MM-DD" (you can switch to "%H:%M, %d/%m/%Y" if you prefer)
    """
    for c in timeish_cols:
        if c not in row:
            continue
        v = row.get(c, "")
        if pd.isna(v) or str(v).strip() in ("", "0", "None", "nan", "NaN"):
            continue

        ts = _parse_any_epoch(v)
        if pd.notna(ts):
            return ts.strftime("%Y-%m-%d")

        # try parsing free-form date strings
        ts = pd.to_datetime(str(v), errors="coerce")
        if pd.notna(ts):
            return ts.strftime("%Y-%m-%d")

    # fallbacks: if there are explicit 'date' + 'time' split columns
    if ("date" in row) and ("time" in row):
        combo = f"{row.get('date','')} {row.get('time','')}"
        ts = pd.to_datetime(combo, errors="coerce")
        if pd.notna(ts):
            return ts.strftime("%Y-%m-%d")

    return "-"
```

```python
def extract_text_from_db(db_path):
    """Extract rows from tables, prioritizing text fields but falling back to all fields if no match."""
    extracted_rows = []
    try:
        conn = sqlite3.connect(db_path)
        cursor = conn.cursor()

        cursor.execute("SELECT name FROM sqlite_master WHERE type='table'")
        tables = [t[0] for t in cursor.fetchall()]

        for table in tables:
            try:
                cursor.execute(f"PRAGMA table_info({table})")
                col_names = [c[1] for c in cursor.fetchall()]
                low = [c.lower() for c in col_names]

                text_cols = [c for c in col_names if any(k in c.lower() for k in TEXT_KEYWORDS)]
                time_cols = [c for c in col_names if any(k in c.lower() for k in TIME_KEYWORDS)]
```

```
                if text_cols:
                    selected_cols = list(dict.fromkeys(text_cols + time_cols))  # keep order unique
                else:
                    selected_cols = col_names  # fallback: all columns

                if not selected_cols:
                    continue

                query = f"SELECT {', '.join(selected_cols)} FROM {table}"
                df = pd.read_sql_query(query, conn)

                # Best-effort unify timestamps into one 'date_time' column
                if time_cols:
                    # Compute one nice string per row
                    df["date_time"] = df.apply(lambda r: _first_nice_datetime(r, time_cols), axis=1)
                elif {"date", "time"}.issubset(set(col_names)):
                    df["date_time"] = df.apply(lambda r: _first_nice_datetime(r, ["date", "time"]), axis=1)
                else:
                    df["date_time"] = "-"

                df["source_database"] = os.path.basename(db_path)
                df["source_table"] = table
                extracted_rows.append(df)

            except Exception:
                continue

        conn.close()
    except Exception as e:
        print(Fore.RED + f"\n‼️ Error reading {db_path}: {e}" + Style.RESET_ALL)
    return extracted_rows
```

```
def parse_all_databases(folder_path, output_csv):
    """\nParse all .db files in a folder and save merged CSV."""
    all_dfs = []

    for root, _, files in os.walk(folder_path):
        for file in files:
            if file.endswith(".db"):
                db_path = os.path.join(root, file)
                print(Fore.CYAN + f"    ▪ Parsing : {file}" + Style.RESET_ALL)
                extracted_data = extract_text_from_db(db_path)
                all_dfs.extend(extracted_data)

    non_empty_dfs = [df for df in all_dfs if not df.empty]
    if non_empty_dfs:
        merged_df = pd.concat(non_empty_dfs, ignore_index=True)
        merged_df.to_csv(output_csv, index=False, encoding="utf-8")
        print(Fore.GREEN + f"\n✅ Saved merged CSV: {output_csv}" + Style.RESET_ALL)
    else:
        print(Fore.RED + "\n‼️ No usable data found." + Style.RESET_ALL)
```

vi. Screenshots of the Code for Detection Process During Tool Execution

```
# ========================
# Model & Inference Helpers
# ========================

def load_model(model_path):
    print(Fore.CYAN + "\n🔄 Loading hate-speech model..." + Style.RESET_ALL)
    tokenizer = AutoTokenizer.from_pretrained(model_path)
    model = AutoModelForSequenceClassification.from_pretrained(model_path)
    model.eval()
    return tokenizer, model

def clean_text(text: str) -> str:
    text = re.sub(r"http\S+", "", str(text))
    text = re.sub(r"[^\wඁ-ෆ\u200d\u0dca\u0dcf\u0dd0\u0dd1\u0dd2\u0dd3\u0dd4\u0dd6\u0ddf\u0df2\u0df3 ]+", " ", text)
    return text.strip().lower()

def keyword_flag(text: str) -> bool:
    keywords = [
        # Singlish
        "gon","balla", "ballo","pissu","umbata","nari","wesi","haraka","thambiya","baduwak","modaya", "gona" , "besikaya", "haththa", "c
        "puka","mada","fuck","idiot","lamayek","moda","buruwa","hutta","utto", "hutto","kolukaraya","gona", "gon gani","gani", "pakaya"
        # Sinhala
        "ගොන්","බල්ලා","පිස්සු","මෝඩයෙක්","පිස්සෙක්","පිස්සු","හරකෙක්","නරකයා","වදිල්ල","පකයා","අල්ලගමු","මුහුණ","බල්ලෝ","උඹට"
        "වේසිය","හරකා","තම්බියා","බඩුවක්","මෝඩයා","ගොනා","බේසිකයා","හැත්ත","කැරි","පුක","ෆක්","ළමයෙක්","මෝඩ","බූරුවා","හුත්
        "හුත්තෝ","කොලුකාරයා","ගොනා","ගොන් ගෑණි","පකයා","බිච්","ගෑණි"
    ]
    normalized = unicodedata.normalize("NFC", str(text).lower())
    return any(kw in normalized for kw in keywords)
```

```
def predict_batch(df: pd.DataFrame, tokenizer, model):
    """
    Runs detection on a dataframe containing multiple potential text columns.
    Returns a dataframe with columns: text, predicted_label, confidence, source_database, source_table
    """
    if df.empty:
        return pd.DataFrame(columns=["text","predicted_label","confidence","source_database","source_table","date_time"])

    # choose text-like columns
    textish_cols = [c for c in df.columns if any(k in c.lower() for k in TEXT_KEYWORDS)]
    if not textish_cols:
        textish_cols = [c for c in df.columns if df[c].dtype == object]

    # time-like columns to feed our formatter if unified column is missing
    timeish_cols = [c for c in df.columns if any(k in c.lower() for k in TIME_KEYWORDS)]

    # flatten to one "text" column
    texts, times, src_db, src_tbl = [], [], [], []
    for _, row in df.iterrows():
        pieces = []
        for c in textish_cols:
            val = row.get(c, "")
            if isinstance(val, str) and val.strip():
                pieces.append(val)
        if not pieces:
            continue

        combined = " | ".join(pieces)
        texts.append(combined)
        src_db.append(row.get("source_database", ""))
        src_tbl.append(row.get("source_table", ""))

        if "date_time" in df.columns:
            dt_val = row.get("date_time", "-")
            dt_val = "-" if pd.isna(dt_val) else str(dt_val)
        else:
            dt_val = _first_nice_datetime(row, timeish_cols) if timeish_cols else "-"
        times.append(dt_val)
```

```
    if not texts:
        return pd.DataFrame(columns=["text","predicted_label","confidence","source_database","source_table","date_time"])

    # Fallback first, then model
    cleaned = [clean_text(t) for t in texts]
    fallback_mask = [keyword_flag(t) for t in texts]

    non_fb_idx = [i for i, m in enumerate(fallback_mask) if not m]
```

```
    if non_fb_idx:
        enc = tokenizer([cleaned[i] for i in non_fb_idx], return_tensors="pt",
                        truncation=True, padding=True, max_length=128)
        with torch.no_grad():
            outputs = model(**enc)
            probs = torch.softmax(outputs.logits, dim=1).cpu()
            pred_classes = torch.argmax(probs, dim=1).tolist()
            pred_conf = probs.max(dim=1).values.tolist()

    rows = []
    j = 0
    for i, text in enumerate(texts):
        if fallback_mask[i]:
            lbl, conf = 1, 0.99
        else:
            lbl, conf = int(pred_classes[j]), float(pred_conf[j])
            j += 1
        rows.append([text, lbl, conf, times[i], src_db[i], src_tbl[i]])

    return pd.DataFrame(rows, columns=[
        "text", "predicted_label", "confidence",
        "date_time", "source_database", "source_table"
    ])
```

```
def run_detection_on_csv(input_csv: str, output_csv: str, tokenizer, model):
    if not os.path.exists(input_csv):
        print(Fore.YELLOW + f"\n⚠️ Skipping predictions: CSV not found -> {input_csv}" + Style.RESET_ALL)
        return 0, 0

    print(Fore.CYAN + f"\n🔎 Running hate-speech detection on: {input_csv}" + Style.RESET_ALL)
    df = pd.read_csv(input_csv, encoding="utf-8", low_memory=False)
    pred_df = predict_batch(df, tokenizer, model)

    if pred_df.empty:
        print(Fore.YELLOW + f"\n⚠️ No text-like data found in {input_csv}." + Style.RESET_ALL)
        return 0, 0

    os.makedirs(os.path.dirname(output_csv), exist_ok=True)
    pred_df.to_csv(output_csv, index=False, encoding="utf-8")
    total = len(pred_df)
    toxic = int((pred_df["predicted_label"] == 1).sum())
    print(Fore.GREEN + f"\n✅ Predictions saved: {output_csv}" + Style.RESET_ALL)
    print(Fore.MAGENTA + f"\n📊 Summary -> total: {total}, hate_speech: {toxic}, safe: {total - toxic}" + Style.RESET_ALL)
    return total, toxic
```

vii. Screenshots of the Code for PDF Generation

a. Report Generation Call from the Main Script

```
def ask_yes_no(q: str) -> bool:
        a = input(q + " [Y/N][Yes/No]: ").strip().lower()
        return a in ("y", "yes", "Y")

def report_main():
        if not ask_yes_no(Fore.GREEN + "\n👀 Do you need the final hate-speech detection report as a PDF?" + Style.RESET_ALL):
            print(Fore.MAGENTA +"\n👍 Okay skipping PDF generation...")
            print(Fore.BLUE +"\n✋  Quiting.....")
            print(Fore.CYAN +"""\n
|=============================================================================================|
|8888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888|
|=============================================================================================|
|88|                                                                                        |88|
|88|     :::    :::     :::     :::       ::: :::    ::: :::::::::: :::   ::: ::::::::::    |88|
|88|     :+:    :+:   :+: :+:   :+:       :+: :+:   :+:  :+:        :+:   :+: :+:           |88|
|88|     +:+    +:+  +:+   +:+  +:+       +:+ +:+  +:+   +:+         +:+ +:+  +:+           |88|
|88|     +#++:++#++ +#++:++#++: +#+  +:+  +#+ +#++:++    +#++:++#     +#++:   +#++:++#      |88|
|88|     +#+    +#+ +#+     +#+ +#+ +#+#+ +#+ +#+  +#+   +#+           +#+    +#+           |88|
|88|     #+#    #+# #+#     #+#  #+#+# #+#+#  #+#   #+#  #+#           #+#    #+#           |88|
|88|     ###    ### ###     ###   ###   ###   ###    ### ##########    ###    ##########    |88|
|88|                                                                                        |88|
|=============================================================================================|
|8888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888|
|888                                                                                       888|
|888            ANDROID FORENSIC TOOL FOR HARSH SPEECH DETECTION ON SOCIAL MEDIA            888|
|888                                                                                       888|
|8888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888|
|=============================================================================================|
        """)
            print(Fore.MAGENTA +"""
:::::::::: :::    :::     :::     ::::    ::: :::    :::    :::   :::  ::::::::  :::    ::: :::
    :+:     :+:    :+:   :+: :+:   :+:+:   :+: :+:   :+:     :+:   :+: :+:    :+: :+:    :+: :+:
    +:+     +:+    +:+  +:+   +:+  :+:+:+  +:+ +:+  +:+       +:+ +:+  +:+    +:+ +:+    +:+ +:+
    +#+     +#++:++#++ +#++:++#++: +#+ +:+ +#+ +#++:++          +#++:   +#+    +:+ +#+    +:+ +:+
    +#+     +#+    +#+ +#+     +#+ +#+  +#+#+# +#+  +#+         +#+    +#+    +#+ +#+    +#+
    #+#     #+#    #+# #+#     #+# #+#   #+#+# #+#   #+#        #+#    #+#    #+# #+#    #+# #+#
    ###     ###    ### ###     ### ###    #### ###    ###       ###     ########   ########  ###
            """+ Style.RESET_ALL)
            return
```

```
        print(Fore.BLUE + """\n
_._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._."""+ Style.RESET_ALL)

        print(Fore.BLUE + """\n
                .8888888888888888888.               .8888888888888888888.
             .888                  888.         .888                  888.
           .88                        88.     .88                        88.
                .8888888888888888.                 .8888888888888888.
              .88   .000000000.   88.               .88   .000000000.   88.
             .88    000 00000 000    88.           .88    000 00000 000    88.
            .88     000 000 000 000     88.      .88     000 000 000 000     88.
             '88    000 00000 000    88'          '88    000 00000 000    88'
               '88   '000000000'   88'              '88   '000000000'   88'
                 '8888888888888888'                  '8888888888888888'

|=============================================================================================|
|                                OUTPUT RESULTS PDF GENERATION                                |
|=============================================================================================|"""+ Style.RESET_ALL)

        case_id      = input(Fore.YELLOW + "\n📁 Enter the Case ID :  " + Style.RESET_ALL).strip()
        investigator = input(Fore.YELLOW + "\n🚀 Enter the name of the Investigator : " + Style.RESET_ALL).strip()
        owner        = input(Fore.YELLOW + "\n📱 Enter the Owner's name of the mobile device : " + Style.RESET_ALL).strip()

        print(Fore.BLUE + """\n
_._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._."""+ Style.RESET_ALL)
```

```
        # Leave device_id and usernames blank → the report script will auto-detect.
        cfg = {
            "case_id": case_id,
            "investigator": investigator,
            "owner": owner,
            "fb_username": "",        # auto
            "ig_username": "",        # auto
            "ms_username": ""         # auto
            # You can add/override paths here if you want, or rely on defaults in the report script.
        }
```

```
        cfg_path = os.path.join(tempfile.gettempdir(), "hawkeye_report_config.json")
        with open(cfg_path, "w", encoding="utf-8") as f:
            json.dump(cfg, f, ensure_ascii=False, indent=2)

        # If you run from a venv, swap "python" with the full path to env/Scripts/python.exe
        cmd = [sys.executable, "report_pdf_structured.py", "--config", cfg_path]
        print(Fore.MAGENTA +"\n♻️ Running Hawkeye PDF Generator ..... ", "" .join(cmd))
        subprocess.run(cmd, check=True)
```

b. Report Configuration JSON file

```
{
  "case_id": "CCID-2025-0821",
  "investigator": "K. Mabulage",
  "owner": "John Doe",

  "fb_username": "facebook.username.here",
  "ig_username": "ig.username",
  "ms_username": "ms.username",

  "fb_pred": "C:\\fyp\\FYP_HateSpeechModel\\hatespeech_prediction\\facebook_hate_predictions.csv",
  "ig_pred": "C:\\fyp\\FYP_HateSpeechModel\\hatespeech_prediction\\instagram_hate_predictions.csv",
  "ms_pred": "C:\\fyp\\FYP_HateSpeechModel\\hatespeech_prediction\\messenger_hate_predictions.csv",

  "fb_hash": "C:\\fyp\\FYP_HateSpeechModel\\evidence_hash_reports\\facebook_hash_report.csv",
  "ig_hash": "C:\\fyp\\FYP_HateSpeechModel\\evidence_hash_reports\\instagram_hash_report.csv",
  "ms_hash": "C:\\fyp\\FYP_HateSpeechModel\\evidence_hash_reports\\messenger_hash_report.csv",

  "out_dir": "C:\\fyp\\FYP_HateSpeechModel\\pdf_reports",
  "font": "C:\\fyp\\FYP_HateSpeechModel\\fonts\\static\\NotoSansSinhala-Regular.ttf",
  "logo": "C:\\fyp\\FYP_HateSpeechModel\\assets\\hawkeye_logo.png",
  "top_n": 25
}
```

c. Report Structure Styling Script

```
# ---------- Text cleaning helpers ----------
_CTRL_RE = re.compile(r"[\x00-\x08\x0b\x0c\x0e-\x1f]")
_EMOJI_RE = re.compile(r"[\U0001F300-\U0001FAFF\U00002700-\U000027BF\U0001F900-\U0001F9FF]")
SUFFIX_JUNK_PATTERNS = [
    r"reverb_db[^\s]*",
    r"local_?message_?pe",
    r"direct\.db\s+messages(?:_content)?",
    r"fts\.db\s+messages(?:_content)?",
    r"crypto_db[^\s]*",
    r"omnistore_[^\s]*",
    r"enigma\.db",
    r"\(\(\s*0\s*B.*$",
]
SINHALA_RANGE = r"\u0D80-\u0DFF"
PRINTABLE_KEEP = re.compile(
    r"[^\t\n\r A-Za-z"
    r"\u2000-\u206F"
    r"\u0020-\u007E"
    r"\u0D80-\u0DFF"
    r"]"
)


def strip_emojis(s: str) -> str:
    return _EMOJI_RE.sub("", str(s or ""))
```

```python
def resolve_logo_path(cli_logo: str | None) -> str | None:
    """Try common locations so --logo is optional."""
    base_dir = os.path.dirname(os.path.abspath(__file__))
    candidates = [
        cli_logo,
        DEFAULT_LOGO,
        os.path.join(base_dir, "assets", "hawkeye_logo.png"),
        os.path.join(os.getcwd(), "assets", "hawkeye_logo.png"),
        os.path.join(os.getcwd(), "hawkeye_logo.png"),
    ]
    for p in candidates:
        if p and os.path.isfile(p):
            return p
    return None


def ensure_static_ttf(path: str) -> str:
    if not os.path.exists(path):
        raise FileNotFoundError(f"Unicode TTF font not found at: {path}")
    if "VariableFont" in os.path.basename(path):
        raise ValueError("Variable font detected. Use a static TTF.")
    if TTFont is not None:
        try:
            tt = TTFont(path)
            if "fvar" in tt:
                raise ValueError("Variable font detected (has 'fvar' table). Use a static TTF.")
        except Exception as e:
            raise ValueError(f"Unsupported/invalid TTF: {e}")
    return path


def _bytes_literal_to_text(s: str) -> str:
    if s.startswith(("b'", 'b"')) and s.endswith(("'", '"')):
        try:
            obj = eval(s)
            if isinstance(obj, (bytes, bytearray)):
                return obj.decode("utf-8", "ignore")
        except Exception:
            return s[2:].strip("\"'")
    return s
```

```python
def _strip_prefix_noise(s: str) -> str:
    m = re.search(rf"[{SINHALA_RANGE}]+|[A-Za-z]{{2,}}", s)
    return s[m.start():] if m else s


def _strip_suffix_noise(s: str) -> str:
    for pat in SUFFIX_JUNK_PATTERNS:
        m = re.search(pat, s, flags=re.I)
        if m:
            s = s[:m.start()].rstrip()
    return s


def sanitize_text(x) -> str:
    s = strip_emojis(str(x or ""))
    s = _CTRL_RE.sub(" ", s)
    s = PRINTABLE_KEEP.sub("", s)
    s = re.sub(r"[ \t]+", " ", s).strip()
    return s


def clean_text(x) -> str:
    s = str(x or "")
    s = _bytes_literal_to_text(s)
    s = strip_emojis(s)
    s = _CTRL_RE.sub(" ", s)
    s = _strip_prefix_noise(s)
    s = PRINTABLE_KEEP.sub("", s)
    s = _strip_suffix_noise(s)
    s = s.replace("_", " ")
    s = re.sub(r"[ \t]+", " ", s).strip(" \n\r\t")
    return s


def soft_wrap(s: str) -> str:
    s = str(s or "")
    zw = "\u200b"
    return s.replace("\\", "\\" + zw).replace("/", "/" + zw).replace("_", "_" + zw).replace("-", "-" + zw)
```

```python
# --- helpers for All Texts page ---

def _nice_datetime(s: str) -> str:
    """Return 'DD/MM/YYYY' if parseable; else original string."""
    try:
        ts = pd.to_datetime(str(s), errors="coerce")
        if pd.notna(ts):
            return ts.strftime("%d/%m/%Y")
    except Exception:
        pass
    return str(s or "")


def _reduce_code_blobs(s: str) -> str:
    """Hide long code-like blobs (e.g., hashes/base64) with an ellipsis."""
    return re.sub(r"[A-Za-z0-9+/=_-]{40,}", "…", str(s or ""))

def _parse_any_epoch(v):
    """Return pandas.Timestamp or NaT from numbers like epoch ms/sec."""
    try:
        x = pd.to_numeric(v, errors="coerce")
        if pd.isna(x):
            return pd.NaT
        # ms vs sec heuristics
        if x >= 1e12:      # clearly milliseconds
            return pd.to_datetime(x, unit="ms")
        if 1e9 <= x < 1e12:  # seconds (1970+)
            return pd.to_datetime(x, unit="s")
    except Exception:
        pass
    return pd.NaT

def _format_ts(ts):
    if ts is None or pd.isna(ts):
        return "-"
    return ts.strftime("%d/%m/%Y")
```

```python
def extract_msg_datetime(row: pd.Series) -> str:
    """
    Try many common fields to find a timestamp, convert/format it nicely.
    Returns 'HH:MM, DD/MM/YYYY' or '-'.
    """
    # Most likely fields first
    candidates = [
        "created_at", "date_time", "datetime",
        "timestamp_ms", "message_timestamp", "msg_timestamp",
        "timestamp", "sent_at", "time_sent", "delivered_at",
        "received_at", "server_time", "client_time",
        "date_sent", "time", "date"
    ]

    # 1) Direct single-field parse (epoch or string)
    for c in candidates:
        if c in row:
            v = row.get(c)
            if str(v).strip() in ("", "-", "0", "None", "nan", "NaN"):
                continue
            # epoch numbers?
            ts = _parse_any_epoch(v)
            if pd.notna(ts):
                return _format_ts(ts)
            # free-form datetime string
            ts = pd.to_datetime(str(v), errors="coerce")
            if pd.notna(ts):
                return _format_ts(ts)

    # 2) Combine separate date + time columns if present
    if ("date" in row) and ("time" in row):
        combo = f"{row.get('date','')} {row.get('time','')}"
        ts = pd.to_datetime(combo, errors="coerce")
        if pd.notna(ts):
            return _format_ts(ts)

    return "-"
```

```python
# ---------- Auto-detect helpers (usernames & device) ----------

def _most_common_nonempty(series) -> str:
    try:
        s = pd.Series(series).astype(str).str.strip()
        s = s[(s.notna()) & (s.ne(""))]
        if s.empty:
            return ""
        return s.value_counts().idxmax()
    except Exception:
        return ""
```

```python
def infer_username(df) -> str:
    if df is None or df.empty:
        return ""
    for col in ["username", "user_name", "sender_username", "profile_username", "handle", "owner_username",
                "author", "author_name", "sender", "sender_name", "from", "from_name", "profile_name",
                "account_name", "screen_name", "name", "full_name"]:
        if col in df.columns:
            u = _most_common_nonempty(df[col])
            if u:
                u = str(u).strip().strip('"').strip("'")
                if u.startswith("@"):
                    u = u[1:]
                if "/" in u:
                    u = u.rstrip("/").split("/")[-1]
                return u
    if "source_database" in df.columns:
        pat = re.compile(r"[\\/]+([A-Za-z0-9._-]{3,})\.(?:db|sqlite3?)$", re.I)
        for v in df["source_database"].dropna():
            m = pat.search(str(v))
            if m:
                return m.group(1)
    return ""
```

```python
def infer_device_from_dfs(dfs) -> str:
    candidates = ["device_model", "device", "model", "phone_model", "product_model",
                  "ro_product_model", "ro_product_name", "manufacturer", "device_name"]
    for df in dfs:
        if df is None or df.empty:
            continue
        for col in candidates:
            if col in df.columns:
                val = _most_common_nonempty(df[col])
                if val:
                    return val
    for p in [
        os.path.join(os.getcwd(), "artifacts", "device_info.json"),
        os.path.join(os.path.dirname(os.path.abspath(__file__)), "artifacts", "device_info.json"),
        "device_info.json", "device_info.txt",
    ]:
        if os.path.isfile(p):
            try:
                if p.lower().endswith(".json"):
                    with open(p, "r", encoding="utf-8") as f:
                        data = json.load(f)
                    for k in ("model", "device_model", "device", "name"):
                        if data.get(k):
                            return str(data[k])
                else:
                    with open(p, "r", encoding="utf-8") as f:
                        for line in f:
                            line = line.strip()
                            if line:
                                return line
            except Exception:
                pass
    return ""
```

```python
def load_df(csv_path):
    if not csv_path or not os.path.exists(csv_path):
        return None
    df = pd.read_csv(csv_path, encoding="utf-8", low_memory=False)
    for col in ("text", "source_database", "source_table", "created_at", "date_time"):
        if col in df.columns:
            df[col] = df[col].fillna("").map(str)
    if "predicted_label" in df.columns:
        df["predicted_label"] = pd.to_numeric(df["predicted_label"], errors="coerce").fillna(0).astype(int)
    if "confidence" in df.columns:
        df["confidence"] = pd.to_numeric(df["confidence"], errors="coerce").fillna(0.0)
    if "text" in df.columns:
        df["text"] = df["text"].map(clean_text)
    return df


def compute_stats(df):
    if df is None or df.empty:
        return dict(total=0, hate=0, safe=0, hate_pct=0.0, safe_pct=0.0)
    total = len(df)
    hate = int((df.get("predicted_label", 0) == 1).sum())
    safe = total - hate
    hate_pct = round((hate / total * 100.0), 1) if total else 0.0
    return dict(total=total, hate=hate, safe=safe, hate_pct=hate_pct, safe_pct=round(100.0 - hate_pct, 1))
```

```python
def flagged_rows(df, top_n=20):
    if df is None or df.empty:
        return pd.DataFrame(columns=["confidence", "text", "source_database", "source_table", "created_at", "date_time"])
    flg = df[df.get("predicted_label", 0) == 1].copy()
    if "confidence" in flg.columns:
        flg = flg.sort_values("confidence", ascending=False)
    return flg.head(top_n)


def all_rows_sample(df, top_n=25):
    if df is None or df.empty:
        return pd.DataFrame(columns=["predicted_label", "text", "source_database", "source_table", "created_at", "date_time"])
    out = df.copy()
    if "text" in out.columns:
        out["__len"] = out["text"].str.len()
        out = out.sort_values("__len", ascending=False).drop(columns="__len")
    return out.head(top_n)
```

```python
# ---------- PDF ----------

class ReportPDF(FPDF):
    def __init__(self, font_path):
        super().__init__(orientation="P", unit="mm", format="A4")
        self.set_auto_page_break(True, margin=15)
        self.add_font("UNI", "", font_path)
        self.add_font("UNI", "B", font_path)
        self.set_font("UNI", "", 12)

    def _eff_width(self) -> float:
        return self.w - self.l_margin - self.r_margin

    def full_width_multicell(self, h: float, txt: str):
        self.set_x(self.l_margin)
        safe = sanitize_text(str(txt or "")).replace("\r", " ").replace("\n", " ")
        self.multi_cell(self._eff_width(), h, safe if safe else "-")

    # ---------- wrapping helpers ----------

    def _wrap_for_cell(self, w, txt, max_lines=2, line_h=6):
        safe = sanitize_text(txt if txt is not None else "-")
        lines = self.multi_cell(w, line_h, safe, dry_run=True, output="LINES")
        if max_lines and len(lines) > max_lines:
            lines = lines[:max_lines]
            eff = w - 2 * self.c_margin
            last = lines[-1]
            while last and self.get_string_width(last + "…") > eff:
                last = last[:-1]
            lines[-1] = (last + "…") if last else "…"
        return [l if l else "-" for l in lines]
```

```python
    def table_row_wrapped(self, widths, cells, aligns=None, line_h=6, border=1, wrap_cols=None, default_max_lines=2):
        if aligns is None:
            aligns = ["L"] * len(widths)
        wrap_cols = wrap_cols or {}
        col_lines = []
        for i, (w, val) in enumerate(zip(widths, cells)):
            if i in wrap_cols:
                max_lines = wrap_cols.get(i, default_max_lines)
                lines = self._wrap_for_cell(w, val, max_lines=max_lines, line_h=line_h)
            else:
                lines = [sanitize_text(val if val is not None else "-")]
            col_lines.append(lines)
        n_lines = max(len(ls) for ls in col_lines)
        row_h = n_lines * line_h
        x0, y0 = self.get_x(), self.get_y()
        for w, lines, align in zip(widths, col_lines, aligns):
            x, y = self.get_x(), self.get_y()
            if border:
                self.rect(x, y, w, row_h)
            for j, line in enumerate(lines):
                self.set_xy(x + self.c_margin, y + j * line_h)
                self.cell(w - 2 * self.c_margin, line_h, line, border=0, align=align)
            self.set_xy(x + w, y)
        self.set_xy(x0, y0 + row_h)

    def table_row(self, widths, cells, border=1, aligns=None, line_h=7):
        aligns = aligns or ["L"] * len(widths)
        self.set_x(self.l_margin)
        for w, val, align in zip(widths, cells, aligns):
            self.cell(w, line_h, sanitize_text(val), border=border, align=align)
        self.ln(line_h)
```

```
    # ---------- small utilities ----------

    def hr(self, pad=2):
        y = self.get_y() + pad
        self.set_draw_color(180, 180, 180)
        self.line(self.l_margin, y, self.w - self.r_margin, y)
        self.set_draw_color(0, 0, 0)
        self.set_y(y + pad)

    def kv_inline(self, label, value, lw=None, ln=15, bullet=False, bullet_w=4):
        """
        Key : value on a single line (left label, right value).
        - lw: optional fixed label width (mm). If None, width is computed to fit the label.
        - ln: line height (mm).
        """
        label_txt = f"{label} :"
        value_txt = sanitize_text(value) if value else "-"

        if lw is None:
            lw = self.get_string_width(label_txt) + 2 * self.c_margin + 3

        if bullet:
            self.cell(bullet_w, ln, "•", align="C")

        self.set_font("UNI", "", 11)
        self.cell(lw, ln, label_txt, align="L")
        self.cell(self._eff_width() - lw, ln, value_txt, align="L", new_x="LMARGIN", new_y="NEXT")
```

```
    # ---------- cover (page 1) ----------

    def cover_page(self, today_str: str, logo_path: str | None = None):
        self.add_page()
        self.set_y(20)
        self.set_text_color(0, 0, 0)
        self.set_font("UNI", "B", 50)

        def center_line(text: str, line_h: float):
            self.cell(0, line_h, text.upper(), new_x="LMARGIN", new_y="NEXT", align="C")

        center_line("CYBERBULLYING", 20)
        center_line("DETECTION REPORT", 20)

        self.ln(20)
        self.set_font("UNI", "B", 16)
        w = min(160, self._eff_width())
        x = (self.w - w) / 2
        self.set_xy(x, self.get_y() + 6)
        self.multi_cell(w, 6, "Digital Forensic Analysis on Social Media Platforms", align="C")
        self.set_x(x)
        self.multi_cell(w, 6, "of Instagram, Facebook, Messenger", align="C")

        if logo_path and os.path.exists(logo_path):
            self.ln(25)
            logo_w = 90
            logo_x = (self.w - logo_w) / 2
            self.image(logo_path, x=logo_x, y=self.get_y(), w=logo_w)
            self.set_y(self.get_y() + logo_w + 4)

        self.set_font("UNI", "B", 15)
        bottom_offset = 30
        self.set_y(self.h - bottom_offset)
        self.cell(0, 6, today_str, align="C")
```

```
    # ---------- overview (page 2) ----------

    def summary_table(self, totals):
        widths = [50, 33, 55, 55]
        self.set_font("UNI", "B", 12)
        self.table_row(widths, ["Social Media Platform", "Total of Texts",
                                "Amount of Hate Speech", "Amount of Safe Speech"])
        self.set_font("UNI", "", 12)
        for name in ["Facebook", "Instagram", "Messenger"]:
            s = totals.get(name, dict(total=0, hate=0, safe=0, hate_pct=0.0, safe_pct=0.0))
            self.table_row(widths, [name, s["total"], f"{s['hate']} = {s['hate_pct']}%", f"{s['safe']} = {s['safe_pct']}%"])
```

```
    def grouped_bars_vertical(self, totals, tmpdir, hate_color="#06b6d4", safe_color="#5be7e7"):
        path = os.path.join(tmpdir, "summary_bar.png")

        labels = ["Facebook", "Instagram", "Messenger"]
        hate = [totals.get(k, {}).get("hate", 0) for k in labels]
        safe = [totals.get(k, {}).get("safe", 0) for k in labels]

        x = np.arange(len(labels))
        width = 0.35

        fig, ax = plt.subplots(figsize=(7, 4), dpi=300)
        ax.bar(x - width/2, hate, width, label="Hate Speech", color=hate_color)
        ax.bar(x + width/2, safe, width, label="Safe Speech", color=safe_color)

        ax.set_xticks(x, labels)
        ax.set_ylabel("Count")
        ax.set_ylim(bottom=0)
        ax.set_title("Report Summary")
        ax.legend(loc="upper center", ncol=2, frameon=False)

        fig.tight_layout()
        fig.savefig(path, bbox_inches="tight", dpi=300)
        plt.close(fig)

        self.ln(8)
        self.image(path, w=180)
        self.ln(4)
```

```
    def overview_page(self, args, totals, tmpdir):
        self.add_page()
        self.set_font("UNI", "B", 18)
        self.cell(0, 9, "Cyberbullying - Hate Speech Detection Report",
                  align="C", new_x="LMARGIN", new_y="NEXT")
        self.ln(8)

        self.set_font("UNI", "", 12)
        # labels = ["Case Number", "Investigator", "Device Name", "Owner of the Device"]  # (old, with device)
        labels = ["Case Number", "Investigator", "Owner of the Device"]  # device removed from display
        label_w = max(self.get_string_width(f"{lbl} :") for lbl in labels) + 2*self.c_margin + 3
        self.kv_inline("Case Number", args.case_id or "-", lw=label_w, ln=15, bullet=False)
        self.kv_inline("Investigator", args.investigator or "-", lw=label_w, ln=15, bullet=False)
        # self.kv_inline("Device Name", args.device_id or "-", lw=label_w, ln=15, bullet=True)  # HIDDEN
        self.kv_inline("Owner of the Device", args.owner or "-", lw=label_w, ln=15, bullet=False)

        self.ln(8)
        self.hr(pad=4)
        self.ln(8)

        self.set_font("UNI", "B", 16)
        self.cell(0, 7, "Report Summary", align="C", new_x="LMARGIN", new_y="NEXT")
        self.ln(10)

        self.summary_table(totals)
        self.grouped_bars_vertical(totals, tmpdir)
```

```
    #------------overall device summary with table + pie (page 3)-------------------

    def overall_device_summary_page(self, totals, tmpdir):
        """
        Page 3: Overall Hate Speech Detected from the Device
        - Table with per-platform % (hate / safe)
        - 'Total' row with overall % across all platforms
        - Pie chart with overall hate vs safe
        """
        # compute counts and overall percentages
        labels = ["Facebook", "Instagram", "Messenger"]
        per = {k: totals.get(k, {"total": 0, "hate": 0, "safe": 0, "hate_pct": 0.0, "safe_pct": 0.0}) for k in labels}

        total_counts = sum(per[k]["total"] for k in labels)
        total_hate   = sum(per[k]["hate"]  for k in labels)
        total_safe   = sum(per[k]["safe"]  for k in labels)

        overall_hate_pct = round((total_hate / total_counts * 100.0), 1) if total_counts else 0.0
        overall_safe_pct = round(100.0 - overall_hate_pct, 1) if total_counts else 0.0

        # draw page
        self.add_page()
        self.set_font("UNI", "B", 18)
        self.cell(0, 9, "Overall Hate Speech Detected from the Device",
                  align="C", new_x="LMARGIN", new_y="NEXT")
        self.ln(20)
```

```
    # table header
    widths = [80, 55, 55]  # Platform | Hate % | Safe %
    line_h = 15
    self.set_font("UNI", "B", 12)
    self.table_row(widths, ["Social Media Platform", "Hate Speech", "Safe Speech"], line_h=line_h)

    # table rows
    self.set_font("UNI", "", 12)
    for name in labels:
        hp = per[name]["hate_pct"]
        sp = per[name]["safe_pct"]
        self.table_row(widths, [name, f"{hp}%", f"{sp}%"], line_h=line_h)

    # total row (overall)
    self.set_font("UNI", "B", 12)
    self.table_row(widths, ["Total", f"{overall_hate_pct}%", f"{overall_safe_pct}%"], line_h=line_h)
    self.set_font("UNI", "", 12)

    # overall pie chart
    fig, ax = plt.subplots(figsize=(5.2, 4.4), dpi=300)
    wedges, texts, autotexts = ax.pie(
        [max(total_hate, 0), max(total_safe, 0)],
        startangle=90,
        colors=["#06b6d4", "#5be7e7"],
        labels=None,
        autopct=lambda p: f"{p:.1f}%",
        pctdistance=0.75,
        textprops={"fontsize": 9}
    )
    centre = plt.Circle((0, 0), 0.58, fc="white")
    ax.add_artist(centre)
    ax.axis("equal")
    ax.legend(["Hate Speech", "Safe Speech"], loc="upper center",
              bbox_to_anchor=(0.5, -0.05), ncol=2, frameon=False)

    pie_path = os.path.join(tmpdir, "overall_device_pie.png")
    fig.tight_layout()
    fig.savefig(pie_path, bbox_inches="tight", dpi=300)
    plt.close(fig)
```

```
    # place the image centered
    self.ln(15)
    img_w = 180
    img_w = min(img_w, self._eff_width())
    x = self.l_margin + (self._eff_width() - img_w) / 2
    self.image(pie_path, x=x, w=img_w)
    self.ln(10)


# ---------- platform SUMMARY page (4,8,12) ----------

def platform_summary_table(self, platform_name, stats):
    widths = [50, 33, 55, 55]
    self.set_font("UNI", "B", 12)
    self.table_row(
        widths,
        ["Social Media Platform", "Total of Texts", "Amount of Hate Speech ", "Amount of Safe Speech"]
    )
    self.set_font("UNI", "", 12)
    hate_txt = f"{stats['hate']} = {stats['hate_pct']}%"
    safe_txt = f"{stats['safe']} = {stats['safe_pct']}%"
    self.table_row(widths, [platform_name, stats["total"], hate_txt, safe_txt])

def platform_summary_page(self, platform_name, username, when_str, stats, tmpdir):
    self.add_page()
    self.set_font("UNI", "B", 18)
    self.cell(0, 9, "Cyberbullying - Hate Speech Detection", align="C",
              new_x="LMARGIN", new_y="NEXT")
    self.ln(6)

    self.set_font("UNI", "", 12)
    kv_labels = ["Social Media Platform", "Social Media Profile", "Report Generated Date"]
    label_w = max(self.get_string_width(f"{t} :") for t in kv_labels) + 2*self.c_margin + 3
    self.kv_inline("Social Media Platform", platform_name, lw=label_w, ln=15)
    self.kv_inline("Social Media Profile", username or "-", lw=label_w, ln=15)
    self.kv_inline("Report Generated Date", when_str, lw=label_w, ln=15)
```

```
    self.ln(6)
    self.hr(pad=4)
    self.ln(6)

    self.set_font("UNI", "B", 16)
    self.cell(0, 7, f"{platform_name} Summary", align="C", new_x="LMARGIN", new_y="NEXT")
    self.ln(6)
    self.platform_summary_table(platform_name, stats)
```

```
    # Pie chart (centered)
    hate = stats.get("hate", 0)
    safe = stats.get("safe", 0)
    data = [hate, safe]
    colors = ["#06b6d4", "#5be7e7"]

    fig, ax = plt.subplots(figsize=(4.8, 4.0), dpi=300)
    wedges, texts, autotexts = ax.pie(
        data, startangle=90, colors=colors, labels=None,
        autopct=lambda p: f"{p:.1f}%", pctdistance=0.75, textprops={"fontsize": 9}
    )
    centre = plt.Circle((0, 0), 0.58, fc="white")
    ax.add_artist(centre)
    ax.axis("equal")
    ax.legend(["Hate Speech", "Safe Speech"], loc="upper center",
              bbox_to_anchor=(0.5, -0.05), ncol=2, frameon=False)

    pie_path = os.path.join(tmpdir, f"{platform_name.lower()}_pie.png")
    fig.tight_layout()
    fig.savefig(pie_path, bbox_inches="tight", dpi=300)
    plt.close(fig)

    self.ln(8)
    img_w = 150
    img_w = min(img_w, self._eff_width())
    x = self.l_margin + (self._eff_width() - img_w) / 2
    self.image(pie_path, x=x, w=img_w)
    self.ln(4)
```

```
# ---------- All Texts Extracted pages (5,9,13) ----------

def all_texts_pages(self, platform_name: str, df: pd.DataFrame):
    """
    Render ALL extracted rows for the given platform.
    Uses fixed layout like your screenshot, wraps key columns to 2 lines,
    and automatically paginates.
    """
    widths = [6, 26, 60, 25, 40, 40]  # #, Hate/Safe, Text, DateTime, DB, Table
    line_h = 8

    def add_page_with_header():
        self.add_page()
        # Title
        self.set_font("UNI", "B", 13)
        self.cell(0, 7, f"All Texts Extracted {platform_name}", align="C",
                  new_x="LMARGIN", new_y="NEXT")
        self.ln(4)
        # Header row
        self.set_font("UNI", "B", 10)
        self.table_row(
            widths,
            ["#", "Hate or Safe", "Text", "Date", "Source DB", "Source Table"],
            line_h=line_h
        )
        self.set_font("UNI", "", 10)

    add_page_with_header()

    # No rows case
    if df is None or df.empty:
        self.full_width_multicell(6, "No rows.")
        return
```

```
    row_no = 1
    for _, r in df.reset_index(drop=True).iterrows():
        # Each row will be at most 2 lines tall for wrapped cols
        required_h = 5 * line_h
        # If we're too close to the bottom, start a new page with header
        if self.get_y() + required_h > (self.h - self.b_margin):
            add_page_with_header()

        label = "Hate" if int(r.get("predicted_label", 0)) == 1 else "Safe"
        txt = clean_text(r.get("text", ""))  # <- show the full (cleaned) message
        dtm = extract_msg_datetime(r)
        sdb = soft_wrap(r.get("source_database", ""))
        stb = soft_wrap(r.get("source_table", ""))

        self.table_row_wrapped(
            widths,
            [row_no, label, txt, dtm, sdb, stb],
            aligns=["C", "L", "L", "L", "L", "L"],
            line_h=line_h,
            wrap_cols={2: 3, 3: 3, 4: 3, 5: 3},  # Text / Date / DB / Table
            default_max_lines=3
        )
        row_no += 1
```

```
    # ---------- Flagged (Hate) Texts pages (6,10,14) ----------

    def flagged_texts_pages(self, platform_name: str, df: pd.DataFrame, top_n: int = 25):
        """
        Render ONLY model-flagged hate texts (predicted_label == 1).
        Texts are cleaned for human readability and long code-ish blobs are
        trimmed with an ellipsis.
        """
        # Filter to hate only
        if df is None or df.empty:
            flg = pd.DataFrame(columns=df.columns if df is not None else [])
        else:
            flg = df[df.get("predicted_label", 0) == 1].copy()
            if "confidence" in flg.columns:
                flg = flg.sort_values("confidence", ascending=False)
        if top_n and len(flg) > top_n:
            flg = flg.head(top_n)

        widths = [7, 30, 60, 28, 38, 34]   # #, Confidence, Text, DateTime, DB, Table
        line_h = 8

        def header_page():
            self.add_page()
            self.set_font("UNI", "B", 13)
            self.cell(0, 7, f"{platform_name} Flagged Messages", align="C",
                      new_x="LMARGIN", new_y="NEXT")
            self.ln(4)
            self.set_font("UNI", "B", 10)
            self.table_row(
                widths,
                ["#", "Confidence Rate", "Text", "Date", "Source DB", "Source Table"],
                line_h=line_h
            )
            self.set_font("UNI", "", 10)

        header_page()
```

```
        header_page()

        if flg.empty:
            self.full_width_multicell(6, "No flagged messages.")
            return

        row_no = 1
        for _, r in flg.reset_index(drop=True).iterrows():
            # keep page nicely paginated
            required_h = 5 * line_h
            if self.get_y() + required_h > (self.h - self.b_margin):
                header_page()


            # confidence as percentage (if present)
            conf = "-"
            try:
                conf = f"{float(r.get('confidence', 0))*100:.1f}%"
            except Exception:
                pass

            # human-readable, code-lite text
            txt = clean_text(r.get("text", ""))
            txt = _reduce_code_blobs(txt)

            dtm = extract_msg_datetime(r)
            sdb = soft_wrap(r.get("source_database", ""))
            stb = soft_wrap(r.get("source_table", ""))

            self.table_row_wrapped(
                widths,
                [row_no, conf, txt, dtm, sdb, stb],
                aligns=["C", "C", "L", "L", "L", "L"],
                line_h=line_h,
                wrap_cols={2: 3, 3: 3, 4: 3, 5: 3},
                default_max_lines=3
            )
            row_no += 1
```

```python
    # ---------- appendix: database SHA-256 hashes pages (7,11,15) ----------

    def appendix_hashes_page(self, platform_name: str, hash_df: pd.DataFrame | None):
        """
        Appendix : Database SHA-256 Hashes
        - Two columns (Databases | Hashes)
        - Auto-wrap long values
        - Auto-add new pages when space runs out, repeating the header/table header
        """
        widths = [95, 95]          # two equal columns
        line_h = 7                  # line height used everywhere
        wrap_cfg = {0: 2, 1: 2}     # wrap both columns to max 2 lines

        def _page_header():
            """Title, subtitle, blurb, then table header."""
            self.add_page()
            self.set_font("UNI", "B", 16)
            self.cell(0, 8, "Appendix : Database SHA-256 Hashes", align="C",
                      new_x="LMARGIN", new_y="NEXT")
            self.ln(6)

            self.set_font("UNI", "", 12)
            self.cell(0, 6, f"Platform: {platform_name}", new_x="LMARGIN", new_y="NEXT")
            self.ln(6)

            # table header
            self.set_font("UNI", "B", 11)
            self.table_row(widths, ["Databases", "Hashes"], line_h=line_h)
            self.set_font("UNI", "", 10)

        def _ensure_space_for(row_h: float):
            """Start a new page with headers if the next row won't fit."""
            if self.get_y() + row_h > (self.h - self.b_margin):
                _page_header()

        # start first page
        _page_header()
```

```python
        # Rows
        if hash_df is None or hash_df.empty:
            self.full_width_multicell(6, "No hashes CSV provided.")
        else:
            # pick sensible columns
            db_col = next((c for c in ["database_file", "file", "database", "db"] if c in hash_df.columns),
                          hash_df.columns[0])
            hash_col = next((c for c in ["sha256_hash", "sha256", "hash", "sha_256"] if c in hash_df.columns),
                            (hash_df.columns[1] if len(hash_df.columns) > 1 else hash_df.columns[0]))

            for _, r in hash_df.iterrows():
                db_val = soft_wrap(r.get(db_col, ""))
                sha_val = soft_wrap(r.get(hash_col, ""))

                # Pre-compute wrapped lines to know the row height *before* drawing
                l_db  = self._wrap_for_cell(widths[0], db_val,  max_lines=wrap_cfg[0], line_h=line_h)
                l_sha = self._wrap_for_cell(widths[1], sha_val, max_lines=wrap_cfg[1], line_h=line_h)
                row_h = max(len(l_db), len(l_sha)) * line_h

                _ensure_space_for(row_h)

                # Now actually render the row (wrapping applied inside)
                self.table_row_wrapped(
                    widths, [db_val, sha_val],
                    aligns=["L", "L"], line_h=line_h, border=1,
                    wrap_cols=wrap_cfg, default_max_lines=2
                )
```

```python
        # Bottom note
        self.ln(6)
        self.set_text_color(110, 110, 110)
        self.set_font("UNI", "", 10)
        self.full_width_multicell(
            5,
            "Note: Below is an original source-file list of extracted data displayed in the"
            "report for quick reference. The full CSV remains the authoritative record. "
            "Furthermore, this report summarizes automated analysis results. "
            "Always refer to the original evidence for deep, sensitive, or critical investigations and decisions."
        )
        self.set_text_color(0, 0, 0)
        self.set_font("UNI", "", 12)
```

```
# ---------- Build ----------
def parse_args():
    p = argparse.ArgumentParser(description="Generate structured PDF report.")
    p.add_argument("--config", help="Optional path to JSON config from Hawkeye wrapper", default="")

    p.add_argument("--case-id", default="")
    p.add_argument("--investigator", default="")
    # p.add_argument("--device-id", default="")  # <<< DISABLED: we no longer accept/show device id
    p.add_argument("--owner", default="")

    p.add_argument("--fb-username", default="")
    p.add_argument("--ig-username", default="")
    p.add_argument("--ms-username", default="")

    p.add_argument("--fb-pred", default=DEFAULT_FB_PRED)
    p.add_argument("--ig-pred", default=DEFAULT_IG_PRED)
    p.add_argument("--ms-pred", default=DEFAULT_MS_PRED)

    p.add_argument("--fb-hash", default=DEFAULT_FB_HASH)
    p.add_argument("--ig-hash", default=DEFAULT_IG_HASH)
    p.add_argument("--ms-hash", default=DEFAULT_MS_HASH)

    p.add_argument("--out-dir", default=DEFAULT_OUT_DIR)
    p.add_argument("--font", default=DEFAULT_FONT)
    p.add_argument("--logo", default="", help="Optional path to a PNG/JPG logo for the cover page")
    p.add_argument("--top-n", type=int, default=25)
    return p.parse_args()
```

```
def merge_config(args):
    """If --config is supplied, merge it over CLI defaults (CLI still wins)."""
    if not args.config:
        return args
    try:
        with open(args.config, "r", encoding="utf-8") as f:
            cfg = json.load(f)
        for k, v in cfg.items():
            if getattr(args, k, "") in ("", None):
                setattr(args, k, v)
    except Exception:
        pass
    return args


def prompt_if_missing(args):
    """Interactive prompts for the fields you wanted if missing."""
    try:
        if not args.case_id:
            args.case_id = input(Fore.YELLOW + "\n 📁 Enter the Case ID : ").strip()
        if not args.investigator:
            args.investigator = input(Fore.YELLOW + "\n 🕵️ Enter the name of the Investigator : ").strip()
        if not args.owner:
            args.owner = input(Fore.YELLOW + "\n 📱 Enter the Owner's name of the mobile device : ").strip()
    except EOFError:
        pass
    return args
```

```
def build_report(args):
    os.makedirs(args.out_dir, exist_ok=True)
    args.font = ensure_static_ttf(args.font)

    # load data
    fb = load_df(args.fb_pred)
    ig = load_df(args.ig_pred)
    ms = load_df(args.ms_pred)

    # NEW: load hash CSVs for appendix pages
    fb_hash = load_df(args.fb_hash)
    ig_hash = load_df(args.ig_hash)
    ms_hash = load_df(args.ms_hash)

    # auto-fill usernames/device if blank
    if not args.fb_username:
        args.fb_username = infer_username(fb) or ""
    if not args.ig_username:
        args.ig_username = infer_username(ig) or ""
    if not args.ms_username:
        args.ms_username = infer_username(ms) or ""
    # if not args.device_id:
    #     args.device_id = infer_device_from_dfs([fb, ig, ms]) or "UNKNOWN-DEVICE"  # <<< DISABLED

    # prompt for human fields if needed
    args = prompt_if_missing(args)
```

```
    fb_stats = compute_stats(fb)
    ig_stats = compute_stats(ig)
    ms_stats = compute_stats(ms)
    totals = {"Facebook": fb_stats, "Instagram": ig_stats, "Messenger": ms_stats}

    today = dt.datetime.now()
    today_str = today.strftime("%d of %B, %Y")
    when = today.strftime("%d/%m/%Y")
    ts_name = today.strftime("%Y%m%d_%H%M%S")
    out_path = os.path.join(args.out_dir, f"Structured_Report_{ts_name}.pdf")
    logo_path = resolve_logo_path(args.logo)
```

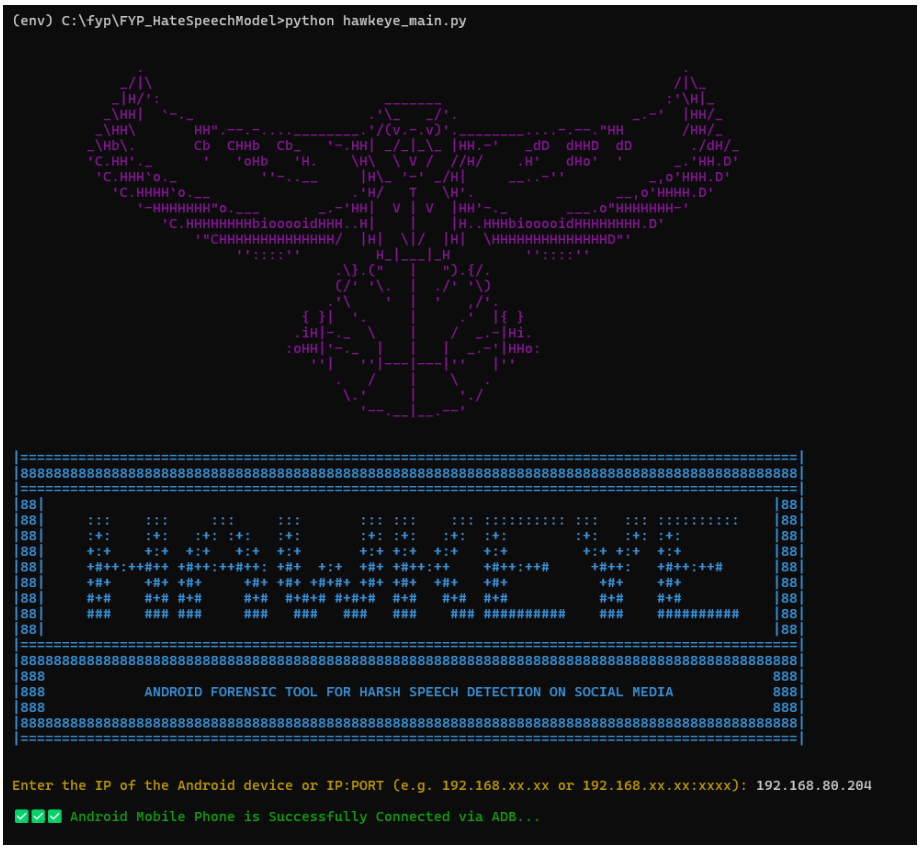
```
    tmp = tempfile.mkdtemp(prefix="pdf_tmp_")
    try:
        pdf = ReportPDF(args.font)

        # Page 1 (cover)
        pdf.cover_page(today_str, logo_path=logo_path)

        # Page 2 (overview)
        pdf.overview_page(args, totals, tmp)

        # Page 3 (overall device summary with table + pie)
        pdf.overall_device_summary_page(totals, tmp)

        # Pages 4,8,12 (summary) + 5,9,13 (all texts) + 6,10,14 (Flaged texts)
        # Facebook
        pdf.platform_summary_page("Facebook",  args.fb_username, when, fb_stats, tmp)
        #pdf.all_texts_pages("Facebook", fb)
        pdf.flagged_texts_pages("Facebook", fb, top_n=args.top_n)
        pdf.appendix_hashes_page("Facebook", fb_hash)

        # Instagram
        pdf.platform_summary_page("Instagram", args.ig_username, when, ig_stats, tmp)
        #pdf.all_texts_pages("Instagram", ig)
        pdf.flagged_texts_pages("Instagram", ig, top_n=args.top_n)
        pdf.appendix_hashes_page("Instagram", ig_hash)

        # Messenger
        pdf.platform_summary_page("Messenger", args.ms_username, when, ms_stats, tmp)
        #pdf.all_texts_pages("Messenger", ms)
        pdf.flagged_texts_pages("Messenger", ms, top_n=args.top_n)
        pdf.appendix_hashes_page("Messenger", ms_hash)
```

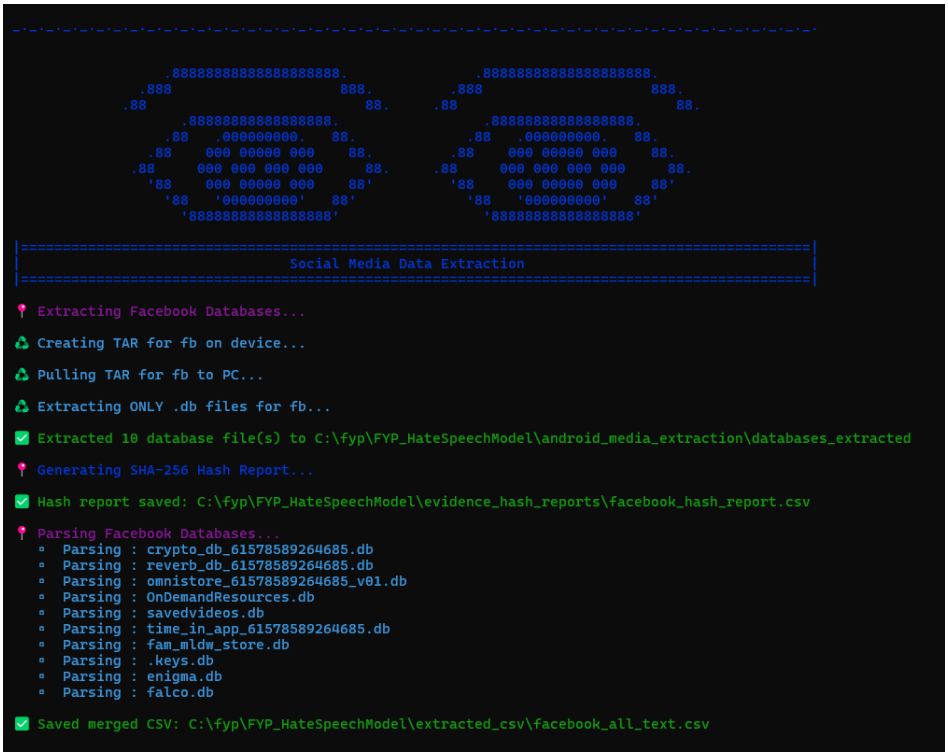
```
        pdf.output(out_path)
        print(Fore.GREEN + "\n✅✅✅ Report Generation Completed Successfully")
        print(Fore.GREEN + "\n📍 Report Saved Location → ")
        print(Fore.YELLOW + f"\n         👉 {out_path}")
        print(Fore.BLUE + """\n
_.-._.-._.-._.-._.-._.-._.-._.-._.-._.-._.-._.-._.-._.-._.-._.-._.-._.-._.-._.-._.-._.-._.-._.-._.-._."""+ Style.RESET_ALL)
        print(Fore.MAGENTA +"\n✋  Quiting.....")
        print(Fore.CYAN +"""\n
|=============================================================================================|
|8888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888|
|=============================================================================================|
|88|                                                                                       |88|
|88|     :::    :::     :::     :::       ::: :::    ::: :::::::::: :::   ::: ::::::::::    |88|
|88|     :+:    :+:   :+: :+:   :+:       :+: :+:   :+:  :+:        :+:   :+: :+:           |88|
|88|     +:+    +:+  +:+   +:+  +:+       +:+ +:+  +:+   +:+         +:+ +:+  +:+          |88|
|88|     +#++:++#++ +#++:++#++: +#+  +:+  +#+ +#++:++    +#++:++#     +#++:   +#++:++#      |88|
|88|     +#+    +#+ +#+     +#+ +#+ +#+#+ +#+ +#+  +#+   +#+           +#+    +#+           |88|
|88|     #+#    #+# #+#     #+#  #+#+# #+#+#  #+#   #+#  #+#           #+#    #+#           |88|
|88|     ###    ### ###     ###   ###   ###   ###    ### ##########    ###    ##########    |88|
|88|                                                                                       |88|
|=============================================================================================|
|8888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888|
|888                                                                                       888|
|888            ANDROID FORENSIC TOOL FOR HARSH SPEECH DETECTION ON SOCIAL MEDIA            888|
|888                                                                                       888|
|8888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888|
|=============================================================================================|
    """)
        print(Fore.MAGENTA +"""
:::::::::: :::    :::     :::     ::::    ::: :::    :::    :::   :::  ::::::::  :::    ::: :::
    :+:     :+:    :+:   :+: :+:   :+:+:   :+: :+:   :+:     :+:   :+: :+:    :+: :+:    :+: :+:
    +:+     +:+    +:+  +:+   +:+  :+:+:+:+  +:+ +:+  +:+       +:+ +:+  +:+    +:+ +:+    +:+ +:+
    +#+     +#++:++#++ +#++:++#++: +#+ +:+ +#+ +#++:++          +#++:   +#+    +:+ +#+    +:+ +:+
    +#+     +#+    +#+ +#+     +#+ +#+  +#+#+# +#+  +#+         +#+    +#+    +#+ +#+    +#+
    #+#     #+#    #+# #+#     #+# #+#   #+#+# #+#   #+#        #+#    #+#    #+# #+#    #+# #+#
    ###     ###    ### ###     ### ###    #### ###    ###       ###     ########   ########  ###
        """+ Style.RESET_ALL)
    finally:
        shutil.rmtree(tmp, ignore_errors=True)
```

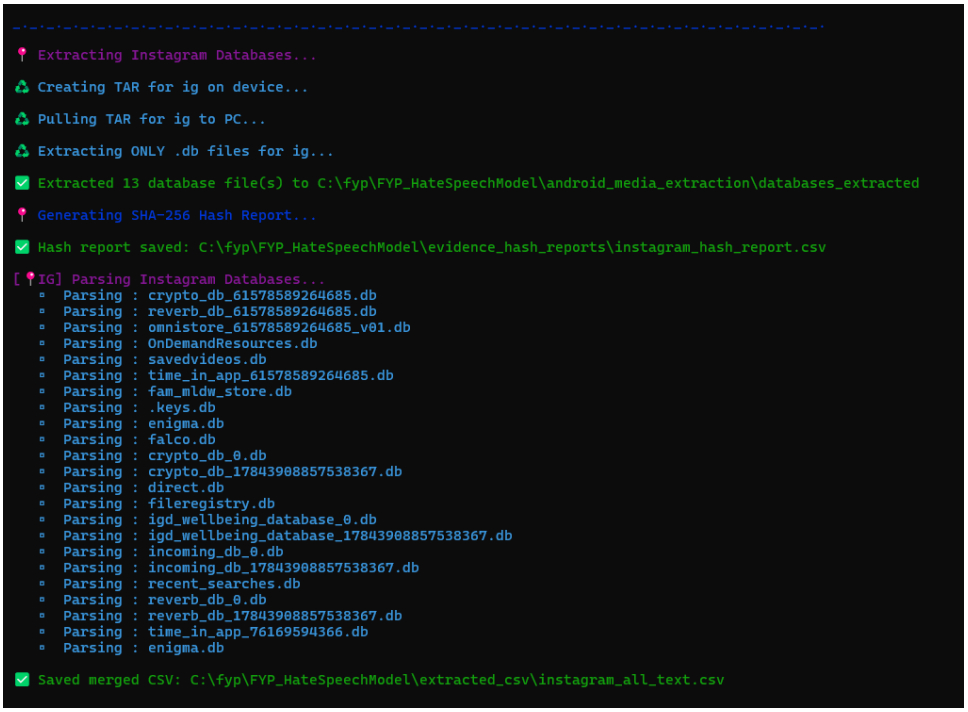
```
def main():
    args = parse_args()
    args = merge_config(args)
    build_report(args)


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

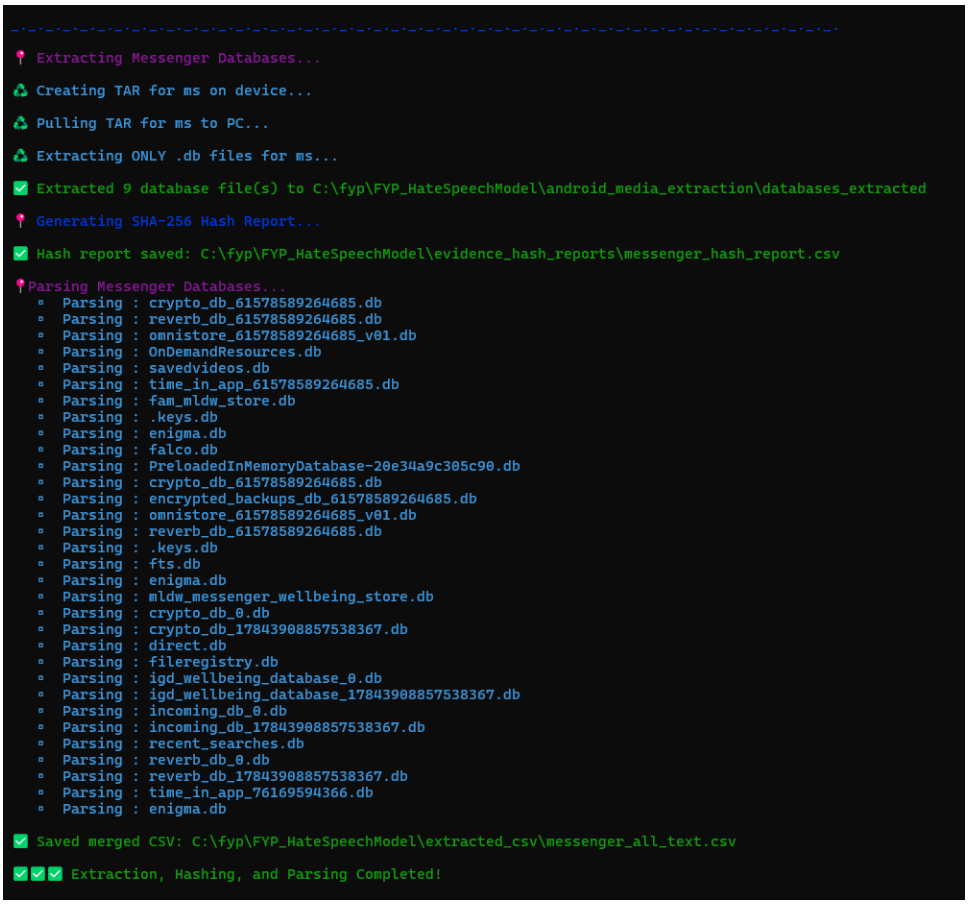
APPENDIX 3 - User Interface of the Tool

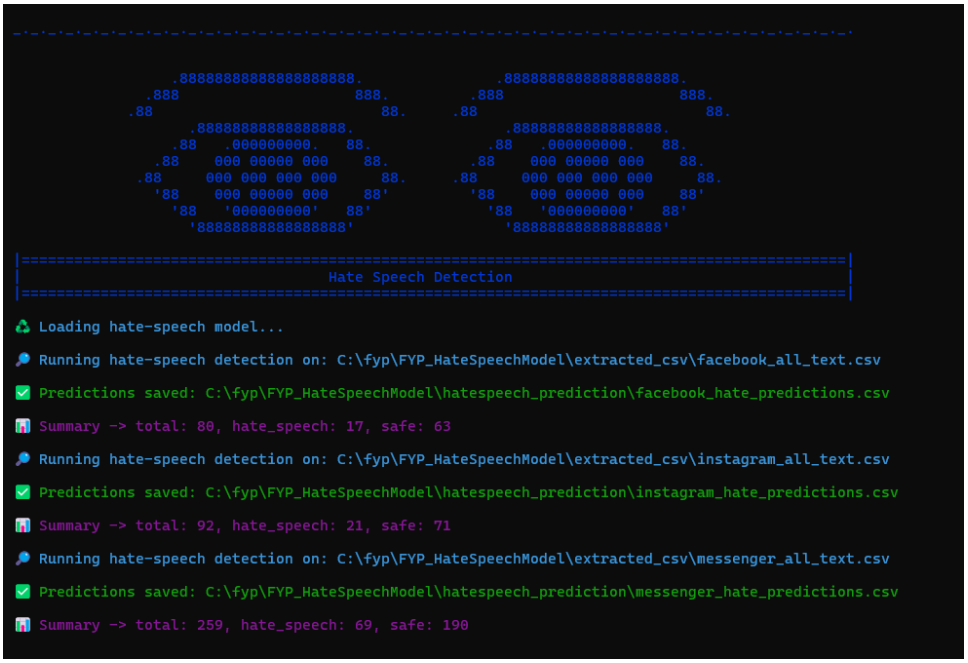
i. Screenshots of the Final Outcome of the Tool



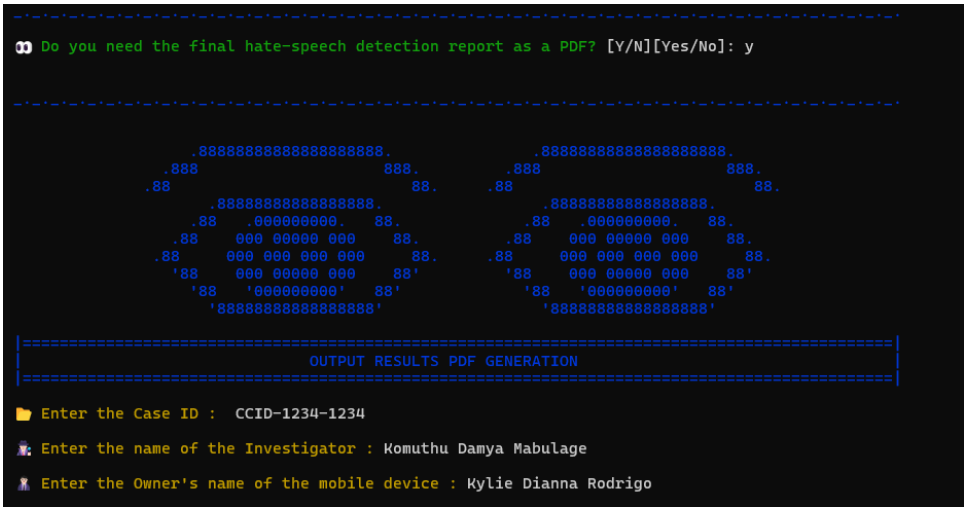


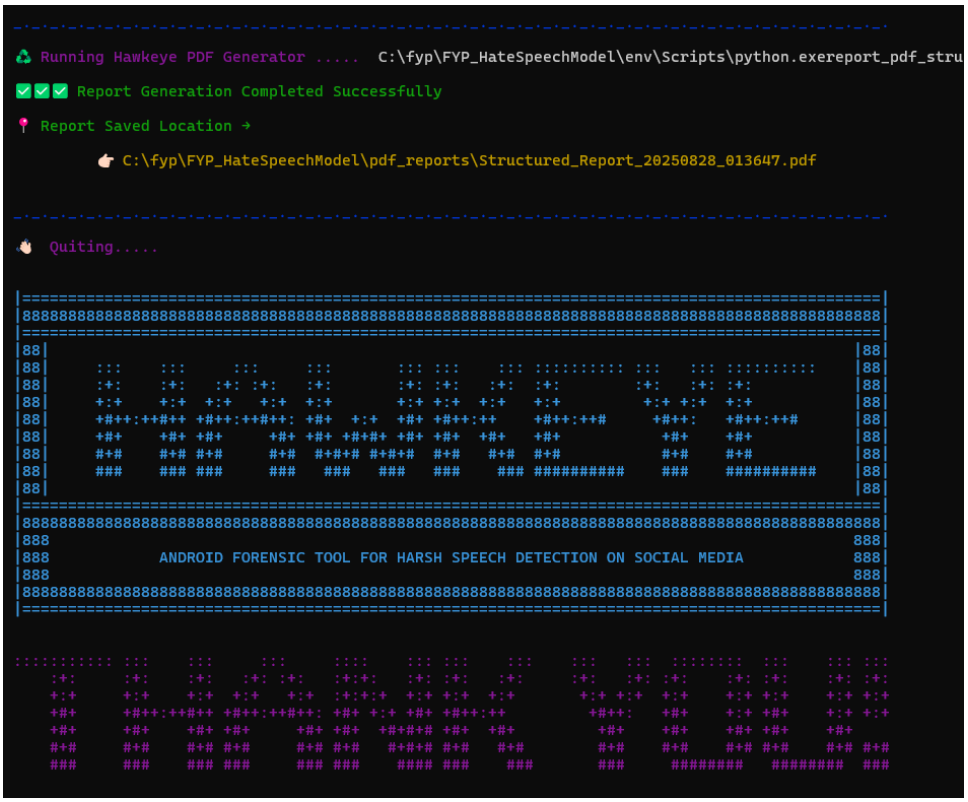

Extracting Instagram Databases...
Creating TAR for ig on device...
Pulling TAR for ig to PC...
Extracting ONLY .db files for ig...
Extracted 13 database file(s) to C:\fyp\FYP_HateSpeechModel\android_media_extraction\databases_extracted
Generating SHA-256 Hash Report...
Hash report saved: C:\fyp\FYP_HateSpeechModel\evidence_hash_reports\instagram_hash_report.csv
[IG] Parsing Instagram Databases...
Parsing : crypto_db_61578589264685.db
Parsing : reverb_db_61578589264685.db
Parsing : omnistore_61578589264685_v01.db
Parsing : OnDemandResources.db
Parsing : savedvideos.db
Parsing : time_in_app_61578589264685.db
Parsing : fam_mldw_store.db
Parsing : .keys.db
Parsing : enigma.db
Parsing : falco.db
Parsing : crypto_db_0.db
Parsing : crypto_db_17843908857538367.db
Parsing : direct.db
Parsing : fileregistry.db
Parsing : igd_wellbeing_database_0.db
Parsing : igd_wellbeing_database_17843908857538367.db
Parsing : incoming_db_0.db
Parsing : incoming_db_17843908857538367.db
Parsing : recent_searches.db
Parsing : reverb_db_0.db
Parsing : reverb_db_17843908857538367.db
Parsing : time_in_app_76169594366.db
Parsing : enigma.db
Saved merged CSV: C:\fyp\FYP_HateSpeechModel\extracted_csv\instagram_all_text.csv


Extracting Messenger Databases...
Creating TAR for ms on device...
Pulling TAR for ms to PC...
Extracting ONLY .db files for ms...
Extracted 9 database file(s) to C:\fyp\FYP_HateSpeechModel\android_media_extraction\databases_extracted
Generating SHA-256 Hash Report...
Hash report saved: C:\fyp\FYP_HateSpeechModel\evidence_hash_reports\messenger_hash_report.csv
Parsing Messenger Databases...
Parsing : crypto_db_61578589264685.db
Parsing : reverb_db_61578589264685.db
Parsing : omnistore_61578589264685_v01.db
Parsing : OnDemandResources.db
Parsing : savedvideos.db
Parsing : time_in_app_61578589264685.db
Parsing : fam_mldw_store.db
Parsing : .keys.db
Parsing : enigma.db
Parsing : falco.db
Parsing : PreloadedInMemoryDatabase-20e34a9c305c90.db
Parsing : crypto_db_61578589264685.db
Parsing : encrypted_backups_db_61578589264685.db
Parsing : omnistore_61578589264685_v01.db
Parsing : reverb_db_61578589264685.db
Parsing : .keys.db
Parsing : fts.db
Parsing : enigma.db
Parsing : mldw_messenger_wellbeing_store.db
Parsing : crypto_db_0.db
Parsing : crypto_db_17843908857538367.db
Parsing : direct.db
Parsing : fileregistry.db
Parsing : igd_wellbeing_database_0.db
Parsing : igd_wellbeing_database_17843908857538367.db
Parsing : incoming_db_0.db
Parsing : incoming_db_17843908857538367.db
Parsing : recent_searches.db
Parsing : reverb_db_0.db
Parsing : reverb_db_17843908857538367.db
Parsing : time_in_app_76169594366.db
Parsing : enigma.db
Saved merged CSV: C:\fyp\FYP_HateSpeechModel\extracted_csv\messenger_all_text.csv
Extraction, Hashing, and Parsing Completed!


Hate Speech Detection
Loading hate-speech model...
Running hate-speech detection on: C:\fyp\FYP_HateSpeechModel\extracted_csv\facebook_all_text.csv
Predictions saved: C:\fyp\FYP_HateSpeechModel\hatespeech_prediction\facebook_hate_predictions.csv
Summary -> total: 80, hate_speech: 17, safe: 63
Running hate-speech detection on: C:\fyp\FYP_HateSpeechModel\extracted_csv\instagram_all_text.csv
Predictions saved: C:\fyp\FYP_HateSpeechModel\hatespeech_prediction\instagram_hate_predictions.csv
Summary -> total: 92, hate_speech: 21, safe: 71
Running hate-speech detection on: C:\fyp\FYP_HateSpeechModel\extracted_csv\messenger_all_text.csv
Predictions saved: C:\fyp\FYP_HateSpeechModel\hatespeech_prediction\messenger_hate_predictions.csv
Summary -> total: 259, hate_speech: 69, safe: 190


Hate Speech Detection Summary
Facebook :
Total = 80
Hate Speech = 17
Safe = 63
Instagram :
Total = 92
Hate Speech = 21
Safe = 71
Messenger :
Total = 259
Hate Speech = 69
Safe = 190
Extractions and Hate Speech Detection All Completed!
Saved Locations of the Generated Reports :
C:\fyp\FYP_HateSpeechModel\hatespeech_prediction\facebook_hate_predictions.csv
C:\fyp\FYP_HateSpeechModel\hatespeech_prediction\instagram_hate_predictions.csv
C:\fyp\FYP_HateSpeechModel\hatespeech_prediction\messenger_hate_predictions.csv


Do you need the final hate-speech detection report as a PDF? [Y/N][Yes/No]: y
OUTPUT RESULTS PDF GENERATION
Enter the Case ID : CCID-1234-1234
Enter the name of the Investigator : Komuthu Damya Mabulage
Enter the Owner's name of the mobile device : Kylie Dianna Rodrigo


Running Hawkeye PDF Generator ..... C:\fyp\FYP_HateSpeechModel\env\Scripts\python.exereport_pdf_stru
Report Generation Completed Successfully
Report Saved Location →
C:\fyp\FYP_HateSpeechModel\pdf_reports\Structured_Report_20250828_013647.pdf
Quiting.....
ANDROID FORENSIC TOOL FOR HARSH SPEECH DETECTION ON SOCIAL MEDIA

ii. Screenshots of the of the PDF Report Generated

CYBERBULLYING DETECTION REPORT

Digital Forensic Analysis on Social Media Platforms of Instagram, Facebook, Messenger

28 of August, 2025

Cyberbullying - Hate Speech Detection Report

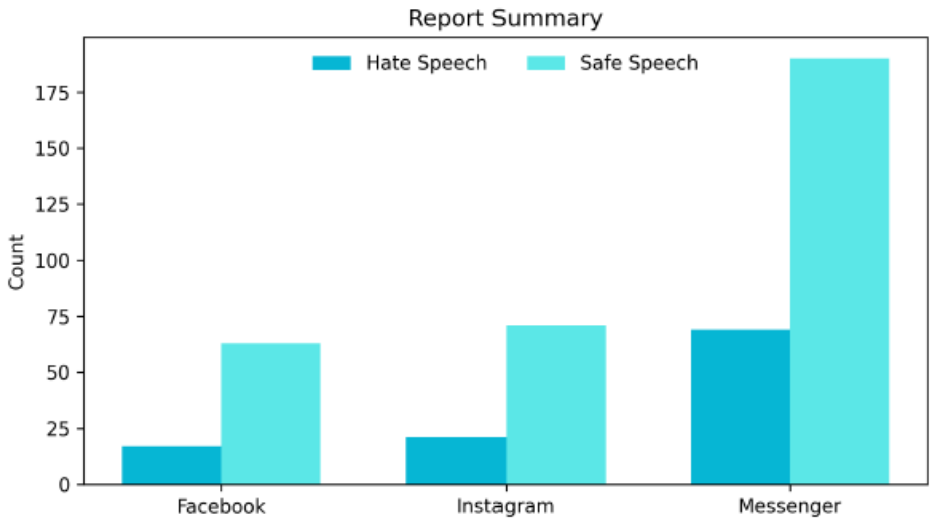
Case Number : CCID-1234-1234

Investigator : Komuthu Damya Mabulage

Owner of the Device : Kylie Dianna Rodrigo

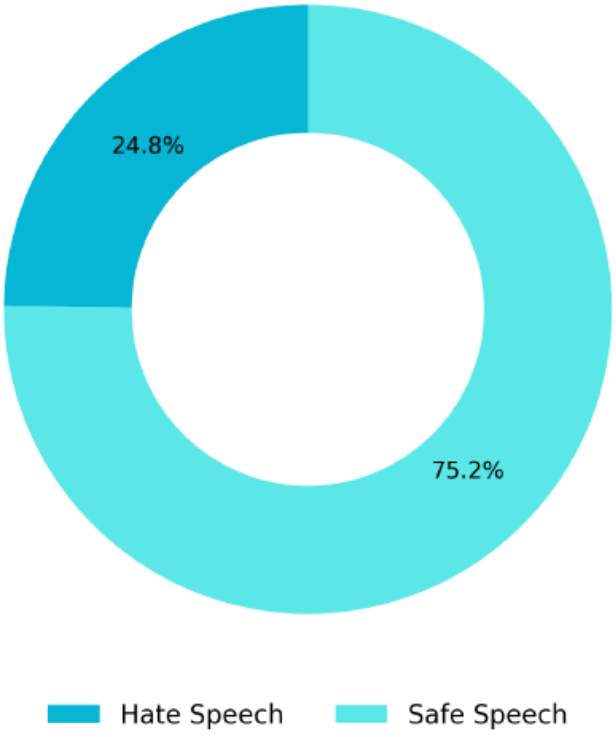
Report Summary

Social Media Platform	Total of Texts	Amount of Hate Speech	Amount of Safe Speech
Facebook	80	17 = 21.2%	63 = 78.8%
Instagram	92	21 = 22.8%	71 = 77.2%
Messenger	259	69 = 26.6%	190 = 73.4%



Overall Hate Speech Detected from the Device

Social Media Platform	Hate Speech	Safe Speech
Facebook	21.2%	78.8%
Instagram	22.8%	77.2%
Messenger	26.6%	73.4%
Total	24.8%	75.2%



Cyberbullying - Hate Speech Detection

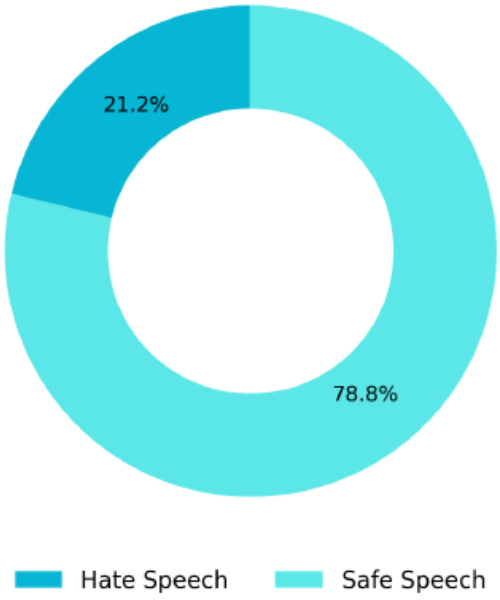
Social Media Platform : Facebook

Social Media Profile : -

Report Generated Date : 28/08/2025

Facebook Summary

Social Media Platform	Total of Texts	Amount of Hate Speech	Amount of Safe Speech
Facebook	80	17 = 21.2%	63 = 78.8%



Facebook Flagged Messages

#	Confidence Rate	Text	Date	Source DB	Source Table
1	99.0%	vZ*(0 8 Q %"# ! ...	12/08/2025	reverb_db_ 61578589264685.db	local_message_ persistence_store
2	99.0%	shut up bitch	12/08/2025	reverb_db_ 61578589264685.db	local_message_ persistence_store
3	99.0%	Caneh meh ballo kata wahagena inna.... mage yaka awussaganne nathuwa	12/08/2025	reverb_db_ 61578589264685.db	local_message_ persistence_store
4	99.0%	Rd0 @hA6V(0 8 3"1 / +...	12/08/2025	reverb_db_ 61578589264685.db	local_message_ persistence_store
5	99.0%	uba hithan inne uba lassanaii kiyala da... uba harakek bn	12/08/2025	reverb_db_ 61578589264685.db	local_message_ persistence_store
6	99.0%	gon gani	12/08/2025	reverb_db_ 61578589264685.db	local_message_ persistence_store
7	99.0%	hgpC*P4(0 8 W +") ' #...	12/08/2025	reverb_db_ 61578589264685.db	local_message_ persistence_store
8	99.0%	fuck you 0 B eqa(K)W?bH	12/08/2025	reverb_db_ 61578589264685.db	local_message_ persistence_store
9	99.0%	yOR (0 8 E ("& " ...	12/08/2025	reverb_db_ 61578589264685.db	local_message_ persistence_store
10	99.0%	JO(0 8 U)"' % !...	12/08/2025	reverb_db_ 61578589264685.db	local_message_ persistence_store
11	99.0%	Caneh meh ballo kata wahagena inna.... mage yaka awussaganne nathuwa 0 B W< @F ~yH	12/08/2025	reverb_db_ 61578589264685.db	local_message_ persistence_store

Appendix : Database SHA-256 Hashes

Platform: Facebook

Databases	Hashes
crypto_db_61578589264685.db	430b6d633505a7ea32b0ed45f38d97046d509816715852 57d4b7fc16892c8542
reverb_db_61578589264685.db	f3cdf6f117dec0e261a81201d230f96c7e83fc8e68565910 5afe36a3196cfa45
omnistore_61578589264685_v01.db	e787d121c62490959f0c7013db17ff4bb6ddf1073deffc99 10ca102e2ee721f0
OnDemandResources.db	6ceb294fc843c80845c8ff6babc21ab32c3d06fa0a623355 5883890f388b49b1
savedvideos.db	701dca9de408836195f93821c6e973ff174c6774c83168fc 31f9a3d33b278b6c
time_in_app_61578589264685.db	e88219d5b79023180e7ad67529e3aaf22ccf5080356baf2 b0c7c898b3d2bc7f3
fam_mldw_store.db	1225ebca3b24ba2b173b48ccf7161a00fe1b3d7faf754d9 74c39638070724877
.keys.db	00edb430179082b72291484b4ebb06a25f1bd8547ac141 3c8bf20eb5f45bae25
enigma.db	7143eb7bc2397b0a085bc4e764f268d520a22d8e71719f ccf3597b03d2c87e9e
falco.db	f6434a7c4e062596af6f761a7c29981754d10a6a9b9f69d a6feec1b1b88b4598

Note: Below is an original source-file list of extracted data displayed in thereport for quick reference. The full CSV remains the authoritative record. Furthermore, this report summarizes automated analysis results. Always refer to the original evidence for deep, sensitive, or critical investigations and decisions.

Cyberbullying - Hate Speech Detection

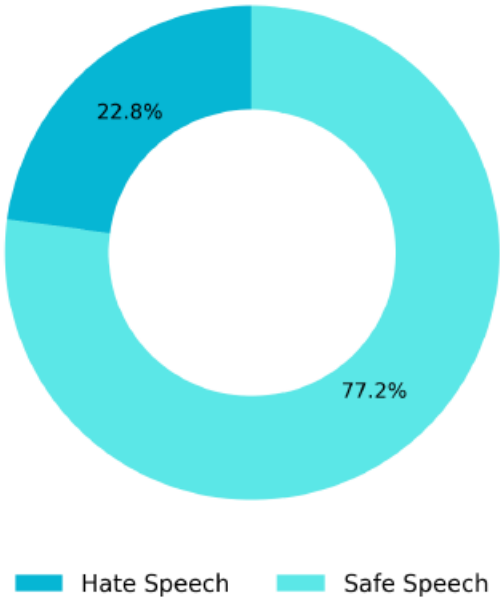
Social Media Platform : Instagram

Social Media Profile : -

Report Generated Date : 28/08/2025

Instagram Summary

Social Media Platform	Total of Texts	Amount of Hate Speech	Amount of Safe Speech
Instagram	92	21 = 22.8%	71 = 77.2%



Instagram Flagged Messages

#	Confidence Rate	Text	Date	Source DB	Source Table
1	99.0%	vZ*(0 8 Q %"# ! ...	12/08/2025	reverb_db_ 61578589264685.db	local_message_ persistence_store
2	99.0%	shut up bitch	12/08/2025	reverb_db_ 61578589264685.db	local_message_ persistence_store
3	99.0%	Caneh meh ballo kata wahagena inna.... mage yaka awussaganne nathuwa	12/08/2025	reverb_db_ 61578589264685.db	local_message_ persistence_store
4	99.0%	Rd0 @hA6V(0 8 3"1 / +...	12/08/2025	reverb_db_ 61578589264685.db	local_message_ persistence_store
5	99.0%	uba hithan inne uba lassanaii kiyala da... uba harakek bn	12/08/2025	reverb_db_ 61578589264685.db	local_message_ persistence_store
6	99.0%	gon gani	12/08/2025	reverb_db_ 61578589264685.db	local_message_ persistence_store
7	99.0%	hgpC*P4(0 8 W +") ' #...	12/08/2025	reverb_db_ 61578589264685.db	local_message_ persistence_store
8	99.0%	fuck you 0 B eqa(K)W?bH	12/08/2025	reverb_db_ 61578589264685.db	local_message_ persistence_store
9	99.0%	yOR (0 8 E ("& " ...	12/08/2025	reverb_db_ 61578589264685.db	local_message_ persistence_store
10	99.0%	JO(0 8 U)"' % !...	12/08/2025	reverb_db_ 61578589264685.db	local_message_ persistence_store
11	99.0%	Caneh meh ballo kata wahagena inna.... mage yaka awussaganne nathuwa 0 B W< @F ~yH	12/08/2025	reverb_db_ 61578589264685.db	local_message_ persistence_store

Appendix : Database SHA-256 Hashes

Platform: Instagram

Databases	Hashes
crypto_db_61578589264685.db	430b6d633505a7ea32b0ed45f38d97046d509816715852 57d4b7fc16892c8542
reverb_db_61578589264685.db	f3cdf6f117dec0e261a81201d230f96c7e83fc8e685659105afe36a3196cfa45
omnistore_61578589264685_v01.db	e787d121c62490959f0c7013db17ff4bb6ddf1073deffc9910ca102e2ee721f0
OnDemandResources.db	6ceb294fc843c80845c8ff6babc21ab32c3d06fa0a6233555883890f388b49b1
savedvideos.db	701dca9de408836195f93821c6e973ff174c6774c83168fc31f9a3d33b278b6c
time_in_app_61578589264685.db	e88219d5b79023180e7ad67529e3aaf22ccf5080356baf2b0c7c898b3d2bc7f3
fam_mldw_store.db	1225ebca3b24ba2b173b48ccf7161a00fe1b3d7faf754d974c39638070724877
.keys.db	00edb430179082b72291484b4ebb06a25f1bd8547ac1413c8bf20eb5f45bae25
enigma.db	7143eb7bc2397b0a085bc4e764f268d520a22d8e71719fccf3597b03d2c87e9e
falco.db	f6434a7c4e062596af6f761a7c29981754d10a6a9b9f69da6feec1b1b88b4598
crypto_db_0.db	b16fd42ba7f807515e75417fbd28e946bfb9d3c55f0da3f743b0e9d780e53db6
crypto_db_17843908857538367.db	ab2191b2ac1d4f8a3a8e4c4ebebdef42ade1376837aa8e955a6dc3569996b0d4
direct.db	e78b59cd662addde174ba2c1f158ac90fc2b1fb105965a7c884df6a66fa047e7
fileregistry.db	8bfac9f8cb73f8351d1571eea2c546448b4b6579ccae678d3214f80f09013b47
igd_wellbeing_database_0.db	12dfb4f179910b54fc69c0ea4769870cabeafa8e45b5c876a89a3ca366ed7aa8
igd_wellbeing_database_17843908857538367.db	64209dba95a1c70453b581dd1a60d20070e65da9c13279940ff7e75e1c505a0e
incoming_db_0.db	9bc531a8742fc31dfc3e13d94cff1d76ccb12e5815a78e2d773d8eb67f9b8bcb

Cyberbullying - Hate Speech Detection

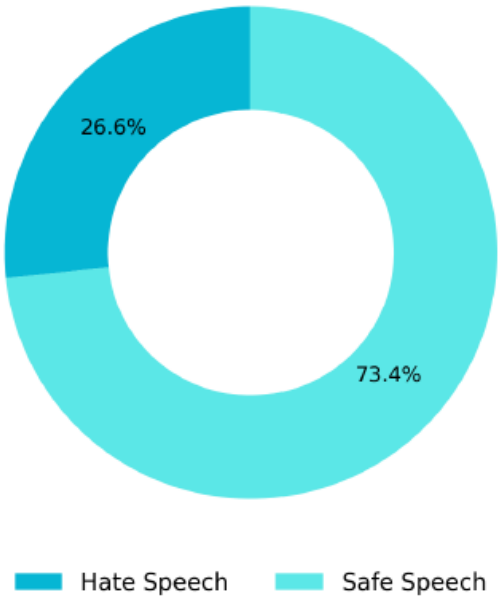
Social Media Platform : Messenger

Social Media Profile : -

Report Generated Date : 28/08/2025

Messenger Summary

Social Media Platform	Total of Texts	Amount of Hate Speech	Amount of Safe Speech
Messenger	259	69 = 26.6%	190 = 73.4%



Messenger Flagged Messages

#	Confidence Rate	Text	Date	Source DB	Source Table
1	99.0%	vZ*(0 8 Q %"# ! ...	12/08/2025	reverb_db_ 61578589264685.db	local_message_ persistence_store
2	99.0%	shut up bitch	12/08/2025	reverb_db_ 61578589264685.db	local_message_ persistence_store
3	99.0%	Caneh meh ballo kata wahagena inna.... mage yaka awussaganne nathuwa	12/08/2025	reverb_db_ 61578589264685.db	local_message_ persistence_store
4	99.0%	Rd0 @hA6V(0 8 3"1 / +...	12/08/2025	reverb_db_ 61578589264685.db	local_message_ persistence_store
5	99.0%	uba hithan inne uba lassanaii kiyala da... uba harakek bn	12/08/2025	reverb_db_ 61578589264685.db	local_message_ persistence_store
6	99.0%	gon gani	12/08/2025	reverb_db_ 61578589264685.db	local_message_ persistence_store
7	99.0%	hgpC*P4(0 8 W +") ' #...	12/08/2025	reverb_db_ 61578589264685.db	local_message_ persistence_store
8	99.0%	fuck you 0 B eqa(K)W?bH	12/08/2025	reverb_db_ 61578589264685.db	local_message_ persistence_store
9	99.0%	yOR (0 8 E ("& " ...	12/08/2025	reverb_db_ 61578589264685.db	local_message_ persistence_store
10	99.0%	JO(0 8 U)"' % !...	12/08/2025	reverb_db_ 61578589264685.db	local_message_ persistence_store
11	99.0%	Caneh meh ballo kata wahagena inna.... mage yaka awussaganne nathuwa 0 B W< @F ~yH	12/08/2025	reverb_db_ 61578589264685.db	local_message_ persistence_store

Appendix : Database SHA-256 Hashes

Platform: Messenger

Databases	Hashes
enigma.db	7143eb7bc2397b0a085bc4e764f268d520a22d8e71719fccf3597b03d2c87e9e
mldw_messenger_wellbeing_store.db	2370a7fdc35df47128081567b1c864c29cd17ca943fb1374583f0fb5759702f6
crypto_db_0.db	b16fd42ba7f807515e75417fbd28e946bfb9d3c55f0da3f743b0e9d780e53db6
crypto_db_17843908857538367.db	ab2191b2ac1d4f8a3a8e4c4ebebdef42ade1376837aa8e955a6dc3569996b0d4
direct.db	e78b59cd662addde174ba2c1f158ac90fc2b1fb105965a7c884df6a66fa047e7
fileregistry.db	8bfac9f8cb73f8351d1571eea2c546448b4b6579ccae678d3214f80f09013b47
igd_wellbeing_database_0.db	12dfb4f179910b54fc69c0ea4769870cabeafa8e45b5c876a89a3ca366ed7aa8
igd_wellbeing_database_17843908857538367.db	64209dba95a1c70453b581dd1a60d20070e65da9c13279940ff7e75e1c505a0e
incoming_db_0.db	9bc531a8742fc31dfc3e13d94cff1d76ccb12e5815a78e2d773d8eb67f9b8bcb
incoming_db_17843908857538367.db	3c267d460f97571f235a1ec02cb903e1b2e8175b4c25fdf8ca67d0a5a652d542
recent_searches.db	ae7a9d5f26898262955de469b1555c7215a84652b7b31ce55a13bbd60a34d49a
reverb_db_0.db	9b5c3915ca79bc9dc0681eecafd5e728e3db69d21472d09915cf69617bdb537b
reverb_db_17843908857538367.db	28e4099a0c38c1946ed21815247180267c2bbee0805425842e66a737c9cd206b
time_in_app_76169594366.db	65838537132bb447311da8e17c34b2b685b2e782c3bedf82ceb837f7fb82aa68
enigma.db	7143eb7bc2397b0a085bc4e764f268d520a22d8e71719fccf3597b03d2c87e9e

Note: Below is an original source-file list of extracted data displayed in thereport for quick reference. The full CSV remains the authoritative record. Furthermore, this report summarizes automated analysis results. Always refer to the original evidence for deep, sensitive, or critical investigations and decisions.

APPENDIX 4 – Model Testing (Confusion Matrix)

i. Facebook

```
📄 Evaluating file: C:\fyp\FYP_HateSpeechModel\testing_score\facebook_hate_predictions.csv
✅ File loaded successfully.

📊 Confusion Matrix:
[[57  0]
 [ 6 17]]

📄 Classification Report:
              precision    recall  f1-score   support

           0     0.9048    1.0000    0.9500        57
           1     1.0000    0.7391    0.8500        23

    accuracy                         0.9250        80
   macro avg     0.9524    0.8696    0.9000        80
weighted avg     0.9321    0.9250    0.9213        80

✅ Accuracy      : 0.9250
✅ Precision     : 1.0000
✅ Recall        : 0.7391
✅ F1 Score      : 0.8500
------------------------------------------------------------
```

ii. Messenger

```
📄 Evaluating file: C:\fyp\FYP_HateSpeechModel\testing_score\messenger_hate_predictions.csv
✅ File loaded successfully.

📊 Confusion Matrix:
[[155   0]
 [ 18  65]]

📄 Classification Report:
              precision    recall  f1-score   support

           0     0.8960    1.0000    0.9451       155
           1     1.0000    0.7831    0.8784        83

    accuracy                         0.9244       238
   macro avg     0.9480    0.8916    0.9118       238
weighted avg     0.9322    0.9244    0.9218       238

✅ Accuracy      : 0.9244
✅ Precision     : 1.0000
✅ Recall        : 0.7831
✅ F1 Score      : 0.8784
------------------------------------------------------------
```

iii. Instagram

```
📄 Evaluating file: C:\fyp\FYP_HateSpeechModel\testing_score\instagram_hate_predictions.csv
✅ File loaded successfully.

📊 Confusion Matrix:
[[65  0]
 [ 6 21]]

📄 Classification Report:
              precision    recall  f1-score   support

           0     0.9155    1.0000    0.9559        65
           1     1.0000    0.7778    0.8750        27

    accuracy                         0.9348        92
   macro avg     0.9577    0.8889    0.9154        92
weighted avg     0.9403    0.9348    0.9321        92

✅ Accuracy      : 0.9348
✅ Precision     : 1.0000
✅ Recall        : 0.7778
✅ F1 Score      : 0.8750
------------------------------------------------------------
```

APPENDIX 5 – Survey Questions and Results

i. Screenshots of the Survey – Questionnaire

Survey on Cyberbullying Attempts, Language, and Usage Patterns on Social Media Platforms in Sri Lanka

B I U

This anonymous survey is part of an undergraduate research project focused on developing a multilingual digital forensic tool to detect cyberbullying on Facebook and Instagram.

The aim is to understand how social media is used in Sri Lanka, how cyberbullying occurs, and the types of language (Unicode Sinhala, Sinhala typed with English letters, or English) commonly used in such incidents.

Your responses will help to create a tool that supports local languages and assists law enforcement with online abuse investigations.

🔒 No personal information will be collected.
⏱ The survey will take less than 5 minutes to complete.

Thank you for your valuable input in making Sri Lanka's digital space safer.

SECTION 1: Demographic Information

Description (optional)

What age group do you fall under? *

- ○ Under 18
- ○ 18 – 24
- ○ 25 – 34
- ○ 35 – 44
- ○ 45+

Your gender? *

- ○ Female
- ○ Male
- ○ Prefer not to say
- ○ Other...

SECTION 2 : Social Media Usage

Description (optional)

Which social media platforms do you use regularly? *

- ☐ Facebook
- ☐ Instagram
- ☐ WhatsApp
- ☐ TikTok
- ☐ Snap Chat
- ☐ YouTube
- ☐ Twitter / X
- ☐ Other...

On which platforms have you experienced or seen cyberbullying (hate speech, hate comments, bullying, trolling, etc.)? *

- ☐ Facebook
- ☐ Instagram
- ☐ WhatsApp
- ☐ TikTok
- ☐ Snap Chat
- ☐ YouTube
- ☐ Twitter / X
- ☐ Other...

How often do you use social media? *

- ○ Daily
- ○ Weekly
- ○ Occasionally
- ○ Rarely

SECTION 3: Cyberbullying Experience & Awareness

Description (optional)

Have you ever personally witnessed or been affected by cyberbullying (hate speech, hate comments, bullying, trolling, etc.)? *

- ○ Yes
- ○ No
- ○ Prefer not to say

If yes, did you report it? *

- ○ Yes – to the platform (e.g., Facebook, Instagram)
- ○ Yes – to the authorities (e.g., Police, SLCERT)
- ○ I didn't know how or whom to report
- ○ No

If you didn't report it, why not?

Short answer text

What types of harmful content did you see in cyberbullying incidents? *

- ☐ Insults or harsh comments
- ☐ Threats or blackmail
- ☐ Spreading false rumors
- ☐ Harassing or repeated messages
- ☐ Sharing personal or private photos
- ☐ Exclusion from groups
- ☐ Sexual or inappropriate content
- ☐ Other...

SECTION 4: Language Usage on Social Media

Description (optional)

In which language(s) do you mostly communicate on social media? *

- [ ] Sinhala (typed in Sinhala letters – Unicode)
- [ ] Sinhala (typed using English letters – e.g., "mokakda?")
- [ ] English
- [ ] Tamil
- [ ] Other...

Do you think cyberbullying often happens in Sinhala? *

- ( ) Yes – mostly in Sinhala Unicode
- ( ) Yes – mostly in Sinhala typed with English letters
- ( ) No – mostly in English
- ( ) Not sure

SECTION 5: Suggestions

Description (optional)

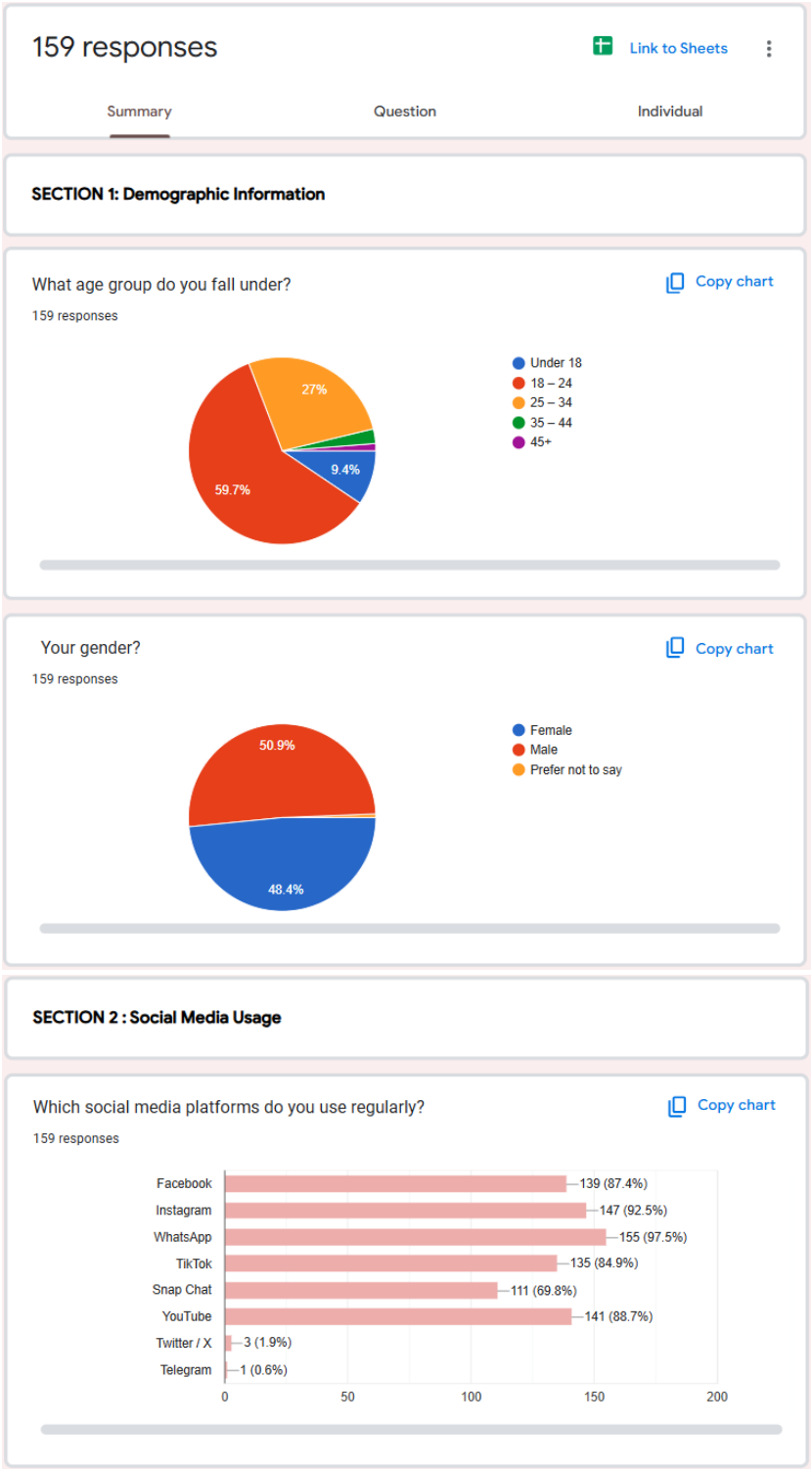
Do you think that Sri Lankan cybercrime units should focus more on solving and addressing these cybercrimes/cyberbullying attempts on social media? *

- ( ) Yes
- ( ) No

Do you have any suggestions for tools or actions that could help reduce cyberbullying in Sri Lanka?

Long answer text

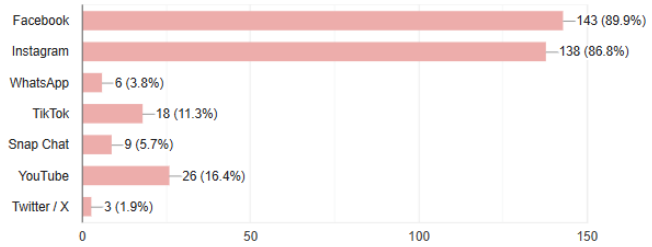
ii. Screenshots of the Survey Results



On which platforms have you experienced or seen cyberbullying (hate speech, hate comments, bullying, trolling, etc.)?

Copy chart

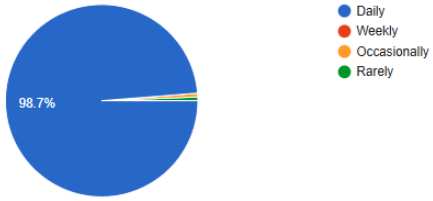
159 responses



How often do you use social media?

Copy chart

159 responses

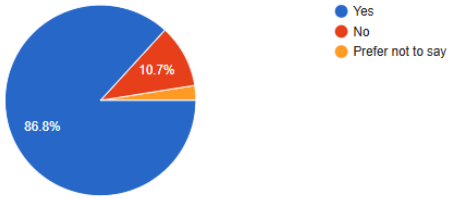


SECTION 3: Cyberbullying Experience & Awareness

Have you ever personally witnessed or been affected by cyberbullying (hate speech, hate comments, bullying, trolling, etc.)?

Copy chart

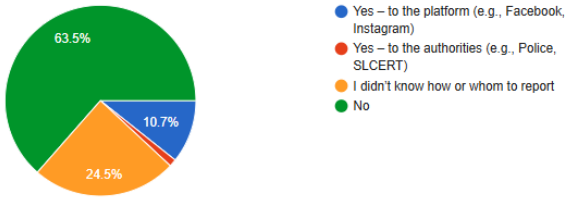
159 responses



If yes, did you report it?

Copy chart

159 responses



If you didn't report it, why not?

6 responses

I didn't knew what to do

I don't know

I reported it

Didn't think of it that much beacuse i was not bullied although i witnessed

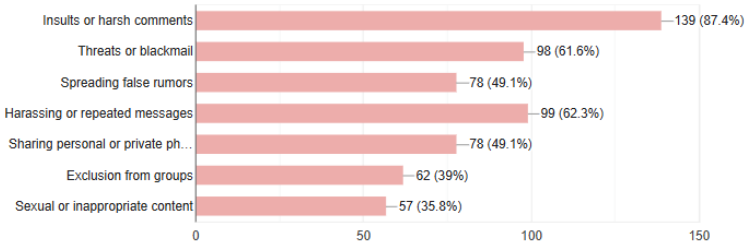
No point.

Didn't want to get in Unecessary Trouble

What types of harmful content did you see in cyberbullying incidents?

Copy chart

159 responses

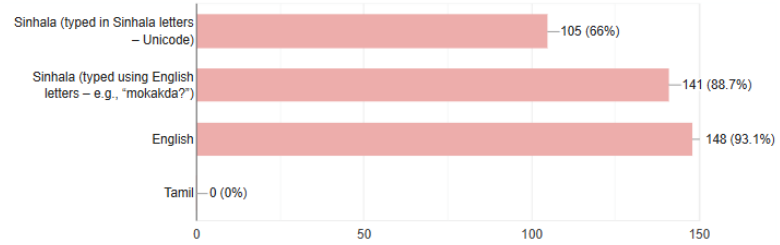


SECTION 4: Language Usage on Social Media

In which language(s) do you mostly communicate on social media?

Copy chart

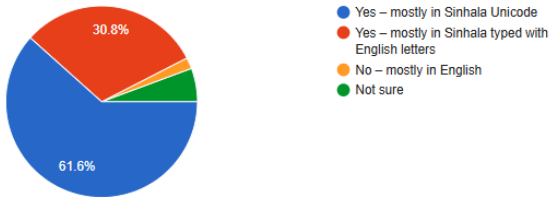
159 responses



Do you think cyberbullying often happens in Sinhala?

Copy chart

159 responses

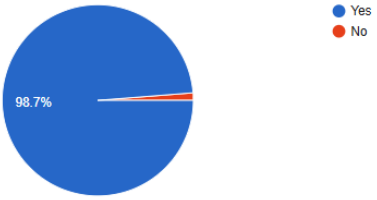


SECTION 5: Suggestions

Do you think that Sri Lankan cybercrime units should focus more on solving and addressing these cybercrimes/cyberbullying attempts on social media?

Copy chart

159 responses



Do you have any suggestions for tools or actions that could help reduce cyberbullying in Sri Lanka?

10 responses

Create a centralized national reporting platform for cyberbullying (like Report Harmful Content UK). Enhance capabilities of existing platforms like CERT|CC Sri Lanka (www.cert.gov.lk) to specifically handle cyberbullying complaints.

They can create a anonymous reporting systems

Make other ones aware of it...

Can we have a mechanism to identify which people do cyberbullying and have a personnel record for everyone in the country which done any bullying.

Now

Yes,

Implement nationwide digital citizenship education in schools, universities, and community centers, teaching youth about responsible online behavior.

Run multilingual public awareness campaigns (Sinhala, Tamil, and English) about the impact of

APPENDIX 6 – Full Code of the Tool

hawkeye_main.py

hawkeye_main.py > predict_batch

```python
import os
import sqlite3
import pandas as pd
import subprocess
import tarfile
import hashlib
import warnings
import re
import unicodedata
import glob
import sys
from colorama import Fore, Style, init as colorama_init
import os, sys, json, tempfile, subprocess, datetime as _dt


# ML imports
from transformers import AutoTokenizer, AutoModelForSequenceClassification
import torch

# ========================
# Setup
# ========================
warnings.simplefilter(action='ignore', category=FutureWarning)
colorama_init(autoreset=True)

OUTPUT_EXTRACT = r"C:\fyp\FYP_HateSpeechModel\android_media_extraction"
OUTPUT_FB_CSV = r"C:\fyp\FYP_HateSpeechModel\extracted_csv\facebook_all_text.csv"
OUTPUT_IG_CSV = r"C:\fyp\FYP_HateSpeechModel\extracted_csv\instagram_all_text.csv"
OUTPUT_MSG_CSV = r"C:\fyp\FYP_HateSpeechModel\extracted_csv\messenger_all_text.csv"
OUTPUT_FB_HASH_REPORT = r"C:\fyp\FYP_HateSpeechModel\evidence_hash_reports\facebook_hash_report.csv"
OUTPUT_IG_HASH_REPORT = r"C:\fyp\FYP_HateSpeechModel\evidence_hash_reports\instagram_hash_report.csv"
OUTPUT_MSG_HASH_REPORT = r"C:\fyp\FYP_HateSpeechModel\evidence_hash_reports\messenger_hash_report.csv"

# Where to save model prediction outputs
OUTPUT_FB_PRED = r"C:\fyp\FYP_HateSpeechModel\hatespeech_prediction\facebook_hate_predictions.csv"
OUTPUT_IG_PRED = r"C:\fyp\FYP_HateSpeechModel\hatespeech_prediction\instagram_hate_predictions.csv"
OUTPUT_MSG_PRED = r"C:\fyp\FYP_HateSpeechModel\hatespeech_prediction\messenger_hate_predictions.csv"

# Model folder
MODEL_PATH = r".\saved_model"

# Keywords for text/timestamp columns
TEXT_KEYWORDS = ["text", "message", "body", "caption", "comment", "title", "content", "description"]
TIME_KEYWORDS = [
    "time", "date", "timestamp", "created", "creation","sent", "received", "delivered", "start_time",
    "client_time", "server_time","message_time", "modified", "updated", "executed_time"
]

# ========================
# Utility
# ========================
#Connecting to the Android Device


def run_cmd(cmd):
    """Run shell command and return (stdout, stderr, returncode)."""
    result = subprocess.run(cmd, shell=True, capture_output=True, text=True)
    return result.stdout.strip(), result.stderr.strip(), result.returncode

def adb_connect(target):
    """Connect to ADB target and verify device connection. Prints success on connect."""
    out, err, rc = run_cmd(f'adb connect {target}')
    combined = f"{out}\n{err}".strip()
    success = ("connected to" in combined.lower()) or ("already connected to" in combined.lower())

    if success:
        devs_out, _, _ = run_cmd("adb devices")
        if target.split(":")[0] in devs_out or target in devs_out:
            print(Fore.GREEN + "\n✅✅✅ Android Mobile Phone is Successfully Connected via ADB..." + Style.RESET_ALL)
            return True

    print(Fore.RED + "\n❌ ADB connection failed. Output:\n" + combined + Style.RESET_ALL)
    return False

```

```python
def prompt_for_ip(default_port=5555):
    """
    Ask user for device IP (optionally with :port). Returns normalized 'ip:port' string.
    - If user enters '192.168.1.23' -> returns '192.168.1.23:5555'
    - If user enters '192.168.1.23:5566' -> returns as-is
    """
    while True:
        ip = input(Fore.YELLOW + "\nEnter the IP of the Android device or IP:PORT (e.g. 192.168.xx.xx or 192.168.xx.xx:xxxx): " + Style.RESET_ALL).strip()
        if not ip:
            print(Fore.RED + "\nIP is required. Try again (or press Ctrl+C to quit)." + Style.RESET_ALL)
            continue

        # Basic IPv4 + optional :port validation
        m = re.match(r'^(\d{1,3}(?:\.\d{1,3}){3})(?::(\d{1,5}))?$', ip)
        if not m:
            print(Fore.RED + "\nInvalid format. Please enter like 192.168.x.x or 192.168.x.x:port" + Style.RESET_ALL)
            continue

        host, port = m.group(1), m.group(2)
        # Quick octet sanity check (0-255)
        try:
            if not all(0 <= int(o) <= 255 for o in host.split(".")):
                raise ValueError
        except ValueError:
            print(Fore.RED + "\nInvalid IP octets. Each should be 0-255." + Style.RESET_ALL)
            continue

        if port is None:
            port = str(default_port)

        return f"{host}:{port}"

#Generating Hash Reports

def sha256sum(file_path):
    h = hashlib.sha256()
    with open(file_path, "rb") as f:
        for chunk in iter(lambda: f.read(4096), b""):
            h.update(chunk)
    return h.hexdigest()
```

```python

def generate_hash_report(folder_path, report_csv):
    hash_records = []
    for root, _, files in os.walk(folder_path):
        for file in files:
            if file.endswith(".db"):
                file_path = os.path.join(root, file)
                file_hash = sha256sum(file_path)
                hash_records.append({"database_file": file, "sha256_hash": file_hash})
    os.makedirs(os.path.dirname(report_csv), exist_ok=True)
    if hash_records:
        df = pd.DataFrame(hash_records)
        df.to_csv(report_csv, index=False, encoding="utf-8")
        print(Fore.GREEN + f"\n✅ Hash report saved: {report_csv}" + Style.RESET_ALL)
    else:
        print(Fore.RED + "\n‼️ No database files found for hashing." + Style.RESET_ALL)

#Pulling the data from Android Device

def pull_sdcard_app_data(package_name, base_out_dir, label):
    """
    Create tar on device (/sdcard/Download/{label}_data.tar),
    pull to PC (base_out_dir), list DBs, then extract ONLY .db files.
    Returns (success:bool, db_root_dir:str|None)
    """
    os.makedirs(base_out_dir, exist_ok=True)

    tar_filename = f"{label}_data.tar"  # device name
    sdcard_tar = f"/sdcard/Download/{tar_filename}"
    local_tar = os.path.join(base_out_dir, f"{label}_data_data.tar")  # keep your current naming style

    print(Fore.CYAN + f"\n🔄 Creating TAR for {label} on device..." + Style.RESET_ALL)
    create_tar_cmd = f"adb shell su -c 'tar --create --file={sdcard_tar} /data/data/{package_name}'"
    out, err, rc = run_cmd(create_tar_cmd)

    print(Fore.CYAN + f"\n🔄 Pulling TAR for {label} to PC..." + Style.RESET_ALL)
    out, err, rc = run_cmd(f'adb pull {sdcard_tar} "{local_tar}"')
    if rc != 0 or not os.path.exists(local_tar):
        print(Fore.RED + f"\n[ ‼️ ] Failed to pull TAR for {label}.\n{out}\n{err}" + Style.RESET_ALL)
        return False, None
```

```python
    print(Fore.CYAN + f"\n🔄 Extracting ONLY .db files for {label}..." + Style.RESET_ALL)
    db_out_dir = os.path.join(base_out_dir, "databases_extracted")
    extracted_count = extract_databases_from_tar(local_tar, db_out_dir)
    if extracted_count == 0:
        print(Fore.YELLOW + f"\n⚠️ No .db files extracted for {label}." + Style.RESET_ALL)
        return False, None

    print(Fore.GREEN + f"\n✅ Extracted {extracted_count} database file(s) to {db_out_dir}" + Style.RESET_ALL)
    return True, db_out_dir
```

```python
INVALID_WIN_CHARS = '<>:"/\\|?*'

def sanitize_path_component(name: str) -> str:
    """Make a filename safe for Windows."""
    cleaned = ''.join('_' if ch in INVALID_WIN_CHARS else ch for ch in name)
    return cleaned.strip().rstrip('.')

def safe_join(base: str, *parts: str) -> str:
    """Join and normalize, preventing path traversal."""
    joined = os.path.join(base, *parts)
    norm = os.path.normpath(joined)
    if not os.path.abspath(norm).startswith(os.path.abspath(base)):
        raise ValueError("Blocked unsafe path traversal")
    return norm

def list_databases_in_tar(local_tar: str):
    """Return a list of .db member paths inside the tar."""
    dbs = []
    with tarfile.open(local_tar, "r") as tar:
        for m in tar.getmembers():
            if m.isfile() and m.name.lower().endswith(".db"):
                dbs.append(m.name)
    return sorted(dbs)

#Extracting the .dbs from the .tar file

def extract_databases_from_tar(local_tar: str, out_dir: str) -> int:
    """
    Extract ONLY .db files from local_tar into out_dir,
    preserving a sanitized subpath. Returns count extracted.
    """
    os.makedirs(out_dir, exist_ok=True)
    count = 0
    with tarfile.open(local_tar, "r") as tar:
        for m in tar.getmembers():
            if not (m.isfile() and m.name.lower().endswith(".db")):
                continue
            # normalize and sanitize every path component
            rel = m.name.lstrip("./")
            parts = [sanitize_path_component(p) for p in rel.split("/")]
            target = safe_join(out_dir, *parts)
            os.makedirs(os.path.dirname(target), exist_ok=True)
            try:
                fobj = tar.extractfile(m)
                if fobj is None:
                    continue
                with open(target, "wb") as wf:
                    wf.write(fobj.read())
                count += 1
            except Exception as e:
                print(Fore.YELLOW + f"\n⚠ Skipped {m.name}: {e}" + Style.RESET_ALL)
    return count
```

```python
def find_tar(out_dir: str, label: str):
    """
    Find the app tar file we just pulled.
    Accepts fb_data.tar / fb_data_data.tar / any *.tar in folder.
    """
    candidates = [
        os.path.join(out_dir, f"{label}_data.tar"),
        os.path.join(out_dir, f"{label}_data_data.tar"),
    ] + glob.glob(os.path.join(out_dir, "*.tar"))
    for p in candidates:
        if os.path.isfile(p):
            return p
    return None
```

```python
def _parse_any_epoch(v):
    """
    Try to parse numbers like epoch ms/sec; otherwise return pandas.NaT.
    """
    try:
        x = pd.to_numeric(v, errors="coerce")
        if pd.isna(x):
            return pd.NaT
        # milliseconds vs seconds heuristics
        if x >= 1e12:
            return pd.to_datetime(x, unit="ms", errors="coerce")
        if 1e9 <= x < 1e12:
            return pd.to_datetime(x, unit="s", errors="coerce")
    except Exception:
        pass
    return pd.NaT


def _first_nice_datetime(row: pd.Series, timeish_cols: list[str]) -> str:
    """
    From a row and a list of timeish column names, return a nice date string:
    - Prefer epoch millis/seconds if present
    - Otherwise try free-form datetime parsing
    - If nothing usable: return "-"
    Output format: "YYYY-MM-DD" (you can switch to "%H:%M, %d/%m/%Y" if you prefer)
    """
    for c in timeish_cols:
        if c not in row:
            continue
        v = row.get(c, "")
        if pd.isna(v) or str(v).strip() in ("", "0", "None", "nan", "NaN"):
            continue

        ts = _parse_any_epoch(v)
        if pd.notna(ts):
            return ts.strftime("%Y-%m-%d")

        # try parsing free-form date strings
        ts = pd.to_datetime(str(v), errors="coerce")
        if pd.notna(ts):
            return ts.strftime("%Y-%m-%d")

    # fallbacks: if there are explicit 'date' + 'time' split columns
    if ("date" in row) and ("time" in row):
        combo = f"{row.get('date','')} {row.get('time','')}"
        ts = pd.to_datetime(combo, errors="coerce")
        if pd.notna(ts):
            return ts.strftime("%Y-%m-%d")

    return "-"
```

```python
# ========================
# Database Parsing
# ========================

def extract_text_from_db(db_path):
    """Extract rows from tables, prioritizing text fields but falling back to all fields if no match."""
    extracted_rows = []
    try:
        conn = sqlite3.connect(db_path)
        cursor = conn.cursor()

        cursor.execute("SELECT name FROM sqlite_master WHERE type='table'")
        tables = [t[0] for t in cursor.fetchall()]

        for table in tables:
            try:
                cursor.execute(f"PRAGMA table_info({table})")
                col_names = [c[1] for c in cursor.fetchall()]
                low = [c.lower() for c in col_names]

                text_cols = [c for c in col_names if any(k in c.lower() for k in TEXT_KEYWORDS)]
                time_cols = [c for c in col_names if any(k in c.lower() for k in TIME_KEYWORDS)]
```

```python
                if text_cols:
                    selected_cols = list(dict.fromkeys(text_cols + time_cols))  # keep order unique
                else:
                    selected_cols = col_names  # fallback: all columns

                if not selected_cols:
                    continue

                query = f"SELECT {', '.join(selected_cols)} FROM {table}"
                df = pd.read_sql_query(query, conn)

                # Best-effort unify timestamps into one 'date_time' column
                if time_cols:
                    # Compute one nice string per row
                    df["date_time"] = df.apply(lambda r: _first_nice_datetime(r, time_cols), axis=1)
                elif {"date", "time"}.issubset(set(col_names)):
                    df["date_time"] = df.apply(lambda r: _first_nice_datetime(r, ["date", "time"]), axis=1)
                else:
                    df["date_time"] = "-"

                df["source_database"] = os.path.basename(db_path)
                df["source_table"] = table
                extracted_rows.append(df)

            except Exception:
                continue

        conn.close()
    except Exception as e:
        print(Fore.RED + f"\n‼️ Error reading {db_path}: {e}" + Style.RESET_ALL)
    return extracted_rows

def parse_all_databases(folder_path, output_csv):
    """\nParse all .db files in a folder and save merged CSV."""
    all_dfs = []
```

```python
    for root, _, files in os.walk(folder_path):
        for file in files:
            if file.endswith(".db"):
                db_path = os.path.join(root, file)
                print(Fore.CYAN + f"    ▪ Parsing : {file}" + Style.RESET_ALL)
                extracted_data = extract_text_from_db(db_path)
                all_dfs.extend(extracted_data)

    non_empty_dfs = [df for df in all_dfs if not df.empty]
    if non_empty_dfs:
        merged_df = pd.concat(non_empty_dfs, ignore_index=True)
        merged_df.to_csv(output_csv, index=False, encoding="utf-8")
        print(Fore.GREEN + f"\n✅ Saved merged CSV: {output_csv}" + Style.RESET_ALL)
    else:
        print(Fore.RED + "\n‼️ No usable data found." + Style.RESET_ALL)

# ========================
# Model & Inference Helpers
# ========================

def load_model(model_path):
    print(Fore.CYAN + "\n♻️ Loading hate-speech model..." + Style.RESET_ALL)
    tokenizer = AutoTokenizer.from_pretrained(model_path)
    model = AutoModelForSequenceClassification.from_pretrained(model_path)
    model.eval()
    return tokenizer, model

def clean_text(text: str) -> str:
    text = re.sub(r"http\S+", "", str(text))
    text = re.sub(r"[^\wඅ-ෟාැෑිීුූෘෙේෛොෝෞංඃ ]+", " ", text)
    return text.strip().lower()
```

```python
def keyword_flag(text: str) -> bool:
    keywords = [
        # Singlish
        "gon","balla", "ballo","pissu","umbata","nari","wesi","haraka","thambiya","baduwak","modaya", "gona" , "besikaya", "haththa", "cari", "kari", "harakek",
        "puka","mada","fuck","idiot","lamayek","moda","buruwa","hutta","utto", "hutto","kolukaraya","gona", "gon gani","gani", "pakaya" , "bitch" , "pissek"
        # Sinhala
        "ගොන්","බල්ලා","පිස්සු","මෝඩයෙක්","පිස්සෙක්","පිස්සු","හරකෙක්","නරකයා","වදිල්ල","පකයා","අල්ලගමු","මුහුණ","බල්ලෝ","උඹට","නරි","වෙසි",
        "වේසිය","හරකා","තම්බියා","බඩුවක්","මෝඩයා","ගොනා","බේසිකයා","හැත්ත","කැරි","පුක","ෆක්","ළමයෙක්","මෝඩ","බූරුවා","හුත්ත","උත්තෝ",
        "හුත්තෝ","කොලුකාරයා","ගොනා","ගොන් ගෑණි","පකයා","බිච්","ගෑණි"
    ]
    normalized = unicodedata.normalize("NFC", str(text).lower())
    return any(kw in normalized for kw in keywords)
```

```python
def predict_batch(df: pd.DataFrame, tokenizer, model):
    """
    Runs detection on a dataframe containing multiple potential text columns.
    Returns a dataframe with columns: text, predicted_label, confidence, source_database, source_table
    """
    if df.empty:
        return pd.DataFrame(columns=["text","predicted_label","confidence","source_database","source_table","date_time"])

    # choose text-like columns
    textish_cols = [c for c in df.columns if any(k in c.lower() for k in TEXT_KEYWORDS)]
    if not textish_cols:
        textish_cols = [c for c in df.columns if df[c].dtype == object]

    # time-like columns to feed our formatter if unified column is missing
    timeish_cols = [c for c in df.columns if any(k in c.lower() for k in TIME_KEYWORDS)]

    # flatten to one "text" column
    texts, times, src_db, src_tbl = [], [], [], []
    for _, row in df.iterrows():
        pieces = []
        for c in textish_cols:
            val = row.get(c, "")
            if isinstance(val, str) and val.strip():
                pieces.append(val)
        if not pieces:
            continue

        combined = " | ".join(pieces)
        texts.append(combined)
        src_db.append(row.get("source_database", ""))
        src_tbl.append(row.get("source_table", ""))

        if "date_time" in df.columns:
            dt_val = row.get("date_time", "-")
            dt_val = "-" if pd.isna(dt_val) else str(dt_val)
        else:
            dt_val = _first_nice_datetime(row, timeish_cols) if timeish_cols else "-"
        times.append(dt_val)

    if not texts:
        return pd.DataFrame(columns=["text","predicted_label","confidence","source_database","source_table","date_time"])

    # Fallback first, then model
    cleaned = [clean_text(t) for t in texts]
    fallback_mask = [keyword_flag(t) for t in texts]

    non_fb_idx = [i for i, m in enumerate(fallback_mask) if not m]
```

```python
    if non_fb_idx:
        enc = tokenizer([cleaned[i] for i in non_fb_idx], return_tensors="pt",
                        truncation=True, padding=True, max_length=128)
        with torch.no_grad():
            outputs = model(**enc)
            probs = torch.softmax(outputs.logits, dim=1).cpu()
            pred_classes = torch.argmax(probs, dim=1).tolist()
            pred_conf = probs.max(dim=1).values.tolist()

    rows = []
    j = 0
    for i, text in enumerate(texts):
        if fallback_mask[i]:
            lbl, conf = 1, 0.99
        else:
            lbl, conf = int(pred_classes[j]), float(pred_conf[j])
            j += 1
        rows.append([text, lbl, conf, times[i], src_db[i], src_tbl[i]])

    return pd.DataFrame(rows, columns=[
        "text", "predicted_label", "confidence",
        "date_time", "source_database", "source_table"
    ])

    #out_df = pd.DataFrame(rows, columns=["text","predicted_label","confidence"])
    #out_df["source_database"] = src_db
    #out_df["source_table"] = src_tbl
    #return out_df
```

```python
def run_detection_on_csv(input_csv: str, output_csv: str, tokenizer, model):
    if not os.path.exists(input_csv):
        print(Fore.YELLOW + f"\n⚠️ Skipping predictions: CSV not found -> {input_csv}" + Style.RESET_ALL)
        return 0, 0

    print(Fore.CYAN + f"\n🔎 Running hate-speech detection on: {input_csv}" + Style.RESET_ALL)
    df = pd.read_csv(input_csv, encoding="utf-8", low_memory=False)
    pred_df = predict_batch(df, tokenizer, model)

    if pred_df.empty:
        print(Fore.YELLOW + f"\n⚠️ No text-like data found in {input_csv}." + Style.RESET_ALL)
        return 0, 0

    os.makedirs(os.path.dirname(output_csv), exist_ok=True)
    pred_df.to_csv(output_csv, index=False, encoding="utf-8")
    total = len(pred_df)
    toxic = int((pred_df["predicted_label"] == 1).sum())
    print(Fore.GREEN + f"\n✅ Predictions saved: {output_csv}" + Style.RESET_ALL)
    print(Fore.MAGENTA + f"\n📊 Summary -> total: {total}, hate_speech: {toxic}, safe: {total - toxic}" + Style.RESET_ALL)
    return total, toxic

# ========================
# Banner
# ========================
def print_banner():
    print(Fore.MAGENTA +"""\n
                .                                                          .
              _/|\                                                       /|\_
             _|H/':                          _______                   :'\H|_
            _\HH|  `-._                  .'\_   _/'.                 _.-'  |HH/_
           _\HH\       HH".---.-....________.'/(v.-.v)\'._______.....-.--."HH       /HH/_
          _\Hb\.       Cb  CHHb  Cb_   '-.HH| _/_|_\_ |HH.-'   _dD  dHHD  dD       ./dH/_
         'C.HH'._      '   'oHb   'H.    \H\  \ V /  //H/    .H'   dHo'  '       _.'HH.D'
          'C.HHH`o._          ''-..__     |H\_ '-' _/H|     __.-''           _,o'HHH.D'
            'C.HHHH`o.__                 .'H/   T   \H'.                 __,o'HHHH.D'
              '-HHHHHHH"o.___         _.-'HH|  V | V  |HH'-._        ___.o"HHHHHHH-'
                'C.HHHHHHHHbioooooidHHH..H|     |    |H..HHHbioooooidHHHHHHHH.D'
                   '"CHHHHHHHHHHHHHH/  |H|  \|/  |H|  \HHHHHHHHHHHHHHD"'
                        ''::::''        H_|___|_H         ''::::''
                                     .\}.("   |   ").{/.
                                     (/' '\.  |  ./' '\)
                                    .'\    '  |  '    ,/'.
                                 { }|  '.     |     .'  |{ }
                                .iH|-._  \     |    /  _.-|Hi.
                               :oHH|'-._  |    |   |  _.-'|HHo:
                                  ''|   ''|---|---|''   |''
                                    .   /    |    \   .
                                     \.'     |     './
                                             '--.__|__.--'
    """+ Style.RESET_ALL)
```

```python
    print(Fore.CYAN +"""
|=============================================================================================|
|8888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888|
|=============================================================================================|
|88|                                                                                       |88|
|88|     :::    :::     :::     :::       ::: :::    ::: :::::::::: :::   ::: ::::::::::    |88|
|88|     :+:    :+:   :+: :+:   :+:       :+: :+:   :+:  :+:        :+:   :+: :+:           |88|
|88|     +:+    +:+  +:+   +:+  +:+       +:+ +:+  +:+   +:+         +:+ +:+  +:+          |88|
|88|     +#++:++#++ +#++:++#++: +#+  +:+  +#+ +#++:++    +#++:++#     +#++:   +#++:++#      |88|
|88|     +#+    +#+ +#+     +#+ +#+ +#+#+ +#+ +#+  +#+   +#+           +#+    +#+           |88|
|88|     #+#    #+# #+#     #+#  #+#+# #+#+#  #+#   #+#  #+#           #+#    #+#           |88|
|88|     ###    ### ###     ###   ###   ###   ###    ### ##########    ###    ##########    |88|
|88|                                                                                       |88|
|=============================================================================================|
|8888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888|
|888                                                                                     888|
|888            ANDROID FORENSIC TOOL FOR HARSH SPEECH DETECTION ON SOCIAL MEDIA          888|
|888                                                                                     888|
|8888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888|
|=============================================================================================|
    """+ Style.RESET_ALL)

def find_db_root(folder):
    for r, dirs, _ in os.walk(folder):
        if os.path.basename(r) == "databases":
            return r
    return folder

#=========================
# report generation
#=========================
def ask_yes_no(q: str) -> bool:
        a = input(q + " [Y/N][Yes/No]: ").strip().lower()
        return a in ("y", "yes", "Y")
```

```
def report_main():
        if not ask_yes_no(Fore.GREEN + "\n👀 Do you need the final hate-speech detection report as a PDF?" + Style.RESET_ALL):
            print(Fore.MAGENTA +"\n👍 Okay skipping PDF generation...")
            print(Fore.BLUE +"\n👋  Quiting.....")
            print(Fore.CYAN +"""\n
|=============================================================================================|
|8888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888|
|=============================================================================================|
|88|                                                                                       |88|
|88|     :::    :::     :::     :::       ::: :::    ::: ::::::::::: :::   ::: ::::::::::    |88|
|88|     :+:    :+:   :+: :+:   :+:       :+: :+:   :+:  :+:        :+:   :+: :+:           |88|
|88|     +:+    +:+  +:+   +:+  +:+       +:+ +:+  +:+   +:+         +:+ +:+  +:+           |88|
|88|     +#++:++#++ +#++:++#++: +#+  +:+  +#+ +#++:++    +#++:++#     +#++:   +#++:++#      |88|
|88|     +#+    +#+ +#+     +#+ +#+ +#+#+ +#+ +#+  +#+   +#+           +#+    +#+           |88|
|88|     #+#    #+# #+#     #+#  #+#+# #+#+#  #+#   #+#  #+#           #+#    #+#           |88|
|88|     ###    ### ###     ###   ###   ###   ###    ### ##########    ###    ##########    |88|
|88|                                                                                       |88|
|=============================================================================================|
|8888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888|
|888                                                                                       888|
|888            ANDROID FORENSIC TOOL FOR HARSH SPEECH DETECTION ON SOCIAL MEDIA            888|
|888                                                                                       888|
|8888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888|
|=============================================================================================|
        """)
            print(Fore.MAGENTA +"""
::::::::::: :::    :::     :::     ::::    ::: :::    :::    :::   :::  ::::::::  :::    ::: :::
    :+:     :+:    :+:   :+: :+:   :+:+:   :+: :+:   :+:     :+:   :+: :+:    :+: :+:    :+: :+:
    +:+     +:+    +:+  +:+   +:+  :+:+:+:+ +:+ +:+  +:+       +:+ +:+  +:+    +:+ +:+    +:+ +:+
    +#+     +#++:++#++ +#++:++#++: +#+ +:+ +#+ +#++:++           +#++:   +#+    +:+ +#+    +:+ +:+
    +#+     +#+    +#+ +#+     +#+ +#+  +#+#+# +#+  +#+         +#+    +#+    +#+ +#+    +#+
    #+#     #+#    #+# #+#     #+# #+#   #+#+# #+#   #+#        #+#    #+#    #+# #+#    #+# #+#
    ###     ###    ### ###     ### ###    #### ###    ###       ###     ########   ########  ###
            """+ Style.RESET_ALL)
            return
```

```
        print(Fore.BLUE + """\n
_.-._.-._.-._.-._.-._.-._.-._.-._.-._.-._.-._.-._.-._.-._.-._.-._.-._.-._.-._.-._.-._.-._.-._."""+ Style.RESET_ALL)

        print(Fore.BLUE + """\n
                .8888888888888888888.              .8888888888888888888.
              .888                  888.         .888                  888.
             .88                      88.     .88                      88.
                .8888888888888888.                .8888888888888888.
               .88   .000000000.   88.            .88   .000000000.   88.
              .88    000 00000 000    88.          .88    000 00000 000    88.
             .88     000 000 000 000     88.     .88     000 000 000 000     88.
              '88    000 00000 000    88'         '88    000 00000 000    88'
                '88   '000000000'   88'            '88   '000000000'   88'
                  '8888888888888888'                '8888888888888888'

|=============================================================================================|
|                                OUTPUT RESULTS PDF GENERATION                                |
|=============================================================================================|"""+ Style.RESET_ALL)

        case_id      = input(Fore.YELLOW + "\n📂 Enter the Case ID :  " + Style.RESET_ALL).strip()
        investigator = input(Fore.YELLOW + "\n👮 Enter the name of the Investigator : " + Style.RESET_ALL).strip()
        owner        = input(Fore.YELLOW + "\n📱 Enter the Owner's name of the mobile device : " + Style.RESET_ALL).strip()

        print(Fore.BLUE + """\n
_.-._.-._.-._.-._.-._.-._.-._.-._.-._.-._.-._.-._.-._.-._.-._.-._.-._.-._.-._.-._.-._.-._."""+ Style.RESET_ALL)

        # Leave device_id and usernames blank → the report script will auto-detect.
        cfg = {
            "case_id": case_id,
            "investigator": investigator,
            "owner": owner,
            "fb_username": "",        # auto
            "ig_username": "",        # auto
            "ms_username": ""         # auto
            # You can add/override paths here if you want, or rely on defaults in the report script.
        }

        cfg_path = os.path.join(tempfile.gettempdir(), "hawkeye_report_config.json")
        with open(cfg_path, "w", encoding="utf-8") as f:
            json.dump(cfg, f, ensure_ascii=False, indent=2)

        # If you run from a venv, swap "python" with the full path to env/Scripts/python.exe
        cmd = [sys.executable, "report_pdf_structured.py", "--config", cfg_path]
        print(Fore.MAGENTA +"\n♻️ Running Hawkeye PDF Generator ..... ", "" .join(cmd))
        subprocess.run(cmd, check=True)
```

```python
# ========================
# Main Workflow
# ========================
def main():
    # Show banner
    print_banner()

    # Connect to device
    target = prompt_for_ip(default_port=5555)
    if not adb_connect(target):
        return
    os.environ["ANDROID_SERIAL"] = target

    print(Fore.BLUE + """\n
_._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._."""+ Style.RESET_ALL)

    print(Fore.BLUE + """\n
                 .88888888888888888888.              .88888888888888888888.
              .888                     888.       .888                     888.
            .88                           88.    .88                          88.
                   .888888888888888888.                .888888888888888888.
                 .88   .000000000.   88.             .88   .000000000.   88.
               .88    000 00000 000    88.         .88    000 00000 000    88.
             .88     000 000 000 000     88.     .88     000 000 000 000     88.
               '88    000 00000 000    88'         '88    000 00000 000    88'
                 '88   '000000000'   88'             '88   '000000000'   88'
                   '888888888888888888'                '888888888888888888'

|============================================================================================|
|                               Social Media Data Extraction                                 |
|============================================================================================|"""+ Style.RESET_ALL)

    # === Facebook Extraction ===
    print(Fore.MAGENTA + "\n📍 Extracting Facebook Databases..." + Style.RESET_ALL)
    fb_ok, fb_db_root = pull_sdcard_app_data("com.facebook.katana", OUTPUT_EXTRACT, "fb")
    if fb_ok and fb_db_root:
        print(Fore.BLUE + "\n📍 Generating SHA-256 Hash Report..." + Style.RESET_ALL)
        generate_hash_report(fb_db_root, OUTPUT_FB_HASH_REPORT)
        print(Fore.MAGENTA + "\n📍 Parsing Facebook Databases..." + Style.RESET_ALL)
        parse_all_databases(fb_db_root, OUTPUT_FB_CSV)

    print(Fore.BLUE + """\n
_._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._."""+ Style.RESET_ALL)

    # === Instagram Extraction ===
    print(Fore.MAGENTA + "\n📍 Extracting Instagram Databases..." + Style.RESET_ALL)
    ig_ok, ig_db_root = pull_sdcard_app_data("com.instagram.android", OUTPUT_EXTRACT, "ig")
    if ig_ok and ig_db_root:
        print(Fore.BLUE + "\n📍 Generating SHA-256 Hash Report..." + Style.RESET_ALL)
        generate_hash_report(ig_db_root, OUTPUT_IG_HASH_REPORT)
        print(Fore.MAGENTA + "\n[📍IG] Parsing Instagram Databases..." + Style.RESET_ALL)
        parse_all_databases(ig_db_root, OUTPUT_IG_CSV)

    print(Fore.BLUE + """\n
_._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._."""+ Style.RESET_ALL)

    # === Messenger Extraction ===
    print(Fore.MAGENTA + "\n📍 Extracting Messenger Databases..." + Style.RESET_ALL)
    msg_ok, msg_db_root = pull_sdcard_app_data("com.facebook.orca", OUTPUT_EXTRACT, "ms")
    if msg_ok and msg_db_root:
        print(Fore.BLUE + "\n📍 Generating SHA-256 Hash Report..." + Style.RESET_ALL)
        generate_hash_report(msg_db_root, OUTPUT_MSG_HASH_REPORT)
        print(Fore.MAGENTA + "\n📍Parsing Messenger Databases..." + Style.RESET_ALL)
        parse_all_databases(msg_db_root, OUTPUT_MSG_CSV)

    print(Fore.GREEN + "\n✅✅✅ Extraction, Hashing, and Parsing Completed!" + Style.RESET_ALL)

    print(Fore.BLUE + """\n
_._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._."""+ Style.RESET_ALL)

    # === Hate Speech Detection ===
    print(Fore.BLUE + """\n
                 .88888888888888888888.              .88888888888888888888.
              .888                     888.       .888                     888.
            .88                           88.    .88                          88.
                   .888888888888888888.                .888888888888888888.
                 .88   .000000000.   88.             .88   .000000000.   88.
               .88    000 00000 000    88.         .88    000 00000 000    88.
             .88     000 000 000 000     88.     .88     000 000 000 000     88.
               '88    000 00000 000    88'         '88    000 00000 000    88'
                 '88   '000000000'   88'             '88   '000000000'   88'
                   '888888888888888888'                '888888888888888888'

|============================================================================================|
|                                  Hate Speech Detection                                     |
|============================================================================================|""" + Style.RESET_ALL)
```

```
    tokenizer, model = load_model(MODEL_PATH)
    fb_total, fb_toxic = run_detection_on_csv(OUTPUT_FB_CSV, OUTPUT_FB_PRED, tokenizer, model)
    ig_total, ig_toxic = run_detection_on_csv(OUTPUT_IG_CSV, OUTPUT_IG_PRED, tokenizer, model)
    msg_total, msg_toxic = run_detection_on_csv(OUTPUT_MSG_CSV, OUTPUT_MSG_PRED, tokenizer, model)

    print(Fore.BLUE + """\n
_._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._."""+ Style.RESET_ALL)

    # === Final Summary ===
    print(Fore.BLUE + """\n
                .8888888888888888888.              .8888888888888888888.
             .888                  888.         .888                  888.
           .88                       88.     .88                       88.
                .8888888888888888.                .8888888888888888.
               .88   .000000000.   88.            .88   .000000000.   88.
              .88    000 00000 000    88.         .88    000 00000 000    88.
             .88     000 000 000 000     88.     .88     000 000 000 000     88.
              '88    000 00000 000    88'         '88    000 00000 000    88'
               '88   '000000000'   88'             '88   '000000000'   88'
                 '8888888888888888'                 '8888888888888888'

|=============================================================================================|
|                                Hate Speech Detection Summary                                |
|=============================================================================================|""" + Style.RESET_ALL)
    print(Fore.BLUE + """\n
    Facebook :""" + Style.RESET_ALL)
    print(Fore.MAGENTA + f"""
        👉 Total = {fb_total}
        👉 Hate Speech = {fb_toxic}
        👉 Safe = {fb_total - fb_toxic}""" + Style.RESET_ALL)
    print(Fore.BLUE + """\n
    Instagram :""" + Style.RESET_ALL)
    print(Fore.MAGENTA + f"""
        👉 Total = {ig_total}
        👉 Hate Speech = {ig_toxic}
        👉 Safe = {ig_total - ig_toxic}""" + Style.RESET_ALL)
    print(Fore.BLUE + """\n
    Messenger :""" + Style.RESET_ALL)
    print(Fore.MAGENTA + f"""
        👉 Total = {msg_total}
        👉 Hate Speech = {msg_toxic}
        👉 Safe = {msg_total - msg_toxic}""" + Style.RESET_ALL)

    print(Fore.BLUE + """\n
_._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._."""+ Style.RESET_ALL)

    print(Fore.GREEN + "\n✅✅✅ Extractions and Hate Speech Detection All Completed!" + Style.RESET_ALL)
    print(Fore.GREEN + "\n💾 Saved Locations of the Generated Reports :" + Style.RESET_ALL)
    print(Fore.YELLOW +"\n    👉 " + OUTPUT_FB_PRED)
    print(Fore.YELLOW +"\n    👉 " + OUTPUT_IG_PRED)
    print(Fore.YELLOW +"\n    👉 " + OUTPUT_MSG_PRED)
```

```
    print(Fore.BLUE + """\n
_._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._."""+ Style.RESET_ALL)

    # === Final report ===
    report_main()

if __name__ == "__main__":
    # your existing main workflow:
    main()
```

hawkeye_main.py | report_pdf_structured.py

report_pdf_structured.py > build_report

```python
# report_pdf_structured.py — clean text + logo cover + wrapped tables + platform summary pages
import os
import re
import json  # NEW
import argparse
import datetime as dt
import tempfile
import shutil
import warnings
import numpy as np

import pandas as pd
from fpdf import FPDF

import glob
from colorama import Fore, Style, init as colorama_init


# silence fpdf2 deprecation noise from older examples
warnings.filterwarnings("ignore", category=DeprecationWarning, module="fpdf")

# ---- matplotlib: non-interactive backend BEFORE pyplot
from matplotlib import use as mpl_use
mpl_use("Agg")
import matplotlib.pyplot as plt

try:
    from fontTools.ttLib import TTFont
except Exception:
    TTFont = None


# ---------- Defaults ----------
DEFAULT_OUT_DIR = r"C:\fyp\FYP_HateSpeechModel\pdf_reports"

DEFAULT_FB_PRED = r"C:\fyp\FYP_HateSpeechModel\hatespeech_prediction\facebook_hate_predictions.csv"
DEFAULT_IG_PRED = r"C:\fyp\FYP_HateSpeechModel\hatespeech_prediction\instagram_hate_predictions.csv"
DEFAULT_MS_PRED = r"C:\fyp\FYP_HateSpeechModel\hatespeech_prediction\messenger_hate_predictions.csv"

DEFAULT_FB_HASH = r"C:\fyp\FYP_HateSpeechModel\evidence_hash_reports\facebook_hash_report.csv"
DEFAULT_IG_HASH = r"C:\fyp\FYP_HateSpeechModel\evidence_hash_reports\instagram_hash_report.csv"
DEFAULT_MS_HASH = r"C:\fyp\FYP_HateSpeechModel\evidence_hash_reports\messenger_hash_report.csv"

DEFAULT_FONT = r"C:\fyp\FYP_HateSpeechModel\fonts\static\NotoSansSinhala-Regular.ttf"
DEFAULT_LOGO = r"C:\fyp\FYP_HateSpeechModel\assets\hawkeye_logo.png"

```

```python
# ---------- Text cleaning helpers ----------
_CTRL_RE = re.compile(r"[\x00-\x08\x0b\x0c\x0e-\x1f]")
_EMOJI_RE = re.compile(r"[\U0001F300-\U0001FAFF\U00002700-\U000027BF\U0001F900-\U0001F9FF]")
SUFFIX_JUNK_PATTERNS = [
    r"reverb_db[^\s]*",
    r"local_?message_?pe",
    r"direct\.db\s+messages(?:_content)?",
    r"fts\.db\s+messages(?:_content)?",
    r"crypto_db[^\s]*",
    r"omnistore_[^\s]*",
    r"enigma\.db",
    r"\(\(\s*0\s*B.*$",
]
SINHALA_RANGE = r"\u0D80-\u0DFF"
PRINTABLE_KEEP = re.compile(
    r"[^\t\n\r A-Za-z"
    r"\u2000-\u206F"
    r"\u0020-\u007E"
    r"\u0D80-\u0DFF"
    r"]"
)


def strip_emojis(s: str) -> str:
    return _EMOJI_RE.sub("", str(s or ""))

```

```python
def ensure_static_ttf(path: str) -> str:
    if not os.path.exists(path):
        raise FileNotFoundError(f"Unicode TTF font not found at: {path}")
    if "VariableFont" in os.path.basename(path):
        raise ValueError("Variable font detected. Use a static TTF.")
    if TTFont is not None:
        try:
            tt = TTFont(path)
            if "fvar" in tt:
                raise ValueError("Variable font detected (has 'fvar' table). Use a static TTF.")
        except Exception as e:
            raise ValueError(f"Unsupported/invalid TTF: {e}")
    return path


def _bytes_literal_to_text(s: str) -> str:
    if s.startswith(("b'", 'b"')) and s.endswith(("'", '"')):
        try:
            obj = eval(s)
            if isinstance(obj, (bytes, bytearray)):
                return obj.decode("utf-8", "ignore")
        except Exception:
            return s[2:].strip("\"'")
    return s


def _strip_prefix_noise(s: str) -> str:
    m = re.search(rf"[{SINHALA_RANGE}]+|[A-Za-z]{{2,}}", s)
    return s[m.start():] if m else s


def _strip_suffix_noise(s: str) -> str:
    for pat in SUFFIX_JUNK_PATTERNS:
        m = re.search(pat, s, flags=re.I)
        if m:
            s = s[:m.start()].rstrip()
    return s


def sanitize_text(x) -> str:
    s = strip_emojis(str(x or ""))
    s = _CTRL_RE.sub(" ", s)
    s = PRINTABLE_KEEP.sub("", s)
    s = re.sub(r"[ \t]+", " ", s).strip()
    return s
```

```python
def clean_text(x) -> str:
    s = str(x or "")
    s = _bytes_literal_to_text(s)
    s = strip_emojis(s)
    s = _CTRL_RE.sub(" ", s)
    s = _strip_prefix_noise(s)
    s = PRINTABLE_KEEP.sub("", s)
    s = _strip_suffix_noise(s)
    s = s.replace("_", " ")
    s = re.sub(r"[ \t]+", " ", s).strip(" \n\r\t")
    return s


def soft_wrap(s: str) -> str:
    s = str(s or "")
    zw = "\u200b"
    return s.replace("\\", "\\" + zw).replace("/", "/" + zw).replace("_", "_" + zw).replace("-", "-" + zw)
```

```python

def _nice_datetime(s: str) -> str:
    """Return 'DD/MM/YYYY' if parseable; else original string."""
    try:
        ts = pd.to_datetime(str(s), errors="coerce")
        if pd.notna(ts):
            return ts.strftime("%d/%m/%Y")
    except Exception:
        pass
    return str(s or "")

```

```python

def _reduce_code_blobs(s: str) -> str:
    """Hide long code-like blobs (e.g., hashes/base64) with an ellipsis."""
    return re.sub(r"[A-Za-z0-9+/=_-]{40,}", "…", str(s or ""))

def _parse_any_epoch(v):
    """Return pandas.Timestamp or NaT from numbers like epoch ms/sec."""
    try:
        x = pd.to_numeric(v, errors="coerce")
        if pd.isna(x):
            return pd.NaT
        # ms vs sec heuristics
        if x >= 1e12:      # clearly milliseconds
            return pd.to_datetime(x, unit="ms")
        if 1e9 <= x < 1e12:  # seconds (1970+)
            return pd.to_datetime(x, unit="s")
    except Exception:
        pass
    return pd.NaT

def _format_ts(ts):
    if ts is None or pd.isna(ts):
        return "-"
    return ts.strftime("%d/%m/%Y")

def extract_msg_datetime(row: pd.Series) -> str:
    """
    Try many common fields to find a timestamp, convert/format it nicely.
    Returns 'HH:MM, DD/MM/YYYY' or '-'.
    """
    # Most likely fields first
    candidates = [
        "created_at", "date_time", "datetime",
        "timestamp_ms", "message_timestamp", "msg_timestamp",
        "timestamp", "sent_at", "time_sent", "delivered_at",
        "received_at", "server_time", "client_time",
        "date_sent", "time", "date"
    ]

    # 1) Direct single-field parse (epoch or string)
    for c in candidates:
        if c in row:
            v = row.get(c)
            if str(v).strip() in ("", "-", "0", "None", "nan", "NaN"):
                continue
            # epoch numbers?
            ts = _parse_any_epoch(v)
            if pd.notna(ts):
                return _format_ts(ts)
            # free-form datetime string
            ts = pd.to_datetime(str(v), errors="coerce")
            if pd.notna(ts):
                return _format_ts(ts)
```

```python
    # 2) Combine separate date + time columns if present
    if ("date" in row) and ("time" in row):
        combo = f"{row.get('date','')} {row.get('time','')}"
        ts = pd.to_datetime(combo, errors="coerce")
        if pd.notna(ts):
            return _format_ts(ts)

    return "-"

# ---------- Auto-detect helpers (usernames & device) ----------

def _most_common_nonempty(series) -> str:
    try:
        s = pd.Series(series).astype(str).str.strip()
        s = s[(s.notna()) & (s.ne(""))]
        if s.empty:
            return ""
        return s.value_counts().idxmax()
    except Exception:
        return ""
```

```python

def infer_username(df) -> str:
    if df is None or df.empty:
        return ""
    for col in ["username", "user_name", "sender_username", "profile_username", "handle", "owner_username",
                "author", "author_name", "sender", "sender_name", "from", "from_name", "profile_name",
                "account_name", "screen_name", "name", "full_name"]:
        if col in df.columns:
            u = _most_common_nonempty(df[col])
            if u:
                u = str(u).strip().strip('"').strip("'")
                if u.startswith("@"):
                    u = u[1:]
                if "/" in u:
                    u = u.rstrip("/").split("/")[-1]
                return u
    if "source_database" in df.columns:
        pat = re.compile(r"[\\/]+([A-Za-z0-9._-]{3,})\.(?:db|sqlite3?)$", re.I)
        for v in df["source_database"].dropna():
            m = pat.search(str(v))
            if m:
                return m.group(1)
    return ""
```

```python

def infer_device_from_dfs(dfs) -> str:
    candidates = ["device_model", "device", "model", "phone_model", "product_model",
                  "ro_product_model", "ro_product_name", "manufacturer", "device_name"]
    for df in dfs:
        if df is None or df.empty:
            continue
        for col in candidates:
            if col in df.columns:
                val = _most_common_nonempty(df[col])
                if val:
                    return val
    for p in [
        os.path.join(os.getcwd(), "artifacts", "device_info.json"),
        os.path.join(os.path.dirname(os.path.abspath(__file__)), "artifacts", "device_info.json"),
        "device_info.json", "device_info.txt",
    ]:
        if os.path.isfile(p):
            try:
                if p.lower().endswith(".json"):
                    with open(p, "r", encoding="utf-8") as f:
                        data = json.load(f)
                    for k in ("model", "device_model", "device", "name"):
                        if data.get(k):
                            return str(data[k])
                else:
                    with open(p, "r", encoding="utf-8") as f:
                        for line in f:
                            line = line.strip()
                            if line:
                                return line
            except Exception:
                pass
    return ""

```

```python
def load_df(csv_path):
    if not csv_path or not os.path.exists(csv_path):
        return None
    df = pd.read_csv(csv_path, encoding="utf-8", low_memory=False)
    for col in ("text", "source_database", "source_table", "created_at", "date_time"):
        if col in df.columns:
            df[col] = df[col].fillna("").map(str)
    if "predicted_label" in df.columns:
        df["predicted_label"] = pd.to_numeric(df["predicted_label"], errors="coerce").fillna(0).astype(int)
    if "confidence" in df.columns:
        df["confidence"] = pd.to_numeric(df["confidence"], errors="coerce").fillna(0.0)
    if "text" in df.columns:
        df["text"] = df["text"].map(clean_text)
    return df
```

```python
def compute_stats(df):
    if df is None or df.empty:
        return dict(total=0, hate=0, safe=0, hate_pct=0.0, safe_pct=0.0)
    total = len(df)
    hate = int((df.get("predicted_label", 0) == 1).sum())
    safe = total - hate
    hate_pct = round((hate / total * 100.0), 1) if total else 0.0
    return dict(total=total, hate=hate, safe=safe, hate_pct=hate_pct, safe_pct=round(100.0 - hate_pct, 1))


def flagged_rows(df, top_n=20):
    if df is None or df.empty:
        return pd.DataFrame(columns=["confidence", "text", "source_database", "source_table", "created_at", "date_time"])
    flg = df[df.get("predicted_label", 0) == 1].copy()
    if "confidence" in flg.columns:
        flg = flg.sort_values("confidence", ascending=False)
    return flg.head(top_n)


def all_rows_sample(df, top_n=25):
    if df is None or df.empty:
        return pd.DataFrame(columns=["predicted_label", "text", "source_database", "source_table", "created_at", "date_time"])
    out = df.copy()
    if "text" in out.columns:
        out["__len"] = out["text"].str.len()
        out = out.sort_values("__len", ascending=False).drop(columns="__len")
    return out.head(top_n)


# ---------- PDF ----------

class ReportPDF(FPDF):
    def __init__(self, font_path):
        super().__init__(orientation="P", unit="mm", format="A4")
        self.set_auto_page_break(True, margin=15)
        self.add_font("UNI", "", font_path)
        self.add_font("UNI", "B", font_path)
        self.set_font("UNI", "", 12)

    def _eff_width(self) -> float:
        return self.w - self.l_margin - self.r_margin

    def full_width_multicell(self, h: float, txt: str):
        self.set_x(self.l_margin)
        safe = sanitize_text(str(txt or "")).replace("\r", " ").replace("\n", " ")
        self.multi_cell(self._eff_width(), h, safe if safe else "-")
```

```python
    # ---------- wrapping helpers ----------

    def _wrap_for_cell(self, w, txt, max_lines=2, line_h=6):
        safe = sanitize_text(txt if txt is not None else "-")
        lines = self.multi_cell(w, line_h, safe, dry_run=True, output="LINES")
        if max_lines and len(lines) > max_lines:
            lines = lines[:max_lines]
            eff = w - 2 * self.c_margin
            last = lines[-1]
            while last and self.get_string_width(last + "…") > eff:
                last = last[:-1]
            lines[-1] = (last + "…") if last else "…"
        return [l if l else "-" for l in lines]

    def table_row_wrapped(self, widths, cells, aligns=None, line_h=6, border=1, wrap_cols=None, default_max_lines=2):
        if aligns is None:
            aligns = ["L"] * len(widths)
        wrap_cols = wrap_cols or {}
        col_lines = []
        for i, (w, val) in enumerate(zip(widths, cells)):
            if i in wrap_cols:
                max_lines = wrap_cols.get(i, default_max_lines)
                lines = self._wrap_for_cell(w, val, max_lines=max_lines, line_h=line_h)
            else:
                lines = [sanitize_text(val if val is not None else "-")]
            col_lines.append(lines)
        n_lines = max(len(ls) for ls in col_lines)
        row_h = n_lines * line_h
        x0, y0 = self.get_x(), self.get_y()
        for w, lines, align in zip(widths, col_lines, aligns):
            x, y = self.get_x(), self.get_y()
            if border:
                self.rect(x, y, w, row_h)
            for j, line in enumerate(lines):
                self.set_xy(x + self.c_margin, y + j * line_h)
                self.cell(w - 2 * self.c_margin, line_h, line, border=0, align=align)
            self.set_xy(x + w, y)
        self.set_xy(x0, y0 + row_h)

    def table_row(self, widths, cells, border=1, aligns=None, line_h=7):
        aligns = aligns or ["L"] * len(widths)
        self.set_x(self.l_margin)
        for w, val, align in zip(widths, cells, aligns):
            self.cell(w, line_h, sanitize_text(val), border=border, align=align)
        self.ln(line_h)
```

```python
    # ---------- small utilities ----------

    def hr(self, pad=2):
        y = self.get_y() + pad
        self.set_draw_color(180, 180, 180)
        self.line(self.l_margin, y, self.w - self.r_margin, y)
        self.set_draw_color(0, 0, 0)
        self.set_y(y + pad)

    def kv_inline(self, label, value, lw=None, ln=15, bullet=False, bullet_w=4):
        """
        Key : value on a single line (left label, right value).
        - lw: optional fixed label width (mm). If None, width is computed to fit the label.
        - ln: line height (mm).
        """
        label_txt = f"{label} :"
        value_txt = sanitize_text(value) if value else "-"

        if lw is None:
            lw = self.get_string_width(label_txt) + 2 * self.c_margin + 3

        if bullet:
            self.cell(bullet_w, ln, "•", align="C")

        self.set_font("UNI", "", 11)
        self.cell(lw, ln, label_txt, align="L")
        self.cell(self._eff_width() - lw, ln, value_txt, align="L", new_x="LMARGIN", new_y="NEXT")

    # ---------- cover (page 1) ----------

    def cover_page(self, today_str: str, logo_path: str | None = None):
        self.add_page()
        self.set_y(20)
        self.set_text_color(0, 0, 0)
        self.set_font("UNI", "B", 50)

        def center_line(text: str, line_h: float):
            self.cell(0, line_h, text.upper(), new_x="LMARGIN", new_y="NEXT", align="C")

        center_line("CYBERBULLYING", 20)
        center_line("DETECTION REPORT", 20)

        self.ln(20)
        self.set_font("UNI", "B", 16)
        w = min(160, self._eff_width())
        x = (self.w - w) / 2
        self.set_xy(x, self.get_y() + 6)
        self.multi_cell(w, 6, "Digital Forensic Analysis on Social Media Platforms", align="C")
        self.set_x(x)
        self.multi_cell(w, 6, "of Instagram, Facebook, Messenger", align="C")
```

```python
        if logo_path and os.path.exists(logo_path):
            self.ln(25)
            logo_w = 90
            logo_x = (self.w - logo_w) / 2
            self.image(logo_path, x=logo_x, y=self.get_y(), w=logo_w)
            self.set_y(self.get_y() + logo_w + 4)

        self.set_font("UNI", "B", 15)
        bottom_offset = 30
        self.set_y(self.h - bottom_offset)
        self.cell(0, 6, today_str, align="C")

    # ---------- overview (page 2) ----------

    def summary_table(self, totals):
        widths = [50, 33, 55, 55]
        self.set_font("UNI", "B", 12)
        self.table_row(widths, ["Social Media Platform", "Total of Texts",
                                "Amount of Hate Speech", "Amount of Safe Speech"])
        self.set_font("UNI", "", 12)
        for name in ["Facebook", "Instagram", "Messenger"]:
            s = totals.get(name, dict(total=0, hate=0, safe=0, hate_pct=0.0, safe_pct=0.0))
            self.table_row(widths, [name, s["total"], f"{s['hate']} = {s['hate_pct']}%", f"{s['safe']} = {s['safe_pct']}%"])
```

```
    def grouped_bars_vertical(self, totals, tmpdir, hate_color="#06b6d4", safe_color="#5be7e7"):
        path = os.path.join(tmpdir, "summary_bar.png")

        labels = ["Facebook", "Instagram", "Messenger"]
        hate = [totals.get(k, {}).get("hate", 0) for k in labels]
        safe = [totals.get(k, {}).get("safe", 0) for k in labels]

        x = np.arange(len(labels))
        width = 0.35

        fig, ax = plt.subplots(figsize=(7, 4), dpi=300)
        ax.bar(x - width/2, hate, width, label="Hate Speech", color=hate_color)
        ax.bar(x + width/2, safe, width, label="Safe Speech", color=safe_color)

        ax.set_xticks(x, labels)
        ax.set_ylabel("Count")
        ax.set_ylim(bottom=0)
        ax.set_title("Report Summary")
        ax.legend(loc="upper center", ncol=2, frameon=False)

        fig.tight_layout()
        fig.savefig(path, bbox_inches="tight", dpi=300)
        plt.close(fig)

        self.ln(8)
        self.image(path, w=180)
        self.ln(4)

    def overview_page(self, args, totals, tmpdir):
        self.add_page()
        self.set_font("UNI", "B", 18)
        self.cell(0, 9, "Cyberbullying - Hate Speech Detection Report",
                  align="C", new_x="LMARGIN", new_y="NEXT")
        self.ln(8)

        self.set_font("UNI", "", 12)
        # labels = ["Case Number", "Investigator", "Device Name", "Owner of the Device"]  # (old, with device)
        labels = ["Case Number", "Investigator", "Owner of the Device"]  # device removed from display
        label_w = max(self.get_string_width(f"{lbl} :") for lbl in labels) + 2*self.c_margin + 3
        self.kv_inline("Case Number", args.case_id or "-", lw=label_w, ln=15, bullet=False)
        self.kv_inline("Investigator", args.investigator or "-", lw=label_w, ln=15, bullet=False)
        # self.kv_inline("Device Name", args.device_id or "-", lw=label_w, ln=15, bullet=True)  # HIDDEN
        self.kv_inline("Owner of the Device", args.owner or "-", lw=label_w, ln=15, bullet=False)

        self.ln(8)
        self.hr(pad=4)
        self.ln(8)
```

```
    def overview_page(self, args, totals, tmpdir):
        self.add_page()
        self.set_font("UNI", "B", 18)
        self.cell(0, 9, "Cyberbullying - Hate Speech Detection Report",
                  align="C", new_x="LMARGIN", new_y="NEXT")
        self.ln(8)

        self.set_font("UNI", "", 12)
        # labels = ["Case Number", "Investigator", "Device Name", "Owner of the Device"]  # (old, with device)
        labels = ["Case Number", "Investigator", "Owner of the Device"]  # device removed from display
        label_w = max(self.get_string_width(f"{lbl} :") for lbl in labels) + 2*self.c_margin + 3
        self.kv_inline("Case Number", args.case_id or "-", lw=label_w, ln=15, bullet=False)
        self.kv_inline("Investigator", args.investigator or "-", lw=label_w, ln=15, bullet=False)
        # self.kv_inline("Device Name", args.device_id or "-", lw=label_w, ln=15, bullet=True)  # HIDDEN
        self.kv_inline("Owner of the Device", args.owner or "-", lw=label_w, ln=15, bullet=False)

        self.ln(8)
        self.hr(pad=4)
        self.ln(8)

        self.set_font("UNI", "B", 16)
        self.cell(0, 7, "Report Summary", align="C", new_x="LMARGIN", new_y="NEXT")
        self.ln(10)

        self.summary_table(totals)
        self.grouped_bars_vertical(totals, tmpdir)
```

```python
    #-----------overall device summary with table + pie (page 3)------------------

    def overall_device_summary_page(self, totals, tmpdir):
        """
        Page 3: Overall Hate Speech Detected from the Device
        - Table with per-platform % (hate / safe)
        - 'Total' row with overall % across all platforms
        - Pie chart with overall hate vs safe
        """
        # compute counts and overall percentages
        labels = ["Facebook", "Instagram", "Messenger"]
        per = {k: totals.get(k, {"total": 0, "hate": 0, "safe": 0, "hate_pct": 0.0, "safe_pct": 0.0}) for k in labels}

        total_counts = sum(per[k]["total"] for k in labels)
        total_hate   = sum(per[k]["hate"]  for k in labels)
        total_safe   = sum(per[k]["safe"]  for k in labels)

        overall_hate_pct = round((total_hate / total_counts * 100.0), 1) if total_counts else 0.0
        overall_safe_pct = round(100.0 - overall_hate_pct, 1) if total_counts else 0.0

        # draw page
        self.add_page()
        self.set_font("UNI", "B", 18)
        self.cell(0, 9, "Overall Hate Speech Detected from the Device",
                  align="C", new_x="LMARGIN", new_y="NEXT")
        self.ln(20)

        # table header
        widths = [80, 55, 55]  # Platform | Hate % | Safe %
        line_h = 15
        self.set_font("UNI", "B", 12)
        self.table_row(widths, ["Social Media Platform", "Hate Speech", "Safe Speech"], line_h=line_h)

        # table rows
        self.set_font("UNI", "", 12)
        for name in labels:
            hp = per[name]["hate_pct"]
            sp = per[name]["safe_pct"]
            self.table_row(widths, [name, f"{hp}%", f"{sp}%"], line_h=line_h)

        # total row (overall)
        self.set_font("UNI", "B", 12)
        self.table_row(widths, ["Total", f"{overall_hate_pct}%", f"{overall_safe_pct}%"], line_h=line_h)
        self.set_font("UNI", "", 12)
```

```python
        # overall pie chart
        fig, ax = plt.subplots(figsize=(5.2, 4.4), dpi=300)
        wedges, texts, autotexts = ax.pie(
            [max(total_hate, 0), max(total_safe, 0)],
            startangle=90,
            colors=["#06b6d4", "#5be7e7"],
            labels=None,
            autopct=lambda p: f"{p:.1f}%",
            pctdistance=0.75,
            textprops={"fontsize": 9}
        )
        centre = plt.Circle((0, 0), 0.58, fc="white")
        ax.add_artist(centre)
        ax.axis("equal")
        ax.legend(["Hate Speech", "Safe Speech"], loc="upper center",
                  bbox_to_anchor=(0.5, -0.05), ncol=2, frameon=False)

        pie_path = os.path.join(tmpdir, "overall_device_pie.png")
        fig.tight_layout()
        fig.savefig(pie_path, bbox_inches="tight", dpi=300)
        plt.close(fig)

        # place the image centered
        self.ln(15)
        img_w = 180
        img_w = min(img_w, self._eff_width())
        x = self.l_margin + (self._eff_width() - img_w) / 2
        self.image(pie_path, x=x, w=img_w)
        self.ln(10)


    # ---------- platform SUMMARY page (4,8,12) ----------

    def platform_summary_table(self, platform_name, stats):
        widths = [50, 33, 55, 55]
        self.set_font("UNI", "B", 12)
        self.table_row(
            widths,
            ["Social Media Platform", "Total of Texts", "Amount of Hate Speech ", "Amount of Safe Speech"]
        )
        self.set_font("UNI", "", 12)
        hate_txt = f"{stats['hate']} = {stats['hate_pct']}%"
        safe_txt = f"{stats['safe']} = {stats['safe_pct']}%"
        self.table_row(widths, [platform_name, stats["total"], hate_txt, safe_txt])
```

```
    # ---------- platform SUMMARY page (4,8,12) ----------

    def platform_summary_table(self, platform_name, stats):
        widths = [50, 33, 55, 55]
        self.set_font("UNI", "B", 12)
        self.table_row(
            widths,
            ["Social Media Platform", "Total of Texts", "Amount of Hate Speech ", "Amount of Safe Speech"]
        )
        self.set_font("UNI", "", 12)
        hate_txt = f"{stats['hate']} = {stats['hate_pct']}%"
        safe_txt = f"{stats['safe']} = {stats['safe_pct']}%"
        self.table_row(widths, [platform_name, stats["total"], hate_txt, safe_txt])

    def platform_summary_page(self, platform_name, username, when_str, stats, tmpdir):
        self.add_page()
        self.set_font("UNI", "B", 18)
        self.cell(0, 9, "Cyberbullying - Hate Speech Detection", align="C",
                  new_x="LMARGIN", new_y="NEXT")
        self.ln(6)

        self.set_font("UNI", "", 12)
        kv_labels = ["Social Media Platform", "Social Media Profile", "Report Generated Date"]
        label_w = max(self.get_string_width(f"{t} :") for t in kv_labels) + 2*self.c_margin + 3
        self.kv_inline("Social Media Platform", platform_name, lw=label_w, ln=15)
        self.kv_inline("Social Media Profile", username or "-", lw=label_w, ln=15)
        self.kv_inline("Report Generated Date", when_str, lw=label_w, ln=15)

        self.ln(6)
        self.hr(pad=4)
        self.ln(6)

        self.set_font("UNI", "B", 16)
        self.cell(0, 7, f"{platform_name} Summary", align="C", new_x="LMARGIN", new_y="NEXT")
        self.ln(6)
        self.platform_summary_table(platform_name, stats)

        # Pie chart (centered)
        hate = stats.get("hate", 0)
        safe = stats.get("safe", 0)
        data = [hate, safe]
        colors = ["#06b6d4", "#5be7e7"]
```

```
        fig, ax = plt.subplots(figsize=(4.8, 4.0), dpi=300)
        wedges, texts, autotexts = ax.pie(
            data, startangle=90, colors=colors, labels=None,
            autopct=lambda p: f"{p:.1f}%", pctdistance=0.75, textprops={"fontsize": 9}
        )
        centre = plt.Circle((0, 0), 0.58, fc="white")
        ax.add_artist(centre)
        ax.axis("equal")
        ax.legend(["Hate Speech", "Safe Speech"], loc="upper center",
                  bbox_to_anchor=(0.5, -0.05), ncol=2, frameon=False)

        pie_path = os.path.join(tmpdir, f"{platform_name.lower()}_pie.png")
        fig.tight_layout()
        fig.savefig(pie_path, bbox_inches="tight", dpi=300)
        plt.close(fig)

        self.ln(8)
        img_w = 150
        img_w = min(img_w, self._eff_width())
        x = self.l_margin + (self._eff_width() - img_w) / 2
        self.image(pie_path, x=x, w=img_w)
        self.ln(4)
```

```
Click to add a breakpoint
    # ---------- All Texts Extracted pages (5,9,13) ----------

    def all_texts_pages(self, platform_name: str, df: pd.DataFrame):
        """
        Render ALL extracted rows for the given platform.
        Uses fixed layout like your screenshot, wraps key columns to 2 lines,
        and automatically paginates.
        """
        widths = [6, 26, 60, 25, 40, 40]  # #, Hate/Safe, Text, DateTime, DB, Table
        line_h = 8

        def add_page_with_header():
            self.add_page()
            # Title
            self.set_font("UNI", "B", 13)
            self.cell(0, 7, f"All Texts Extracted {platform_name}", align="C",
                      new_x="LMARGIN", new_y="NEXT")
            self.ln(4)
            # Header row
            self.set_font("UNI", "B", 10)
            self.table_row(
                widths,
                ["#", "Hate or Safe", "Text", "Date", "Source DB", "Source Table"],
                line_h=line_h
            )
            self.set_font("UNI", "", 10)

        add_page_with_header()

        # No rows case
        if df is None or df.empty:
            self.full_width_multicell(6, "No rows.")
            return

        row_no = 1
        for _, r in df.reset_index(drop=True).iterrows():
            # Each row will be at most 2 lines tall for wrapped cols
            required_h = 5 * line_h
            # If we're too close to the bottom, start a new page with header
            if self.get_y() + required_h > (self.h - self.b_margin):
                add_page_with_header()

            label = "Hate" if int(r.get("predicted_label", 0)) == 1 else "Safe"
            txt = clean_text(r.get("text", ""))  # <- show the full (cleaned) message
            dtm = extract_msg_datetime(r)
            sdb = soft_wrap(r.get("source_database", ""))
            stb = soft_wrap(r.get("source_table", ""))
```

```
            self.table_row_wrapped(
                widths,
                [row_no, label, txt, dtm, sdb, stb],
                aligns=["C", "L", "L", "L", "L", "L"],
                line_h=line_h,
                wrap_cols={2: 3, 3: 3, 4: 3, 5: 3},  # Text / Date / DB / Table
                default_max_lines=3
            )
            row_no += 1

    # ---------- Flagged (Hate) Texts pages (6,10,14) ----------

    def flagged_texts_pages(self, platform_name: str, df: pd.DataFrame, top_n: int = 25):
        """
        Render ONLY model-flagged hate texts (predicted_label == 1).
        Texts are cleaned for human readability and long code-ish blobs are
        trimmed with an ellipsis.
        """
        # Filter to hate only
        if df is None or df.empty:
            flg = pd.DataFrame(columns=df.columns if df is not None else [])
        else:
            flg = df[df.get("predicted_label", 0) == 1].copy()
            if "confidence" in flg.columns:
                flg = flg.sort_values("confidence", ascending=False)
        if top_n and len(flg) > top_n:
            flg = flg.head(top_n)
```

```
        widths = [7, 30, 60, 28, 38, 34]   # #, Confidence, Text, DateTime, DB, Table
        line_h = 8

        def header_page():
            self.add_page()
            self.set_font("UNI", "B", 13)
            self.cell(0, 7, f"{platform_name} Flagged Messages", align="C",
                      new_x="LMARGIN", new_y="NEXT")
            self.ln(4)
            self.set_font("UNI", "B", 10)
            self.table_row(
                widths,
                ["#", "Confidence Rate", "Text", "Date", "Source DB", "Source Table"],
                line_h=line_h
            )
            self.set_font("UNI", "", 10)

        header_page()

        if flg.empty:
            self.full_width_multicell(6, "No flagged messages.")
            return

        row_no = 1
        for _, r in flg.reset_index(drop=True).iterrows():
            # keep page nicely paginated
            required_h = 5 * line_h
            if self.get_y() + required_h > (self.h - self.b_margin):
                header_page()


            # confidence as percentage (if present)
            conf = "-"
            try:
                conf = f"{float(r.get('confidence', 0))*100:.1f}%"
            except Exception:
                pass

            # human-readable, code-lite text
            txt = clean_text(r.get("text", ""))
            txt = _reduce_code_blobs(txt)

            dtm = extract_msg_datetime(r)
            sdb = soft_wrap(r.get("source_database", ""))
            stb = soft_wrap(r.get("source_table", ""))
```

```
            self.table_row_wrapped(
                widths,
                [row_no, conf, txt, dtm, sdb, stb],
                aligns=["C", "C", "L", "L", "L", "L"],
                line_h=line_h,
                wrap_cols={2: 3, 3: 3, 4: 3, 5: 3},
                default_max_lines=3
            )
            row_no += 1

    # ---------- appendix: database SHA-256 hashes pages (7,11,15) ----------

    def appendix_hashes_page(self, platform_name: str, hash_df: pd.DataFrame | None):
        """
        Appendix : Database SHA-256 Hashes
        - Two columns (Databases | Hashes)
        - Auto-wrap long values
        - Auto-add new pages when space runs out, repeating the header/table header
        """
        widths = [95, 95]          # two equal columns
        line_h = 7                  # line height used everywhere
        wrap_cfg = {0: 2, 1: 2}     # wrap both columns to max 2 lines

        def _page_header():
            """Title, subtitle, blurb, then table header."""
            self.add_page()
            self.set_font("UNI", "B", 16)
            self.cell(0, 8, "Appendix : Database SHA-256 Hashes", align="C",
                      new_x="LMARGIN", new_y="NEXT")
            self.ln(6)
```

```
            self.set_font("UNI", "", 12)
            self.cell(0, 6, f"Platform: {platform_name}", new_x="LMARGIN", new_y="NEXT")
            self.ln(6)

            # table header
            self.set_font("UNI", "B", 11)
            self.table_row(widths, ["Databases", "Hashes"], line_h=line_h)
            self.set_font("UNI", "", 10)

        def _ensure_space_for(row_h: float):
            """Start a new page with headers if the next row won't fit."""
            if self.get_y() + row_h > (self.h - self.b_margin):
                _page_header()

        # start first page
        _page_header()

        # Rows
        if hash_df is None or hash_df.empty:
            self.full_width_multicell(6, "No hashes CSV provided.")
        else:
            # pick sensible columns
            db_col = next((c for c in ["database_file", "file", "database", "db"] if c in hash_df.columns),
                          hash_df.columns[0])
            hash_col = next((c for c in ["sha256_hash", "sha256", "hash", "sha_256"] if c in hash_df.columns),
                            (hash_df.columns[1] if len(hash_df.columns) > 1 else hash_df.columns[0]))

            for _, r in hash_df.iterrows():
                db_val = soft_wrap(r.get(db_col, ""))
                sha_val = soft_wrap(r.get(hash_col, ""))

                # Pre-compute wrapped lines to know the row height *before* drawing
                l_db  = self._wrap_for_cell(widths[0], db_val,  max_lines=wrap_cfg[0], line_h=line_h)
                l_sha = self._wrap_for_cell(widths[1], sha_val, max_lines=wrap_cfg[1], line_h=line_h)
                row_h = max(len(l_db), len(l_sha)) * line_h

                _ensure_space_for(row_h)

                # Now actually render the row (wrapping applied inside)
                self.table_row_wrapped(
                    widths, [db_val, sha_val],
                    aligns=["L", "L"], line_h=line_h, border=1,
                    wrap_cols=wrap_cfg, default_max_lines=2
                )
```

```
        # Bottom note
        self.ln(6)
        self.set_text_color(110, 110, 110)
        self.set_font("UNI", "", 10)
        self.full_width_multicell(
            5,
            "Note: Below is an original source-file list of extracted data displayed in the"
            "report for quick reference. The full CSV remains the authoritative record. "
            "Furthermore, this report summarizes automated analysis results. "
            "Always refer to the original evidence for deep, sensitive, or critical investigations and decisions."
        )
        self.set_text_color(0, 0, 0)
        self.set_font("UNI", "", 12)

# ---------- Build ----------
def parse_args():
    p = argparse.ArgumentParser(description="Generate structured PDF report.")
    p.add_argument("--config", help="Optional path to JSON config from Hawkeye wrapper", default="")

    p.add_argument("--case-id", default="")
    p.add_argument("--investigator", default="")
    # p.add_argument("--device-id", default="")  # <<< DISABLED: we no longer accept/show device id
    p.add_argument("--owner", default="")

    p.add_argument("--fb-username", default="")
    p.add_argument("--ig-username", default="")
    p.add_argument("--ms-username", default="")

    p.add_argument("--fb-pred", default=DEFAULT_FB_PRED)
    p.add_argument("--ig-pred", default=DEFAULT_IG_PRED)
    p.add_argument("--ms-pred", default=DEFAULT_MS_PRED)
```

```
    p.add_argument("--fb-pred", default=DEFAULT_FB_PRED)
    p.add_argument("--ig-pred", default=DEFAULT_IG_PRED)
    p.add_argument("--ms-pred", default=DEFAULT_MS_PRED)

    p.add_argument("--fb-hash", default=DEFAULT_FB_HASH)
    p.add_argument("--ig-hash", default=DEFAULT_IG_HASH)
    p.add_argument("--ms-hash", default=DEFAULT_MS_HASH)

    p.add_argument("--out-dir", default=DEFAULT_OUT_DIR)
    p.add_argument("--font", default=DEFAULT_FONT)
    p.add_argument("--logo", default="", help="Optional path to a PNG/JPG logo for the cover page")
    p.add_argument("--top-n", type=int, default=25)
    return p.parse_args()


def merge_config(args):
    """If --config is supplied, merge it over CLI defaults (CLI still wins)."""
    if not args.config:
        return args
    try:
        with open(args.config, "r", encoding="utf-8") as f:
            cfg = json.load(f)
        for k, v in cfg.items():
            if getattr(args, k, "") in ("", None):
                setattr(args, k, v)
    except Exception:
        pass
    return args


def prompt_if_missing(args):
    """Interactive prompts for the fields you wanted if missing."""
    try:
        if not args.case_id:
            args.case_id = input(Fore.YELLOW + "\n 📁 Enter the Case ID : ").strip()
        if not args.investigator:
            args.investigator = input(Fore.YELLOW + "\n 🕵️ Enter the name of the Investigator : ").strip()
        if not args.owner:
            args.owner = input(Fore.YELLOW + "\n 👤 Enter the Owner's name of the mobile device : ").strip()
    except EOFError:
        pass
    return args


def build_report(args):
    os.makedirs(args.out_dir, exist_ok=True)
    args.font = ensure_static_ttf(args.font)
```

```
    # load data
    fb = load_df(args.fb_pred)
    ig = load_df(args.ig_pred)
    ms = load_df(args.ms_pred)

    # NEW: load hash CSVs for appendix pages
    fb_hash = load_df(args.fb_hash)
    ig_hash = load_df(args.ig_hash)
    ms_hash = load_df(args.ms_hash)

    # auto-fill usernames/device if blank
    if not args.fb_username:
        args.fb_username = infer_username(fb) or ""
    if not args.ig_username:
        args.ig_username = infer_username(ig) or ""
    if not args.ms_username:
        args.ms_username = infer_username(ms) or ""
    # if not args.device_id:
    #     args.device_id = infer_device_from_dfs([fb, ig, ms]) or "UNKNOWN-DEVICE"  # <<< DISABLED

    # prompt for human fields if needed
    args = prompt_if_missing(args)

    fb_stats = compute_stats(fb)
    ig_stats = compute_stats(ig)
    ms_stats = compute_stats(ms)
    totals = {"Facebook": fb_stats, "Instagram": ig_stats, "Messenger": ms_stats}

    today = dt.datetime.now()
    today_str = today.strftime("%d of %B, %Y")
    when = today.strftime("%d/%m/%Y")
    ts_name = today.strftime("%Y%m%d_%H%M%S")
    out_path = os.path.join(args.out_dir, f"Structured_Report_{ts_name}.pdf")
    logo_path = resolve_logo_path(args.logo)
```

```
    tmp = tempfile.mkdtemp(prefix="pdf_tmp_")
    try:
        pdf = ReportPDF(args.font)

        # Page 1 (cover)
        pdf.cover_page(today_str, logo_path=logo_path)

        # Page 2 (overview)
        pdf.overview_page(args, totals, tmp)

        # Page 3 (overall device summary with table + pie)
        pdf.overall_device_summary_page(totals, tmp)

        # Pages 4,8,12 (summary) + 5,9,13 (all texts) + 6,10,14 (Flaged texts)
        # Facebook
        pdf.platform_summary_page("Facebook",  args.fb_username, when, fb_stats, tmp)
        #pdf.all_texts_pages("Facebook", fb)
        pdf.flagged_texts_pages("Facebook", fb, top_n=args.top_n)
        pdf.appendix_hashes_page("Facebook", fb_hash)

        # Instagram
        pdf.platform_summary_page("Instagram", args.ig_username, when, ig_stats, tmp)
        #pdf.all_texts_pages("Instagram", ig)
        pdf.flagged_texts_pages("Instagram", ig, top_n=args.top_n)
        pdf.appendix_hashes_page("Instagram", ig_hash)

        # Messenger
        pdf.platform_summary_page("Messenger", args.ms_username, when, ms_stats, tmp)
        #pdf.all_texts_pages("Messenger", ms)
        pdf.flagged_texts_pages("Messenger", ms, top_n=args.top_n)
        pdf.appendix_hashes_page("Messenger", ms_hash)
```

```
        pdf.output(out_path)
        print(Fore.GREEN + "\n✅✅✅ Report Generation Completed Successfully")
        print(Fore.GREEN + "\n📍 Report Saved Location → ")
        print(Fore.YELLOW + f"\n          👉 {out_path}")
        print(Fore.BLUE + """\n
_._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._."""+ Style.RESET_ALL)
        print(Fore.MAGENTA +"\n🤚  Quiting.....")
        print(Fore.CYAN +"""\n
|=============================================================================================|
|8888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888|
|=============================================================================================|
|88|                                                                                       |88|
|88|    :::    :::     :::     :::       ::: :::    ::: :::::::::: :::   ::: ::::::::::   |88|
|88|    :+:    :+:   :+: :+:   :+:       :+: :+:   :+:  :+:        :+:   :+: :+:          |88|
|88|    +:+    +:+  +:+   +:+  +:+       +:+ +:+  +:+   +:+         +:+ +:+  +:+          |88|
|88|    +#++:++#++ +#++:++#++: +#+  +:+  +#+ +#++:++    +#++:++#     +#++:   +#++:++#     |88|
|88|    +#+    +#+ +#+     +#+ +#+ +#+#+ +#+ +#+  +#+   +#+           +#+    +#+          |88|
|88|    #+#    #+# #+#     #+#  #+#+# #+#+#  #+#   #+#  #+#           #+#    #+#          |88|
|88|    ###    ### ###     ###   ###   ###   ###    ### ##########    ###    ##########   |88|
|88|                                                                                       |88|
|=============================================================================================|
|8888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888|
|888                                                                                     888|
|888           ANDROID FORENSIC TOOL FOR HARSH SPEECH DETECTION ON SOCIAL MEDIA           888|
|888                                                                                     888|
|8888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888|
|=============================================================================================|
    """)
        print(Fore.MAGENTA +"""
:::::::::: :::    :::     :::     ::::    ::: :::    :::    :::   :::  ::::::::  :::    ::: :::
    :+:     :+:    :+:   :+: :+:   :+:+:   :+: :+:   :+:     :+:   :+: :+:    :+: :+:    :+: :+:
    +:+     +:+    +:+  +:+   +:+  :+:+:+  +:+ +:+  +:+       +:+ +:+  +:+    +:+ +:+    +:+ +:+
    +#+     +#++:++#++ +#++:++#++: +#+ +:+ +#+ +#++:++         +#++:   +#+    +:+ +#+    +:+ +:+
    +#+     +#+    +#+ +#+     +#+ +#+  +#+#+# +#+  +#+         +#+    +#+    +#+ +#+    +#+
    #+#     #+#    #+# #+#     #+# #+#   #+#+# #+#   #+#        #+#    #+#    #+# #+#    #+# #+#
    ###     ###    ### ###     ### ###    #### ###    ###       ###     ########   ########  ###
    """+ Style.RESET_ALL)
    finally:
        shutil.rmtree(tmp, ignore_errors=True)


def main():
    args = parse_args()
    args = merge_config(args)
    build_report(args)


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