

# **Multi-Purpose AI Password Analysis Tool**

---



**Osama Khalid – 27971**  
**Sardar M. Zeeshan khan – 27969**  
**Salman Ali – 27667**

**Supervised by:**  
**Dr. Muhammad Mansoor Alam**

**Faculty of Computing**  
**Riphah International University, Islamabad**  
**Spring 2024**

**A Dissertation Submitted To**

**Faculty of Computing,**

**Riphah International University, Islamabad**

**As a Partial Fulfillment of the Requirement for the Award of**

**the Degree of**

**Bachelor of Science in Cyber Security**

**Faculty of Computing**  
**Riphah International University, Islamabad**

**Date: 23/6/2024**

## Final Approval

This is to certify that we have read the report submitted by *Osama Khalid 27971, Sardar Muhammad Zeeshan Khan 27969, Salman Ali 27667*, for the partial fulfillment of the requirements for the degree of the Bachelor of Science in Cyber Security (BSCY). It is our judgment that this report is of sufficient standard to warrant its acceptance by Riphah International University, Islamabad for the degree of Bachelor of Science in Cyber Security (BSCY).

### Committee:

1

---

Dr. Muhammad Mansoor Alam  
(Supervisor)

2

---

Dr. Musharraf Ahmed  
(Head of Department/chairman)

## Declaration

We hereby declare that this document “**Multi-Purpose AI Password Analysis Tool**” neither as a whole nor as a part has been copied out from any source. It is further declared that we have done this project with the accompanying report entirely on the basis of our personal efforts, under the proficient guidance of our teachers, especially our supervisor **Dr. Mansoor Alam**. If any part of the system is proved to be copied out from any source or found to be reproduction of any project from anywhere else, we shall stand by the consequences.

s

---

**Osama Khalid**  
**27971**

---

**Sardar M. Zeeshan khan**  
**27969**

---

**Salman Ali**  
**27667**

## **Dedication**

This project is dedicated to our friends, mentors, and educators, whose unwavering support and encouragement have been a constant source of strength throughout this journey. Their belief in our capabilities has fueled our determination to succeed. Additionally, we dedicate this work to our colleagues and the academic community, whose guidance and knowledge have been instrumental in shaping our professional growth. Their commitment to excellence has inspired us to push the boundaries of our potential. Thank you for being our pillars of support and for always believing in us.

## Acknowledgement

First of all, we are obliged to Allah the Almighty the Merciful, the Beneficent and the source of all Knowledge, for granting us the courage and knowledge to complete this Project.

We would like to express our sincere gratitude to the following individuals whose guidance and support were instrumental in the successful completion of this project:

- **Project Supervisor:** We are particularly grateful to our project supervisor, **Dr. Mansoor Alam**, for their invaluable expertise, insightful feedback, and unwavering encouragement throughout the entire project development process. His dedication and support played a pivotal role in shaping the direction and success of our work.
- **Faculty Advisors:** We extend our sincere thanks to our faculty advisors, **Dr. Muhammad Mansoor Alam**, whose passion for AI inspired us to pursue this project, and **Dr. Jawaid Iqbal**, whose knowledge and guidance were invaluable throughout the research phase.
- **Technical Support Staff:** We appreciate the assistance provided by **Mr. Haseeb Ahmed**, **Mr. Ihtesham Ullah**, and **Mr. Tajammul Hussain**. Their technical expertise and willingness to help were crucial in overcoming technical challenges.
- **Project Convener:** We acknowledge the leadership of **Mr. Muhammad Ahmad Nawaz** as the project convener, whose guidance ensured the project's smooth execution within the academic framework.
- **Colleagues and Friends:** Finally, we would like to thank our colleagues, **Mr. Osama Raza** and **Mr. Awais Nawaz**, for their encouragement, collaboration, and support throughout the course of this project. Their positive spirit and willingness to assist contributed significantly to our progress.

---

**Osama Khalid**  
**27971**

---

**Sardar M. Zeeshan khan**  
**27969**

---

**Salman Ali**  
**27667**

## Abstract

In an increasingly digitized world, the security of personal and organizational data is paramount. This project presents a comprehensive solution in the form of a multi-purpose AI-driven password analysis tool. By harnessing the power of artificial intelligence, this tool offers dual functionality, serving both offensive and defensive purposes.

On the offensive front, it employs advanced algorithms to analyze and crack passwords, providing insights into their vulnerabilities and potential points of exploitation. This capability enables security professionals to understand the weaknesses inherent in various password configurations, thereby facilitating the development of more robust defense strategies.

Simultaneously, the tool acts as a guardian on the defensive front, evaluating the strength of passwords and identifying potential breaches through sophisticated pattern recognition and analysis. By proactively identifying compromised credentials, it empowers users to take preemptive action, mitigating the risks associated with data breaches and unauthorized access.

Through its intuitive interface and customizable features, this AI-powered tool becomes an indispensable asset for individuals and organizations seeking to fortify their digital security posture. By enabling users to test the resilience of their passwords and stay ahead of emerging threats, it serves as a proactive safeguard in an ever-evolving cybersecurity landscape.

**Keywords:** Multi-Purpose, Password Analysis Tool, AI-Driven, Artificial Intelligence, Cracking Passwords, Vulnerabilities, Advanced Algorithms.

## TABLE OF CONTENTS

System Design Figures .....	9
Table of Figures .....	10
Chapter 1: Introduction .....	12
1.1 Introduction .....	13
1.2 Opportunity & Stakeholders .....	13
1.3 Problem Statement .....	14
1.4 Motivation and Challenges .....	16
1.5 Goals and Objectives .....	19
1.6 Solution Overview .....	19
1.7 Report Outline .....	20
Chapter 2: Market Review .....	23
2.1 Background .....	24
2.2 Market Review / Technologies Overview .....	24
2.3 Summary .....	27
Chapter 3: System Requirements .....	29
3.1 Hardware Requirements .....	30
3.2 Software Requirements .....	31
Chapter 4: Preparing Datasets .....	34
4.1 Introduction .....	35
4.2 Acquire or prepare relevant password datasets for training .....	35
Chapter 5: Develop and Train the AI Model .....	45
5.1 Introduction .....	46
5.2 Research and select appropriate AI algorithms .....	46
5.3 Train and optimize the AI model with the selected datasets .....	52
Chapter 6: Implementation .....	55
<b>References</b> .....	58
Appendix .....	59
Appendix A: Data Preparation Details .....	59
Appendix B: Data Filtration .....	60
Appendix C: Data Splitting .....	61
Github .....	63



## System Design Figures

Figure 1: Upper Case .....	14
Figure 2 :Upper and Lowercase .....	14
Figure 3: Upper, Lower and Numeric .....	15
Figure 4: Upper, Lower, Numeric, and Special Char .....	15
Figure 5: Datasets Screenshots .....	37
Figure 6: Pseudocode DataFilteration-1 .....	38
Figure 7: Pseudocode DataFilteration-2 .....	38
Figure 8: Pseudocode DataFileration-3 .....	39
Figure 9: Pseudocode DataFilteration-4 .....	39
Figure 10: DataFilteration Flowchart.....	40
Figure 11: Pseudocode DataSplitting-1 .....	41
Figure 12: Pseudocode DataSplitting-2 .....	42
Figure 13: Pseudocode Data Splitting-3 .....	42
Figure 14: DataSplitting Flowchart .....	43
Figure 15: Pseudocode RNN for Generating Passwords .....	48
Figure 16: Pseudocode RNN for Generating Passwords-1 .....	49
Figure 17: flowchart of RNN for Generating Passwords.....	49
Figure 18: Pseudocode for RNN's Training.....	50
Figure 19: Pseudocode for RNN's Traning-2 .....	51
Figure 20: Pseudocode for RNN's Traning-3 .....	51
Figure 21: Flowchart of RNN's Training.....	52
Figure 22: Training AI Model Flow Chart .....	53

## Table of Figures

Table 1: No. of iterations for each length of passwords .....	15
Table 2: Password Combinations.....	16
Table 3: Comparsion Tables of tools .....	26
Table 4: Comparison Table.....	27
Table 5: Datasets Table.....	36
Table 6: No. of Password as per Length .....	40

# **Chapter 1: Introduction**

# **Chapter 1: Introduction**

**1.1 Introduction**

**1.2 Opportunities and Stakeholders**

**1.3 Motivations and Challenges**

**1.4 Goals and Objectives**

**1.5 Solution Overview**

**1.6 Report Outline**

# Chapter 1: Introduction

## Project Title: Multi-Purpose AI Password Analysis Tool

### 1.1 Introduction

In today's world, where everything is becoming digital and online threats are on the rise, having strong password security is crucial. That's where the AI Multipurpose Password Analysis Tool comes in. It's a game-changing software that's here to revolutionize how we manage passwords. By combining cutting-edge artificial intelligence with traditional password analysis tools, this tool offers a complete solution for keeping our digital accounts safe. Whether it's cracking passwords or assessing password's strength, this tool covers multiple aspects of password security. And it doesn't stop there - it seamlessly works with popular tools like Hashcat, making it even more powerful. So, as we step into the future of password management, the AI Multipurpose Password Analysis Tool is leading the way, providing both analysis and cracking abilities to protect our sensitive information from the ever-evolving threats online.

### 1.2 Opportunity & Stakeholders

The key opportunities and primary stakeholders associated with this project are outlined as follows:

#### Opportunities

- **Enhanced Security Assessments:** Businesses and individuals can leverage the tool for penetration testing and vulnerability assessments, identifying potential weaknesses in their password security policies and practices.
- **AI-driven Offensive and Defensive Capabilities:** Utilizing AI for both offensive (password cracking) and defensive (password strength assessment) purposes provides a comprehensive approach to password security management.

#### Stakeholders

- **Businesses:** Companies aim to improve their overall cybersecurity posture by identifying and mitigating password-related vulnerabilities.
- **Security Professionals:** Penetration testers and security consultants who can use the tool for vulnerability assessments and ethical hacking activities.
- **Law Enforcement:** Agencies might leverage the tool for lawful investigations adhering to court orders and legal guidelines.

### 1.3 Problem Statement

With increasing password complexity requirements, brute-force attacks become significantly slower and less efficient.

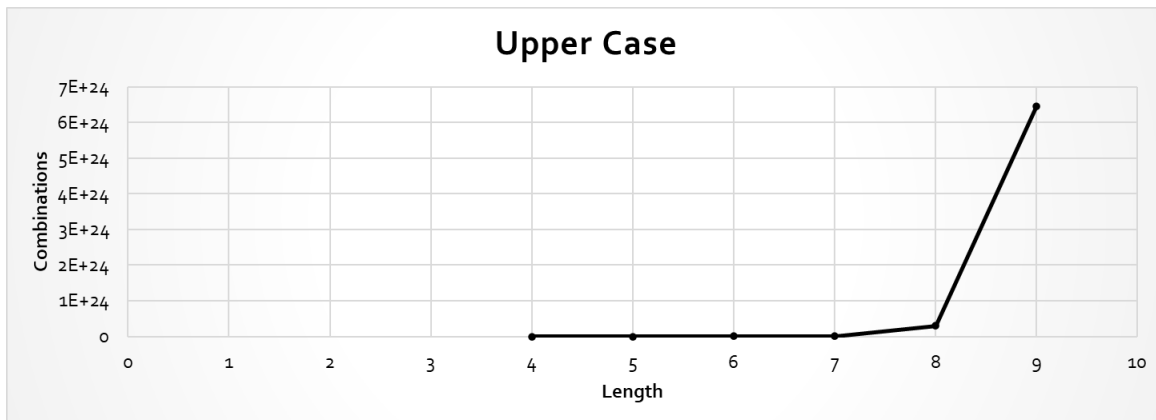


Figure 1: Upper Case

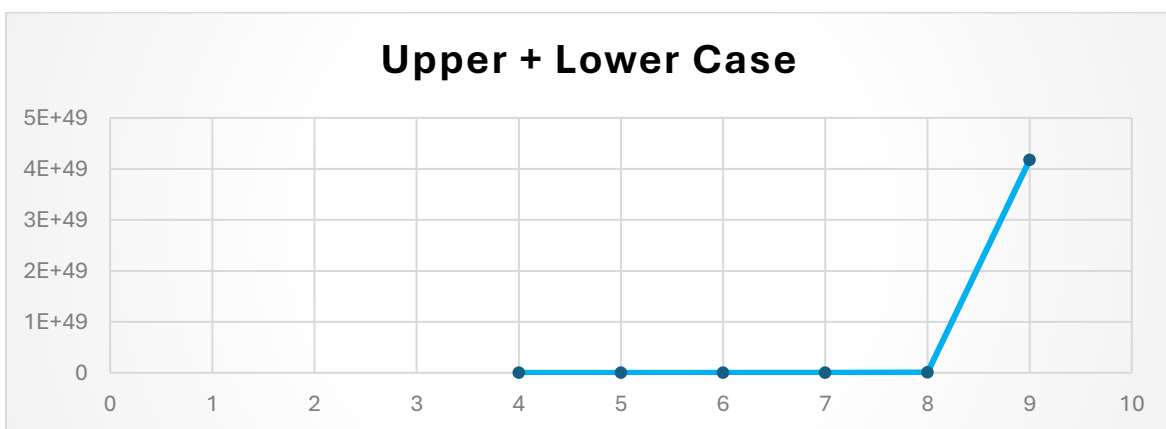
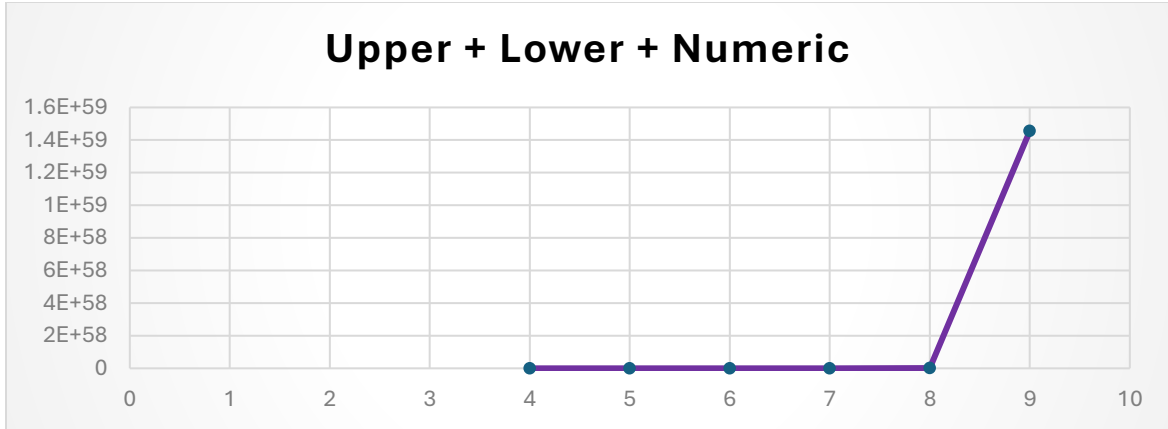
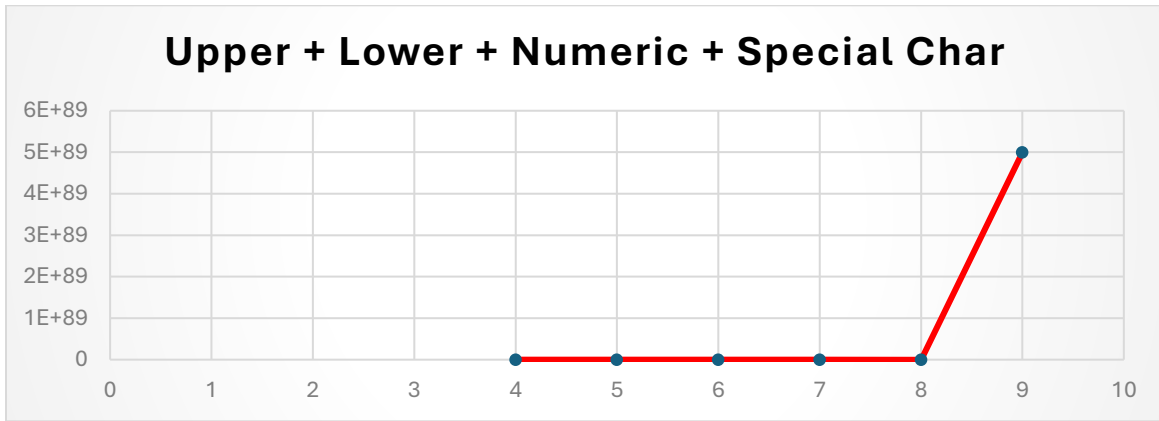


Figure 2 :Upper and Lowercase



**Figure 3: Upper, Lower and Numeric**



**Figure 4: Upper, Lower, Numeric, and Special Char**

Users often employ longer and more complex passwords (e.g., 8+ characters, combination of uppercase, lowercase, symbols) making traditional brute-force methods impractical.

Length	Combination
$4^{26+26}$	2220446049250313080847263336181640625
$4^{26+26+10}$	21267647932558653966460912964485513216
$5^{26+26}$	29098125988731506183153025616435306561536
$5^{26+26+10}$	21684043449710088680149056017398834228515625

**Table 1: No. of iterations for each length of passwords**

Length	Upper Case	Upper + Lower Case	Upper + Lower + Numeric	Upper + Lower + Numeric + Special Char
4	4.5036E+15	2.02824E+31	2.12676E+37	3.92319E+56
5	1.49012E+18	2.22045E+36	2.1684E+43	5.04871E+65
6	1.70582E+20	2.90981E+40	1.75945E+48	1.40029E+73
7	9.38748E+21	8.81248E+43	2.48931E+52	2.74926E+79
8	3.02231E+23	9.13439E+46	9.80797E+55	7.77068E+84
9	6.46108E+24	4.17456E+49	1.45558E+59	4.998E+89

*Table 2: Password Combinations*

## 1.4 Motivation and Challenges

Outlined below are the motivations and challenges associated with our AI-driven password analysis tool.

### Motivation

In the digital age, robust password security is more critical than ever. The need for an advanced, AI-driven password analysis tool arises from several key motivations.

#### Increasing Complexity of Passwords:

- As cyber threats evolve, users are encouraged to create more complex passwords to enhance security. However, the complexity often leads to users adopting predictable patterns, which can be exploited.
- Our tool aims to address this by leveraging AI to recognize and exploit these patterns, thereby improving both offensive and defensive security measures.

#### Advancements in AI and Machine Learning:

- The development of sophisticated AI algorithms offers new possibilities for password analysis and cracking. By integrating AI, our tool can stay ahead of traditional methods, providing more efficient and effective solutions.
- Utilizing state-of-the-art AI techniques ensures that our tool can handle the increasing complexity and diversity of modern passwords.



### **Comprehensive Security Assessments:**

- Businesses and security professionals need comprehensive tools that can provide both offensive and defensive capabilities. By integrating password strength analysis with AI-driven password cracking, our tool offers a holistic approach to password security.
- This dual functionality supports thorough security assessments, identifying vulnerabilities and testing the robustness of password policies.

### **Growing Number of Data Breaches:**

- The frequency and scale of data breaches are increasing, exposing millions of passwords. Our tool can analyze these breaches to understand common patterns and weaknesses, enhancing its cracking capabilities and providing insights into better password practices.
- By incorporating breach-checking features, our tool can alert users if their passwords have been compromised, enabling proactive security measures.

### **Enhancing Cybersecurity Awareness:**

- Educating users about the importance of strong passwords and the risks of weak ones is crucial. Our tool can demonstrate the ease with which weak passwords can be cracked, promoting better password practices.
- Security professionals can use our tool to raise awareness and provide tangible evidence of the need for robust password policies.

## **Challenges**

### **Data Collection and Quality:**

- Obtaining a diverse and representative dataset of passwords is crucial for training the AI model. However, ethical and legal considerations must be adhered to when sourcing data from online leaks and breaches.
- Ensuring the dataset is comprehensive and free from sensitive or personally identifiable information requires rigorous data cleaning and preprocessing.

**Balancing Offensive and Defensive Capabilities:**

- Integrating both offensive (password cracking) and defensive (password strength analysis) functionalities in a single tool presents a significant challenge. The tool must be designed to operate ethically, with proper authorization and adherence to legal guidelines.
- Ensuring that the tool's offensive capabilities do not compromise its defensive functionalities, and vice versa, requires careful design and implementation.

**Adapting to Evolving Password Practices:**

- Password patterns and user behaviors are continuously evolving. Traditional methods often struggle to keep up with these changes, and our AI-driven approach must be flexible and adaptive to remain effective.
- Continuous updates and retraining of the AI model are necessary to handle new password trends and behaviors.

**Efficiency and Scalability**

- As passwords become more complex, traditional brute-force methods become impractical due to their time and resource-intensive nature. Our AI-powered tool must significantly reduce the search space and improve efficiency.
- Ensuring that the tool can scale effectively to handle large datasets and complex passwords without compromising performance is a critical challenge.

**Security and Privacy Concerns:**

- Implementing features such as breach checking involves integrating with online databases, which must be done securely to prevent exposure to sensitive information.
- Ensuring that the tool itself is secure and does not become a vector for attacks is paramount. This includes safeguarding against misuse by malicious actors.

## 1.5 Goals and Objectives

### Goals

- Enhance Password Security
- Efficient Password Cracking
- Advanced Password Strength Analysis
- Comprehensive Vulnerability Assessments

### Objectives

- Focus cracking efforts on statistically probable password patterns.
- Analyze password strength based on complexity metrics and AI-driven pattern recognition.
- Reduced cracking time.
- Improved vulnerability assessment.

## 1.6 Solution Overview

Our proposed AI-powered tool enhances password security by leveraging advanced algorithms and human-like password data. It focuses on both offensive and defensive capabilities:

**Focused Cracking Efforts:** Utilizes AI to identify statistically probable password patterns, reducing the search space and accelerating the cracking process compared to traditional brute-force methods.

**Password Strength Analysis:** Employs AI-driven pattern recognition and complexity metrics to analyze and identify weaknesses in complex passwords, aiding in targeted cracking attempts.

**Reduced Cracking Time:** Enhances efficiency by concentrating on likely password patterns, significantly reducing the time required for cracking.

**Improved Vulnerability Assessment:** Provides comprehensive assessments of password vulnerabilities, helping to identify and address weak points proactively.

## Scope of the Project

### Defensive capabilities:

**Password strength assessment:** Analyze password complexity, identify dictionary words, and check for patterns using AI.

#### Optional

- **Breach checking:** Integrate with online databases to verify if entered passwords and email addresses have been exposed in known data breaches.

### Offensive capabilities: (For Ethical purposes only):

**AI-powered password cracking:** Utilize correlation of password data and advanced algorithms to efficiently crack passwords within a specified scope and with proper authorization.

## 1.7 Report Outline

The project report for the "Multi-Purpose AI Password Analysis Tool" begins with an introduction, providing a comprehensive background on password security and the impetus for developing this tool. It identifies key opportunities and stakeholders who will benefit from enhanced password analysis. The report then reviews existing password analysis tools such as John the Ripper, Brutus, and Wfuzz, highlighting their limitations and setting the context for the problem statement and proposed solution. Our proposed solution leverages AI to focus on statistically probable password patterns and complexity metrics, aiming to reduce cracking time and improve vulnerability assessment. The objectives and scope of the project are clearly defined, emphasizing defensive capabilities like password strength assessment and breach checking, as well as ethical offensive capabilities like AI-powered password cracking. The evaluation plan outlines a comprehensive assessment using diverse datasets from real-world breaches, password dictionaries, and synthetic data. The development and training section details the process of acquiring and preparing relevant datasets, training and optimizing the AI model, and selecting appropriate algorithms. Implementation focuses on core functionalities for password strength analysis and breach checking, with optional automated password cracking. The report also discusses the design and

development of a user-friendly interface, ensuring seamless integration with the AI model and functionalities. Thorough testing and refinement processes are documented to ensure the tool's effectiveness and usability. Finally, the report includes references and a list of figures to support the content and provide visual clarity.

## **Chapter 2: Market Review**

## **Chapter 2: Market Review**

**2.1** Background

**2.2** Market Review / Technologies Overview

**2.3** Summary

## Chapter 2: Market Review

### 2.1 Background

Traditional password cracking tools have relied on brute force techniques or precomputed dictionaries to decipher passwords. While these methods have been effective to some extent, they suffer from several limitations and challenges.

1. **Limited Efficiency:** Brute force attacks iterate through all possible combinations of characters, making them time-consuming and resource-intensive, especially for complex passwords.
2. **Dependence on Dictionaries:** Dictionary attacks rely on precompiled lists of commonly used passwords or words found in dictionaries. However, these lists may not encompass all possible variations, especially when users employ complex or unique passwords.
3. **Lack of Adaptability:** Traditional tools often struggle with adaptive password generation techniques, such as adding special characters or changing letter cases, making them less effective against modern password practices.
4. **Inability to Detect Breached Credentials:** Many password cracking tools do not have built-in mechanisms to cross-reference passwords with known breaches, leaving users unaware if their passwords have already been compromised.
5. **Scalability Issues:** As passwords become longer and more complex to enhance security, traditional methods face scalability challenges in terms of processing power and memory requirements.

### 2.2 Market Review / Technologies Overview

The foundation of password security lies in its ability to resist unauthorized access, primarily achieved through encryption and hashing techniques. Over the years, several tools and methodologies have been developed to analyze and crack passwords, each with its unique strengths and weaknesses.



## **Existing Systems**

The following are the Existing systems that exist in the market for about more than a decade.

### **John the Ripper**

#### **Limitations:**

1. Relies on traditional methods of password cracking such as dictionary attacks, brute-force attacks, and rainbow tables.
2. While powerful, it may not effectively adapt to evolving password patterns and behaviors without continuous updates and modifications.

#### **Problems:**

1. Lack of advanced AI integration for targeted password list generation based on learned patterns.
2. Limited defensive capabilities in analyzing password strength beyond basic dictionary checks.

### **Brutus**

#### **Limitations:**

1. Primarily designed for online password cracking through network protocols like HTTP, FTP, SMB, etc.
2. May lack advanced AI-driven capabilities for generating targeted password lists.

#### **Problems:**

1. Limited applicability for offline password analysis and cracking scenarios.
2. May not integrate well with the defensive aspects of the proposed tool, such as analyzing password strength and checking for breached credentials.

### **Wfuzz**

#### **Limitations:**

1. Primarily focuses on web application security testing, including fuzzing and brute-forcing directories and files.
2. May lack specialized features for password cracking and analysis.

#### **Problems:**

1. Not specifically tailored for password analysis and cracking tasks, thus requiring significant adaptation to fit into the proposed project's scope.
2. Limited or no integration with AI-driven password analysis and cracking techniques.

### Comparison Tables

Tool	John the Ripper	RainbowCrack	OphCrack
Type	Password Cracker	Password Cracker	Password Cracker
Supported Platforms	Windows, Linux, macOS	Windows, Linux	Windows, Linux
Password Hashes Supported	Various (Unix, Windows, etc.)	LM, NTLM, MD5, SHA1, SHA256, SHA512	LM, NTLM
Attack Methods	Dictionary, Brute Force, Hybrid	Precomputed Hash Tables	Rainbow Tables, Brute Force
Speed	Fast	Depends on Rainbow Table size	Moderate
User Interface	Command Line	Command Line	GUI
License	Open Source	Freeware	Open Source
Usage	Penetration Testing, Password Auditing	Password Cracking	Password Recovery

**Table 3: Comparison Tables of tools**

<b>Tool</b>	<b>L0phtCrack</b>	<b>Aircrack-ng</b>
<b>Type</b>	Password Cracker	Wi-Fi Network Security Tool
<b>Supported Platforms</b>	Windows	Linux, macOS
<b>Password Hashes Supported</b>	LM, NTLM	WEP, WPA, WPA2
<b>Attack Methods</b>	Dictionary, Brute Force	Dictionary, Brute Force, WPS PIN
<b>Speed</b>	Fast	Depends on hardware and complexity of the password
<b>User Interface</b>	GUI	Command Line
<b>License</b>	Commercial	Open Source
<b>Usage</b>	Password Cracking	Wi-Fi Network Security Testi

**Table 4: Comparison Table**

## 2.3 Summary

The evolution of password security technologies reflects the ongoing battle between cybersecurity professionals and cybercriminals. Traditional tools like John the Ripper, Brutus, and Wfuzz continue to play crucial roles in password cracking and security testing. However, the advent of AI and machine learning has ushered in a new era of password security, offering more efficient and effective methods for analyzing and cracking passwords.

The integration of human-like passwords data with advanced algorithms represents a significant leap forward, enabling more targeted and successful password cracking attempts. This literature review underscores the need for continuous innovation in password security, highlighting the potential of AI-powered tools to enhance our defenses against evolving cyber threats.

By understanding the strengths and limitations of existing tools and technologies, this review sets the stage for the development of a comprehensive AI-powered password analysis tool. Such a tool aims to address the current gaps in password security, providing a more robust and reliable means of protecting sensitive data in an increasingly digital world.

# **Chapter 3:**

# **System Requirements**

## **Chapter 3: System Requirements**

**3.1** Hardware Requirements

**3.2** Software Requirements

## Chapter 3: System Requirements

### 3.1 Hardware Requirements

#### 3.1.1 Workstation: HP Z840

The HP Z840 workstation is a high-performance desktop designed for demanding tasks such as AI and machine learning model training. It offers robust processing power, extensive memory capacity, and advanced graphics capabilities, making it ideal for computationally intensive applications.

#### 3.1.2 Processor: Intel Xeon E5-2680 V4 Dual

The Intel Xeon E5-2680 V4 is a high-end server processor with 14 cores and 28 threads, offering a base clock speed of 2.4 GHz and a turbo boost up to 3.3 GHz. Utilizing two of these processors in a dual configuration provides a total of 28 cores and 56 threads, significantly enhancing parallel processing capabilities crucial for training complex AI models.

#### 3.1.3 RAM: 64GB DDR4 ECC

64GB of DDR4 Error-Correcting Code (ECC) RAM ensures high data integrity and system stability, which is essential for handling large datasets and preventing data corruption during intensive computations. This amount of memory supports the simultaneous operation of multiple AI models and applications without performance degradation.

#### 3.1.4 GPU: RTX 3070TI 8GB GDDR6x

The NVIDIA RTX 3070TI graphics card, with 8GB of GDDR6x memory, provides powerful parallel processing capabilities and accelerated performance for deep learning tasks. Its CUDA cores and Tensor cores are optimized for AI workloads, significantly speeding up the training and inference processes of neural networks.

#### 3.1.5 SSD: 512GB

A 512GB Solid-State Drive (SSD) offers fast read/write speeds and quick access to data, reducing loading times and improving overall system responsiveness. The SSD is used for storing the operating system, software applications, and project files, ensuring efficient data retrieval and storage.

## **3.2 Software Requirements**

### **3.2.1 Operating System: Windows 11 64bit**

Windows 11 is the latest version of Microsoft's operating system, providing a modern interface, enhanced security features, and support for the latest hardware and software technologies. The 64-bit version is necessary to leverage the full potential of the workstation's hardware, especially the large amount of RAM and dual processors.

### **3.2.2 Integrated Development Environment (IDE): Visual Studio**

Visual Studio is a comprehensive development environment from Microsoft, offering tools and features for coding, debugging, and deploying applications. It supports multiple programming languages and integrates well with various frameworks and libraries, making it suitable for developing and testing the AI password analysis tool.

### **3.2.3 Programming Language: Python 3.12**

Python 3.12 is the latest stable release of the Python programming language, known for its simplicity, readability, and extensive library support. It is widely used in AI and machine learning projects due to its powerful frameworks and community support. Python serves as the primary language for developing the AI models and integrating different components of the tool.

### **3.2.4 Machine Learning Framework: TensorFlow**

TensorFlow is an open-source machine learning framework developed by Google, offering comprehensive tools for building and deploying machine learning models. It supports deep learning and neural network training, providing high performance on both CPUs and GPUs. TensorFlow's flexibility and scalability make it a preferred choice for developing complex AI applications.

### **3.2.5 Deep Learning Library: PyTorch**

PyTorch is an open-source deep learning library developed by Facebook's AI Research lab, known for its dynamic computational graph and ease of use. It is widely adopted for research and production in AI, offering strong support for GPU acceleration and integration with other machine learning tools. PyTorch is used for experimenting with and deploying AI models in the password analysis tool.

### **3.2.6 GPU-Accelerated Library: cuDNN**

cuDNN (CUDA Deep Neural Network library) is a GPU-accelerated library developed by NVIDIA, designed to enhance the performance of deep learning frameworks. It provides highly optimized implementations of standard routines such as forward and backward convolution, pooling, normalization, and activation layers. cuDNN is essential for maximizing the efficiency of TensorFlow and PyTorch models on the RTX 3070TI GPU.



# **Chapter 4: Preparing Datasets**

## **Chapter 4: Preparing Datasets**

**4.1** Introduction

**4.2** Acquire or prepare relevant password datasets for training

## Chapter 5: Preparing Datasets

### 4.1 Introduction

In the development of any machine learning or artificial intelligence project, the quality and preparation of datasets play a pivotal role in determining the success and efficacy of the final model. This chapter delves into the essential processes and considerations involved in acquiring, preparing, and managing datasets specifically tailored for our multipurpose AI password analysis tool. The robustness of our AI models, such as Generative Adversarial Networks (GANs), Recurrent Neural Networks (RNNs), and Transformers, hinges on the richness and accuracy of the data they are trained on.

### 4.2 Acquire or prepare relevant password datasets for training

Getting a dataset ready for password generation involves a series of steps aimed at making sure the data accurately represents different password characteristics and is suitable for training. Here's a simplified breakdown of what we need to do:

#### Data Collection

Get a mix of passwords from various sources, like:

- Online leaks and breaches (follow ethical and legal guidelines).
- Making sure our dataset covers a wide range of password types, including different lengths, complexities, and compositions.

**Collected Datasets Table**

No	Name	Size
1	rockyou2021.txt dictionary from kys234 on RaidForums	12.7 GB
2	36.4GB-18_in_1.lst_2.7z	4.50 GB
3	ASLM.txt.7z	127 MB
4	b0n3z_dictionary-SPLIT-BY-LENGTH-34.6GB_2.7z	3.29 GB
5	b0n3z-wordlist-sorted_REPACK-69.3GB_3.7z	9.07 GB
6	bad-passwords-master.zip	1.34 MB
7	crackstation.txt.gz	4.19 GB
8	dictionaries-master.zip	19.3 MB
9	Password lists.zip	336 MB
10	password-list-main.zip	291 MB
11	password-lists-master.zip	8.86 MB
12	pastePasswordLists-main_2.zip	54.6 MB
13	PowerSniper-master.zip	0.3 MB
14	pwlist-master.zip	8.02 MB
15	rockyou.zip	41.7 MB
16	SecLists-master.zip	554 MB
17	statistically-likely-usernames-master.7z	9.07 MB
18	vietnam-password-lists-master.zip	5.14 MB
19	wpa-passwords-master.zip	5.79 MB
20	WPA-PSK WORDLIST 3 Final (13 GB).rar	4.49 GB
21	cyclone.hashesorg.hashkiller.combined.7z	6.58 GB

**Table 5: Datasets Table**

## Dataset Downloaded Screen Shot

Name	Date modified	Type	Size
rockyou2021.txt dictionary from kys234 on RaidForums	7/15/2022 4:56 AM	File folder	
36.4GB-18_in_1.lst_2.7z	3/10/2024 11:28 AM	7z Archive	4,726,867 KB
ASLM.txt.7z	12/11/2021 12:16 PM	7z Archive	130,261 KB
b0n3z_dictionary-SPLIT-BY-LENGTH-34.6GB_2.7z	3/10/2024 12:47 PM	7z Archive	3,460,722 KB
b0n3z-wordlist-sorted_REPACK-69.3GB_3.7z	3/10/2024 12:26 PM	7z Archive	9,511,223 KB
bad-passwords-master.zip	3/10/2024 10:02 AM	zip Archive	1,376 KB
crackstation.txt.gz	3/10/2024 4:57 PM	gz Archive	4,395,271 KB
dictionaries-master.zip	3/10/2024 10:02 AM	zip Archive	19,766 KB
Password lists.zip	5/25/2024 1:21 AM	zip Archive	344,979 KB
password-list-main.zip	3/10/2024 10:26 AM	zip Archive	298,841 KB
password-lists-master.zip	3/10/2024 9:59 AM	zip Archive	8,894 KB
pastePasswordLists-main_2.zip	3/10/2024 10:26 AM	zip Archive	55,928 KB
PowerSniper-master.zip	3/10/2024 10:02 AM	zip Archive	372 KB
pwlist-master.zip	3/10/2024 10:02 AM	zip Archive	8,220 KB
rockyou.zip	5/25/2024 1:24 AM	zip Archive	42,728 KB
SecLists-master.zip	3/10/2024 10:50 AM	zip Archive	567,740 KB
statistically-likely-usernames-master.7z	5/25/2024 1:14 AM	7z Archive	9,298 KB
vietnam-password-lists-master.zip	3/10/2024 10:00 AM	zip Archive	5,269 KB
wpa-passwords-master.zip	3/10/2024 10:02 AM	zip Archive	5,937 KB
WPA-PSK WORDLIST 3 Final (13 GB).rar	3/10/2024 10:56 AM	rar Archive	4,710,749 KB
cyclone.hashesorg.hashkiller.combined.7z	5/25/2024 1:42 AM	7z Archive	6,902,263 KB

Figure 5: Datasets Screenshots

## Data Cleaning

Filter the password of length of 4 to 10 to keep the datasets clean and manageable.

Get rid of any weird or corrupted entries that could mess up training.

## Pseudocode: Data Filtration

```
≡ pseudocode Data Filtration •
≡ pseudocode Data Filtration
1  START Main Script
2    Parse command-line arguments
3    IF word lengths not provided
4      Set default word lengths to [5, 6, 7, 8]
5    CALL parallel_filter_words with parsed arguments
6  END Main Script
7
8  FUNCTION parallel_filter_words(directory, word_lengths, num_chunks, subchunk_size, encoding, output_directory)
9    Create multiprocessing pool
10   IF output directory does not exist
11     Create output directory
12
13   Open log file in output directory
14
15   FOR each file in directory
16     IF file does not end with '.txt'
17       Continue to next file
18
19     Get file path
20     Get file chunks based on num_chunks
21
22     FOR each word length in word_lengths
23       Create directory for current word length if not exists
24       Create output filename for filtered words
25
26       Initialize list to store results from multiprocessing
27
28       FOR each chunk (chunk_start, chunk_size) in file chunks
```

Figure 6: Pseudocode DataFiltration-1

```
28  ✓   FOR each chunk (chunk_start, chunk_size) in file chunks
29     Apply process_chunk asynchronously with pool
30     Add result to results list
31
32     Open output file for writing
33  ✓   FOR each result in results list
34     Get filtered words from result
35  ✓   IF filtered words exist
36     Write filtered words to output file
37
38     Write log entry for processed file and output file
39
40     Close and join multiprocessing pool
41
42     PRINT message indicating where filtered words are saved
43  END FUNCTION
44
45  ✓ FUNCTION process_chunk(chunk_start, chunk_size, lengths, filename, subchunk_size, encoding)
46     Initialize empty list for filtered words
47
48     Open file with specified encoding and seek to chunk_start position
49
50  ✓   WHILE chunk_size > 0
51     Read subchunk from file (size: min(subchunk_size, chunk_size))
```

Figure 7: Pseudocode DataFiltration-2

```

49
50     WHILE chunk_size > 0
51         Read subchunk from file (size: min(subchunk_size, chunk_size))
52         Split subchunk into lines
53
54         Filter lines by specified lengths
55         Add filtered words to list
56
57         Reduce chunk_size by subchunk_size
58         IF no more lines in subchunk
59             Break loop
60
61     RETURN list of filtered words
62 END FUNCTION
63
64 FUNCTION filter_words_by_length(chunk, lengths)
65     Initialize empty list for filtered words
66
67     FOR each word in chunk
68         IF length of word is in lengths
69             Add word to filtered words list
70
71     RETURN filtered words list
72 END FUNCTION
73
74 FUNCTION get_file_chunks(filename, num_chunks)
75     Get size of file

```

**Figure 8: Pseudocode DataFiltration-3**

```

76     Calculate chunk size as file size divided by num_chunks
77     Initialize empty list for chunks
78
79     FOR i from 0 to num_chunks - 1
80         Calculate chunk start as i * chunk_size
81         Add (chunk_start, chunk_size) to list of chunks
82
83     RETURN list of chunks
84 END FUNCTION
85

```

**Figure 9: Pseudocode DataFiltration-4**

## Flowchart



Figure 10: DataFilteration Flowchart

<b>Total number of Input passwords</b>	8459063135
<b>Length</b>	<b>Total Count of Filtered Passwords</b>
5	10,751,871
6	563,608,354
7	465,597,114
8	1,357,729,013

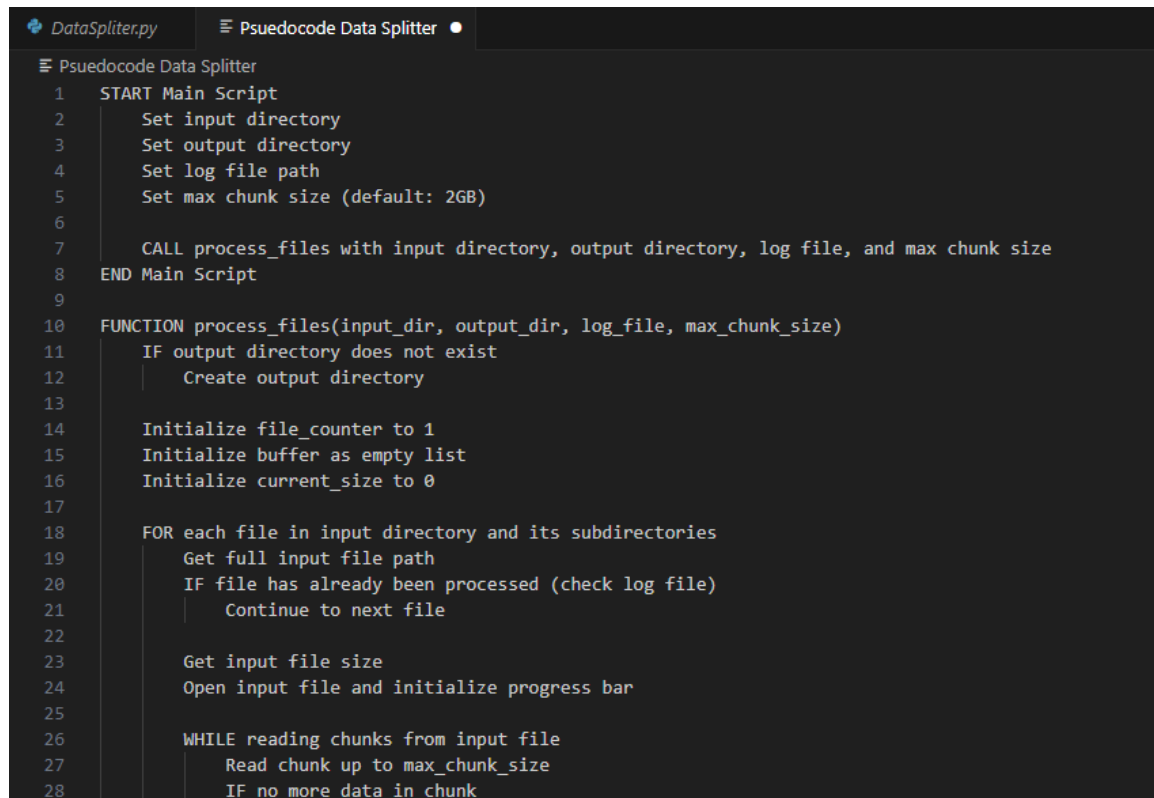
Table 6: No. of Password as per Length



## Splitting the Dataset

We divided our dataset into multiple 2 GB files for efficient processing and management. Our approach involves processing these chunks independently, which helps in handling large datasets without the need for significant memory resources. Each chunk is processed, and the results are aggregated to ensure comprehensive analysis. This method allows us to train, validate, and test our model effectively while managing data size constraints.

## Pseudocode: Data Splitting



```
1  START Main Script
2    Set input directory
3    Set output directory
4    Set log file path
5    Set max chunk size (default: 2GB)
6
7    CALL process_files with input directory, output directory, log file, and max chunk size
8  END Main Script
9
10 FUNCTION process_files(input_dir, output_dir, log_file, max_chunk_size)
11   IF output directory does not exist
12     Create output directory
13
14   Initialize file_counter to 1
15   Initialize buffer as empty list
16   Initialize current_size to 0
17
18   FOR each file in input directory and its subdirectories
19     Get full input file path
20     IF file has already been processed (check log file)
21       Continue to next file
22
23     Get input file size
24     Open input file and initialize progress bar
25
26     WHILE reading chunks from input file
27       Read chunk up to max_chunk_size
28       IF no more data in chunk
```

Figure 11: Pseudocode DataSplitting-1

```

29         Break loop
30
31     FOR each line in chunk
32     TRY
33         Add line to buffer
34         Increment current_size by line length + 1 (for newline character)
35     IF current_size exceeds max_chunk_size
36         Write buffer to new output file
37         Increment file_counter
38         Clear buffer
39         Reset current_size to 0
40     EXCEPT error
41         Print error message
42         Continue to next line
43
44         Update progress bar with chunk size
45         Free memory after processing chunk
46
47     Log processed file in log file
48
49     IF buffer is not empty
50         Write remaining buffer to new output file
51 END FUNCTION

```

Figure 12: Pseudocode DataSplitting-2

```

52
53 FUNCTION create_new_output_file(output_dir, file_counter)
54     Create output file path using file_counter with zero padding
55     Open output file for writing in UTF-8 encoding
56     RETURN output file and its path
57 END FUNCTION
58
59 FUNCTION log_processed_file(log_file, input_file_path)
60     Open log file for appending in UTF-8 encoding
61     Write input file path to log file
62     Close log file
63 END FUNCTION
64
65 FUNCTION is_file_processed(log_file, input_file_path)
66     IF log file does not exist
67         RETURN False
68
69     Open log file for reading in UTF-8 encoding
70     Read all processed file paths
71     Close log file
72
73     IF input file path is in processed file paths
74         RETURN True
75
76     RETURN False
77 END FUNCTION

```

Figure 13: Pseudocode Data Splitting-3

### Flowchart:

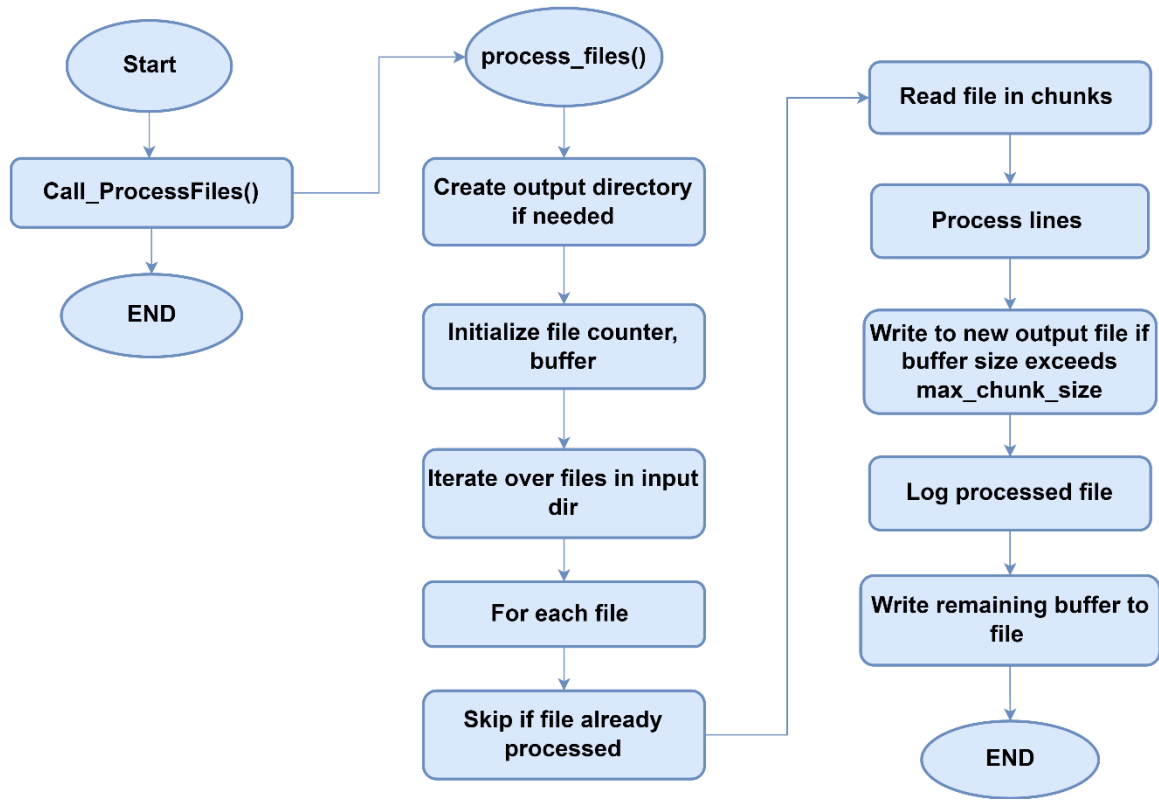


Figure 14: DataSplitting Flowchart

### Save the Preprocessed Dataset

As the files contain datasets of huge volumes i.e. 120GB, 90GB, 100GB. We couldn't run these files. So, we split each file into multiples files which were approx. 2 GB and saved them as txt files and used the Data filtration technique to filter the passwords for training.

# **Chapter 5:**

## **Develop and Train the AI Model**

# **Chapter 5: Develop and Train the AI Model**

## **5.1 Introduction**

## **5.2 Research and select appropriate AI algorithms**

## **5.3 Train and optimize the AI model with the selected datasets**

## Chapter 5: Develop and Train the AI Model

### 5.1 Introduction

The development and training of the RNN model lie at the heart of our multipurpose AI password analysis tool. This chapter delves into the intricate process of building, fine-tuning, and optimizing the RNN architecture to effectively analyze and predict password strength, generate secure passwords, and identify potential vulnerabilities. Recurrent Neural Networks (RNNs) are particularly suited for sequence-based data, making them ideal for understanding patterns within password structures and recognizing common vulnerabilities.

This chapter covers every stage of model development, from selecting suitable RNN architectures to implementing them using frameworks such as TensorFlow and PyTorch, with cuDNN to enhance GPU performance. It highlights the significant advantages of using an RTX 3070TI GPU to efficiently manage the intensive training processes required by RNNs. Additionally, we explore methods to address challenges specific to RNNs, such as vanishing gradients, by integrating techniques like gradient clipping and implementing LSTM or GRU cells where appropriate. This approach ensures a robust foundation for a password analysis tool capable of adapting to varied password complexities and strengthening overall security.

### 5.2 Research and select appropriate AI algorithms

We have considered and selected the RNN model to use for our project. Here is the overview of the RNN model.

- Recurrent Neural Networks (RNNs)

#### **Algorithm Overview**

##### **Recurrent Neural Networks (RNNs)**

Recurrent Neural Networks (RNNs) are a class of neural networks designed to handle sequential data by maintaining a hidden state that captures information about previous inputs in the sequence. RNNs are well-suited for tasks where the input data's temporal

order is important, making them a suitable choice for password generation, where the order of characters matters.

### **Training Phase**

#### **a. Input**

A dataset of passwords represented as sequences of characters.

#### **b. Architecture**

The RNN consists of recurrent units (such as LSTM - Long Short-Term Memory, or GRU - Gated Recurrent Unit) that process one character at a time while maintaining a hidden state. This hidden state captures information about previous characters in the password sequence.

#### **c. Sequence Learning**

The RNN is trained to predict the next character in the sequence given the previous characters. This is done by feeding each character in the sequence into the RNN and comparing the predicted next character with the actual next character in the training data.

#### **d. Backpropagation Through Time (BPTT)**

The errors are backpropagated through time to update the weights of the RNN, allowing it to learn the patterns and dependencies present in the password dataset.

#### **e. Loss Function**

The RNN is trained to minimize a loss function, such as categorical cross-entropy, which measures the difference between the predicted and actual characters in the sequence.

### **Generation Phase**

#### **a. Seed**

During the generation phase, a seed sequence is provided as input to the RNN to initiate the generation process. This seed sequence can be randomly chosen or predefined.

#### **b. Character-by-Character Generation**

RNN generates new characters one at a time by feeding the previous character and the current hidden state back into the network. The output

character is sampled from the predicted probability distribution over the character vocabulary.

### c. Output

The generated characters are concatenated to form a new password sequence.

## Evaluation

The quality of the generated passwords can be evaluated using metrics such as similarity to real passwords, diversity, and entropy. Human evaluators can also assess the usability and security of the generated passwords.

## Pseudocode (For Generating Passwords)

```
Pseudocode RNN Generate > RNNModel > init_hidden
1  import torch
2  import torch.nn as nn
3  import numpy as np
4
5  # Define RNN model for word generation
6  class RNNModel(nn.Module):
7      def __init__(self, input_size, hidden_size, num_layers, num_classes):
8          super(RNNModel, self).__init__()
9          self.hidden_size = hidden_size
10         self.num_layers = num_layers
11         self.rnn = nn.RNN(input_size, hidden_size, num_layers, batch_first=True)
12         self.fc = nn.Linear(hidden_size, num_classes)
13
14         def forward(self, x, h):
15             out, h = self.rnn(x, h)
16             out = self.fc(out.reshape(out.size(0) * out.size(1), out.size(2)))
17             return out, h
18
19         def init_hidden(self, batch_size):
20             return torch.zeros(self.num_layers, batch_size, self.hidden_size).to(device)
21
22         device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
23
24         # Load vocabulary
25         char_to_idx = torch.load('char_to_idx.pth')
26         idx_to_char = torch.load('idx_to_char.pth')
27         chars = sorted(char_to_idx.keys())
28
29         input_size = len(chars)
30         hidden_size = 64
31         num_layers = 2
32
33         # Load trained RNN model
34         model = RNNModel(input_size, hidden_size, num_layers, len(chars)).to(device)
35         model.load_state_dict(torch.load('rnn_model.pth'))
36
37         def generate_word(model, start_char, length):
38             model.eval()
39             h = model.init_hidden(1)
40             input = torch.eye(len(chars))[char_to_idx[start_char]].unsqueeze(0).unsqueeze(0).to(device)
41             word = start_char
```

Figure 15: Pseudocode RNN for Generating Passwords



```

41 word = start_char
42
43 with torch.no_grad():
44     for _ in range(length - 1):
45         output, h = model(input, h)
46         _, top_idx = torch.topk(output[-1], 1)
47         generated_idx = top_idx.item()
48
49         if generated_idx not in idx_to_char:
50             break
51
52         next_char = idx_to_char[generated_idx]
53         word += next_char
54         input = torch.eye(len(chars))[char_to_idx[next_char]].unsqueeze(0).unsqueeze(0).to(device)
55
56     return word
57
58 # Example usage
59 start_char = 's' # Starting character for word generation
60 word_length = 9 # Length of the word to generate
61 new_word = generate_word(model, start_char, word_length)
62 print(f'Generated word: {new_word}')
63

```

Figure 16: Pseudocode RNN for Generating Passwords-1

## Flowchart

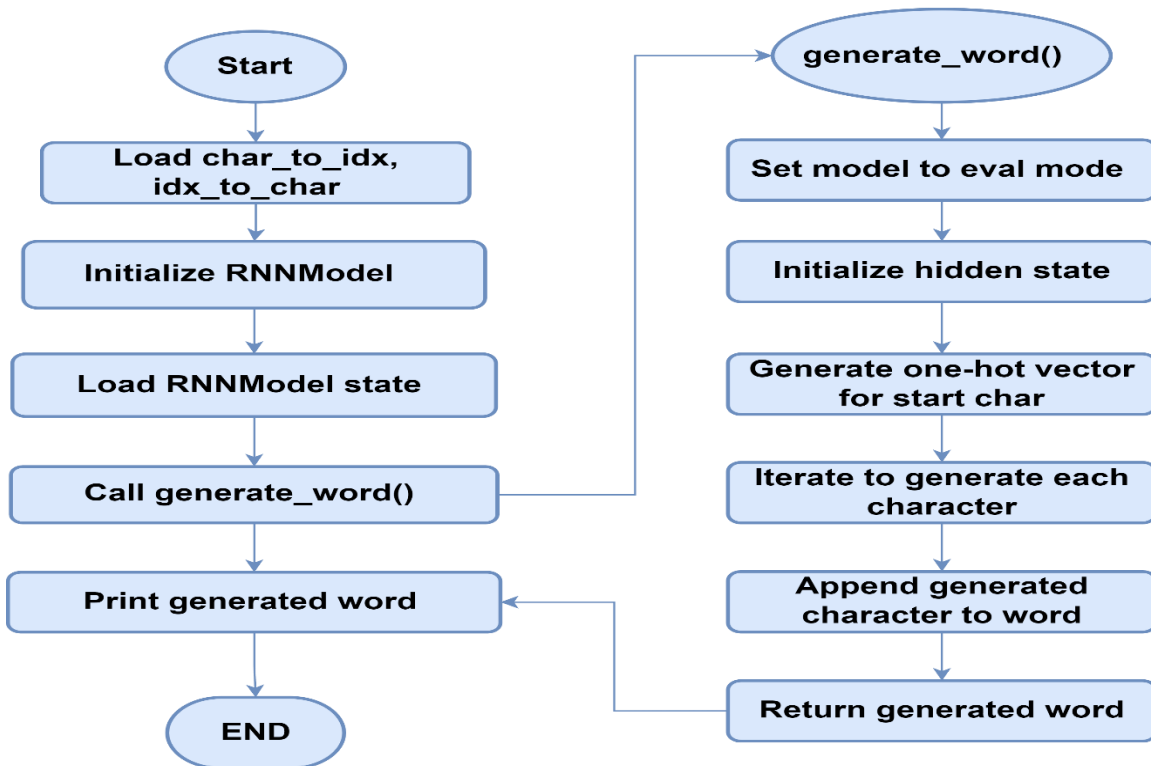


Figure 17: flowchart of RNN for Generating Passwords

## Pseudocode (For Training)

```
1  import torch
2  import torch.nn as nn
3  import torch.optim as optim
4  import numpy as np
5  from tqdm import tqdm
6
7  # Parameters
8  hidden_size = 64
9  num_layers = 2
10 num_epochs = 10
11 learning_rate = 0.003
12
13 # Read dataset
14 def read_words(file_path):
15     with open(file_path, 'r', encoding='utf-8') as file:
16         words = file.read().splitlines()
17     return words
18
19 words = read_words('path_to_your_dataset.txt')
20
21 # Create character vocabulary
22 chars = sorted(list(set(''.join(words))))
23 char_to_idx = {ch: i for i, ch in enumerate(chars)}
24 idx_to_char = {i: ch for i, ch in enumerate(chars)}
25
26 input_size = len(chars)
27
28 # Preprocess dataset
29 def preprocess(words, char_to_idx):
30     sequences = []
31     for word in words:
32         sequences.append([char_to_idx[char] for char in word])
33     return sequences
34
35 sequences = preprocess(words, char_to_idx)
36
37 # Define RNN model
38 class RNNModel(nn.Module):
39     def __init__(self, input_size, hidden_size, num_layers, num_classes):
40         super(RNNModel, self).__init__()
41         self.hidden_size = hidden_size
```

Figure 18: Pseudocode for RNN's Training

```

42     self.num_layers = num_layers
43     self.rnn = nn.RNN(input_size, hidden_size, num_layers, batch_first=True)
44     self.fc = nn.Linear(hidden_size, num_classes)
45
46     def forward(self, x, h):
47         out, h = self.rnn(x, h)
48         out = self.fc(out.reshape(out.size(0) * out.size(1), out.size(2)))
49         return out, h
50
51     def init_hidden(self, batch_size):
52         return torch.zeros(self.num_layers, batch_size, self.hidden_size).to(device)
53
54 device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
55 model = RNNModel(input_size, hidden_size, num_layers, len(chars)).to(device)
56
57 # Loss and optimizer
58 criterion = nn.CrossEntropyLoss()
59 optimizer = optim.Adam(model.parameters(), lr=learning_rate)
60
61 # Training loop
62 for epoch in range(num_epochs):
63     model.train()
64     h = model.init_hidden(1)
65     loss_avg = 0
66     with tqdm(total=len(sequences), desc=f'Epoch {epoch+1}/{num_epochs}') as pbar:
67         for seq in sequences:
68             inputs = torch.eye(len(chars))[seq[:-1]].unsqueeze(0).to(device)
69             targets = torch.tensor(seq[1:], dtype=torch.long).to(device)
70
71             h = h.detach()
72             outputs, h = model(inputs, h)
73
74             loss = criterion(outputs, targets.view(-1))
75             optimizer.zero_grad()
76             loss.backward()
77             optimizer.step()
78

```

Figure 19: Pseudocode for RNN's Training-2

```

79         loss_avg += loss.item()
80         pbar.update(1)
81
82     print(f'Epoch [{epoch+1}/{num_epochs}], Loss: {loss_avg/len(sequences):.4f}')
83
84 # Save model and vocabulary
85 torch.save(model.state_dict(), 'rnn_model.pth')
86 torch.save(char_to_idx, 'char_to_idx.pth')
87 torch.save(idx_to_char, 'idx_to_char.pth')
88

```

Figure 20: Pseudocode for RNN's Training-3

## Flowchart

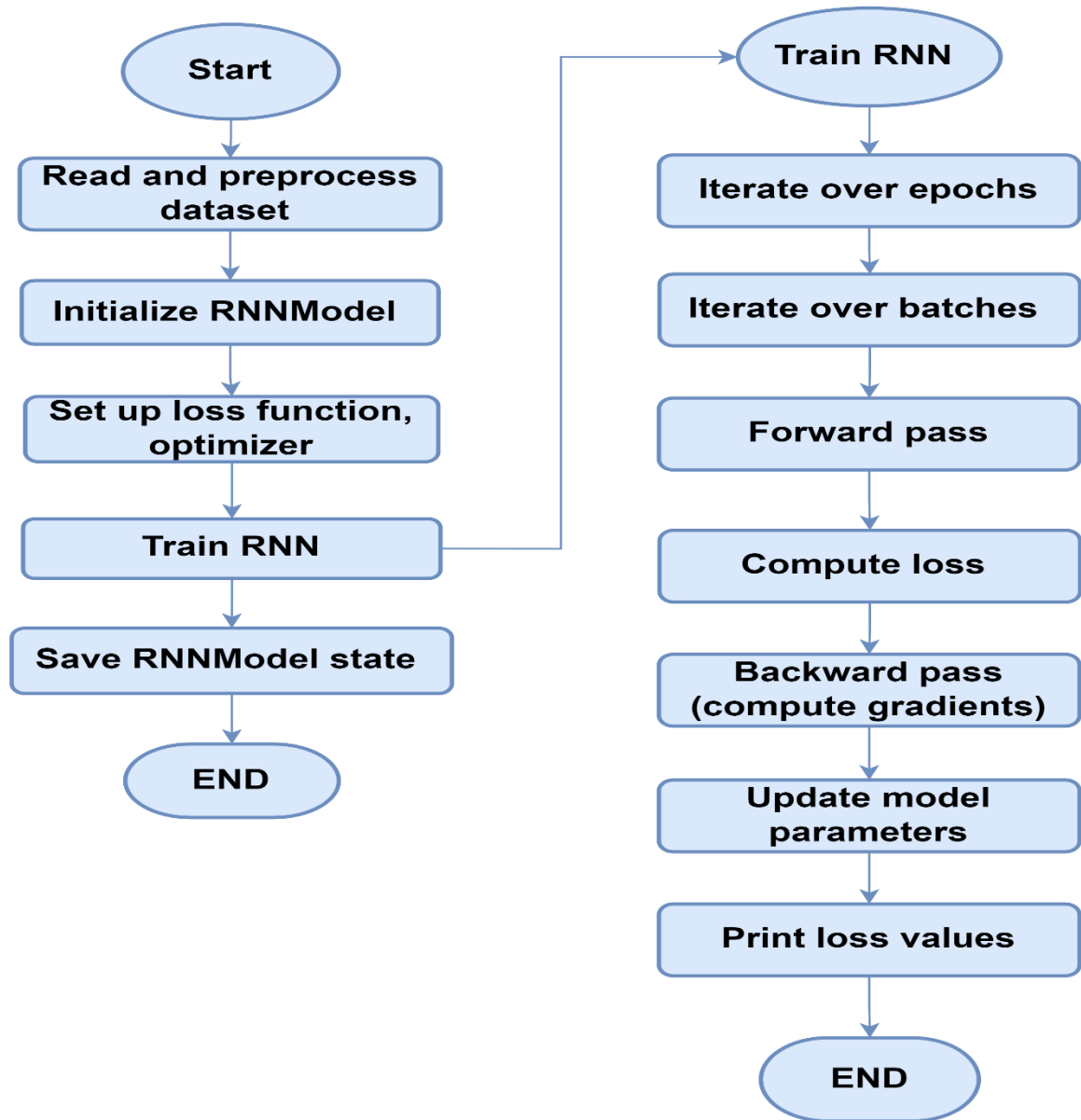
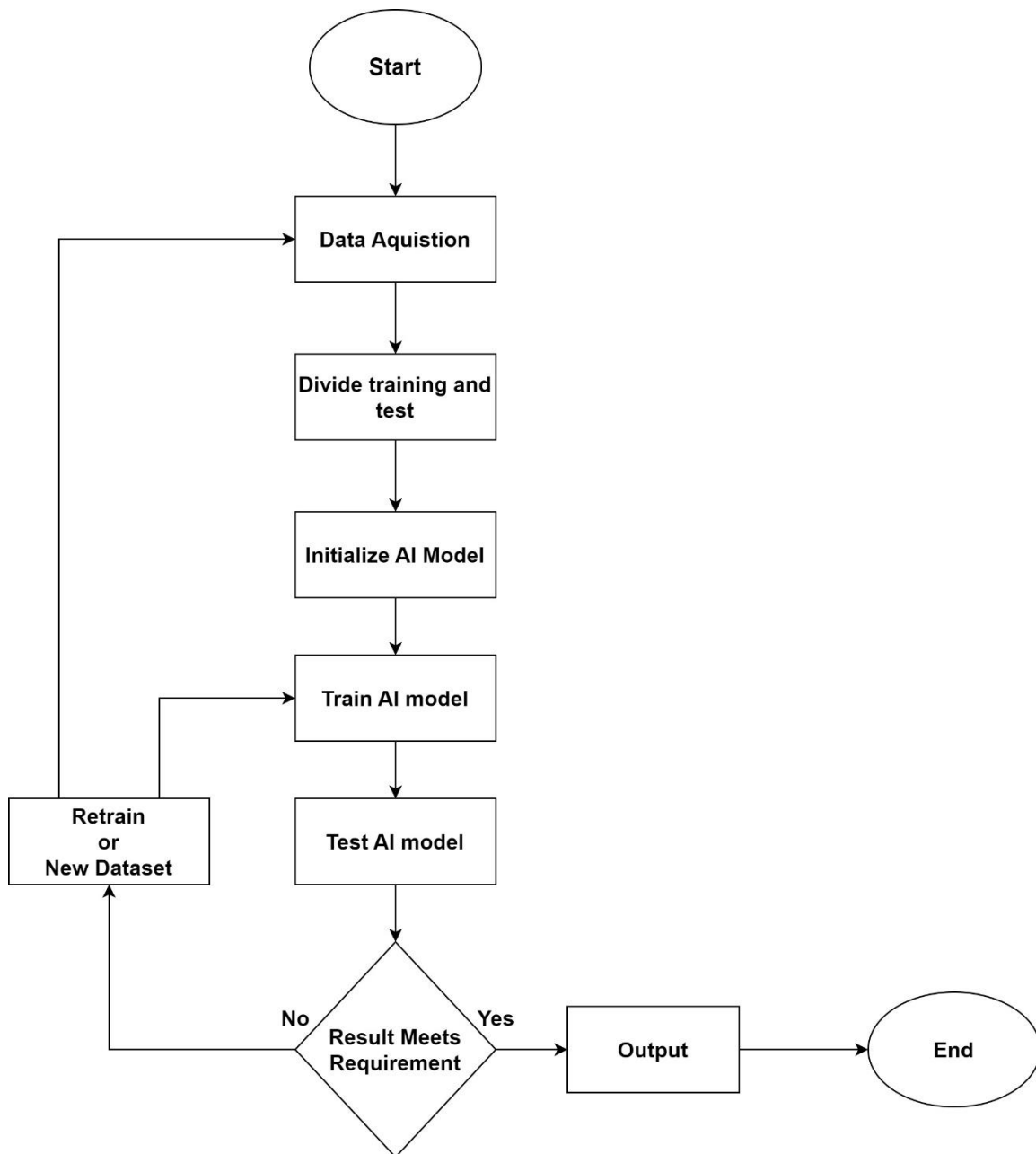


Figure 21: Flowchart of RNN's Training

### 5.3 Train and optimize the AI model with the selected datasets



### Train and optimize the AI models

Figure 22: Training AI Model Flow Chart

# **Chapter 6: Implementation**

## **Chapter 6: Implementation**

**6.1**    Insert Topic name

**6.2**    Insert Topic Name

**6.3**    Insert Topic name

## **Chapter 6: Implementation**





## References

- Hitaj, Briland et al. "PassGAN: A Deep Learning Approach for Password Guessing." *International Conference on Applied Cryptography and Network Security* (2017).
- Hitaj, Briland, et al. "Passgan: A deep learning approach for password guessing." *Applied Cryptography and Network Security: 17th International Conference, ACNS 2019, Bogota, Colombia, June 5–7, 2019, Proceedings 17*. Springer International Publishing, 2019.

# Appendix

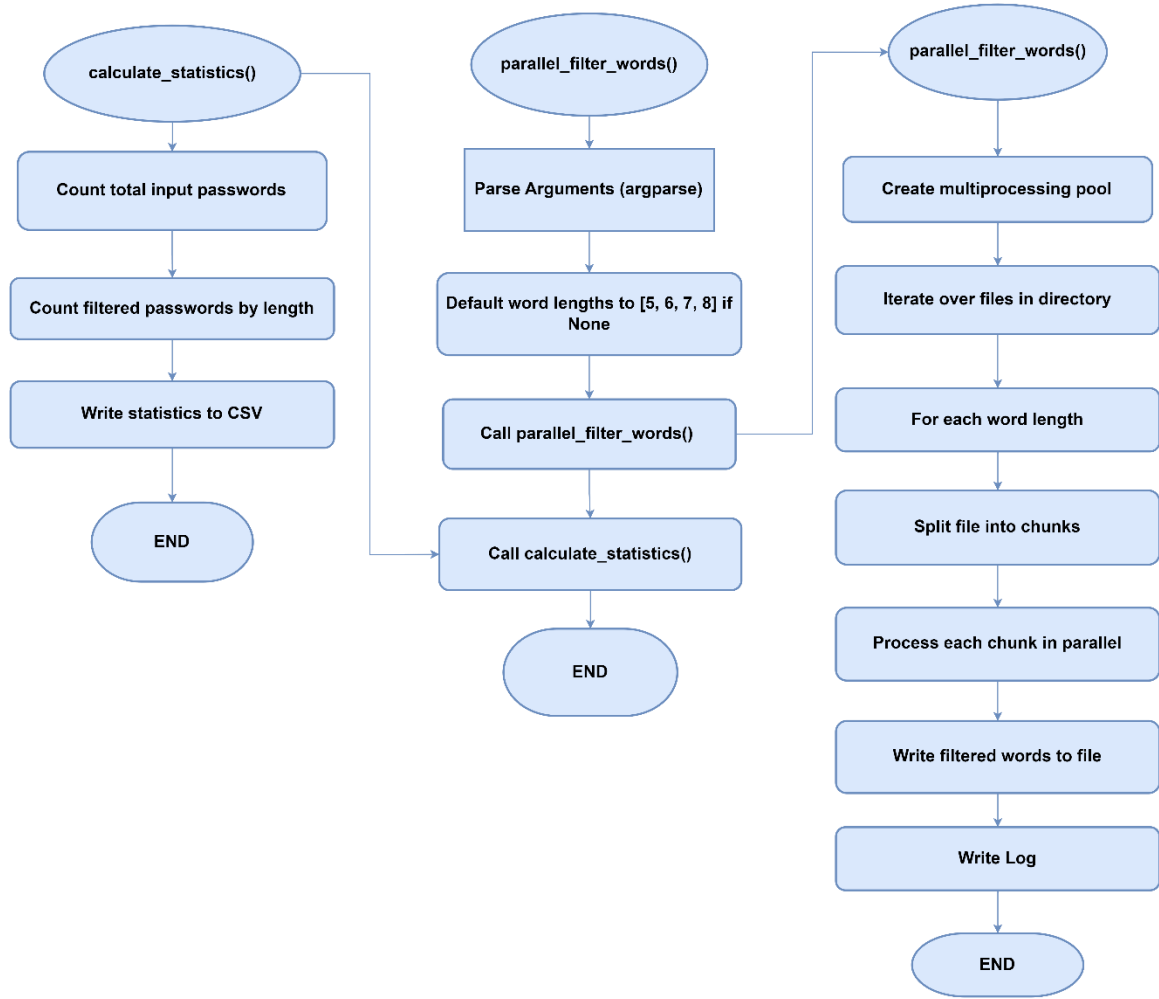
## Appendix A: Data Preparation Details

**Table A.1: Collected Datasets**

No	Name	Size
1	rockyou2021.txt dictionary from kys234 on RaidForums	12.7 GB
2	36.4GB-18_in_1.lst_2.7z	4.50 GB
3	ASLM.txt.7z	127 MB
4	b0n3z_dictionary-SPLIT-BY-LENGTH-34.6GB_2.7z	3.29 GB
5	b0n3z-wordlist-sorted_REPACK-69.3GB_3.7z	9.07 GB
6	bad-passwords-master.zip	1.34 MB
7	crackstation.txt.gz	4.19 GB
8	dictionaries-master.zip	19.3 MB
9	Password lists.zip	336 MB
10	password-list-main.zip	291 MB
11	password-lists-master.zip	8.86 MB
12	pastePasswordLists-main_2.zip	54.6 MB
13	PowerSniper-master.zip	0.3 MB
14	pwlist-master.zip	8.02 MB
15	rockyou.zip	41.7 MB
16	SecLists-master.zip	554 MB
17	statistically-likely-usernames-master.7z	9.07 MB
18	vietnam-password-lists-master.zip	5.14 MB
19	wpa-passwords-master.zip	5.79 MB
20	WPA-PSK WORDLIST 3 Final (13 GB).rar	4.49 GB
21	cyclone.hashesorg.hashkiller.combined.7z	6.59 GB

**Figure A.1: Dataset Downloaded Screenshot**

## Appendix B: Data Filtration



**Table A.2: Filtered Passwords Count**

<b>Total number of Input passwords</b>	8459063135
<b>Length</b>	<b>Total Count of Filtered Passwords</b>
5	10,751,871
6	563,608,354
7	465,597,114
8	1,357,729,013

## Appendix C: Data Splitting

### Pseudocode: Data Splitting

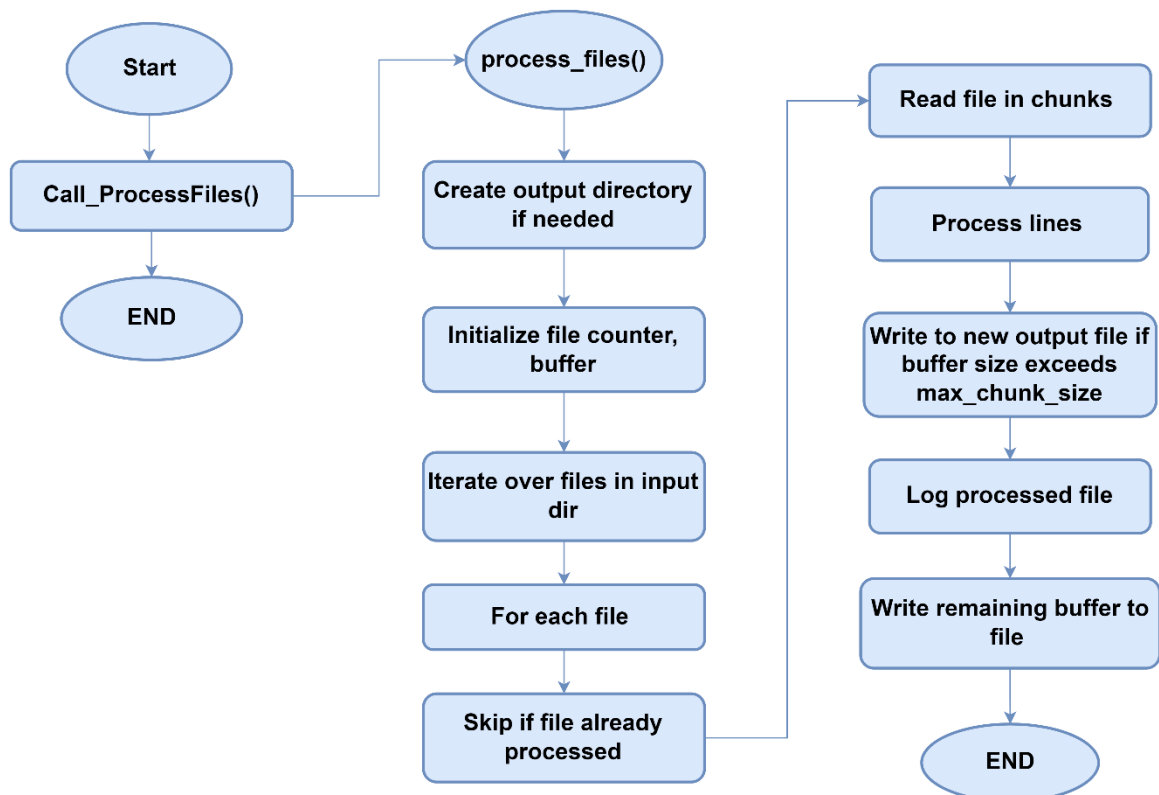
```
DataSplitter.py Psuedocode Data Splitter
Psuedocode Data Splitter
1  START Main Script
2    Set input directory
3    Set output directory
4    Set log file path
5    Set max chunk size (default: 2GB)
6
7    CALL process_files with input directory, output directory, log file, and max chunk size
8  END Main Script
9
10 FUNCTION process_files(input_dir, output_dir, log_file, max_chunk_size)
11   IF output directory does not exist
12     Create output directory
13
14   Initialize file_counter to 1
15   Initialize buffer as empty list
16   Initialize current_size to 0
17
18   FOR each file in input directory and its subdirectories
19     Get full input file path
20     IF file has already been processed (check log file)
21       Continue to next file
22
23     Get input file size
24     Open input file and initialize progress bar
25
26     WHILE reading chunks from input file
27       Read chunk up to max_chunk_size
28       IF no more data in chunk
29         Break loop
30
31       FOR each line in chunk
32         TRY
33           Add line to buffer
34           Increment current_size by line length + 1 (for newline character)
35           IF current_size exceeds max_chunk_size
36             Write buffer to new output file
37             Increment file_counter
38             Clear buffer
39             Reset current_size to 0
40         EXCEPT error
41           Print error message
42           Continue to next line
43
44       Update progress bar with chunk size
45       Free memory after processing chunk
46
47     Log processed file in log file
48
49   IF buffer is not empty
50     Write remaining buffer to new output file
51 END FUNCTION
```

```

52
53 ∨ FUNCTION create_new_output_file(output_dir, file_counter)
54     Create output file path using file_counter with zero padding
55     Open output file for writing in UTF-8 encoding
56     RETURN output file and its path
57 END FUNCTION
58
59 ∨ FUNCTION log_processed_file(log_file, input_file_path)
60     Open log file for appending in UTF-8 encoding
61     Write input file path to log file
62     Close log file
63 END FUNCTION
64
65 ∨ FUNCTION is_file_processed(log_file, input_file_path)
66 ∨     IF log file does not exist
67     |     RETURN False
68
69     Open log file for reading in UTF-8 encoding
70     Read all processed file paths
71     Close log file
72
73 ∨     IF input file path is in processed file paths
74     |     RETURN True
75
76     RETURN False
77 END FUNCTION

```

### Flowchart: Data Splitting



## **Github**

<https://github.com/Zeshankhan03/Multi-Purpose-AI-Password-Analysis-Tool>