

ABSTRACT

The XENON framework is a Web scraper/Content summarization tool that can scrape (scrapes the DOM files page content from the specified URL) and summarize the contents given within the requested domain. The beauty of XENON lies in its ability to scrape domains from both the Clearnet and the darknet and thus by setting a threshold value, XENON can be used to effectively monitor a particular domain and define a notification if the content there gets out of hand.

TABLE OF CONTENTS

CHAPTER	TITLE	PAGE NO
1	INTRODUCTION	
2	PROBLEM STATEMENT	
3	REQUIREMENT	
4	PROPOSED SYSTEM	
5	MODULES USED	
6	MODULE EXPLANATION	
7	PROGRAM	
8	SAMPLE OUTPUT	
9	CONCLUSION	
10	REFERENCE	

INTRODUCTION

In this fast moving world, there is a lot of data/information being spread around through online journals, magazines, newsletters and so on. While most of them give us knowledge that deserves our time, some of them are just plain time wasting mediums. Such mediums usually provide content that are either cloned, or doesn't have enough practical data. Thus the goal of this project is to create a web scraper/summarization framework that scrapes the content present within a given domain and summarizes the content within it.

A: WEB SCRAPERS

Web scrapers are software tools used to extract data from websites automatically. They simulate human browsing behaviour to navigate through web pages, retrieve specific information, and store it in a structured format.

Web scrapers use different techniques to extract data, including parsing HTML, interacting with APIs, and even employing machine learning algorithms for more complex tasks. They can scrape text, images, links, tables, and other types of data from websites.

Web scrapers are widely used for various purposes, such as market research, price comparison, content aggregation, data analysis, and monitoring online presence. They enable businesses and individuals to gather large amounts of data efficiently and automate repetitive tasks.

However, web scraping must be done ethically and legally. Website owners have the right to control access to their content, and scraping can violate their terms of service or even infringe on copyright laws.

It is essential to adhere to rate limits, and respect any restrictions imposed by the site. Many programming languages provide libraries and frameworks for web scraping, such as Python (with popular libraries like BeautifulSoup and Scrapy), Node.js (with libraries like Puppeteer), and Ruby (with libraries like Nokogiri). These tools simplify the process of building web scrapers and handling common scraping tasks.

eg,

- **Beautiful Soup,**
- **Scrapy,**
- **Selenium WebDriver,**
- **Puppeteer,**
- **Requests,**
- **Octoparse,**
- **Parse Hub,**
- **Apify,**
- **Import.io,**
- **Web Harvy.**

B: CONTENT SUMMARIZERS

Content summarizers are tools or algorithms that condense large bodies of text into shorter versions, capturing the most important information and main ideas.

Content summarizers use various techniques, including natural language processing (NLP), machine learning, and text analysis, to identify key sentences, extract relevant information, and generate concise summaries.

Summarizers can be applied to different types of content, such as articles, documents, research papers, news stories, and even social media posts.

Summarizers aim to save time and improve information consumption by providing a condensed version of lengthy texts, allowing readers to quickly grasp the main points without reading the entire document.

Extractive summarization is a common approach used by many content summarizers. It involves selecting sentences or passages from the original text and assembling them to form a summary.

This method preserves the wording and structure of the original text. Abstractive summarization is another approach that involves generating summaries by understanding the content and creating new sentences that convey the main ideas.

Abstractive summarizers have the ability to paraphrase and rephrase information, but they can be more challenging to develop.

Content summarizers can be developed as standalone software applications, integrated into content management systems (CMS), or accessed through online platforms and APIs. Evaluation of summarizers can be subjective, as there is no single correct summary.

However, common evaluation metrics include measures like ROUGE (Recall-Oriented Understudy for Gisting Evaluation) and BLEU (Bilingual Evaluation Understudy), which assess the quality and similarity of summaries compared to reference summaries or human-generated summaries.

Content summarizers have various applications, including news aggregation, document summarization for research or legal purposes, social media post analysis, and even personal productivity tools for digesting large volumes of information.

While content summarizers offer efficiency and convenience, they may not capture all nuances or context present in the original text.

Users should be aware that summaries are abridged versions and may not fully reflect the complete meaning or intent of the source material.

It is important to use summaries as a starting point for understanding and explore the original content when necessary.

eg.

- **IBM Watson Natural Language Understanding.**
- **Hugging Face Transformers,**
- **Summarize.io,**
- **Sumy,**
- **TextRank,**
- **Gensim,**
- **BART (Bidirectional and Auto-Regressive Transformer),**
- **T5 (Text-To-Text Transfer Transformer),**

PROBLEM STATEMENT

PRIMARY

To monitor the internet for desired / specific anomalies

To analyse the web scraped DOM content and predict / categorize their intensity.

To summarize and segment the content using an NLP engine.

To define a threshold and define a system to create a notification when the threshold is surpassed.

SECONDARY

A

Web scrapers often need to handle dynamic websites that rely heavily on JavaScript for content generation. In such cases, headless browsers like Puppeteer or Selenium

WebDriver can be used to render and interact with JavaScript-based content, enabling the scraper to extract data that would be inaccessible through static HTML parsing.

Some websites actively employ anti-scraping techniques to prevent or deter scrapers.

These techniques can include CAPTCHAs, IP blocking, honeypot traps, or user behaviour analysis.

Overcoming these challenges requires implementing countermeasures or using specialized scraping tools designed to bypass such obstacles.

Web scraping can pose challenges related to data quality and consistency. Websites

may change their structure or layout, which can break scraping scripts.

Handling variations in data formats, dealing with missing or incomplete information, and managing updates in the source website are important considerations when building robust and reliable web scrapers.

Scraping too many websites or sending too many requests in a short period can put a strain on the target server's resources and lead to performance issues or even legal consequences. It is crucial to be mindful of rate limits, caching mechanisms, and design scraping scripts to be respectful of the target website's capacity.

Web scraping is a constantly evolving field, and it requires ongoing maintenance and monitoring. Websites change, APIs get updated, and new scraping challenges emerge. Keeping abreast of best practices, legal considerations, and technological advancements is crucial for successful and responsible web scraping.

B

In the summarization field, there is an established dichotomy: summaries can be either extractive (i.e., snippets from the original text), or abstractive (i.e., newly-generated text).

Extractive summaries locate key sentences within the original text, and therefore tend to accurately reflect the main idea (unless sentences are maliciously cherry-picked in a way that misrepresents it). The downside is that extractive summaries are limited in their ability to fully capture the content — the sentences that are dropped are forever lost.

On the other hand, abstractive summaries emulate more closely how humans summarize: we put together new sentences that aim to tell the whole story, but from a higher-level perspective.

However, abstractive summaries are technically difficult to produce, since they require generative models like the GPT family. Currently, such models suffer from hallucination — their outputs can be factually incorrect and/or not supported by the original text.

This is a huge impediment for summarization, since staying true to the input text is not negotiable. While preventing hallucinations is an active field of research, there are no bullet-proof solutions that guarantee a certain quality bar. This is why the abstractive APIs from Google are still experimental (i.e., it is not supported officially like products offered through Google Cloud Platform).

REQUIREMENTSs

HARDWARE

A GPU that can efficiently handle Nvidia's CUDA toolkit

(Preferably with 8 gb RAM)

SOFTWARE

PYTHON

PYTORCH

BS4

PLOTLY

IDE (SPYDER/VSCODE)

PROPOSED SYSTEM

The XENON framework is a Web scraper/Content summarization tool that can scrape and summarize the contents given within the requested domain.

It can efficiently scrape the domains from both Clearnet and darknet.

It makes use of bs4 (i.e) Beautiful soup along with headers to scrape the contents present within a domain.

It makes use of the Sock5 module along with headers to scrape the contents present within a domain.

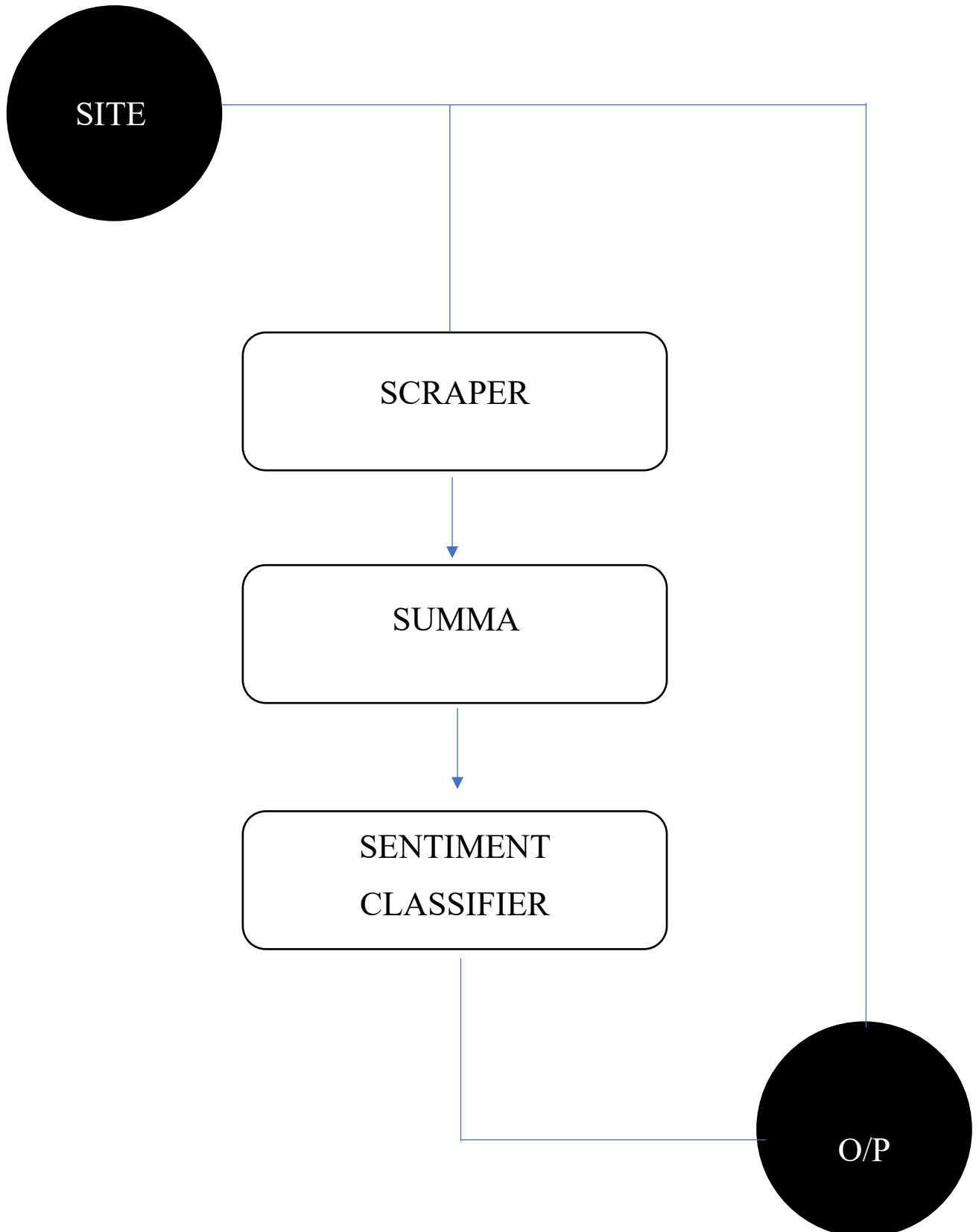
It makes use of the Natural language toolkit commonly known as 'nltk' to summarize the scraped content.

It uses various 'GUI' libraries to present a user-friendly interface.

The XENON highlights are as follows,

- A really efficient scraping tool that takes an average of 2 seconds to completely scrape a 500 words long website
- An accurate summarizer that summarizes the contents and segments them based on the context
- An intuitive sentiment analysis tool that uses the context provided by the summarizer to efficiently predict the emotional intensity of the segmented content

DIAGRAMMATIC REPRESENTATION



MODULES USED

The following are the modules used to build the scraper and the summarizer,

1. OS
2. SYS
3. BEAUTIFUL SOUP
4. NLTK
5. SOCKS
6. SOCKET
7. REQUEST
8. PLATFORM

MODULES EXPLANATION

OS

The OS module in Python provides functions for interacting with the operating system. OS comes under Python's standard utility modules. This module provides a portable way of using operating system-dependent functionality.

The `*os*` and `*os.path*` modules include many functions to interact with the file system.

Python-OS-Module Functions

Here we will discuss some important functions of the Python os module :

- Handling the Current Working Directory
- Creating a Directory
- Listing out Files and Directories with Python
- Deleting Directory or Files using Python

SYS

The sys module in Python provides various functions and variables that are used to manipulate different parts of the Python runtime environment. It allows operating on the interpreter as it provides access to the variables and functions that interact strongly with the interpreter.

BEAUTIFUL SOUP

Beautiful Soup is a Python library that allows parsing and scraping HTML and XML documents from the web, providing us with more options while navigating a hierarchical data tree. The library offers a straightforward, user-friendly API that lets you extract data from the HTML or XML code while also parsing and navigating the website.

Advantages and Disadvantages of Beautiful soup

Advantages:

The advantages of beautiful soup include:

1. It is beginner-friendly and very easy to set up.
2. It is fast.
3. The library functions without relying on browsers.
4. Parsing HTML and XML: The main purpose of Beautiful Soup is to extract data from HTML and XML files. It offers idioms in Python for searching, iterating, and altering the parse tree. Selenium, on the other hand, is more appropriate for interacting with dynamic web pages and is primarily a browser automation tool.
5. Debugging the framework is simpler, and it requires less time to run.

Disadvantages:

1. It supports just python
2. Because Beautiful Soup only allows you to browse through HTML or XML files, you'll need a different module to scrape pages built with JavaScript.

3. It can't interact with webpages.
4. BeautifulSoup can just parse data, so you will need 'Request' and 'HTTPx' to extract information.

Selenium

Selenium is an open-source framework commonly used for automating web applications for testing purposes. Selenium is also widely used for automating web scraping and other tasks involving browsers. The components of selenium include:

- Selenium IDE (Integrated Development Environment):
- Selenium IDE is a browser extension that facilitates the recording and playback of user interactions with a website.
- It is mainly used for prototyping and quick script development.
- Selenium IDE is available as a plugin for Chrome and Firefox, allowing users to record actions, edit scripts, and export them in various programming languages.

NLTK

The Natural Language Toolkit, or more commonly NLTK, is a suite of libraries and programs for symbolic and statistical natural language processing (NLP) for English written in the Python programming language. It supports classification, tokenization, stemming, tagging, parsing, and semantic reasoning functionalities.^[4] It was developed by Steven Bird and Edward Loper in the Department of Computer and Information Science at the University of Pennsylvania.^[5] NLTK includes graphical demonstrations

and sample data. It is accompanied by a book that explains the underlying concepts behind the language processing tasks supported by the toolkit,^[6] plus a cookbook.^[7]

NLTK is intended to support research and teaching in [NLP](#) or closely related areas, including empirical linguistics, cognitive science, A.I and ML. NLTK has been used successfully as a teaching tool, as an individual study tool, and as a platform for prototyping and building research systems. There are 32 universities in the US and 25 countries using NLTK in their courses.

PROGRAM

1.0

MAIN - ABSTRACTED

```
import os
```

```
import sys
```

```
import nltk
```

```
import socks
```

```
import socket
```

```
import requests
```

```
import platform
```

```
import datetime
```

```
from art import *
```

```
from bs4 import BeautifulSoup
```

```
from nltk.tokenize import sent_tokenize
```

```
from nltk.corpus import stopwords
```

```
nltk.download('punkt')
```

```
nltk.download('stopwords')
```

```

def preprocess_text(text):

    sentences = sent_tokenize(text)

    stop_words = set(stopwords.words('english'))

    filtered_sentences = [sentence for sentence in sentences if sentence.lower() not in
stop_words]

    return filtered_sentences

    response.raise_for_status()

    soup = BeautifulSoup(response.content, 'html.parser')

    text = soup.get_text()

    return text

except requests.exceptions.RequestException as e:

    print(f"Error occurred while scraping URL: {url}\n{e}")

    return None

def scrape(urls):

    for url in urls:

        text = scrape_text_from_url(url)

        print("\nURL:", url)

        print("-----")

        output_file = f"output_{datetime.datetime.now().strftime('%Y-%m-%d_%H-%M-%S')}.txt"

```

```
path = os.getcwd()
```

```
path += '.txt'
```

```
output_path = os.path.join(path, output_file)
```

```
with open(output_file, "w", encoding="utf-8") as file:
```

```
    file.write(text)
```

```
print(f'summary saved to {summary_path}')
```

```
file1.close()
```

```
def scrape_tor(url):
```

```
    socks.set_default_proxy(socks.SOCKS5, "localhost", 9050)
```

```
    socket.socket = socks.socksocket
```

```
    #response = requests.get(url)
```

```
    try:
```

```
        response = requests.get(url)
```

```
    except requests.exceptions.ConnectionError as e:
```

```
        print("Connection Error:", e)
```

```
    path = os.getcwd()
```

```
output_file = f'output_{datetime.datetime.now().strftime('%Y-%m-%d_%H-%M-%S')}.txt'
```

```
path = os.getcwd()
```

```
path += '.txt'
```

```
output_path = os.path.join(path, output_file)
```

```
with open(output_file, "w", encoding="utf-8") as file:
```

```
    file.write(text)
```

```
print(f'Output saved to {output_path}')
```

```
print("-----")
```

```
file.close()
```

```
print("BEGINNING PHASE 2")
```

```
tprint("\nSUMMARIZING CONTENT\n")
```

```
summary = summarize_text(text)
```

```
summary_file = f'summary_{datetime.datetime.now().strftime('%Y-%m-%d_%H-%M-%S')}.txt'
```

```
path1 = os.getcwd()
```

```
path1 += '.txt'
```

```
summary_path = os.path.join(path1, summary_file)
```

```
with open(summary_file, "w", encoding="utf-8") as file1:
```

```
file1.write(summary)
```

```
file1.close()
```

```
print(f"summary saved to {summary_path}")
```

```
print("-----")
```

```
print("\nMODE SET TO { CLEARNET }\n")
```

```
print("-----")
```

```
url = input("\nPaste the URL: ")
```

```
print("-----")
```

```
print("BEGINNING PHASE 1")
```

```
tprint(f"\nSCRAPING URL\n",font="mini")
```

```
scrape([url])
```

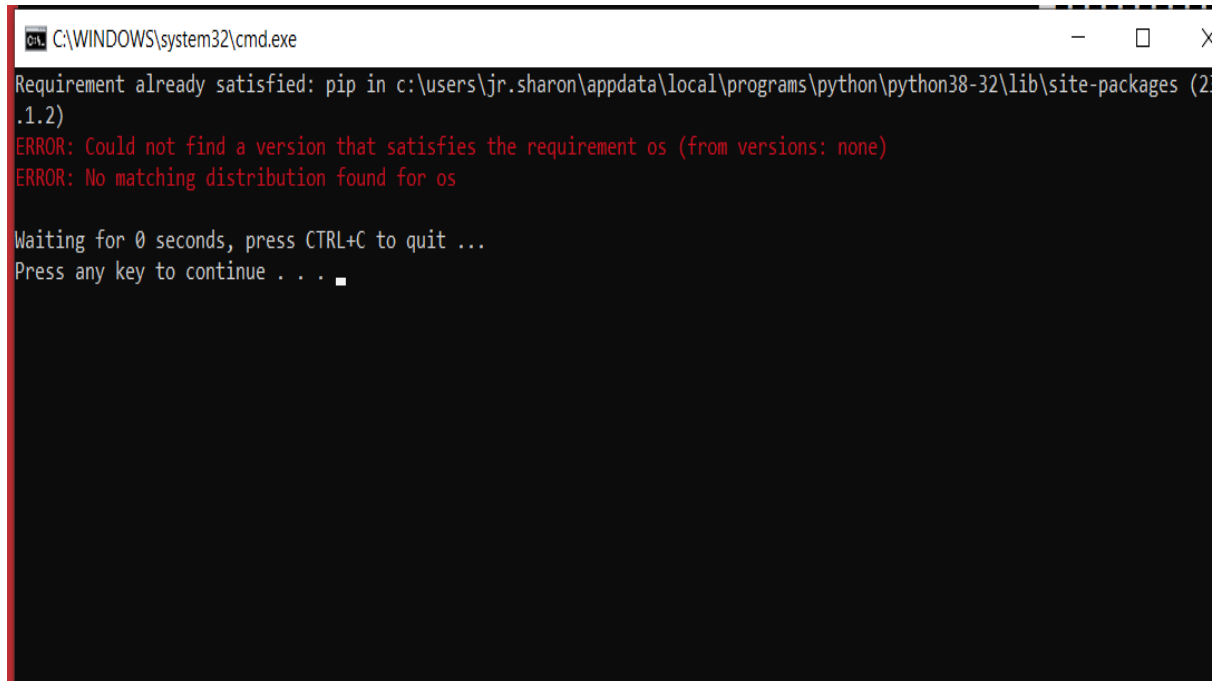
```
print("-----")
```

```
print("\nSite scraped successfully")
```

```
q = input("PLEASE PROVIDE FEEDBACK: ")
```

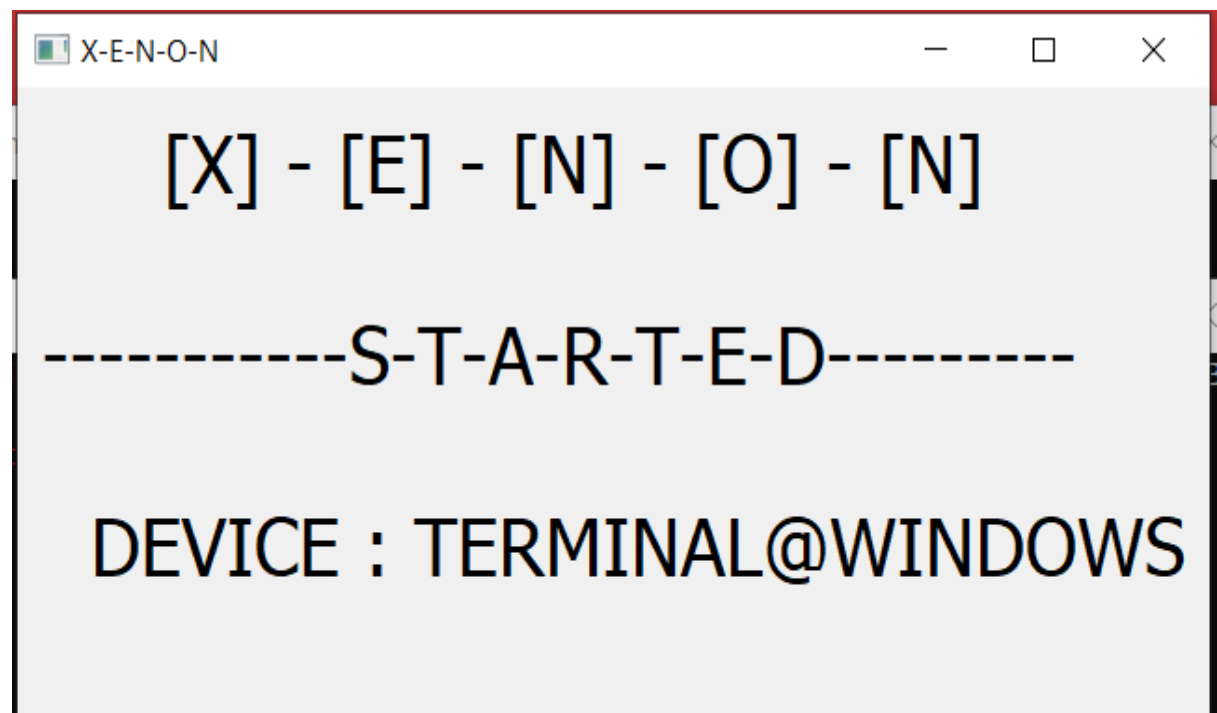
SAMPLE OUTPUT

INITIAL STATE



```
C:\WINDOWS\system32\cmd.exe
Requirement already satisfied: pip in c:\users\jr.sharon\AppData\Local\Programs\Python\Python38-32\lib\site-packages (2.1.2)
ERROR: Could not find a version that satisfies the requirement os (from versions: none)
ERROR: No matching distribution found for os

Waiting for 0 seconds, press CTRL+C to quit ...
Press any key to continue . . . █
```



```
X-E-N-O-N

[X] - [E] - [N] - [O] - [N]

-----S-T-A-R-T-E-D-----

DEVICE : TERMINAL@WINDOWS
```

THE INTERFACE

```
[nltk_data] Downloading package punkt to
[nltk_data]   C:\Users\JR.SHARON\AppData\Roaming\nltk_data...
[nltk_data] Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]   C:\Users\JR.SHARON\AppData\Roaming\nltk_data...
[nltk_data] Package stopwords is already up-to-date!
```

```
-----
      8.8888.      ,8' 8 88888888888 b.      8      ,o888888o.      b.      8
      8.8888.      ,8' 8 8888      888o.      8 . 8888      `88. 888o.      8
      8.8888.      ,8' 8 8888      Y88888o.      8 ,8 8888      `8b Y88888o.      8
      8.8888.,8' 8 8888      .Y888888o.      8 88 8888      `8b .Y888888o.      8
      8.88888' 8 8888888888888 8o. `Y888888o. 8 88 8888      88 8o. `Y888888o. 8
      .88.`8888. 8 8888      8`Y8o. `Y88888o8 88 8888      88 8`Y8o. `Y88888o8
      .8`8.8888. 8 8888      8 `Y8o. `Y8888 88 8888      ,8P 8 `Y8o. `Y8888
      .8' `8.8888. 8 8888      8 `Y8o. `Y8 `8 8888      ,8P 8 `Y8o. `Y8
      .8' `8.8888. 8 8888      8 `Y8o. ` 8888      ,88' 8 `Y8o. `
      .8' `8.8888. 8 88888888888 8 `Yo `8888888P' 8 `Yo
```

THE SCRAPER AND SUMMARIZATION FRAMEWORK

MODE-SELECTION

Press [0] for [[clearnet]] and [1] for [[darknet]] -> 0

MODE SET TO { CLEARNET }

```
-----
Paste the URL: https://ejbpc.springeropen.com/articles/10.1186/s41938-023-00696-x
-----
```



```
BEGINNING PHASE 1

SCRAPING URL

URL: https://ejbpc.springeropen.com/articles/10.1186/s41938-023-00696-x
-----
Output saved to C:\Users\JR.SHARON\Desktop.txt\output_2023-05-15_08-53-24.txt
-----
BEGINNING PHASE 2

SUMMARIZING CONTENT

summary saved to C:\Users\JR.SHARON\Desktop.txt\summary_2023-05-15_08-53-24.txt
-----

Site scraped successfully
PLEASE PROVIDE FEEDBACK: ☐
```

EXAMPLE URL GIVEN

<https://ejbpc.springeropen.com/articles/10.1186/s41938-023-00696-x>

SCRAPED CONTENT

AbstractBackgroundMycotoxins are secondary metabolites made by a variety of molds and fungi. They contaminate a lot of food products and local crops during pre- and post-harvesting under favorable conditions like high temperature and moisture. Aspergillus species are the most common fungi that contaminate food and produce biochemicals known as mycotoxins. Aflatoxins (AFB1, AFB2, AFG1, and AFG2) are the major mycotoxins produced by *A. flavus* and *A. parasiticus* that harm animal and human health. These fungi are controlled by chemical fungicides, but these are harmful to the environment. The aim of this study was to determine whether the aflatoxigenic fungi can be exterminated only by marine algal extracts or not.

ResultsThe findings showed that the tested seaweed extracts inhibited fungal growth and aflatoxins production to varying degrees. The maximum antifungal activity was recorded in *Halimeda opuntia* extract against *A. parasiticus*-24 and *A. flavus*-18 and *Turbunaria decurrens* extract against *A. flavus*-18 (with an inhibition percentage of 77.78%), followed by *Jania rubens* extract against *A. parasiticus*-16 with inhibition percentage 75.88% compared to the control. Aqueous extract of *H. opuntia* effectively eliminated aflatoxins (B1, B2, G1, and G2) in *A. parasiticus*-16 and *A. parasiticus*-24. *T. decurrens* extract could detoxify 100% of aflatoxins in three isolates of *A. parasiticus*. *J. rubens* extract eliminated aflatoxins in *A. parasiticus*-15 and *A. parasiticus*-16 compared to their normal production using high-performance liquid chromatography.**Conclusions**According to this study, the macroalgal species with numerous distinctive antifungal properties constituents significantly inhibited the growth and production of aflatoxin in *A. parasiticus* and *A. flavus* isolates. The findings supported the use of macroalgae as a biological control agent against fungi and their toxins.

SUMMARY

Abstract**Background**Mycotoxins are secondary metabolites made by a variety of molds and fungi. They contaminate a lot of food products and local crops during pre- and post-harvesting under favorable conditions like high temperature and moisture. *Aspergillus* species are the most common fungi that contaminate food and produce biochemicals known as mycotoxins

CONCLUSION

The XENON project is still in development with goals for integration with the transformer architecture to custom train LLM models.

OFFICIAL REPOSITORY:

<https://github.com/Francis-sharon/XENON>

REFERENCES FOR THE PROJECT

<https://towardsdatascience.com/why-text-summarization-is-still-hard-39e5559db86>

<https://realpython.com/python-web-scraping-practical-introduction/>

<https://realpython.com/beautiful-soup-web-scraper-python/>

<https://www.analyticsvidhya.com/blog/2021/10/a-detailed-guide-on-web-scraping-using-python-framework/>

<https://www.geeksforgeeks.org/python-text-summarizer/>

<https://www.mygreatlearning.com/blog/text-summarization-in-python/>

<https://towardsdatascience.com/understand-text-summarization-and-create-your-own-summarizer-in-python-b26a9f09fc70>