BSCS-S25-001

03-134221-025 Muhammad Hassan Arshad

03-134221-022 Muhammad Dilshad Ghauri

OnionOwl: An Autonomous Crawler for Mapping Onion Sites

In partial fulfilment of the requirements for the degree of

Bachelor of Science in Computer Science

Supervisor: Dawood Akram

Department of Computer Sciences

Bahria University, Lahore Campus

January 2026

C e r t i f i c a t e

We accept the work contained in the report titled

"OnionOwl: An Autonomous Crawler for Mapping Onion Sites"

written by

Muhammad Hassan Arshad

Muhammad Dilshad Ghauri

as a confirmation to the required standard for the partial fulfilment of the degree of Bachelor of Science in Computer Science.

Approved by:

Supervisor: Dawood Akram

_____ (Signature)

January 05, 2026

DECLARATION

We hereby declare that this project report is based on our original work except for citations and quotations which have been duly acknowledged. We also declare that it has not been previously and concurrently submitted for any other degree or award at Bahria University or other institutions.

Enrolment	Name	Signature
03-134221-025	Muhammad Hassan Arshad	
03-134221-025	Muhammad Dilshad Ghauri	

Date : January 05, 2026

Specially dedicated to
my beloved grandmother, mother and father
(Muhammad Hassan Arshad)
my beloved grandmother, mother and father
(Muhammad Dilshad Ghauri)

ACKNOWLEDGEMENTS

We would like to thank everyone who had contributed to the successful completion of this project. We would like to express our gratitude to my research supervisor, Mr Dawood Akram for his invaluable advice, guidance and his enormous patience throughout the development of the research.

In addition, We would also like to express my gratitude to our loving parent and friends who had helped and given me encouragement.

Muhammad Hassan Arshad
Muhammad Dilshad Ghauri

OnionOwl: An Autonomous Crawler for Mapping Onion Sites

ABSTRACT

Orion Intelligence needs advance capabilities to automate the classification and extraction of actionable intelligence out of the hidden network where leak sites contain enormous volumes of unstructured and inconsistent HTML information. The existing web scraping, web classification and NLP tools developed for the clear web are unsuitable because of a non-standardized and fragmented structure of hidden network sites, as well as their active masking. In order to overcome this issue, we built a specialized pipeline of structured leak onion sites classification and data extraction, which comprises data gathering, cleaning and normalization in addition to fine-tuning of the model that is particular to the dark web leak sites.

This project discusses the use of machine learning based onion site classification and transformer-based NLP models in HTML-to-structured-information extraction that provided an balance between input capacity, model size and extraction accuracy. Sites classified as leaks were cleaned, cleaned HTML data were converted into a specialized JSONL data format and data normalization was applied so that NLP model training could be compatible with the data. The model was optimized to detect and harvest important entities in the form of titles, dates, web links, records and other appropriate metadata of various dark web resources.

The system is very accurate in converting unstructured HTML from hidden networks to structured and analyzable formats. This module improves the functionality of OSINT, reduce the manual work and assist downstream processes like threat prediction and analysis. It is a framework that is both modular and scalable and can be improved in future as per the further requirements of ***Orion Intelligence*** and can be modified to suit the needs of emerging challenges in dark web monitoring.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS / ABBREVIATIONS

API	Application Programming Interface
BERT	Bidirectional Encoder Representation from Transformers
CNN	Convolutional Neural Network
CSS	Cascading Style Sheets
CSV	Comma-Seperated Value
DOM	Document Object Model
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
JSON	JavaScript Object Notation
JSONL	JavaScript Object Notation Line
JS	JavaScript
LLM	Large Language Model
ML	Machine Learning
NER	Named Entity Recognition
NLP	Natural Language Processing
NLTK	Natural Language Toolkit
OSINT	Open Source Intelligence
PII	Personally Identifiable Information
SVM	Support Vector Machine
SMOTE	Synthetic Minority Oversampling Technique
TF-IDF	Term Frequency-Inverse Document Frequency

CHAPTER 1

INTRODUCTION

1.1 Background

Each day, a huge amount of information is produced and shared online. While search engines help access the surface web, most of the internet, especially the deep and dark web contains data that is not indexed in the usual way. This data includes business listings, leak repositories and content about data breach, much of which is difficult to access, find, analyze or organize. The traditional method of visiting each website through a browser or downloading content from various sites takes too much time, has a high chance for error and is practically impossible due to the huge volume and variety of information available online.

Genesis Technologies is a prominent security operations and threat intelligence solutions provider in Pakistan. The company has designed and developed its leading product, named ***Orion Intelligence***, that various national and international enterprises use for threat monitoring and cyber risk exposure. In the course of my industry collaboration with ***Genesis Technologies***, the company identified a significant gap in ***Orion Intelligence***, specifically on its lack of automated and intelligent crawling module that is capable of obtaining structured information from hidden networks, specifically onion sites that regularly present breach samples and leaked datasets.

Research has shown that well designed data scraping and processing tools can significantly improve quality of data. Moreover, it reduces manual efforts and help

generate useful insights. For example, structured extraction of metadata such as titles, dates of publication, link and content enable effective organization, filtering and analysis of data. In addition, data labeling and grouping can be automated by applying machine learning to categorize content. This is extremely helpful with a dataset that is scraped from a variety of web-based platforms. While these pipelines are still useful, developing stable pipelines to tackle issues, such as imbalanced categories of content being scraped, standardized HTML structures for each website and access to masked networks are significant barriers to development.

This project will focus on addressing this issue, by developing an autonomous crawler module that will be incorporated with ***Orion Intelligence***. The module will automatically categorize onion sites, crawl them, scrape them and extract structured content that is related to leaks. This autonomous crawler module will provide high value enhancements to ***Orion Intelligence's*** mission of enabling rapid and actionable intelligence on emerging data leaks by automatically fetching leaked data as soon as it becomes publicly available**,** which can then be utilized by companies, agencies and law enforcement authorities for timely action and mitigation.

1.2 Problem Statements

The key challenge with ***Orion Intelligence*** is that capturing and sorting data from onion sites takes an extensive amount of manual effort. Analysts are required to visit pages for identifying content and parsing webpages, identifying and deleting unwanted data and removing duplicates, before they begin the process of sorting the data into categories and extracting it. This takes time, involves human errors and results in inconsistencies due to the diversity in page structures and dynamically generated content. To overcome these manual efforts, this project builds a full end-to-end automated pipeline that can classify, parse and sort data from the onion sites. The pipeline will ensure there is less manual work and less human errors resulting in clean and structured datasets that will be used for research, monitoring and threat intelligence purposes within ***Orion Intelligence***.

1.3 Aims and Objectives

The objective of this project is to develop a workflow which is designed for automated web classification and extracting useful information. Our goal is to minimize the manual efforts and fetch breaches in a structured way that are ready for analysis or decision-making.

The objectives of the project are as follows:

i) To utilize machine learning and natural language processing techniques to classify sites and extract important data.
ii) To build an efficient web crawler that can crawl a variety of different sites and access hidden networks such as Tor.
iii) To fetch the important content such as titles, links, content, images, records and other metadata, while removing unnecessary content and duplicate records to ensure consistency.

1.4 Scope of Project

The scope of the project is to develop a system that classifies onion sites and extracts meaningful insights from them, in an automated manner. This system will take unstructured HTML from onion sites and extract useful information from it. The goal is to automate the crawling process of onion sites and reduce much of the manual time and efforts used to identify and organize data from private networks. The system will incorporate machine learning to classify sites and natural language processing to identify and extract relevant and important content from onion sites, thus removing the manual labeling.

CHAPTER 2

LITERATURE REVIEW (and/or SRS)

2.1 OnionOwl: Overview

OnionOwl is a module of ***Orion Intelligence*** and it is designed to increase ***Orion Intelligence*** ability in automated leak data extraction and structured generation of clean, usable data. While the broader Orion system is concerned with OSINT-based monitoring, analysis and alerting, this component is oriented towards one of the most time-intensive challenges of the OSINT team which includes extracting clean, structured and human-readable information out of highly inconsistent leak sites of the dark web [1]. Its irregular HTML structures and the frequently changing designs resulted in the manual extraction being slow, error-prone and non-scalable [2]. To address these issues, our project proposes an NLP-based extraction engine that can convert raw HTML of leak-site to valuable information. To accomplish this, we constructed two datasets by ourselves, one for classification and other for important data extraction from raw HTML. The data set was created from hidden-network leak sites so that it would be fully compatible with the setting in which ***Orion Intelligence*** is operating.

2.1.1 Existing Solutions

There are solutions for this type of problem but currently there is no effective end-to-end solution that can be incorporated into ***Orion Intelligence*** to make automated

process of extracting structured data from dark-web leak sites. The majority of available studies and tools work with Clearnet datasets, where websites follow predictable layouts, clean HTML and normalized metadata. Such techniques fail to work on hidden-network targets, where HTMLs are not uniform, tags frequently have been fragmented and content has deliberately been hidden. Due to this, we have created a completely customized pipeline, including dataset creation and cleaning, normalization and model fine-tuning, which is specific to dark-web leak pages and the business requirements of ***Orion Intelligence***.

2.1.2 Limitations of Existing Solutions

Existing solutions for dark web intelligence suffer from several critical limitations that prevent their direct integration into ***Orion Intelligence*** for automated structured data extraction from leak sites. Most existing crawlers and analysis frameworks are either purpose-specific, limited to surface web environments or require substantial manual configuration, as seen in systems like CRATOR, DIDECT2S and other Selenium-based Tor crawlers, which struggle with highly irregular HTML, fragmented tags, dynamic content and anti-scraping countermeasures. Deep learning approaches in PII detection and dark web classification primarily operate on clean, curated datasets and do not address the severe noise and structural inconsistencies typical of leak-site HTML [3]. Even advanced NER and LLM-based systems such as UniversalNER, GLiNER or LLM-driven threat intelligence pipelines require large annotated datasets [4]. Additionally, most current frameworks focus on classification or entity extraction in marketplaces, forums or generic onion services, not on transforming raw, unstructured leak-sites into reliable, normalized and structured OSINT outputs. As a result, no existing end-to-end model or pipeline can be directly adopted for Orion Intelligence, making it necessary to design a fully customized workflow.

2.2 Existing Research and Work

Recent studies have paid significant attention to automatic detection of Personally Identifiable Information (PII) within unstructured text, highlighting the performance of different machine learning techniques. The rule-based models such as Regex, classic classifiers such as SVM, Random Forest and Conditional Random Fields and current deep learning models such as LSTMs and Transformers such as BERT were explored in one paper. The results have shown that deep learning models were more accurate and recalled better than the conventional ones, but they were computationally costly and required very large training datasets. The proposed study workflow was also a structured process of detecting PII that included preprocessing and tokenization, named entity recognition, confidence scoring, and downstream action modules, including masking or alerting [3].

The other significant contribution in the literature is developing an overall and scalable framework of dark web analysis, which align with the fact that most of the current tools are purpose-specific and might need to generate the software from scratch. The microservice architecture proposed in the case is a three-layer. These layers are control, logic, and operations. The Control Layer deals with infrastructure and API requests, the Logic Layer deals with particular workflows like scraping websites or classifying forums and the Operations Layer deals with autonomous activities like web crawling or port scanning. It uses open-source technologies such as Docker Swarm to create an orchestration, Apache Kafka, asynchronous messaging, PostgreSQL persistence, and the ELK Stack, which is centralized logging and monitoring. Validation was done at the expense of scraping Tor onion services, extracting the titles, description, tags and language metadata of onion sites, and had a very impressive scale with more than 78,000 onion services and over half a million domains scraped [4].

In another study, the authors present CRATOR, a Tor network crawler that attempts to convert dark web pages into actionable Open-Source Intelligence (OSINT). CRATOR is the solution to the problem of dark web crawling that is the lack of structure and randomness of the content. The system uses a breadth-first crawler policy, an integration of seed URLs and link analyzer and the use of proxies, user-agent switching and cookies to make the system efficient and offer anonymity. It can also

bypass simple security systems such as the use of login forms and simple CAPTCHA and both automated and manual session management is supported [1].

The next piece of research is an example of a deep-learning-based search engine that was created with the purpose of illicit activity detection and classification on the dark web. The authors note that the dark web is hard to analyze, with anonymity allowing mass crime and offer a system according to which automatic categorization of material into five groups is made: the trading of drugs, the trade of weapons, stolen bank cards, fake identity documents and illegal money. Their system is a mixture of image and text-based classification. Onion websites images are processed with CNNs, assisted by a transfer-learning-enabled ResNet50 backbone, and text-only websites are processed through NLP using KeyBERT and cosine similarity are used to label each category. The system is highly performing and this proves it has potential capacity to increase greatly in automated detection of illegal activities and minimizing manual work in law enforcement agencies [5].

The other important contribution that has been made in the field is devoted to the systematical review of the already existing dark web crawlers and the creation of a Tor-based crawling model that is optimally suited for digital investigations. The authors conducted a Systematic Literature Review of 34 studies and discovered that a majority of studies make use of custom-built crawlers, with Python becoming the chosen language and Selenium and Scrapy being the most used libraries. Based on these observations, they created a new crawler as part of the Digital Detectives Comprehensive Tor Toolset (DIDECT2S) which they developed using a modified Selenium driver to use the Tor browser and behave as a human operator to maintain forensic integrity. Through experimental analysis, the crawler was shown to be a useful tool in scraping the content of large dark markets, including pages with CAPTCHA and authentication, but it was significantly slower because of Tor intrinsic latency. In spite of this, an analysis of cosine similarity revealed that the content scraped was very similar to the clear-web equivalents which proved the validity of the crawler [2].

The other recent work is the study of how Large Language Models (LLMs) may be used to assist law enforcement in monitoring and analyzing the Dark Web, which would help overcome the problem of the rapidly increasing volumes of hidden,

decentralized and criminally oriented content. The authors compared eight state-of the art LLMs offered by the major providers on three ethically sourced Dark Web datasets, DUTA-10K, CODA and Nemesis Marketplace to compare their performance over the main threat-intelligence tasks. The findings revealed that LLMs had good results in classification and domain specific tagging. Nevertheless, activities that demand a higher level of reasoning, finer-level differences or strategic interpretation were more challenging, which shows the shortcomings of existing models. On the basis of these findings, the authors suggested a modular architecture based on LLM (which serves as a key component of real-time Dark Web intelligence extraction) that combines TOR-compatible data ingestion with multi-agent LLM pipelines and automated threat alerting [6].

Another research is aimed at automating the extraction of the key information in Darknet Markets to aid law enforcement to overcome the shortcomings of the manual data collection method, which is time-consuming, labor-intensive and subject-to-error. The authors compared Named Entity Recognition (NER) three state of the art models (ELMo BiLSTM, UniversalNER and GLiNER) on extracting complex entities like a product name, prices, vendors, categories, stock, and views on DNM product listings. To alleviate a lack of data, they proposed a scalable data collection structure and created a novel collection of annotated data, DNMsListing. A specialized dark web crawler with SOCKS5 proxies, randomized delay and anti-scraping mechanisms was used to conduct crawling which allowed access to the protected marketplaces with high dependability. It was experimentally evaluated that LoRA and k-bit quantization with PEFT techniques resulted in a significant improvement. The best results were obtained with fine-tuned UniversalNER-7B. Tests of Robustness in an unseen market showed high Precision but lower Recall and demonstrate that the model is sensitive to changes in the domain and it is difficult to generalize across heterogeneous darknet settings [7].

CHAPTER 3

DESIGN AND METHODOLOGY

3.1 Development Method

For this project, an Agile Software Development Life Cycle (SDLC) was employed as our development methodology. Due to the complexity of the system, that includes web feature extraction, HTML cleaning, parsing, machine learning based binary classification and natural language processing-based information retrieval, database integration and front-end display, we chose Agile. This approach allowed the team to use iterative development and deliver functionality in short cycles while allowing flexibility for changing when changes were required. By using Agile, each module of the system was developed in stages to enhance functionality and develop the efficient system to achieve the required outcome.

3.1.1 Why Agile Was Chosen?

Due to the iterative nature of Agile, this methodology was chosen as it supports extensive and quick development activities while modules are being developed, utilized and refined based on the feedback. For projects that required step by step development, Agile is used as it ensures that the planned modules such as web scraping, HTML parsing, NLP model building and front-end integration can be developed, tested and improved independently. Using Agile, the team build and test system modules while receiving quick and effective feedback. Furthermore, team developed

a product in small manageable pieces which ultimately reduce the risk of technical failure and errors and enable developers to timely incorporate the feedback into the final product.

3.2 Tools and Technologies

The project lifecycle is dependent upon various tools and technologies that provide support to a range of functions like web scraping, clearing raw data, parsing structured data, classifying the data, storing the data and integrating data into the frontend of the pipeline. Each function of the system will require specific capabilities including, working with a dynamic web environment, unstructured HTML, training machine learning models and data management. This project involves multiple technological components in order to ensure the functionality of the project. Together these tools construct the components of the automated pipeline so the system will extract, cleanse, store and produce meaningful information in an efficient and scalable manner.

3.2.1 Programming Language

Python is the backbone of this project because is offers a flexible environment for smooth management of multiple tasks. It is used to scrape onion sites, as well as preprocessing of data and training and evaluation of machine learning and natural language processing models. Asynchronous web scraping, HTML parsing and data cleaning are also determined by python, which means that the system is able to efficiently fetch and prepare data from different websites for further use.

3.2.2 Web Building Technologies

- **HTML:** It is used to define structure of the web application.

- **CSS:** It is used to style web pages by adding layouts, colors, fonts which enhance the overall appearance of the application.
- **JavaScript:** It is used to add functionality and interactive features into the application.
- **Flask:** It is a python backend framework used to serve APIs, manage client requests and connect the frontend with backend.
- **MongoDB:** It is a NoSQL database which provides a reliable and flexible storage solution for the application. It is used to store the information predicted by the NLP model.

3.2.3 Web Scraping Technologies

- **Request and urllin3:** Web pages are fetched using requests and urllib3 which can handle both, a normal as well as a secure URL. They are reliable to manage network requests. They also including retries to send request and validate URL. This ensure pages are accessible for further processing.
- **BeautifulSoup (bs4):** It is used to parse HTML content and extract structured information which makes the content ready for further process.
- **Playwright:** It is used to automate the accessing of hidden networks and the sites having JavaScript-rendered content. Playwright is selected because it is reliable, efficient and works great for crawling. In this project it runs in a headless way. Playwright is an appropriate choice of a reliable web scraping solution due to the ability to customize timeouts and retries.
- **lxml and XPath:** These two technologies are used to converts HTML into a tree like structure of data, which allows the system to find and extract specific content efficiently.

3.2.4 Data Processing and Analysis

- **CSV:** It is used to store and share structured datasets, which consist of sites features, input/output labels and textual keywords along with the frequency they appear. This format provides a clean and standardized structure for storing input/output pairs, making it easy to use for different parts of the system to load, read and process data consistently.
- **OS:** This module to manage and organize project files as it interacts with the file system to make sure that datasets, models and outputs are saving in the correct locations and are easy to locate.
- **Argument parsing module:** Argparse is used in this pipeline to allow the input through command line parameters, instead of hard-coding values into the script.
- **Pandas:** It is used throughout the pipeline to accommodate tabular data and ease the process of loading, updating, cleaning and preprocessing datasets for feature extraction and classification.
- **Collections:** This module is take part in abstracting and analyzing patterns inside the HTML content like identifying frequently appearing elements.
- **JSONL:** The main format used to store cleaned and parsed HTML chunks for this project is called JSONL (JSON Lines) format, where every line forms its own data element. This format is efficient while reading and writing large amounts of data that are used for training and analysis of NLP models.

3.2.5 Text Processing and Feature Engineering

- **Re (Regular Expression):** It is used in this project to detect patterns in text and extract relevant keywords. Moreover, it also used to remove unnecessary or irrelevant content from HTML.
- **Nltk stopwords module:** It provides a list of common English stopwords which enables the system to filter out words that do not contribute meaningfully to data classification.

- **minify_html:** It is used in this project to clean and compress raw HTML content. By removes unnecessary white spaces, useless formatting tags and other irrelevant scripts minify_html keeps meaningful content which is suitable for parsing.
- **Imblearn:** It provides methods to balance classes in datasets where certain classes are less in number. This improving model performance and reducing biasness. It is used in the binary classification of onion sites.

3.2.6 Netowrking and Proxy

This project uses a Tor SOCKS Proxy, which ensures that each web request is passed through the Tor network. The Tor network will provide anonymity and a secure tunnel to the system so that system can access to hidden and restricted sites. Thus, it is relatively easy to scrape onion site data while protecting systems privacy and the masking of IP address.

3.2.7 Artificial Intelligence Techniques

- **Scikit-learn:** Used for training classical machine learning models for text classification.
- **Label Encoding:** Converts textual category labels into numeric format for machine learning models.
- **TF-IDF Vectorization:** Converts text from website keywords into numerical feature vectors. Captures word importance across the corpus for classifier input.
- **Joblib and from_pretrained by Hugging Face:** Serializes trained models and preprocessing objects for later use. Allows easy loading of trained classifiers in the Orion Intelligence system.
- **Evaluation Metrics:** Helps in selecting the most effective model for integration into Orion Intelligence.

- **PyTorch:** Core deep learning framework used to build, train, and run the FLAN-T5 model.
- **Transformers by Hugging Face:** Provides the pre-trained FLAN-T5 model and tokenizer. Enables sequence-to-sequence text generation for extracting structured information from unstructured HTML.
- **Datasets by Hugging Face:** Converts JSONL data into Hugging Face Dataset objects for seamless training and evaluation. Facilitates efficient batching, tokenization and integration with Seq2SeqTrainer.
- **Seq2SeqTrainer and Seq2SeqTrainingArguments:** High-level API to handle training, evaluation, logging, and checkpointing for sequence-to-sequence models. Manages hyperparameters like epochs, batch size, learning rate, and gradient accumulation.
- **T5 Tokenizer:** Converts input HTML and output JSON into token IDs that the model can understand.
- **Data Collator for Seq2Seq:** Dynamically pads inputs/outputs to the same length in a batch.
- **Evaluation Metrics (ROUGE, BLEU):** Measure similarity between generated output and ground truth for sequence-to-sequence tasks. ROUGE evaluates n-gram overlap. BLEU evaluates exact token match accuracy.
- **Hugging Face accelerate:** Handles device placement and mixed precision for optimized training across GPUs. Reduces memory usage and accelerates model training.

3.3 Development Environment

- **Git:** It is a version control system which helped monitor and manage changes to the project over time. The collaboration and version tracking was done easily using git.
- **GitHub:** It is a remote repository hosting service that supports the collaborative development. It also includes code backup and makes code publicly available to the team.

- **PyCharm:** It was used to write, edit, debug and organizing python code. It helps maintain a smooth and well-structured development workflow.
- **Kaggle:** It was used to train the natural language processing models. Additionally, it was used to maintain datasets for experimentations and to explore many different techniques to train the models within the online notebook environment.
- **Google Colab:** It also assisted in developing models by providing a free cloud-based notebook environment with GPU capabilities. Free GPUs made experimenting fast. Moreover, using GPU made prototyping easy and training scalable.

3.4 System Architecture and Design

The architecture of the autonomous leak crawler is overall structured to develop an automated system that is robust, scalable and reliable. A system which is capable of classifying, processing, extracting useful information and storing that information. This system has multiple modules working together across the entire process, extracting unstructured information from the onion sites and transforming the information into usable content. Each module surves a particular purpose and adds value to the entire workflow. The system is intended to process the information in real time and convert raw HTML into useful, structured data. The entire system consists of layers and components to maintain flexibility and ease of maintenance throughout the architecture.

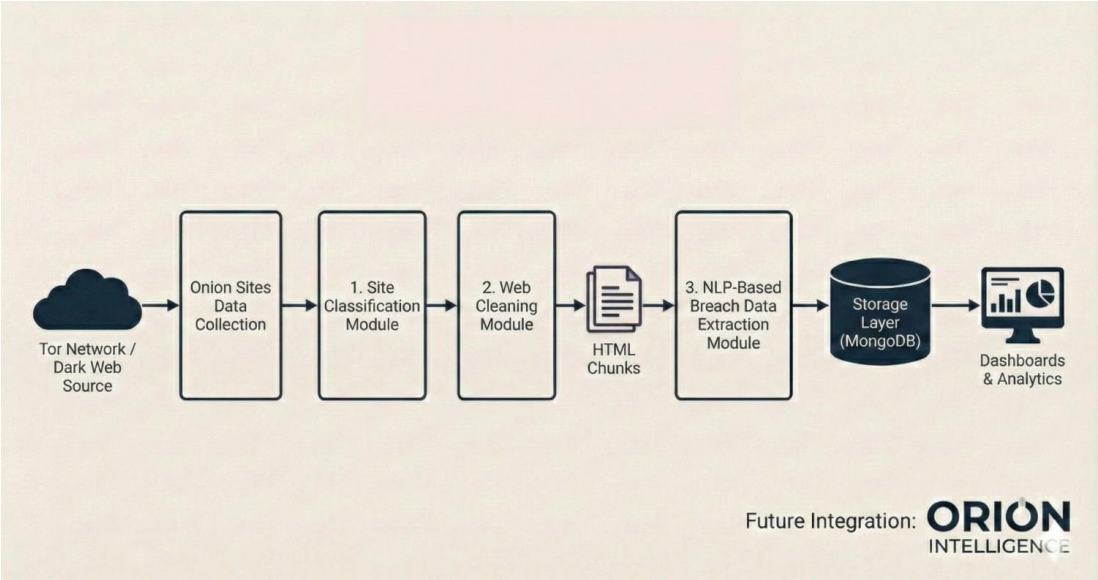


Figure 3.1: System Architecture Diagram

3.4.1 Overview of System Architecture

This project is built in a modular, end-to-end automated pipeline. The system is designed to classify, clean, crawl and store data from onion sites. The system architecture converts unstructured web content into structured, useful data while minimizing human effort and errors. The system is arranged in a client-server environment with pipelines as the main engine supported by components for storage and the frontend functionality.

Layers of the system includes:

i) **Presentation Layer (Frontend):** Allows the user the to interact with structured data, view crawled records and review extracted metadata. This layer acts as a web platform for presenting real-time outcomes.

ii) **Business Logic Layer (Processing Pipeline):** Implements the core steps of the project like:

- **Site Classification:** Classifies onion sites according to their content type.
- **Data Cleaning:** Remove irrelevant content such as ads, styles, scripts and other unnecessary HTML tags.

- **Web Crawling:** Extract important information from the cleaned HTML.
- **Data Storage:** Extracted information is then stored in the database

iii) **Backend Layer:** It defined the workflow of the application. It helps in the communication between the pipeline modules and frontend. It handles client request. Moreover, it stores, retrieve and update data in the database effectively.

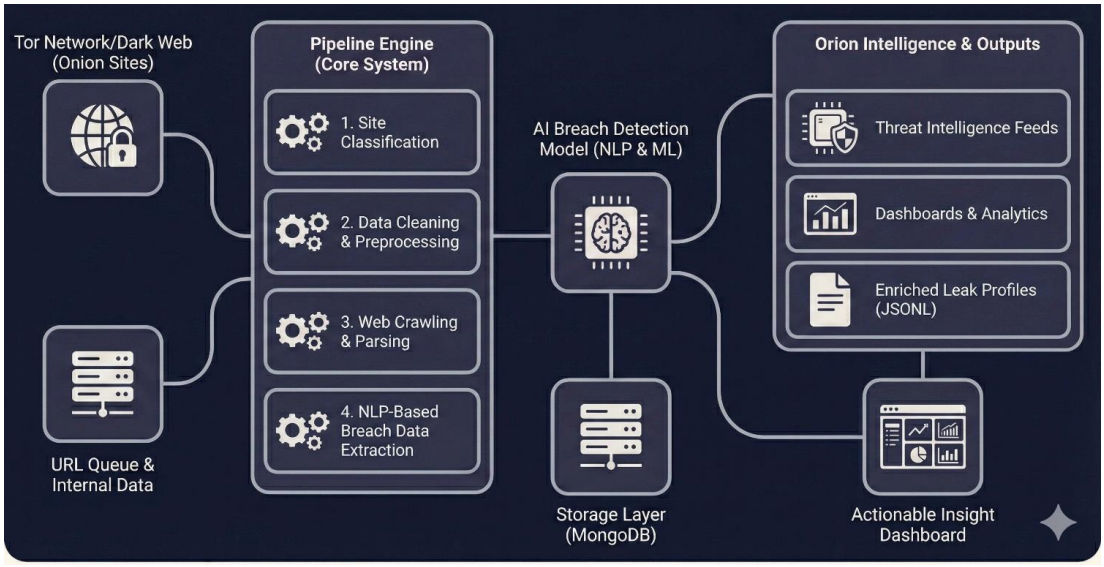


Figure 3.2: Complete Pipeline Design

3.4.2 Interaction of Components and Data Flow

The autonomous crawler follows a seamless pipeline, which ensures the effective extraction, processing and storing of information in a reliable manner. The crawler is purposely designed to tackle issues associated with the extraction of unstructured web content. The overall systematic approach is designed to classify, clean and crawl all information from raw HTML. This system provides the data ready to be used for analysis purpose by different organizations and authorities.

The process begins with the classification module which classifies onion websites as leak or other. These onion sites are analyzed through machine learning to classify them according to their class. The sites that are classified as leak will continue in the sequence of the process. Once the site is classified, the relevant URLs are sent

to the cleaning module. In cleaning module, the HTML of those classified leak sites are processed to remove unnecessary elements such as scripts, advertisements, banners, pop-ups, timers and duplicate blocks. This cleaning step ensures that only meaningful, noise-free and contextually relevant content is kept. This step creates a clean foundation for data extraction and further data processing.

After cleaning is done, the system proceeds to the next step which is web crawling, where each page is crawled using our natural language processing model and the site's meaningful and important information is extracted. Once information is extracted, that structured outputs move to the storage and integration layer, where each record is stored in a database. This step allows systems to safely access clean and structured data, which will be used by various authorities, companies and law enforcement agencies. This guarantees consistency, avoids duplication and makes it easier to manage information.

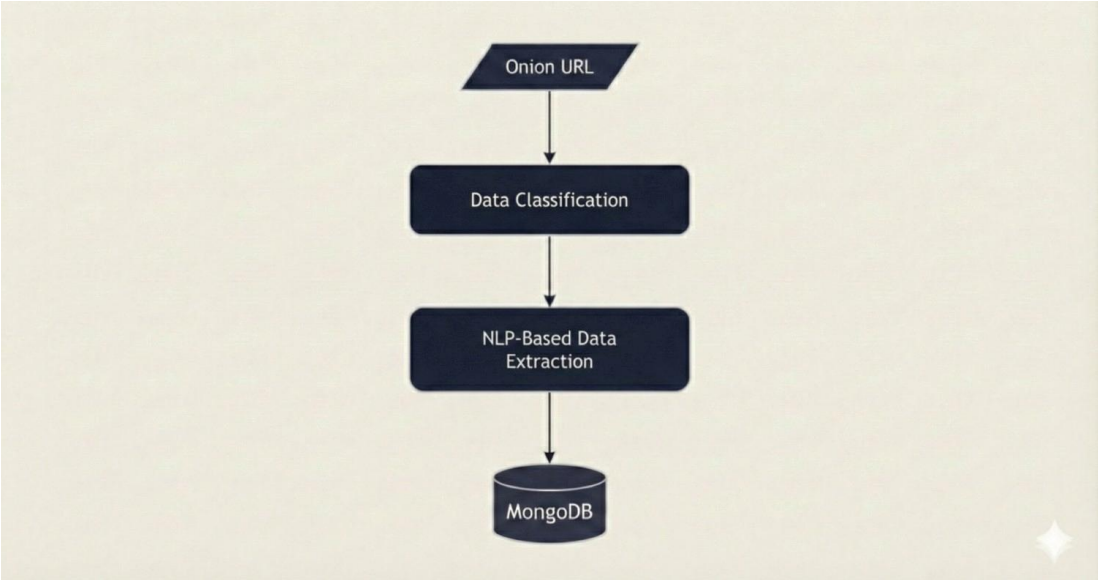


Figure 3.3: Data Flow Diagram

3.5 Use Case Diagram

The use case diagram illustrates how different actors interact with the Orion Intelligence system, highlighting all major functionalities.

Figure 3.4: Use case Diagram

3.5.1 Use Cases

Table 3.1: Use Case 1

USE CASE ID	**UC-1**
USE CASE NAME	**Onion Site Classification and Breach Extraction**
Actors	**Admin / Bot**
Description	The system will allow admin / bot to enter the onion url and rest of the work will be done by pipeline.
Trigger	User will be using it through the *OnionOwl/Orion Intelligence* UI.
Precondition	Site should be a valid onion site.
Postcondition	Important data from leak sites will be fetched.
Normal Flow	1. Open UI 2. Enter the URL
Assumptions	User must understand English language.

Table 3.2: Use Case 2

USE CASE ID	**UC-2**
USE CASE NAME	**View Extracted Breach Information**
Actors	**User**
Description	The User accesses our platform to search and read the breach data that has been processed by the pipeline.
Trigger	User will be using it through the *OnionOwl/Orion Intelligence* UI.
Precondition	Data has been successfully stored in MongoDB.
Postcondition	The user is presented with a visual interface showing potential data leaks.
Normal Flow	1. Open UI 2. Search for data
Assumptions	User must understand English language.

CHAPTER 4

DATA AND EXPERIMENTS (and/or IMPLMENTATION)

4.1 Data Collection for Binary Classification

The first step in constructing this crawler was to create a dataset of onion sites that is both diverse and comprehensive to train the classification model. We have to created our own dataset because there was no standardized dataset available for onion sites. The data was all manually collected from various open-source sites. The objective of this step was to gather a large collection of onion sites, which contain both leak related and onion sites like marketplace forums etc. This vast collection of onion sites enables the classifier to identify valuable distinguishing features between the two classes.

The data collection phase began with exploring several different publicly accessible GitHub repositories that monitor onion services. Generally, these repositories contain a category for marketplaces, blogs, forums, ransomware leaks, scam sites and dead links. We searched multiple repositories to gather valid, non-duplicate and potentially useful onion URLs for model building and classification. Besides GitHub repositories, we also searched Ahmia, a well known Tor search engine that indexes onion sites. Ahmia's public database facilitated searches with keyword like "leaks," "breach," "ransomware," "data dump" etc. This search engine also provide some more relevant sites which were not documented elsewhere.

In addition to these sources, the dataset was enhanced by manually browsing through onion directories, OSINT blogs, communities which monitors the hidden network, cybersecurity reports and ransomware intelligence trackers. This was to

assure that our links were not biased toward any particular class type, while still including traditional, non-leak websites like forums, hosting services, marketplaces, blogs, news articles, file sharing sites, and general-purpose sites. Once the onion sites were collected, they were consolidated into a single CSV file. Each row in the CSV represented one onion website and contained:

- The onion site URL.
- Manual Label: Leak or other.

The manual labeling part was essential as we needed clear examples for the model to learn. Leak sites were identified by a unique feature which confirmed that the site contain breaches, such as sites hosting stolen data, a breach announcement or sites publishing ransomware dumps or companies. All other sites were considered unrelated to data leaks. The manual labeling created the initial dataset, which was now prepared for feature engineering and other processes. This dataset served as the foundation of the full pipeline. It allowed the model to learn how a leak site behaves or looks like, assuring that only the relevant sites proceed to later stages like cleaning, crawling and structured data extraction.

4.2 Feature Extraction for Classification

The process of extracting features is an important step in the classification process. It takes the original website URLs and converts them into meaningful, structured features that can be used to train classification model. To begin, we only have a list of onion site URLs with manual labels, either leak or other. In order to train the model, it was required to extract features that contain the content and characteristics of each site. This step required to scrape web content, cleaning it up and extracting important keywords along with the number of times they appear on the site. These two elements serve as the features which will be utilized by machine learning models. In this section we run through the whole process of extracting features, including obtaining the raw HTML, cleaning the HTML, normalizing the text and then pulling out keywords based

on some frequency. Lastly, we will save these features in a CSV format for further processing.

4.2.1 Setting up Tor for Feature Extractor

To crawl onion sites, the system must send requests through Tor network because onion sites cannot be connected with a standard HTTP request. The only way to connect is to send requests to Tor via a Tor SOCKS5 Proxy. As a result of this system will create requests.Session() object and configure it to use a SOCKS5 Proxy located on the local machine. This ensures that any GET request will always go through Tor. We have also implemented retry strategies based on exponential back-off times and status codes. This allows the feature extractor to minimize the amount of time required to recover from failures. This enables the crawler to withstand the frequent issues associated with crawling onion domains, like the slowness of some services, dropped connections, short-term downtimes and the problem of inconsistent routing across the onion network.

4.2.2 HTML Retrival and Content Filtering

Once the feature extractor establishes a connection to a webpage, from there it is able to retrieve the webpage's raw HTML. This HTML typically contains some combination of the JavaScript code, inline CSS, ads, tracking components, hidden HTML, repetitious navigation elements, pop-up windows, overlays and several font and style tags. In order to remove all of this irrelevant material from the retrieved HTML, the feature extractor uses BeautifulSoup to perform structured filtering. It removes all elements related to script, style and no script tags, gathers all remaining text nodes, eliminated all multiple spaces into single space characters and decodes any HTML-encoded characters back into standard text. The final output after the filtering contains only human-readable, meaningful text, which is suitable for keyword

extraction as it converts a complex HTML webpage containing numerous menus, banners and scripts into a single block of clean, paragraph-like plain text.

4.2.3 Keyword Extraction and Normalization

The process of generating features from the extracted text involves several steps of data normalization. In order to avoid misclassification or duplicate classifications of words based on their case, for example, Data versus data, all visible text is first converted to lower case. Secondly, a specially defined regex pattern is applied to extract only those words that consist entirely of letters and contain at least four letters, thereby discarding any short word with one, two and three letter's words, URL identifiers, numbers, HTML tags and punctuation. The use of the regex pattern allows the process to identify and ignore many forms of noise, maintaining only meaningful tokens. The removal of stop words then takes place in order to eliminate very commonly occurring English words that adds no value to the classification. The primary source for stop words is based upon the NLTK library of stop words and in the absence of NLTK being present, a set of about thirty common stop words is used as a fallback. The combined use of both sources of stop word removal allows for consistent data preprocessing in any environment where this process occurs, allowing for the pipeline to produce reliable outputs whether it relies on NLTK or not.

4.2.4 Keyword Frequency Extraction

To obtain a clean list of relevant keywords, the extractor determines how frequently each term appears within the processed sites content. This is accomplished by creating a frequency matrix for each relevant term and sorting it so that terms are listed by their respective frequencies in descending order. After determining the top ten keywords by their frequencies, the extractor creates tuples containing each keyword and its corresponding frequency count. This step is important because when comparing the most frequently used words within websites that are related to leaks vs. those that are

not, we see that they often have a similarity in the repetition of many of these same words like breach, data, dump, stolen, gang, ransomware, publish. The distribution of keywords is quite different on normal blogs or marketplaces. By being able to determine these distributions of frequency of the keywords, we are able to train models to accurately identify leak sites from non-leak sites.

4.2.5 Handling Failure and Timeouts

The feature extractor is designed to handle variety of common issues that one may experience when interacting with hidden networks. To address this challenge, feature extractor has several error-handling features. Timeout mechanism is ensured by feature extractor that if request hangs while waiting for the response from the server, it will not run indefinitely. Any request that fails after a timeout is automatically reprocessed with a retry mechanism. In cases where an empty response is returned, that particular website will be skipped and will not be processed. All types of errors encountered by the feature extractor are logged without interrupting the flow of requests through the pipeline. Lastly, an additional time delay is added between each request in order to avoid overloading the onion servers. All these precautions ensure a robust extraction process capable of effectively extracting data from multiple URLs using long-running crawls.

4.2.6 Onion Site URL Normalization and Validation

As many sites located in the onion repositories are not correct, therefore, the feature extractor normalizes URLs before sending request to server and adds the http:// to any URL which does not have this prefix. The extractor also removes any whitespace from the end of the URL, which validates the domain by checking the URL structure and discard any URL that is invalid. By following these steps, the feature extractor is able to maintain a clean and working list of URLs for classification purposes and ultimately reduces any errors that might occur during the feature extraction process.

4.2.7 Structured Feature Output

Once everything is processed and features are extracted, rows in a structured CSV file contain information about individual URLs and their associated features. Each row in this file contains three pieces of information:

- The URL, whether it was labelled as a leak or otherwise.
- A list of the extracted keywords associated with it.
- Frequency of those keywords

With the usage of pandas, we can now load up this data, which will be used for training machine-learning models. In short, this format of data allows consistent handling of missing or incomplete entries in the dataset and ensures that the dataset maintain its proper structure.

4.2.8 Importance of Feature Extractor

The feature extractor process is important because it converts an empty dataset which contains only URLs and labels into a robust and knowledge-rich dataset by extracting keywords from the onion sites. The extracted keywords will serve as the primary predictors for the classification process. Without this step, the machine learning model will have no viable means to identify or classify onion leak sites.

4.3 Site Classification Module

In this stage of project pipeline, the onion site category classification module is responsible for automatically determines whether a given onion website belongs to the leak category or the other category. It uses the keyword-frequency pair as features produced by the Feature Extraction module and applies machine learning techniques to learn the different patterns between leak sites and other hidden network web pages.

This stage includes the machine learning models used, the preprocessing steps applied to the data, the training procedure, balancing techniques for handling skewed datasets, evaluation metrics for performance measurement, the saved model assets and how this classification component integrates into the full end-to-end pipeline.

4.3.1 Purpose of Classification Module

At this stage of the pipeline, each website is represented by its URL, its manually assigned label that either is leak or other and the extracted top keywords with their frequencies. The machine learning classifier operates entirely on these keyword-based features, without using any HTML content, titles or screenshot. The purpose of ML classifier is to learn a pattern from the extracted features and use them to predict whether a website is a leak site or not based solely on these extracted textual signals.

4.3.2 Preprocessing Work Flow for Classification

The complete preprocessing pipeline is implemented within the “preprocess_data” function.

4.3.2.1 Load and Standardize Dataset

The dataset is loaded using pandas from “onionowl_input_data.csv” file and all column names are converted to lowercase to ensure consistent formatting.

4.3.2.2 Remove Unnecessary Columns

The URLs from URLs column are removed because the URL itself does not contain any information and is not useful for training.

4.3.2.3 Shuffle Dataset

Shuffling the dataset ensures that the training and testing sets are free from any order related bias.

4.3.2.4 Identify and Clean Keyword Columns

All columns that start with the name "Keyword" are selected at once. Therefore, the function will process all keywords regardless of the number of keyword columns in a file. There are methods used to standardize keyword columns, such as lower-case conversion of text, replacement of blank values and the removal of unnecessary whitespace.

4.3.2.5 Fill Missing Values

Missing values in keyword columns are filled based on their category. This ensures that leak rows receive keywords that reflects the leak sites and other rows receive keywords that reflects not leak, improving category specific patterns.

4.3.2.6 Combine Keywords and Encode Labels

All keyword columns are combined into a single text field. This produces a concatenated string like “database breach hacked stolen ransomware ransom data leak admin dump credentials dump login exposed.” The category labels are encoded into numerical form using label encoder, which converts leak as 1 and other as 0.

4.3.2.7 TF-IDF Vectorization

In the last step of preprocessing, the concatenated text is transformed into numerical vectors using TF-IDF Vectorizer, which captures word importance, bi-gram patterns and frequency contributions.

4.3.3 Machine Learning Model Selection

This next step in pipeline provides five machine learning models for classification and the admin is responsible for selecting the model that achieves the highest performance, specifically models that achieve accuracy above 80%. Each model has unique strengths suitable for text-based TF-IDF features.

4.3.3.1 Random Forest

A Random Forest provides a solid benchmark for onion dataset, as it works well with sparse TF-IDF representations of text and is resistant to noise in dataset. Random Forest is constructed as an ensemble of decision trees and uses a majority-vote approach to minimize over-fitting and maximize generalization.

4.3.3.2 Naïve Bayes

Naive Bayes is an extremely efficient and particularly effective algorithm in the domain of textual classification. This is primarily due to the fact that Naive Bayes can make an assumption regarding features independence, which gives it relatively high accuracy when used on datasets that are primarily composed of keywords.

4.3.3.3 Extreme Gradient Boosting (XGBoost)

XGBoost can capture nonlinear patterns due to which it can achieve high accuracy. It works well with sparse feature sets. Its gradient boosting framework iteratively corrects errors from previous trees, due to which predictive performance of model increase.

4.3.3.4 Categorical Boost (CatBoost)

The CatBoost algorithm is considered the best algorithm for working with categorical variables. IT manages imbalanced datasets and obtaining maximum accuracy without extensive hyperparameter tuning. Additionally, it uses a technique called Ordered Boosting, which can mitigate overfitting and improve computational efficiency.

4.3.3.5 Gradient Boosting

Gradient Boosting uses boosted decision trees to deliver strong performance while reducing the chance of overfitting. It iteratively improves model predictions by focusing on the residuals of previous trees, due to which it offers reliable results across different datasets.

4.3.4 Handling Imbalance Data

Typically, there are fewer leak sites than the other site types and our dataset faced the same issue also. Leak sites were fewer than other sites. This can lead to a model that is biased towards the class which is more in number. To handle this imbalance problem, the pipeline provides two balancing techniques option to select. These balancing techniques helps to improve model performance by reducing biasness.

4.3.4.1 SMOTE

Smote generates a set of synthetic examples for the minority class rather than taking an existing already classifies examples as input. This is done by generating new examples using interpolation between existing minority instances. SMOTE creates diversity within the minority class and assists the classifier to learn patterns more effectively by generating a larger variety of different synthetic examples. This reduces the bias towards the majority class. A large set of diverse synthetic examples enables the model to generalize better across different sites.

4.3.4.2 Random Oversampling

Through random oversampling, the dataset is balanced by simply duplicating existing instances from the minority class. This ensures that model sees enough examples of leak sites during training so that it can distinguish between leak and other class more effectively. Additionally, It provides a very simple method that maintains the sparse nature of text-based features.

4.3.5 Machine Learning Model Training Work Flow

This step implements a structured approach to training an ML model. This workflow ensures that the model is evaluated properly. Each step during training like preprocessing and balancing the data, training and saving model is done in a consistent manner for all of the models that were selected.

4.3.5.1 Train Test Split and Model Training

The dataset is first divided into training and testing sets. This is done before applying balancing technique as applying balancing techniques to only testing set prevents data leakage and ensures that model performance is evaluated on unseen, unbiased data. Once this is done the next step is to train machine learning models. During this process the model learn patterns from the features and adjust their internal parameters to distinguish between leak and other category.

4.3.5.2 Model Evaluation

After training the model, it is necessary to test and evaluate it. Models are tested on a separate test set and each model performance is evaluated using various performance metrics. These matrices include confusion matrix, classification report, precision, recall, F1 score and the overall accuracy. The combination of these metrics helps us to evaluate how well the model is trained and how accurately the model predicts the correct class.

4.3.5.3 Save Model

When the model is trained well now it’s time to save it as it will be used for future predictions. With the help of joblib system stores the classifier, label encoder and TF-IDF vectorizer. Saving the vectorizer and encoder ensures that future predictions follow the exact same preprocessing pipeline, maintaining consistency between training and inference.

4.3.6 Role of Classification Module

The Classifier occupies a central position in the process of converting raw HTML on onion websites into data. Once the initial URLs are collected, their list can be used by the feature extraction module to crawl through the content of the URL list, extract relevant keywords from the content and convert those keywords into a structured CSV file. The Classification Module then uses the information contained in the CSV files to convert keywords into TF-IDF features, train the selected Machine Learning model and provide predictions as to whether a given site is a leak or other using the resulting TF-IDF feature vectors.

The category classifier module directly impacts the efficiency and accuracy of the entire pipeline of OnionOwl. It is responsible for automating the identification and filtering of potentially leak-related onion websites from available onion URLs data. By providing only confirmed leak sites to the HTML cleaning and NLP-driven structured extraction processes, this component also reduces the amount of unnecessary crawling and processing performed by both components. Furthermore, because of the use of a filtered dataset in the NLP module, the dataset will be cleaner and relevant to the deeper analyses being performed by the NLP module.

4.4 HTML Cleaner for NLP Model

The web cleaner module is designed to clean and normalize HTML content from leak sites which are identified by web classifier module. Its goal of this module is to prepare raw HTML of leak onion site for training NLP model. This cleaner is generic**,** which aims to remove noise and irrelevant elements while preserving meaningful content of multiple sites.

4.4.1 Configuration Constants

Multiple configuration constants are required for web cleaner to perform its preprocessing tasks properly and accurately when scraping onion sites. The web cleaner uses the proxy constant “PROXY_SERVER”, which is configured to send all the scraping requests through the Tor network. This proxy configuration will allow cleaning onion sites without ever revealing the system real identity. “REMOVE_TAGS” contains a lists HTML elements that can usually be ignored such as script, style, meta, iframe, header, footer, svg, and others that often contain non-essential or noisy content. All these tags will be removed from the webpage. This will reduce the size of the HTML and streamline page processing. “AD_KEYWORDS” contains the list of keywords that will allow us to identify advertising elements, the keywords which are usually part of class or ID like ad, banner and popup to detect and eliminate advertisement elements that do not contribute to meaningful content. “TIMER_KEYWORDS” refers to keywords associated with time-sensitive elements which change dynamically. This list identifies and removes countdown timers or other time-sensitive elements, which prevent irrelevant dynamic content from interfering with relevant, useful HTML.

4.4.2 Remove Unwanted Elements

The purpose of the function named "remove_unwanted_inside" is to clean any given DOM element by removing all irrelevant or noisy HTML content. The function removes tags specified in the remove tags list, as well as elements whose class or id include keywords related to advertisements or timers. The function also removes the countdown subblocks and removes all forms and images that contain inline base64 encoded data. The purpose of this function is to removes any noise that might interfere with feature extraction, while keeping meaningful HTML blocks.

4.4.3 Remove Inline Styles and Empty Tags

This step is about standardizing HTML in order to provide a consistency across all pages. This is done by eliminating any inline CSS styles like width, heights from the HTML source. The algorithm will go through all child nodes of every parent node and eliminate any nodes whose CSS formatting might negatively affect the NLP model training process later. The algorithm will continue to traverse the DOM and eliminate any empty nodes such as those without children or text or attributes, until there are no more of those types left. Certain tags such as line breaks and image tags are preserved. The algorithm removes as much of the non-usable or invalid content as is possible while maintaining the remaining content for use in future analyses.

4.4.4 Minify HTML

The HTML is minified using the “minify_html” function, which eliminates extra whitespace (spaces and tabs) as well as compresses multiple spaces into a single space. It also replaces the “> <” sequences with “><”. The primary purpose of this minification is to generate smaller HTML strings, thereby reducing the amount of storage space required, in addition to improving the efficiency of the NLP task by facilitating quicker processing of the resulting data.

4.4.5 Identify Main Content Block

The main purpose of the "find_main_classes" function is to identify the most common classes in block level html body tag such as div, article, section, tr, td, etc. based upon its relation to its parent and child tags. The function will remove duplicate nesting classes so that there remain a clean set of classes that represent the majority of the class containers on the website. The output of this will be a list of tags for the main class, the blocks that are likely to contain meaningful content. This method will help to identify which areas on the website may contain significant or important content while ignoring all irrelevant content.

4.4.6 Grouping HTML Blocks

The purpose of “get_grouped_chunks” function is to process and organize the HTML content into meaningful, manageable information for further processing. This step includes handling structures for table rows, sections and any other main class that is identified by function as part of the HTML document. This ensures that duplicated or nested content which is unnecessary is excluded. The goal of this function is to to provide a better format to facilitate further cleaning and extraction of valuable information from complex and large HTML pages by breaking them down into smaller, more manageable pieces.

4.4.7 Converting Relative URLs to Absolute URLs

The function "absolutize_links", will convert the relative URLs from tags such as images, hyperlinks, scripts, etc., and create a full path for those relative URLs using the base URL of the website we are working with. This process ensures that even after formatting the HTML file, all links continue to be operational and will work in further processes when the data is represented in a JSONL file format.

4.4.8 Normalizing Text

This step removes all the words in a certain HTML element, removes the extra spaces and combines them to form a single line. This is in order to normalize the text to the greatest extent possible to eliminate duplicates to ensure that when it is created as metadata inputs to NLP models, it will be formatted as a single line of text.

4.4.9 Checking for Necessary Information

This is to determine the relevance of an HTML block. It verifies that there are headings, links, paragraphs, list items, table cell and address or location keywords. It also implements a limit of over 20 characters per visible text to filter irrelevant content. This functionality is aimed at making sure that the data contained in the JSONL dataset is not irrelevant and informative, enhancing the quality of the model training later.

4.4.10 Removing Duplicates

This function combines the beginning URL and title text of each block to form a unique identifier, that eliminates duplicate content. When dealing with table rows, it takes into account the text of every cell in order to deal with duplicates. In the presence of numerous duplicates, the function retains the block with the most content. This would remove duplication and only unique and informative blocks would be left at the end.

4.4.11 Pagination Handling

This step helps determine which URL corresponds to the next page of multi-page listings based on commonly understood formats like "Next" button, generic formats such as "See all posts" and through various methods of numbered pagination. The

system uses absolute URL links to provide accurate references when attempting to scrape multiple pages. As a result of this, cleaner will be able to automatically navigate through all paginated content. This makes the cleaner to obtain comprehensive datasets from large number of pages.

4.4.12 Scraping and Cleaning Pages

The scraping can be done to both individual web pages and sites with multiple pages to produce clean and normalized data. The scrape and clean single page function relies on Playwright through Tor to open one page, scrape the HTML content of the page and after scraping it, it is structured into sets of meaningful blocks of content. Once this process has been finished, then it cleans, minifies, removes duplicates and normalizes the HTML and saves the HTML chunks in a JSONL format. The final product in this workflow is to deliver quality structured data in a format suitable to be utilized in Natural Language Processing activities.

The scrape and clean multiple pages function automates the scraping operation of information on different pages of a given site using the next page links. The functionality will assist in all forms of pagination that comprises of link based, button based and numbered pagination. This function also set a limit on the number of pages to prevent excessive scraping and a seen_texts set to track previously extracted test, ensuring that duplicate content across pages is not processed again. All the content collected from each page of a particular website will then be compiled into a single resource called a JSONL file, which allows for a complete website reference, with each single line item being assigned its own entry. Structured in this way, the information contained within this file can be further processed by natural language processing techniques.

4.4.13 Role of Web Cleaner Module

The web cleaner module is the processing unit that converts the raw leak site HTML into the structured and noise free dataset for NLP model. This module also makes sure that the pages that are extracted are meaningful, standardized and can be used in other downstream applications. It eliminates all the unnecessary noise like scripts, styles, forms, ads, inline styles, timers and any other distracting content and all that is left is the content that is relevant to NLP based data extraction.

The module also clean and normalize the content by changing all links to absolute URLs, minifying HTML and standardizing spacing, ensuring that all pages have the same formatting. It also determines meaningful blocks like article, div, tr. It eliminates the repetitive content and retrieves readable and normalized text. The process of pagination enables it to navigate through multi-page web sites automatically and retrieve entire body of data.

In summary, web cleaner module will convert the unstructured leak site HTML into clean, deduplicated and structured JSONL output, which can be used to train an NLP model. In its absence, the dataset would be noisy, inconsistent and unreliable.

4.5 Data Parser Module

After cleaning of the webpages in web cleaner module, the second stage of the pipeline was to convert every clean HTML chunk into a structured input output pair which could be fed into a sequence to sequence, encoder decoder model. This change was made on the data parser module. This module aimed at processing cleaned HTML chunk as input and extracting meaningful fields related to leaks as output. The BeautifulSoup is used in the module to extract the HTML DOM and find several key fields (title, date, company URL, content description, record count, data size, images and dump links). Since each site of the leak has different HTML structure, the parser is based on CSS selectors and all the selectors are always kept configurable so that they can be substituted with others based on the target site.

The parser generates a dictionary of the form {"input": "<html code>", "output": { extracted fields }}. The dictionaries are then written one line each in a JSON Lines (JSONL) file. The reason behind the choice of the JSONL format is that it is lightweight, scalable, line separated and can be fed into the modern encoder decoder transformer architectures easily without having to load the entire dataset into memory.

4.5.1 Role of Data Parser

The role of data parser is very important in facilitating the transition between the raw cleaned HTML chunks and model-ready structured data. Once the web cleaner has created normalized HTML chunks, this parser transmits all meaningful entities and creates the input output examples needed to train the encoder decoder NLP model. Without this module, the cleaned HTML chunks would have been incomplete and the encoder decoder model would not have supervised labels to learn. Concisely, data parser converts cleaned HTML chunks into high quality training samples, which can help in automatically identifying fundamental fields in hidden network pages that would otherwise remain unnoticed.

4.6 Data Normalization Module

After the data is parsed and arranged into the form of JSONL with the help of the data parser module, the second most significant action is data normalization. Normalization is used to make sure that the data is coherent, complete and ready to train NLP model. It normalizes missing values, processes list and makes all fields in the appropriate format to natural language processing.

4.6.1 Input and Output File

The input for the normalization process is the JSONL file created by data parser module. This file has one record per input field with the raw HTML chunk and the other record is the output fields with the extracted information like the title, date, weblink and the other extracted information.

The output is a normalized JSONL data file, a cleaned and standardized JSONL file, ready to be used by the crawler model. In this file, empty and missing fields are set to the value none and list-type fields are set to a comma-separated string. This is to make sure the dataset is homogenous, complete and applicable to train natural language processing task.

4.6.2 Output Cleaner

Normalizing of all the outputs dictionaries in the dataset is done by the clean output function. An empty strings or empty lists is replaced by the word none. If there is a list containing multiple URLs or images links, it turns the list into a comma-separated string. The rest of the values are held constant. This makes all the fields of the output always formatted and appropriate to process downstream.

4.6.3 Purpose of Data Normalization

The information collected from leak sites is often full of missing values or empty strings or lists that cannot be directly given to an NLP model. The process of normalizing solves these problems by substituting the empty fields or empty lists with the text “none”, so that all the records have the same structure. Significant data, like URLs or images links, are turned into comma-separated strings in order to be consistent. The process also ensures that each record contains the same set of fields as it is crucial for batch training of encoder-decoder models. Also, it removes redundant

white space in the output of the JSONL. All in all, data normalization will provide the NLP model with a clean, standardized and structured dataset that is reliable to be trained and processed.

4.7 Natural Language Processing Model Training

The last phase of the OnionOwl pipeline is to train a Natural Language Processing (NLP) model that has the potential to automatically extract structured information from cleaned HTML chunks. This component aims to acquire a sequence-to-sequence mapping that transforms raw leak-site HTML to valuable fields including title, records, data size, dump link, and weblink.

To do so, we treat the task as an encoder decoder text generation problem, in which the input is the cleaned HTML string and the output is a formatted JSONL dictionary with extracted attributes. We tried two transformer-based models of the T5 family, such as Flan-T5 Small to support faster training and finally settling on Flan-T5 Base as offering the best balance between performance matrices, generalization and training feasibility.

4.7.1 Fine-Tuning Flan-T5 Small

First we fine-tune Flan-T5 Small, a model with 80 million parameters. This model was selected because it could be experimented faster and also consume less resources. Training was successful but the model had weak results. It had a hard time decoding the structure of cleaned chunks of HTML and never could extract the necessary fields.

Flan-T5 Small lacks capacity to model the different patterns and site layouts of leak websites that are in existence. It also produced partial or wrong output, sometimes without key or hallucinatory values. More training epochs were not significantly better in performance and eventually resulted in overfitting. With such limitations, Flan-T5

Small was not a final model and we switched to a bigger variant, Flan-T5 Base that had much better performance.

4.7.2 Fine-Tuning Flan-T5 Base

Flan-T5 Base model was chosen as the primary model for HTML-to-structured-information extraction task. The name FLAN means Fine-tuned Language Net, which implies that Google already fine-tunes these models. Flan-T5 Base offered a reasonable compromise between the size of the model, the size of the input and the accuracy of extraction. The model is pre-finetuned and can then be finetuned well on one's own data, to attain high task-specific performance. This part explains the dataset preparation, strategy of tokenization, model configuration and training, evaluation, challenges and reasons why it was selected in the end.

4.7.2.1 Dataset Preparation and Train-Test Split

The training dataset set consisted of cleaned HTML chunks that had their respective structured outputs, created with the data parser module and normalized with the data normalizer module. The data has been saved in the form of JSONL format with a single input-output pair per line. To check the appropriateness of evaluation, we shuffled the dataset and use 80% of the dataset for training and 20% for validation. Each of the inputs was a whole HTML snippet. Their output was in the form of a JSONL string of all the structured fields which had a similar format that could be used to finetune sequence-to-sequence NLP training. This arrangement guaranteed that the model was provided with equal and high-quality instances to study how to transform plain HTML into organized results.

4.7.2.2 Tokenization Strategy

T5Tokenizer of the Hugging face was used to tokenize input HTML snippets and output JSONL strings. There was a limit of 512 tokens on input and output length to trade-off between the memory limits on the GPUs and the sufficient coverage of sequences. Truncation and padding were used to ensure that the dataset had the same shape of tensors. JSONL strings of output were tokenized to labels to enable sequence-to-sequence supervised learning. This method of tokenization meant that the model was able to learn the mapping of variable-length HTML chunks to structured JSON outputs in an efficient manner while maintaining the consistency for batch training.

4.7.2.3 Model Architecture

This NLP model was the one of Google/flan-t5-base, which is Transformer-based, an encoder-decoder architecture that has been trained to complete instruction-following tasks. The device map auto was loaded to the model so that available GPU resources were automatically used and “torch.float32” was used as the torch data type so that there were no disruptions which occur during the training. Flan-T5 Base has about 250 million parameters, which is comfortably fitted in a single GPU unlike Long-T5, which surpassed the memory capacity. The tokenizer and pre-trained Flan-T5 weights were also initialized as part of model initialization and allowed finetuning to utilize general language understanding knowledge to solve the information extraction task.

4.7.2.4 Training Configurations and Hyperparameters

The model was trained using Hugging Face Seq2SeqTrainer with a carefully selected configuration to balance performance and GPU memory constraints. The configurations are:

- **Epochs:** The number of epochs were set to 7, which means model see the data 7 times so that it could learn pattern effectively.
- **Batch Size:** A batch size of 4 per device was set. With gradient accumulation over 4 steps results in an efficient batch size of 16, which prevents the GPU memory from being exceed.
- **Learning Rate:** It was set to 3e-5 which ensures that the step size with which parameter updates are done was controlled, in order to maintain a stable convergence.
- **Warmup Steps:** These were set to 500 which gradually changed the learning rate during the beginning of the training to avoid abrupt weight changes.
- **Weight Decay:** It was set to 0.01, which helps the model to minimize overfitting.
- **Token Length:** Maximum input/output token length was set to 512. This ensures that the sequences fits within the GPU memory while covering sufficient HTML content.
- **Evaluation Frequency:** It was set to 200 steps, which means model performance will be monitored during training.
- **Checkpointing:** It was set to every 200 steps. It ensure that the 3 best performing models are saved, which prevents loss of progress and enables the selection of best performing model later.
- **FP16:** It was turned off to maintain stability.
- **Optimizer:** AdamW optimizer was used, which is an efficient and commonly used optimizer in case of transformer models.
- **Logging:** It was set to every 50 steps. It provides with visual monitoring of loss, matrices and training progress.
- **Data Collator:** It ensure that sequences were dynamically padded using “DataCollatorForSeq2Seq”, which guarantees consistent tensor shapes across each batch.

4.7.2.5 Training Loop

The Seq2SeqTrainer controlled the whole training process, such as batching, backpropagation, gradient accumulation and evaluation. Input HTML snippets and target JSONL outputs were converted into tensors during training and tokenized. The loss was computed each forward pass between the model predicted sequences and the target JSONL strings and then gradient accumulation and optimization steps were performed. To monitor performance, the validation set was used to evaluate it after every 200 steps. The approximate time of training on a single GPU was 4-5 hours. During training, training loss, evaluation loss and samples processed per second were monitored as the main indicators of convergence and a good model learning.

4.7.2.6 Evaluation Strategy

The data were evaluated with the help of “Seq2SeqTrainer.evaluate()” which estimated the loss on the validation set. The model was making predictions based on input HTML snippets and the results were compared to the target JSONL string results. Evaluation loss was the main measure of model selection and qualitative checks were also conducted on the main extracted items like title, date, links and records in order to ensure semantic correctness. Checkpoints were automatically saved throughout the training process and the model with the lowest evaluation loss was kept as the final version to be deployed.

4.7.2.7 Purpose of Selecting Flan-T5 Base

Flan-T5 Base is selected as the final model to be used in the NLP extraction module due to a number of reasons. It is memory friendly and can be trained on one single GPU without the out of memory errors. It is instruction-tuned and optimized on tasks that can be defined as mapping input instructions like HTML snippets to structured output, like JSONL. Compared to smaller variants like Flan-T5 Small, Flan-T5 Base

was better in terms of performance, with better qualitative extraction accuracy and a more stable training. Its flexibility allowed truncation of the length of token to extract most of the HTML content, allowing successful extraction of structured data. Lastly, it was practical enough to enable training and testing to be executed within the accessible hardware capabilities and it was able to support the NLP pipeline of HTML input to structured JSONL output successfully.

4.7.3 Model Loading and Prediction Pipeline

When the Flan-T5 Base model is fine-tuned on our dataset, it can be loaded locally to be inferred. This pipeline is intended to work with raw HTML snippets and predict the structured output with the help of finetuned model.

4.7.3.1 Loading the Fine-Tuned Model and Tokenizer

The tokenizer and fine-tuned Flan-T5 model are loaded to a local directory in which checkpoints are saved. It makes sure that model files are read as a whole out of disk and prevents any attempt to download files out of the Hugging Face Hub and ensures offline reproducibility.

4.7.3.2 Prediction Function

The “predict_from_html” function accepts an input of a single HTML chunk or a list of HTML chunks. All the inputs are preprocessed in the same way they were trained. Then the inputs are tokenized with the same maximum length of 512 tokens. Padding and truncation are used to make them consistent. To enhance the quality of output beam search with num beams 4 and early stopping are used to generate predictions in the model. Lastly, the generated sequences are decrypted into human readable JSONL

strings that contain structured information like title, date, weblink, dumplink and records.

4.7.3.3 Key Features

This NLP extraction pipeline has a number of advantages. It promotes consistency between training and inference preprocessing, such that the model takes as inputs the same form as it was trained on. It is capable of dealing with raw HTML of any onion site after normalization and cleaning and therefore is flexible to various content of leak-sites. It also includes a ready to use function to directly incorporate structured data extraction into the OnionOwl pipeline and simplify the process of converting raw HTML into useful JSON outputs.

CHAPTER 5

RESULTS AND DISCUSSIONS (or USER MANUAL)

5.1 Results of Machine Learning Models

This part provides the experimental results obtained from of the development and testing of the onion site classification module which acts as the gatekeeper to the whole system. We evaluate the output of five machine learning models Random Forest, Naive Bayes, XGBoost, CatBoost and Gradient Boosting coupled with two methods of balancing data, first one SMOTE (Synthetic Minority Over-sampling Technique) and second one Random Oversampling. The accuracy, precision, recall and F1 Score are common metrics of the classification used to measure the performance of each model. Also, Confusion Matrices were examined to know the distribution of True Positives, True Negatives, False Positives and False Negatives.

5.1.1 Evaluation Matrices for ML Models

The data was in the form of manually labelled onion URLs, Class 0 represents Leak sites and Class 1 represents other sites. The data was divided into a training set which consist of 80% entities and testing set which consist of 20% entities. Since data set was not balanced, there were fewer leak sites than non-leak sites, balancing techniques were applied only on the training set and not on the test set of data, which ensures that the test set remains the realistic representation of real-world data and resolve the problem of data leakage. The models were evaluated based on the following matrices:

- **Accuracy:** It is the proportion of all classifications that were correct, whether positive or negative.
- **Precision:** It is the proportion of all the model's positive classifications that are actually positive.
- **Recall:** It is proportion of all actual positives that were classified correctly as positives.
- **F1-Score:** It is the harmonic mean of Precision and Recall.

5.1.2 Detailed Machine Learning Model Analysis

We conduct a detail examination of each machine learning model's performance in this section and determine its specific strengths and weaknesses beyond simple accuracy metrics. Based on the confusion matrices and classification reports, we evaluate the capability of each classifier to address the significant trade-off between recalls and precision. Such a comprehensive analysis of the model performance matrices allows us to discover the most stable classifier for the automated nature of the project.

5.1.2.1 Random Forest Classifier

Random Forest proved to be the most robust model for this text-based classification task. It handled the high-dimensionality of the TF-IDF features effectively.

5.1.2.1.1 Using SMOTE

Performance: The model achieved the accuracy of 73%.

Confusion Matrix Analysis:

Table 5.1: Confusion Matrix of Random Forest Using SMOTE

	Predict 0	**Predict 1**
True 0	12	7
True 1	6	24

The confusion matrix provides a good understanding of the accuracy that the Random Forest Classifier provides after the unbalanced data had been balanced with the help of SMOTE. It tells that the model has observed the correct number of 12 true negative (Class 0) cases and 24 true positive (Class 1) cases, meaning that the model has made the right predictions. However, It had 7 false positives, in which a prediction was made on Class 0 which should have been predicted as Class 1 and 6 false negatives in which prediction was made on Class 1 which should have been predicted as Class 0. These are case of false alarms and false negative errors respectively.

Classification Report:

Table 5.2: Classification Report of Random Forest Using SMOTE

Class	**Precision**	**Recall**	**F1-Score**
Leak (0)	0.67	0.63	0.65
Other (1)	0.77	0.80	0.79
Macro Avg	0.72	0.72	0.72
Weighted Avg	0.73	0.73	0.73

5.1.2.1.2 Using Random Oversampling

Performance: The model achieved the accuracy of 78%.

Confusion Matrix Analysis:

Table 5.3: Confusion Matrix of Random Forest Using Random Oversampling

	Predict 0	**Predict 1**
True 0	14	5
True 1	6	24

The confusion matrix provides a good understanding of the accuracy that the Random Forest Classifier provides after the unbalanced data had been balanced with the help of Random Oversampling. It tells that the model has observed the correct number of 14 true negative (Class 0) cases and 24 true positive (Class 1) cases, meaning that the model has made the right predictions. However, It had 5 false positives, in which a prediction was made on Class 0 which should have been predicted as Class 1 and 6 false negatives in which prediction was made on Class 1 which should have been predicted as Class 0. These are case of false alarms and false negative errors respectively.

Classification Report:

Table 5.4: Classification Report of Random Forest Using Random Oversampling

Class	**Precision**	**Recall**	**F1-Score**
Leak (0)	0.70	0.74	0.72
Other (1)	0.83	0.80	0.81
Macro Avg	0.76	0.77	0.77
Weighted Avg	0.78	0.78	0.78

5.1.2.2 Naïve Bayes Classifier

Naive Bayes has a high performance in the classification of text, in the sense that it uses each word as individual features therefore enabling it to effectively work with extremely large vocabularies without involving complicated calculations.

5.1.2.2.1 Using SMOTE

Performance: The model achieved the accuracy of 78%.

Confusion Matrix Analysis:

Table 5.5: Confusion Matrix of Naive Bayes Using SMOTE

	Predict 0	**Predict 1**
True 0	19	0
True 1	11	19

The confusion matrix provides a good understanding of the accuracy that the Naïve Bayes Classifier provides after the unbalanced data had been balanced with the help of SMOTE. It tells that the model has observed the correct number of 19 true negative (Class 0) cases and 19 true positive (Class 1) cases, meaning that the model has made the right predictions. However, It had 0 false positives and 11 false negatives in which prediction was made on Class 1 which should have been predicted as Class 0. These are case of false alarms and false negative errors respectively.

Classification Report:

Table 5.6: Classification Report of Naïve Bayes Using SMOTE

Class	Precision	Recall	F1-Score
Leak (0)	0.63	1	0.78
Other (1)	1	0.63	0.78
Macro Avg	0.82	0.82	0.78
Weighted Avg	0.86	0.78	0.78

5.1.2.2.2 Using Random Oversampling

Performance: The model achieved the accuracy of 78%.

Confusion Matrix Analysis:

Table 5.7: Confusion Matrix of Naïve Bayes Using Random Oversampling

	Predict 0	Predict 1
True 0	19	0
True 1	11	19

The confusion matrix provides a good understanding of the accuracy that the Naïve Bayes Classifier provides after the unbalanced data had been balanced with the help of Random Oversampling. It tells that the model has observed the correct number of 19 true negative (Class 0) cases and 19 true positive (Class 1) cases, meaning that the model has made the right predictions. However, It had 0 false positives and 11 false negatives in which prediction was made on Class 1 which should have been predicted as Class 0. These are case of false alarms and false negative errors respectively.

Classification Report:

Table 5.8: Classification Report of Naïve Bayes Using Random Oversampling

Class	Precision	Recall	F1-Score
Leak (0)	0.63	1	0.78
Other (1)	1	0.63	0.78
Macro Avg	0.82	0.82	0.78
Weighted Avg	0.86	0.78	0.78

5.1.2.3 XGBoost Classifier

XGBoost is an optimized gradient-boosting algorithm, which sequentially builds the decision trees, where each new decision tree is targeted specifically correct the mistakes made by the previous decision trees. It is considered the state-of-the-art for structured data due to its speed of execution, scalability and inherent regularization which prevents overfitting than regular Random Forests.

5.1.2.3.1 Using SMOTE

Performance: The model achieved the accuracy of 78%.

Confusion Matrix Analysis:

Table 5.9: Confusion Matrix of XGBoost Using SMOTE

	Predict 0	Predict 1
True 0	14	5
True 1	6	24

The confusion matrix provides a good understanding of the accuracy that the XGBoost Classifier provides after the unbalanced data had been balanced with the help of SMOTE. It tells that the model has observed the correct number of 14 true negative (Class 0) cases and 24 true positive (Class 1) cases, meaning that the model has made the right predictions. However, It had 05 false positives, in which a prediction was made on Class 0 which should have been predicted as Class 1 and 6 false negatives in which prediction was made on Class 1 which should have been predicted as Class 0. These are case of false alarms and false negative errors respectively.

Classification Report:

Table 5.10: Classification Report of XGBoost Using SMOTE

Class	Precision	Recall	F1-Score
Leak (0)	0.70	0.74	0.72
Other (1)	0.83	0.80	0.81
Macro Avg	0.76	0.77	0.77
Weighted Avg	0.78	0.78	0.78

5.1.2.3.2 Using Random Oversampling

Performance: The model achieved the accuracy of 76%.

Confusion Matrix Analysis:

Table 5.11: Confusion Matrix of XGBoost Using Random Oversampling

	Predict 0	Predict 1
True 0	15	4
True 1	8	22

The confusion matrix provides a good understanding of the accuracy that the XGBoost Classifier provides after the unbalanced data had been balanced with the help of

Random Oversampling. It tells that the model has observed the correct number of 15 true negative (Class 0) cases and 22 true positive (Class 1) cases, meaning that the model has made the right predictions. However, It had 4 false positives, in which a prediction was made on Class 0 which should have been predicted as Class 1 and 8 false negatives in which prediction was made on Class 1 which should have been predicted as Class 0. These are case of false alarms and false negative errors respectively.

Classification Report:

Table 5.12: Classification Report of XGBoost Using Random Oversampling

Class	Precision	Recall	F1-Score
Leak (0)	0.65	0.79	0.71
Other (1)	0.85	0.73	0.79
Macro Avg	0.75	0.76	0.75
Weighted Avg	0.77	0.76	0.76

5.1.2.4 CatBoost Classifier

CatBoost is an advance gradient boosting algorithm that is unique in the sense that it is capable of operating on categorical features without involving complex preprocessing, like one-hot encoding. It has an advanced method known as ordered boosting that reduces overfitting hence it is highly stable and precise.

5.1.2.4.1 Using SMOTE

Performance: The model achieved the accuracy of 80%.

Confusion Matrix Analysis:

Table 5.13: Confusion Matrix of CatBoost Using SMOTE

	Predict 0	**Predict 1**
True 0	15	4
True 1	6	24

The confusion matrix provides a good understanding of the accuracy that the CatBoost Classifier provides after the unbalanced data had been balanced with the help of SMOTE. It tells that the model has observed the correct number of 15 true negative (Class 0) cases and 24 true positive (Class 1) cases, meaning that the model has made the right predictions. However, It had 4 false positives, in which a prediction was made on Class 0 which should have been predicted as Class 1 and 6 false negatives in which prediction was made on Class 1 which should have been predicted as Class 0. These are case of false alarms and false negative errors respectively.

Classification Report:

Table 5.14: Classification Report of CatBoost Using SMOTE

Class	**Precision**	**Recall**	**F1-Score**
Leak (0)	0.71	0.79	0.75
Other (1)	0.86	0.80	0.83
Macro Avg	0.79	0.79	0.79
Weighted Avg	0.80	0.80	0.80

5.1.2.4.2 Using Random Oversampling

Performance: The model achieved the accuracy of 82%.

Confusion Matrix Analysis:

Table 5.15: Confusion Matrix of CatBoost Using Random Oversampling

	Predict 0	**Predict 1**
True 0	16	3
True 1	6	24

The confusion matrix provides a good understanding of the accuracy that the CatBoost Classifier provides after the unbalanced data had been balanced with the help of Random Oversampling. It tells that the model has observed the correct number of 16 true negative (Class 0) cases and 24 true positive (Class 1) cases, meaning that the model has made the right predictions. However, It had 3 false positives, in which a prediction was made on Class 0 which should have been predicted as Class 1 and 6 false negatives in which prediction was made on Class 1 which should have been predicted as Class 0. These are case of false alarms and false negative errors respectively.

Classification Report:

Table 5.16: Classification Report of CatBoost Using Random Oversampling

Class	**Precision**	**Recall**	**F1-Score**
Leak (0)	0.73	0.84	0.78
Other (1)	0.89	0.80	0.84
Macro Avg	0.81	0.82	0.81
Weighted Avg	0.83	0.82	0.82

5.1.2.5 Gradient Boosting Classifier

Gradient Boosting is an effective ensemble machine learning method, which constructs the model step by step with each subsequent tree being focused on correcting the errors of the previous trees. The combination of these weak learners in

a step-by-step manner forms a very accurate strong model that in many cases has better performance than single decision trees.

5.1.2.5.1 Using SMOTE

Performance: The model achieved the accuracy of 80%.

Confusion Matrix Analysis:

Table 5.17: Confusion Matrix of Gradient Boosting Using SMOTE

	Predict 0	**Predict 1**
True 0	15	4
True 1	6	24

The confusion matrix provides a good understanding of the accuracy that the Gradient Boosting Classifier provides after the unbalanced data had been balanced with the help of SMOTE. It tells that the model has observed the correct number of 15 true negative (Class 0) cases and 24 true positive (Class 1) cases, meaning that the model has made the right predictions. However, It had 4 false positives, in which a prediction was made on Class 0 which should have been predicted as Class 1 and 6 false negatives in which prediction was made on Class 1 which should have been predicted as Class 0. These are case of false alarms and false negative errors respectively.

Classification Report:

Table 5.18: Classification Report of Gradient Boosting Using SMOTE

Class	Precision	Recall	F1-Score
Leak (0)	0.71	0.79	0.75
Other (1)	0.86	0.80	0.83
Macro Avg	0.79	0.79	0.79
Weighted Avg	0.80	0.80	0.80

5.1.2.5.2 Using Random Oversampling

Performance: The model achieved the accuracy of 80%.

Confusion Matrix Analysis:

Table 5.19: Confusion Matrix of Gradient Boosting Using Random Oversampling

	Predict 0	Predict 1
True 0	15	4
True 1	6	24

The confusion matrix provides a good understanding of the accuracy that the Gradient Boosting Classifier provides after the unbalanced data had been balanced with the help of Random Oversampling. It tells that the model has observed the correct number of 16 true negative (Class 0) cases and 24 true positive (Class 1) cases, meaning that the model has made the right predictions. However, It had 3 false positives, in which a prediction was made on Class 0 which should have been predicted as Class 1 and 6 false negatives in which prediction was made on Class 1 which should have been predicted as Class 0. These are case of false alarms and false negative errors respectively.

Classification Report:

Table 5.20: Classification Report of Gradient Boosting Using Random Oversampling

Class	Precision	Recall	F1-Score
Leak (0)	0.71	0.79	0.75
Other (1)	0.86	0.80	0.83
Macro Avg	0.79	0.79	0.79
Weighted Avg	0.80	0.80	0.80

5.1.3 Overall Performance of All ML Models

CatBoost, with the help of Random Oversampling, was the most effective in the test. It was found to have the best overall accuracy of approximately 82 with a great balance of false alarms and cases of a positive appearance. It had a recall of 80% of the true positive cases and 84% of the true negative cases and precision was high because it generated a lower number of false positive and negative, less than any other leading model. This accuracy and high recall with a low false alarm is what makes CatBoost with Random Oversampling the obvious choice between the tested techniques.

Gradient Boosting was the strong candidate appear as the second-best, with great and stable performance in terms of oversampling methods. The results of both SMOTE and Random Oversampling were the same. Hence both were equally effective in this model. Gradient Boosting was very stable and precise with an accuracy of approximately 80% and 4 false positive and 6 false negative. It was slightly less accurate and in false-alarm control than CatBoost, but still, it is a very reliable Tier-1 model that is very strong and repeatable.

Random Forest was used as a strong baseline model and it was found to perform consistently with a fair level of accuracy but not as accurate as the other boosting-based methods. Its highest performance was with Random Oversampling because it achieved the 80% recall, indicating good capabilities of identifying the positive category and reaching 74% of recall indicating the negative category. It was

however, a little noisier than CatBoost and Gradient Boosting with 5 false positives and 6 false negative. Moreover, achieved the overall accuracy of about 78%. It may have been reliable and strong, but not as refined and efficient as the highest performing models.

XGBoost proved to be a more balanced opponent in the comparison and its highest score was obtained when it was used with SMOTE and overall accuracy was 78%. It was almost identical to the best Random Forest setup, with a result of 14 True Negatives, 24 True positive, 6 False Negative and 5 False Positives. Although it showed good models and there was consistency in false alarm and detection, it appeared to reach a limit of performance on this dataset. Consequently, it was unable to outperform the precision or the general accuracy offered by the CatBoost or Gradient Boosting, which put it squarely in the middle of the model performance spectrum.

Naive Bayes was accurate but not as accurate as rest of the models and both the resampling methods yielded the same results. The overall accuracy of this model was 78%. It achieved a perfect precision score of 100% on class 1, which means it did not raise a false alarm. But for class 0 precision score was 63% which is very low. The Recall is too low to be used in a leak-detection pipeline where false alarms are less harmful than missing real ones.

5.1.4 Comparison of All Machine Learning Models

This is the detailed comparison of all model which we train for classification task.

Table 5.21: Comparison of All ML Models

Model	Balancing Technique	Accuracy	Precision	Recall	F1-Score
Random Forest	SMOTE	0.73	0.67	0.63	0.65
Random Forest	Random Oversampling	0.78	0.70	0.74	0.72
Naïve Bayes	SMOTE	0.78	0.63	1.00	0.78
Naïve Bayes	Random Oversampling	0.78	0.63	1.00	0.78
XGBoost	SMOTE	0.78	0.70	0.74	0.72
XGBoost	Random Oversampling	0.76	0.65	0.79	0.71
CatBoost	SMOTE	0.80	0.71	0.79	0.75
CatBoost	Random Oversampling	0.82	0.73	0.84	0.78
Gradient Boosting	SMOTE	0.80	0.71	0.79	0.75
Gradient Boosting	Random Oversampling	0.80	0.71	0.79	0.75

5.1.5 Final Selected ML Model (CatBoost)

After the analysis of the overall performance, we have chosen a final model to be CatBoost with Random Oversampling to be used in the OnionOwl pipeline. It was also the most successful in terms of accuracy with 82% of overall accuracy and better precision, which minimized false alarms to a minimum of 3, the lowest of all high-performing. Importantly, it had a high recall of 84% for leak detection, which presents the optimal safety versus reliability ratio to application in the real world.

5.2 Results of Natural Language Processing Models

The section contains the results of the experiment conducted on the fine-tuned NLP models, which is the base extraction engine of the system. We compare the results of the two Transformer-based models, FLAN-T5 Small, and FLAN-T5 Base, which are fine-tuned on our own data, consisting of cleaned HTML and structured JSONL pairs. The standard text generation measures, such as Training and Validation Loss, ROUGE scores (ROUGE-1, ROUGE-L) and BLEU scores were used to measure the performance of each model. Also, learning curves and outputs were examined to determine the stability of training, convergence behavior and structural integrity of the extracted JSON data.

5.2.1 Evaluation Matrices for NLP Models

The dataset is in the form of cleaned HTML chunks and structured JSONL strings respectively. To have a strong evaluation, it was shuffled and split into a training set that contains 80% of the entity and a validation set that contains the other 20% of the entities. This division made sure that the model was tested on the unknown examples to test its generalization abilities and its capability to work with the new structures of the sites. The results were then evaluated on the following matrices:

- **Training Loss:** Training Loss is a metric that measures the level of model performance on the training data. In training the model the model makes predictions and compares the predictions with the actual target values. The loss functional then gets the difference between these outputs and the actual labels.
- **Validation Loss:** Validation loss is a metric that evaluates model performance on data that the model has never seen during training.
- **BLEU:** The BLEU score is used to evaluate the quality of response by comparing it with reference.
- **ROUGE:** The ROUGE score is a set of metrics used to evaluate the quality of natural language generations. It measures the overlap between the generated response and the reference text based on n-gram recall, precision, and F1 score.
    - **ROUGE1:** This metric measures the overlap of the unigrams (words by themselves) between the text generated and the text in the reference meaning how accurately the model is able to identify particular keywords such as titles or dates.
    - **ROUGEL:** It is a metric that evaluates the number of common characters in the generated and reference text indicating how the model can preserve the correct sentence structure and syntax of the JSON.

5.2.2 Detailed NLP Model Analysis

In this part we conduct a detailed analysis of the individual model performance of each Natural Language Processing model and establish the particular strengths and weaknesses of the model beyond the basic validation loss values. Depending on the training dynamics and generation scores, we consider the ability of each model to solve the enormous problem of unstructured HTML to valid, structured JSONL without hallucinations. With the help of such an in-depth analysis of the model performance metrics, we can learn about the most robust extractor to the automated nature of the project.

5.2.2.1 FLAN-T5 Small

In the experiment, the FLAN-T5 Small, the model with around 80 million parameters was chosen as the baseline to evaluate whether the extraction task was possible under the conditions of severe hardware limitations.

5.2.2.1.1 Overall Performance of FLAN-T5 Small

In the fine-tuning phase, the model had several symptoms pointing to optimization failure and gradient explosion. The training loss decreased to 0.000000, which is mathematically impossible in a real sequence-to-sequence task this is an indication of complete breakdown of the model's learning dynamics or a loss function corruption. Meanwhile the validation loss went to NaN, a traditional indicator of exploding gradients, with values becoming large enough that the hardware is unable to represent them, resulting in weights and activations taking up invalid numerical values.

Further examination revealed that the model ceased to learn altogether. The ROUGE scores were not increasing in any of the five epochs and the scores did not improve at all. ROUGE-L was not moving and had zero variance at 0.133176. Such lack of progress is a sign of the collapse of the model, it goes to produce the same output on all inputs without any reference to the context of the HTML. This is expected when gradients separate quickly, making the network lose the capacity to meaningfully update. During this unsuccessful training, the model essentially went into a recursive loop of outputs, which proved that optimization has completely failed.

5.2.2.1.2 Evaluation Matrices

There are the evaluation matrices and their performance in case of FLAN-T5 small.

Table 5.22: Evaluation Matrices Results for FLAN-T5 Small

Model	Training Loss	Validation Loss	ROUGE-1	ROUGE-L	BLEU
FLAN-T5 Small	0.00	Nan	0.13	0.09	0.08

5.2.2.2 FLAN-T5 Base

The FLAN-T5 Base model with 250 million parameters was fine-tuned after the limitations experienced by the small variant. This model was more stable and was able to learn the patterns.

5.2.2.2.1 Overall Performance of FLAN-T5 Base

The convergence pattern of the learning curve is very healthy and smooth. The training loss continued to reduce steadily at step 200 through step 1600 by constant reduction of 3.32 to 0.04 respectively, proving that the model did not become unstable in the process of learning the underlying features. Meanwhile, the validation loss was 2.06 at step 200 but reduce to 0.06 at step 1600, which is very similar to the final training loss. This small margin indicates that the model is not overfitting and is trained well.

ROUGE test results display a high text-matching. ROUGE-1 was 0.88 that means that the degree of unigram overlap of ROUGE-1 is very high such that the model is highly confident in the ability of selecting the most meaningful words from source text. Similarly, ROUGE-L also achieved the score of 0.88 underlines the fact that the model does not disintegrate the long strings of meaningful words and preserves

the original phrasing and structure. These scores tells that the model can not only find the important content but can recreate it very closely to the original text.

The model exceptional performance is also supported by the BLEU score. The model scores 0.71, which is above average standards of NLP since 0.3 or 0.4 is good with generation tasks. Such a score indicates that the model is making text which is incredibly accurate. The text being produced exactly corresponds to the reference text at the n-gram level. Practically, this implies that the extracted content is accurate and closely matched the targeted output.

5.2.2.2.2 Evaluation Matrices

There are the evaluation matrices and their performance in case of FLAN-T5 Base.

Table 5.23: Evaluation Matrices Results for FLAN-T5 Base

Model	Training Loss	Validation Loss	ROUGE-1	ROUGE-L	BLEU
FLAN-T5 Small	0.04	0.06	0.88	0.88	0.71

5.2.3 Comparison of Both NLP Models

This is the detailed comparison of all model which we fine- tuned for text extraction task.

Table 5.24: Comparison of Both NLP Models

Model	Training Loss	Validation Loss	ROUGE-1	ROUGE-L	BLEU
FLAN-T5 Small	0.00	Nan	0.13	0.09	0.08
FLAN-T5 Base	0.04	0.06	0.88	0.88	0.71

5.2.4 Final Selected NLP Model

According to the performance evaluation, FLAN-T5 Base was chosen as the ultimate project pipeline model. It showed a remarkable semantic knowledge with an impressive ROUGE-1 and ROUGE-L scores of 88% and BLEU score of 71% which indicates highly accurate information retrieval. The model has a loss of only 0.06 in validation, which means that model is reliable and it can extract complex and unstructured information.

CHAPTER 6

CONCLUSION AND RECOMMENDATIONS

6.1 Conclusion

In conclusion, the project aimed to solve a gap in dark web intelligence, the lack of a reliable, automated and scalable way to extract structured information out of highly irregular, noisy and masked leak-sites. The current literature and tools, including standard crawlers and sophisticated NER models, LLM-driven threat intelligence systems and PII detection models show great advancements in dark web research, yet none can be a perfect end-to-end system to be integrated into the ***Orion Intelligence***. These solutions are either based on clean and predictable clearnet datasets or are marketplace-based. Having identified these drawbacks, this project came up with a fully customized system that can cater the needs of Orion Intelligence in relation to monitoring of hidden network leaks.

In general, this project does not only extract structured information from leak onion sites, it shows that the combination of domain-specific dataset generation, cautious model choice and a solid pipeline can help current NLP systems to outperform challenges that once regarded as too irregular or too risky to be automated. Finally, the system developed during this project will not only address capability gap but will also offer ***Orion Intelligence*** a scalable, extensible and intelligence-ready module for hidden network leak data extraction, which is critical in analysis of modern cyber-threats.

6.2 Recommendations/Future Enhancements

Although the designed module has been able to perform strongly in retrieving structured information on the dark web leak websites, there are various areas in which it can be improved to enhance the utility, scalability and adaptability of the system in ***Orion intelligence***. To start with, by adding more leak sites, forums and marketplaces to the dataset, the model will have better capabilities in its generalization and it will be possible to keep it working despite the changes in the dark web ecosystem.

Second, the system would be enhanced with adaptive learning mechanisms, in which feedback of the ***Orion Intelligence*** analysts will be employed to constantly adjust the NLP model, making it more accurate with time. The pipeline is modular, which can be integrated with other modules of the ***Orion Intelligence***, like threat prediction and anomaly detection, and form a full intelligence ecosystem. Through these improvements, ***Orion Intelligence*** will be able to not only continue to have a competitive edge in automated dark web surveillance, but also deliver actionable insights in a more effective and reliable way. All these improvements can be implemented in the system if company decides to do so and will ensure that the module will remain align with the operational requirements and the evolving challenges experienced by the ***Orion Intelligence***.

REFERENCES

Journal Papers:

[1] D. De Pascale, G. Cascavilla, D. A. Tamburri, and W. J. Van Den Heuvel, "CRATOR a CRAwler for TOR: Turning Dark Web Pages into Open Source INTelligence," in *Computer Security – ESORICS 2024*, vol. 14983, J. Garcia-Alfaro, R. Kozik, M. Choraś, and S. Katsikas, Eds., in Lecture Notes in Computer Science, vol. 14983. , Cham: Springer Nature Switzerland, 2024, pp. 144–161. doi: 10.1007/978-3-031-70890-9_8.

[2] J. Bergman and O. B. Popov, "Exploring Dark Web Crawlers: A Systematic Literature Review of Dark Web Crawlers and Their Implementation," *IEEE Access*, vol. 11, pp. 35914–35933, 2023, doi: 10.1109/ACCESS.2023.3255165.

[3] A. Makhambet and A. Moldagulova, "A comparative analysis of machine learning methods for personal information recognition (PII) in unstructured texts," . *Computing*, 2025.

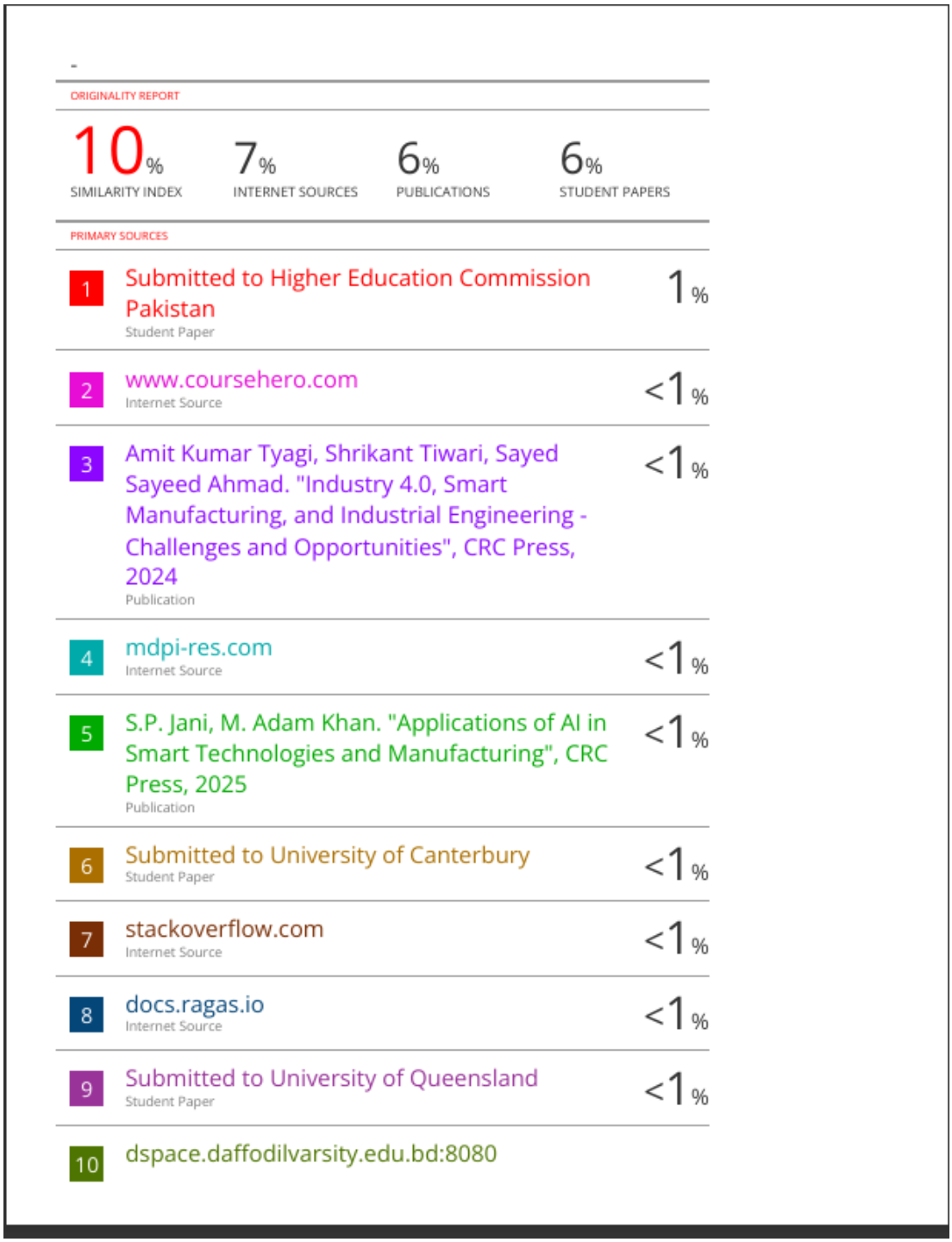
[4] G. Cascavilla, I. Bakermans, D. De Pascale, G. Marcelino, and Z. Geradts, "Scraping the Shadows: Deep Learning Breakthroughs in Dark Web Intelligence," 2025, *SSRN*. doi: 10.2139/ssrn.5078577.

[5] J. M. Ruiz Ródenas, J. Pastor-Galindo, and F. Gómez Mármol, "A general and modular framework for dark web analysis," *Clust. Comput.*, vol. 27, no. 4, pp. 4687–4703, July 2024, doi: 10.1007/s10586-023-04189-2.

[6] A. Fayzi, M. Fayzi, and K. D. Ahmadi, "Dark Web Activity Classification Using Deep Learning".

[7] R. N. Ribeiro, "Exploring the Feasibility of Using Large Language Models for Dark Web Threat Intelligence in Security and Defence".

Appendex



-

ORIGINALITY REPORT

10%	7%	6%	6%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to Higher Education Commission Pakistan Student Paper	1%
2	www.coursehero.com Internet Source	<1%
3	Amit Kumar Tyagi, Shrikant Tiwari, Sayed Sayeed Ahmad. "Industry 4.0, Smart Manufacturing, and Industrial Engineering - Challenges and Opportunities", CRC Press, 2024 Publication	<1%
4	mdpi-res.com Internet Source	<1%
5	S.P. Jani, M. Adam Khan. "Applications of AI in Smart Technologies and Manufacturing", CRC Press, 2025 Publication	<1%
6	Submitted to University of Canterbury Student Paper	<1%
7	stackoverflow.com Internet Source	<1%
8	docs.ragas.io Internet Source	<1%
9	Submitted to University of Queensland Student Paper	<1%
10	dspace.daffodilvarsity.edu.bd:8080	

	Internet Source	<1%
11	eprints.utar.edu.my Internet Source	<1%
12	Submitted to Sheffield Hallam University Student Paper	<1%
13	deeplearning.neuromatch.io Internet Source	<1%
14	docs.google.com Internet Source	<1%
15	www.jatit.org Internet Source	<1%
16	Thangaprakash Sengodan, Sanjay Misra, M Murugappan. "Advances in Electrical and Computer Technologies", CRC Press, 2025 Publication	<1%
17	Shankar Babu, Mahesh Babu Kota. "Synergies in Smart and Virtual Systems using computational intelligence", CRC Press, 2025 Publication	<1%
18	Submitted to University of Hertfordshire Student Paper	<1%
19	Yahyaeian, Amir Abbas. "Enhancing Mechanical Engineering Education Through a Virtual Instructor in an AI-Driven Virtual Reality Fatigue Test Lab", Purdue University, 2025 Publication	<1%
20	bibliotekanauki.pl Internet Source	<1%
21	Submitted to Liverpool John Moores University	<1%

	Student Paper	
22	Khalifa Afane, Yijun Zhao. "Selecting Classifiers and Resampling Techniques for Imbalanced Datasets: A New Perspective", Procedia Computer Science, 2024 Publication	<1%
23	www.fernuni-hagen.de Internet Source	<1%
24	Ashok Kumar, Geeta Sharma, Anil Sharma, Pooja Chopra, Punam Rattan. "Advances in Networks, Intelligence and Computing - International Conference on Networks, Intelligence and Computing (ICONIC-2023)", CRC Press, 2024 Publication	<1%
25	Submitted to University of Greenwich Student Paper	<1%
26	eprints.utm.my Internet Source	<1%
27	gpttutorpro.com Internet Source	<1%
28	ce.journal.satbayev.university Internet Source	<1%
29	core.ac.uk Internet Source	<1%
30	www.icondata.org Internet Source	<1%
31	Submitted to University of Adelaide Student Paper	<1%
32	www.mdpi.com Internet Source	<1%

33	Alketbi, Abdulla Matar. "Real-Time Fraud Detection Using Big Data", Rochester Institute of Technology Publication	<1%
34	Submitted to University of Exeter Student Paper	<1%
35	lisatwyw.github.io Internet Source	<1%
36	umpir.ump.edu.my Internet Source	<1%
37	"Data Science and Big Data Analytics", Springer Science and Business Media LLC, 2025 Publication	<1%
38	Shivam R Solanki, Drupad K Khublani. "Generative Artificial Intelligence", Springer Science and Business Media LLC, 2024 Publication	<1%
39	Sushil Kamboj, Pardeep Singh Tiwana. "Innovations in Computing", CRC Press, 2025 Publication	<1%
40	herkules.oulu.fi Internet Source	<1%
41	repository.lcu.edu.ng Internet Source	<1%
42	O. P. Verma, Seema Verma, Thinagaran Perumal. "Advancement of Intelligent Computational Methods and Technologies - Proceedings of I International Conference on Advancement of Intelligent Computational Methods and Technologies (AICMT2023)", CRC Press, 2024 Publication	<1%

43	Submitted to Strathmore University (Main Account) Student Paper	<1 %
44	Submitted to University of Durham Student Paper	<1 %
45	Submitted to University of Birmingham Student Paper	<1 %
46	Submitted to University of Sheffield Student Paper	<1 %
47	Submitted to University of Sydney Student Paper	<1 %
48	al-kindipublishers.org Internet Source	<1 %
49	gitlab.sliit.lk Internet Source	<1 %
50	www.scribd.com Internet Source	<1 %
51	A.K. Indira Kumar, Gayathri Sthanusubramoniani, Deepa Gupta, Aarathi Rajagopalan Nair, Yousef Ajami Alotaibi, Mohammed Zakariah. "Multi-task detection of harmful content in code-mixed meme captions using large language models with zero-shot, few-shot, and fine-tuning approaches", Egyptian Informatics Journal, 2025 Publication	<1 %
52	journal.unisza.edu.my Internet Source	<1 %
53	jurnal.yoctobrain.org Internet Source	<1 %

54	www.kcsfoundation.org Internet Source	<1%
55	Submitted to Monash University Student Paper	<1%
56	Chakraborty, Avipriyo. "Effects of Deep-Rooted Vetiver Grass as a Bio-Anchor on Slope Stabilization", Jackson State University Publication	<1%
57	aircconline.com Internet Source	<1%
58	capuana.ifi.uzh.ch Internet Source	<1%
59	"Computational Intelligence in Pattern Recognition", Springer Science and Business Media LLC, 2022 Publication	<1%
60	Abelha, João Paulo Gomes Torres. "Automatic Categorization of Health-Related Messages in Online Health Communities", Universidade do Porto (Portugal), 2024 Publication	<1%
61	Amir Gharavi, Mohamed Hassan, Jebraeel Gholinezhad, Hesam Ghoochaninejad, Hossein Barati, James Buick, Karrar A. Abbas. "Application of machine learning techniques for identifying productive zones in unconventional reservoir", International Journal of Intelligent Networks, 2022 Publication	<1%
62	Anshul Verma, Pradeepika Verma. "Research Advances in Intelligent Computing - Volume 2", CRC Press, 2024 Publication	<1%

63	Ara Cho, Eugene C Yi, Yun Jong Lee, Yeong Wook Song, Yoshiya Tanaka, Kristine M Kim. "Predictive Biomarkers for TNF Inhibitor Response in Rheumatoid Arthritis: A Proteomics-based Machine Learning Approach", Springer Science and Business Media LLC, 2024 Publication	<1%
64	Ehtesham Hashmi, Sule Yildirim Yayilgan. "Multi-class hate speech detection in the Norwegian language using FAST-RNN and multilingual fine-tuned transformers", Complex & Intelligent Systems, 2024 Publication	<1%
65	Gun-Yoon Shin, Dong-Wook Kim, SungJin Park, A-ran Park, Younghwan Kim, Myung-Mook Han. "Identifying Similar Users Between Dark Web and Surface Web Using BERTopic and Authorship Attribution", Electronics, 2025 Publication	<1%
66	Mohammad Sadegh Sheikhaei, Yuan Tian, Shaowei Wang, Bowen Xu. "An empirical study on the effectiveness of large language models for SATD identification and classification", Empirical Software Engineering, 2024 Publication	<1%
67	Sumon, Maruful Abedin. "Emergency Care Patient Prediction Using Electronic Health Records Data: An End-to-End Machine Learning Pipeline.", Jackson State University Publication	<1%
68	fenix.tecnico.ulisboa.pt Internet Source	<1%

69	link.springer.com Internet Source	<1%
70	nova.newcastle.edu.au Internet Source	<1%
71	www.slideshare.net Internet Source	<1%
72	www.theseus.fi Internet Source	<1%
73	"Intelligent and Fuzzy Systems", Springer Science and Business Media LLC, 2024 Publication	<1%
74	Acharige, Tharindu Madusanka Batawala. "Reasoning With Natural Language: Probing Transformer Models' Ability to Perform Formal Reasoning in Natural Language", The University of Manchester (United Kingdom), 2025 Publication	<1%
75	Clift, Jennifer E.. "A Predictive Model to Increase Technology Transfer and Transition", The George Washington University, 2023 Publication	<1%
76	Mohabir, Ashir Harish. "Preventing Model Poisoning Through Artificial Immune Networks", University of Johannesburg (South Africa), 2025 Publication	<1%
77	lutpub.lut.fi Internet Source	<1%
78	pdfcoffee.com Internet Source	<1%
79	repository.tilburguniversity.edu Internet Source	

		<1%
80	www.intechopen.com Internet Source	<1%
81	www.preprints.org Internet Source	<1%
82	"Text, Speech, and Dialogue", Springer Science and Business Media LLC, 2026 Publication	<1%
83	"Artificial Intelligence Applications and Innovations", Springer Science and Business Media LLC, 2025 Publication	<1%
84	Sukhpreet Kaur, Amanpreet Kaur, Manish Kumar. "Recent Advances in Computational Methods in Science and Technology", CRC Press, 2026 Publication	<1%
85	do Couto Gonçalves, Luís. "Automatic Lung Nodule Classification in Chest Computerized Tomography Images", Universidade do Porto (Portugal), 2024 Publication	<1%

Exclude quotes	Off	Exclude matches	Off
Exclude bibliography	Off		

*% detected as AI

AI detection includes the possibility of false positives. Although some text in this submission is likely AI generated, scores below the 20% threshold are not surfaced because they have a higher likelihood of false positives.

Caution: Review required.

It is essential to understand the limitations of AI detection before making decisions about a student's work. We encourage you to learn more about Turnitin's AI detection capabilities before using the tool.

Disclaimer
Our AI writing assessment is designed to help educators identify text that might be prepared by a generative AI tool. Our AI writing assessment may not always be accurate (i.e., our AI models may produce either false positive results or false negative results), so it should not be used as the sole basis for adverse actions against a student. It takes further scrutiny and human judgment in conjunction with an organization's application of its specific academic policies to determine whether any academic misconduct has occurred.

Frequently Asked Questions

How should I interpret Turnitin's AI writing percentage and false positives?
The percentage shown in the AI writing report is the amount of qualifying text within the submission that Turnitin's AI writing detection model determines was either likely AI-generated text from a large-language model or likely AI-generated text that was likely revised using an AI paraphrase tool or word spinner.

False positives (incorrectly flagging human-written text as AI-generated) are a possibility in AI models.

AI detection scores under 20%, which we do not surface in new reports, have a higher likelihood of false positives. To reduce the likelihood of misinterpretation, no score or highlights are attributed and are indicated with an asterisk in the report (*%).

The AI writing percentage should not be the sole basis to determine whether misconduct has occurred. The reviewer/instructor should use the percentage as a means to start a formative conversation with their student and/or use it to examine the submitted assignment in accordance with their school's policies.

What does 'qualifying text' mean?
Our model only processes qualifying text in the form of long-form writing. Long-form writing means individual sentences contained in paragraphs that make up a longer piece of written work, such as an essay, a dissertation, or an article, etc. Qualifying text that has been determined to be likely AI-generated will be highlighted in cyan in the submission, and likely AI-generated and then likely AI-paraphrased will be highlighted purple.

Non-qualifying text, such as bullet points, annotated bibliographies, etc., will not be processed and can create disparity between the submission highlights and the percentage shown.