National College of Ireland

Advanced Google Scholar Scraper: A Content-Based Filtering Approach for Literature Recommendation Using BERT

MSc Research Project
Artificial Intelligence

Praneeth Bandi
Student ID: 22183922

School of Computing
National College of Ireland

Supervisor: Rejwnaul Haque

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Praneeth Bandi
Student ID:	22183922
Programme:	Artificial Intelligence
Year:	2023
Module:	MSc Research Project
Supervisor:	Rejwanul Haque
Submission Due Date:	05/01/2024
Project Title:	Advanced Google Scholar Scraper: A Content-Based Filtering Approach for Literature Recommendation Using BERT
Word Count:	6327
Page Count:	20

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Praneeth Bandi
Date:	05th January 2023

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Advanced Google Scholar Scraper: A Content-Based Filtering Approach for Literature Recommendation Using BERT

Praneeth Bandi
22183922

Abstract

Advanced Google Scholar Scraper is a complex recommendation system for reading materials. It employs various techniques including web scraping, natural language processing, and content-based filtering. It uses Selenium, Beautiful Soup, and the Hugging Face Transformers library (with a focus on BERT) to make literature referrals more accurate and relevant. The scraper was developed to provide researchers, students, and practitioners with a simple but flexible tool that will allow them to find relevant articles across all fields. Domains include Natural Language Processing (NLP), Machine Learning (ML), and BERT Models. Trying to provide contextually and semantically accurate recommendations, the system is based on BERT embeddings and cosine similarity metrics. An assessment of the scraper verifies its capacity to collect articles from specific domains and offers examples of successful applications for natural language processing methods and web scraping functions. The results show high similarity scores in different fields of research and are timely. The results are that the Advanced Google Scholar Scraper succeeds in getting over these obstacles for dynamic Web scraping, error handling, and user interface design. This is one solution appropriate to all the applications of literature suggestions. The scraper's adaptability, real-time progress monitoring, and error tolerance make it an extremely useful tool in many research environments.

1 Introduction

With web scraping having been applied to so many different fields, over the last few years, it has received attention that never would have come its way. It is an essential tool for gathering information from websites. Experts in web scraping Under the negative impact of modern methods, fields as remote from one another as robotics and criminal activity have been studied (Gheorghe et al. (2018); Krotov and Silva (2018)) since then. (Singrodia et al.) Legal and moral concerns related to web scraping have become another center of attention, focusing on problems existing in this activity (Bhardwaj et al. (2021); Diouf et al. (2019)) These studies are an important step toward developing a nuanced conception of web scraping and its wider impact.

The spread of methods for web scraping, however, has been propelled by the ever-increasing need to collect data in all fields efficiently. The use of web scraping has been explored in many applications such as extraction of weather data (Dewi et al. (2018)),

and big data analytics. Cloud-based web scraping has also lent itself as a way to process data at scale (Uriawan et al. (2020); Matta et al. (2022)). As for using the output of Web crawlers within content analysis studies on social networks like Facebook or Instagram. In addition, a comparative study of scraping tools has yielded useful clues concerning the respective advantages and disadvantages in implementation (Chaudhari et al. (2020)). For all researchers and practitioners in data science, the most crucial thing is understanding web scraping. It's a tremendously effective tool for collecting, analyzing, and deriving insights from immense datasets sitting on the servers at giant companies such as Uber or Yahoo. With the insights gleaned from web scraping, people in many different fields can make sounder judgments and better planning with strategy. As a result, it certainly can be considered an essential part of a data scientist's equipment kit. In research paper recommender systems, where the identification of relevant information is a crucial first step in accurate recommendations (MOHAMED (2020); Putrama and Martinek (2023)), it has particular significance.

Our overarching research question is concerned with discovering the most advanced methods, ethical considerations and applications of web-scraping. Its primary objective is to give a comprehensive overview of the field, examining legality and ethics and applications in practice. These objectives involve analyzing the methods involved, dealing with ethical issues, and describing applications studied in recent research. This paper will help our understanding of the multifaceted web scraping landscape.

Acknowledging that web scraping is an important tool, it is also important to recognize the limitations and assumptions inherent within such a practice. One of these limitations could involve data privacy, or possibly legal restrictions, or the fact that websites are constantly changing. One assumption concerns the ethics of web scraping. The other is about information retrieved and validity. Nonetheless, web scraping plays a vital role in harvesting valuable leaves.

Briefly, using this broad introduction we set the stage for exploring web scraping, pinpoint its importance and varied usage, and establish our premises for discussions about the challenges connected to it and ethical concerns.

2 Literature Review

A glimpse into web scraping or crawling is an essential practice. Specialized software tools are used to collect data from websites by this method. They emphasize how critical online scraping has become today, particularly for businesses that must have access to structured data. This study also points out that when one cannot get data by machine-readable formats like XML or JSON, the value of online scraping is very great indeed. It can help provide real-time access to pricing on retail websites and even collect intelligence for law enforcement on covert activities such as those on drug marketplaces in the dark net. The authors point out that in the information age, online scraping is a very effective and useful tool because the data it provides is more complete and accurate than hand-entered data.

Secondly, it highlights how important code reuse and maintenance are in the case of

online scraping. The reuse of code is stressed as a key approach that allows all the code written before to be used in project after project, especially when visiting websites. Webs are inherently dynamic things, and changes to behavior or appearance can make extraction more difficult; both bring up stress on how the new code will have been maintained. The authors point out that Python is the best programming language for carrying out online scraping because it is easy to pick up and allows the construction of a huge library of reusable code, which accelerates the development and deployment of spider applications. The in-depth examination highlights the moral and jurisprudential dilemmas that must be resolved, while further illuminating our grasp of web scraping and its implications both positive and negative (Khder (2021)).

This novel way of collecting data for the "Istat sampling survey on 'ICT in enterprises'" is described. The objective of this innovative research was to provide an in-depth analysis of how Italian businesses employ information and communication technology (ICT), paying special attention to the use of the Internet for functions ranging from e-commerce through e-government, e-tendering, e-recruitment, and advertising. Researchers therefore looked at how to substitute text and data mining methods, as well as online scraping equipment for regular surveys' processes of collecting data and estimation procedures. Taking answers to an ICT survey from 2013, data were extracted from 8600 websites and then processed to yield essentially the same information that would have resulted had we used traditional questionnaires. The preliminary results suggested that a number of models, particularly those using Naïve Bayes algorithms, had adequate predictive power. The Content Analysis method was also adopted and compared with the traditional teaching approach. To speed up the whole process, an advanced mining and scraping system based on the open-source Apache suite Nutch-Solr-Lucene was implemented. Based on the study's final results, the researchers intended to build a system that integrated survey data and Internet data to estimate needed figures as efficiently as possible (Barcaroli et al. (2016)).

We considered the expansive geography of the internet as a great storehouse of human-generated data. They pointed out that although there is a lot of relevant information on the Internet, it's usually scattered and disorganized, making it difficult to gather physically and apply in automatic procedures. To address the obstacles, researchers studied the field of scraping online, explaining its basic ideas and analyzing what several programs and services offer for this purpose. The study on the development and application of web scraping helped readers understand some of the subtleties of web scraping. Furthermore, the author comprehensively covered the web scraping procedure with a detailed explanation for different ways to get important information from sites or other online resources. A balanced evaluation It concluded with a comprehensive discussion of the advantages and disadvantages associated with online scraping. For an in-depth look at the issues, the writers also examined many applications in which online scraping can be used. They particularly noted its potential for application to such areas as Big Data, Open Government Data, and Business Intelligence; building creative apps and mashups (Singrodia et al. (2019)). One issue that numerous cities in South Sumatra faced as a result of the multiple weather measurement locations operated by different agencies, including BMKG, AngkasaPura, Lapan, and others was resolved. With these measuring capabilities bureaucratic procedures involving several agencies prevented you from getting the most recent and comprehensive weather data in time. It was a problem for scientists

and analysts, who use current weather datasets for weather forecasting or decision support system (DSS) analysis–much of which is dependent on detailed information about the atmosphere.

All the information available in today's digital world–as summed up by Donny Miller via the assertion that In this era of information, ignorance is a choice (Bhardwaj et al. (2021))–makes it clear that compiling and processing data is one of the most important things. Because the internet is so widespread, it has become an unprecedented reservoir of data and knowledge. The outcome is that several industries, covering the financial markets, corporate businesses, and internet companies themselves, have taken to using advanced tools and methods to dig up information from the web. This is one of these methods that has become very popular and made the process of extracting data from webpages much easier than before, without you needing to have an in-depth grasp on the Document Object Model (DOM) structure. However, with the increase in demand for large-scale data extraction, captcha error problems have occurred, along with storage restrictions and processing requirements. Data dependability has also turned into a bottleneck, demanding inventive solutions. Interestingly, a cloud-based web scraper architecture has been developed (Chaulagain et al. (2017)) which uses AWS for elastic storage and computing resources. In addition, web scraping is complex and computationally demanding, so automated tools are becoming more and more popular. Thus different techniques and tools were born which this review will take inventory and classify while describing the numerous applications of web scraping.

According to the Indonesian Internet Service Providers Association (APJII), more and more people in Indonesia are logging on. This is why it is so important to understand online consumer behaviour, especially on popular social media like Instagram. Since its widespread use, Instagram has become an outlet to share information, particularly religious information. Engagement metrics, such as likes on individual videos, have now become a way of measuring user preferences. Web scraping techniques are used to gather data related to Instagram video postings with Islamic themes. We use the Pearson correlation method to determine the degree of relationship between variables. More precisely, the relationship between how many movies were shown and how many likes they collected is scrutinized. With this methodology, the study hopes to establish a standard that allows pastors to determine the kinds of postings that Instagram users most enjoy. This will provide valuable ideas about how the internet has been shaping religious material (Uriawan et al. (2020)).

In today's dynamically advancing technological world, data–structured or unstructured and polished or refined–is the single most critical resource. These different forms of data all coalesced together to form the idea of Big Data, and what distinguishes them is that large in volume and fast in speed. This flood of information has become a problem for society. One must understand the power of historical data and see how it concerns different fields. One important step to this project is the discipline of data extraction the break-through procedure which has altered our conception of the world. In that environment, scraping the web becomes an essential instrument for promoting corporate growth, helping scientific research, and offering insight into popular topics on the Web. The most potent web scraping technologies currently available are scrutinized. A comparison is made that describes the unique advantages and disadvantages of each in

different applications. With technology reshaping the way we interact with and extract value from data, an understanding of these technologies is key to navigating the difficult information retrieval and use landscape for people and organizations (Matta et al. (2022)).

In today's age of hurried lives and random eating habits, the requirement for customized diet regimes has intensified. One large problem people face when attempting to follow specific ingredient guidelines is how to find a variety of recipes that fit their dietary requirements. There are plenty of websites that offer recipes but a big problem is there's no specific information about ingredient amounts or possible side effects. This work fills this void by proposing an algorithm that uses Python, MongoDB, and web scraping methods to gather rich recipe information from several sources. Its integration with MongoDB helps get a structure module up on the Internet, which allows for further research and development of an online smart chef application dedicated to foods in accordance with a healthy diet. This suggested procedure makes it easier to find recipes that contain certain ingredients as well as making extraction of recipe particulars a breeze. What they produce through their labor is a web and mobile application that powers people by giving them the means to create meals of many kinds which help maintain health. This paper is a contribution to the field of technology and nutrition. This solution integrates dietary needs with culinary exploration to meet the demands of more knowledgeable and health-conscious consumers (Chaudhari et al. (2020)). The writers overcame these obstacles with web scraping technology to fill the need for weather data. Like this, they obtained real-time weather information from dozens of websites related to cities in South Sumatra and nearby regions. Information can then be scraped off websites selectively using web scrapping technology. The information can be maintained in a shared database or data warehouse. South Sumatra Specific Data Mining and Weather Forecast (Dewi et al. (2018)). Secondly, this weather data repository can be used for further research and analysis. In the field of research paper recommender systems, there's been quite a lot thrown at it over recent years. Motivated to increase the efficiency and accuracy of research paper recommender systems, MOHAMED (2020) went exploring deep learning models. Through extracting complex patterns and hidden features from scientific articles with deep learning algorithms, the recommendation process is improved.

Putrama and Martinek (2023) proposes an innovative idea of integrating platforms through content-based graph representation learning. A work published in the International Journal of Information Management Data Insights points out that with graph-based representations, recommended articles must be both more connected and relevant. Proceeding in this way fosters a wider appreciation of the relationships among research papers, which thus becomes most pertinent to any particular situation. The techniques of natural language processing applied to content-based book recommender systems were covered in depth by Berbatova (2019). Here we can see that NLP is capable of helping in the extraction of semantic meaning, thus increasing the accuracy of content-based filtering systems. An autonomous multi-agent system for personalized scientific literature recommendation was introduced by Herrouz et al. (2023) Their tool has proven most popular with academic users, researchers and students. Its strongest feature is its adaptability to different usages. The introduction of autonomous agents into the process allows for a new degree of personalization. To users, literature recommendations are only related to their special interests. In terms of the field of interactively exploring scientific publications, Horn (2017) proposed a radical idea–PubVis. The use of this interactive

tool enables users to jump from one big repository to another and locate related papers easily. By using visualization techniques, PubVis is an easy-to-use means of glancing at and identifying relevant research papers.

Totla et al. (2021) proposed a method of employing topic modeling for testing project idea similarity. This contribution was given at the 2021 International Conference on Advances in Computing, Communication, and Control. It refers to an application for idea similarity evaluation of a project itself used by practice in topic modeling. If such methodologies are adopted, it could refine the recommendation algorithm along thematic lines.

Houshmand et al. (2023) carried out a study on NLP of social media data and strategic decision-making for pedagogical course planning The necessity to extract vital information from social media data and use natural language processing tools to allow strategic decisions on educational planning is clear. The use of NLP in this case shows that language processing techniques are not limited to the traditional literary repositories.

The authors studied in detail how web scraping has already been used by numerous researchers. It shows how online scraping is a highly important way of gathering information for research and business alike. The paper stresses the ethicality and legality of web scraping, and suggests that to avoid further disputes, researchers should first deal with these questions.

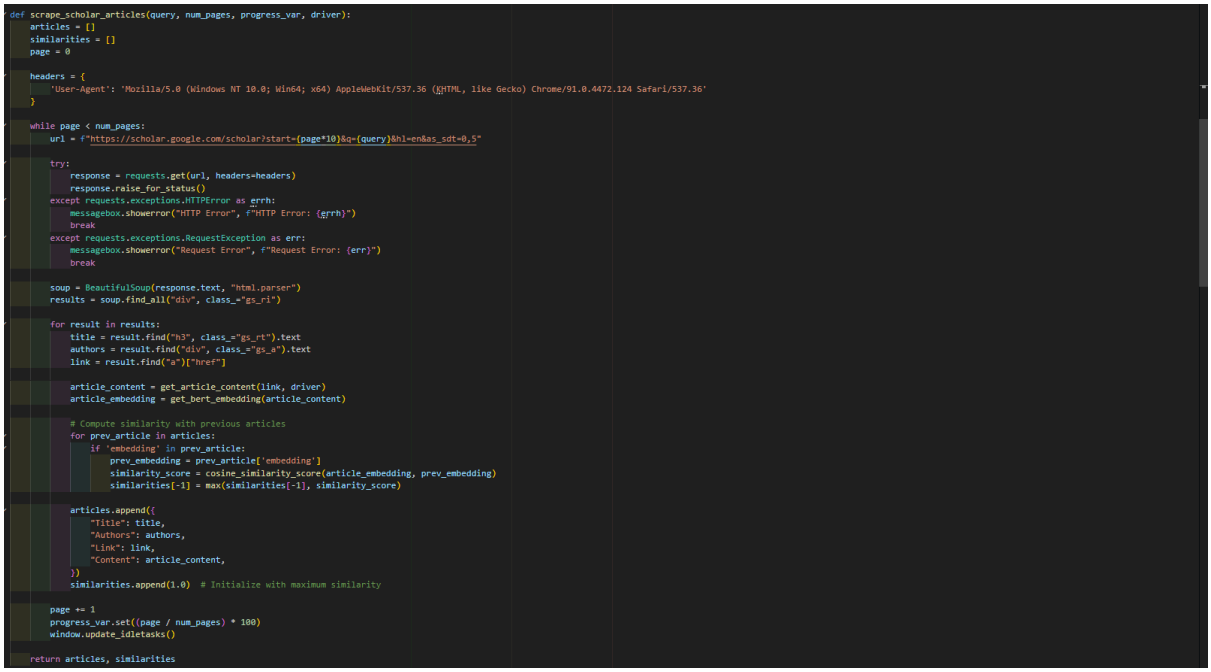
Finally, the paper presents a comprehensive and enlightening review of how web scraping is used in research by scientists scholarly as well as commercial. The worked-over research spanning the years 2014 to those just past suggests how online scraping methods have developed and today are very important in a data society. While web scraping is sometimes ethically and legally questionable, it has practical applications from business intelligence to weather forecasts. Web scraping also forms a vital means of data extraction in the boundless world that is the Internet. Research by a lot of people has shown that the moral and legal issues posed by web scraping need to be further debated to formulate ethical and lawful ways of collecting data. What's more, the applications of web scraping in data mining and government data use are increasing at breakneck speed. So you can see just how important a role this is playing within an information-age society. Despite the changes in digital landscapes, web scraping remains an important instrument for researchers and organizations to collect as much unstructured data from the four corners of the internet world as possible.

3 Methodology

Implemented in Python using various libraries, the advanced Google Scholar scraper is supposed to retrieve scholarly articles based on a user-entered query. The methodology involves several steps such as web scraping, BERT embeddings-based natural language processing (NLP), and data storage in CSV format. This section introduces the main components of the methodology. We'll first take a look at an overview workflow for our solution, then delve into how we integrate BERT embeddings with other methods and describe some error-handling strategies, finally wrapping up by showing readers how to construct a user-friendly interface using Tkinter

3.1 Web Scraping Workflow:

The real function of the scraper is to crawl along Google Scholar's search results, carve out only meaningful information from their HTML code, and store it for future computation.



```
def scrape_scholar_articles(query, num_pages, progress_var, driver):
    articles = []
    similarities = []
    page = 0

    headers = {
        'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.124 Safari/537.36'
    }

    while page < num_pages:
        url = f"https://scholar.google.com/scholar?start={page*10}&q={query}&hl=en&as_sdt=0,5"

        try:
            response = requests.get(url, headers=headers)
            response.raise_for_status()
        except requests.exceptions.HTTPError as errh:
            messagebox.showerror("HTTP Error", f"HTTP Error: {errh}")
            break
        except requests.exceptions.RequestException as err:
            messagebox.showerror("Request Error", f"Request Error: {err}")
            break

        soup = BeautifulSoup(response.text, "html.parser")
        results = soup.find_all("div", class_="gs_ri")

        for result in results:
            title = result.find("h3", class_="gs_rt").text
            authors = result.find("div", class_="gs_a").text
            link = result.find("a")["href"]

            article_content = get_article_content(link, driver)
            article_embedding = get_bert_embedding(article_content)

            # Compute similarity with previous articles
            for prev_article in articles:
                if 'embedding' in prev_article:
                    prev_embedding = prev_article['embedding']
                    similarity_score = cosine_similarity_score(article_embedding, prev_embedding)
                    similarities[-1] = max(similarities[-1], similarity_score)

            articles.append({
                "Title": title,
                "Authors": authors,
                "Link": link,
                "Content": article_content,
            })
            similarities.append(1.0)  # Initialize with maximum similarity

        page += 1
        progress_var.set((page / num_pages) * 100)
        window.update_idletasks()

    return articles, similarities
```

Figure 1: Web Scraping Workflow

The script relies on the BeautifulSoup library to parse HTML content, extracting things like article titles and authors 'names as well as snippets. Based on the user's instructions, this process is repeated over several pages. You end up with a complete set of articles on your topic.

3.2 BERT Embeddings for Content Analysis:

Its script form deepens the level of information extraction, using BERT (Bidirectional Encoder Representations from Transformers) embeddings.

```
# Initialize BERT model and tokenizer
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
model = BertModel.from_pretrained('bert-base-uncased')

def get_bert_embedding(text):
    input_ids = tokenizer.encode(text, return_tensors="pt", truncation=True)
    with torch.no_grad():
        outputs = model(input_ids)
    return outputs.last_hidden_state.mean(dim=1).squeeze().tolist()
```

Figure 2: Bert Embeddings

The transformers library offers support for integrating a pre-trained BERT model and tokenizer. Each article's content is tokenized and fed into the BERT model to get embeddings. These embeddings are numerical representations of textual content, which let us perform complex analyses like calculating cosine similarity.

3.3 Cosine Similarity for Article Comparison:

The script uses cosine similarity as a measure to determine the degree of similarity between embeddings for two articles.

```
def cosine_similarity_score(embedding1, embedding2):
    return cosine_similarity([embedding1], [embedding2])[0][0]
```

Figure 3: Similarity Analysis

It records the most similar articles by comparing each new article with those previously extracted. By this means, the collected data is of higher quality due to duplicate or highly similar content being properly flagged. Each article's similarity scores are stored in a list so they can be easily looked up when the CSV file is being created.

3.4 Error Handling and Robustness:

Some uncertainties are inherent to web scraping: the website structure may vary from one occasion to another, and you might unexpectedly make a mistake during content extraction.

```
while page < num_pages:
    url = f"https://scholar.google.com/scholar?start={page*10}&q={query}&hl=en&as_sdt=0,5"

    try:
        response = requests.get(url, headers=headers)
        response.raise_for_status()
    except requests.exceptions.HTTPError as errh:
        messagebox.showerror("HTTP Error", f"HTTP Error: {errh}")
        break
    except requests.exceptions.RequestException as err:
        messagebox.showerror("Request Error", f"Request Error: {err}")
        break
```

Figure 4: Error Handling

These challenges are addressed by the script's strong error-handling capabilities. For example, while trying to gather article content from external links the script uses exception handling so that failure is managed as gracefully as possible. This means that the scraper can recover from failures and continue running, preserving data integrity.

3.5 User Interface with Tkinter:

A graphical user interface (GUI) built into the Tkinter library was added to provide greater accessibility to a wider audience. Optionally specify the number of pages to scrape and choose an output folder. Only article title or keyword input is required.

```
# Main program
window = tk.Tk()
window.title("Advanced Google Scholar Scraper")
window.geometry("500x300")

label_query = tk.Label(window, text="Article Title or Keyword:")
label_query.grid(row=0, column=0, padx=10, pady=5)
entry_query = tk.Entry(window, width=40)
entry_query.grid(row=0, column=1, padx=10, pady=5)

label_pages = tk.Label(window, text="Number of Pages:")
label_pages.grid(row=1, column=0, padx=10, pady=5)
entry_pages = tk.Entry(window, width=40)
entry_pages.grid(row=1, column=1, padx=10, pady=5)

label_folder = tk.Label(window, text="Output Folder (optional):")
label_folder.grid(row=2, column=0, padx=10, pady=5)
entry_folder = tk.Entry(window, width=40)
entry_folder.grid(row=2, column=1, padx=10, pady=5)

button_browse = tk.Button(window, text="Browse", command=browse_folder)
button_browse.grid(row=2, column=2, padx=5, pady=5)

button_extract = tk.Button(window, text="Extract Data", command=scrape_articles)
button_extract.grid(row=3, column=1, pady=10)
label_status = tk.Label(window, text="", wraplength=400)
label_status.grid(row=4, column=0, columnspan=3, pady=5)

progress_var = tk.DoubleVar()
progress_bar = ttk.Progressbar(window, variable=progress_var, length=400, mode="determinate")
progress_bar.grid(row=5, column=0, columnspan=3, pady=5)

window.mainloop()
```

Figure 5: User Interface using Tkinter

The GUI has a progress bar to inform users of the scraping status. An intuitive design increases usability, enabling even those with little or no programming experience to take advantage of the potential power behind Google Scholar.

3.6 Data Storage in CSV Format:

After the scraping process is finished, the script stores all extracted information in a CSV file.

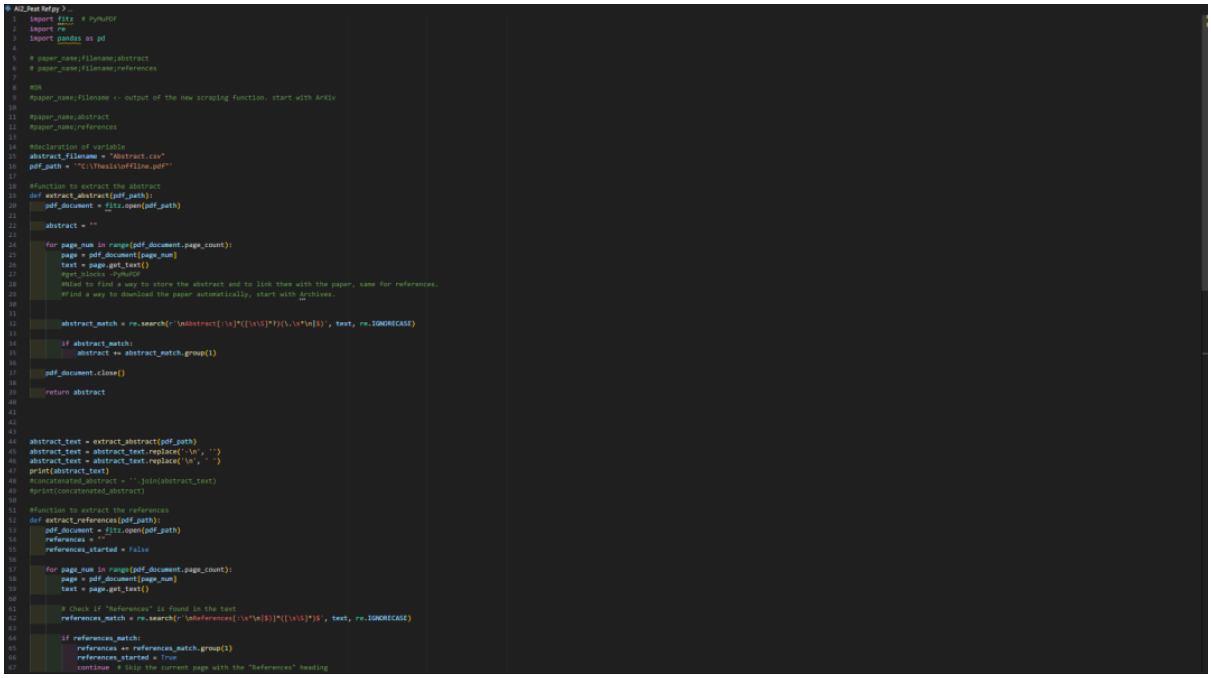
```python
def save_to_csv(articles, similarities, filename):
    with open(filename, 'w', newline='', encoding='utf-8') as csvfile:
        fieldnames = ['Title', 'Authors', 'Link', 'Content', 'Similarity']
        writer = csv.DictWriter(csvfile, fieldnames=fieldnames)

        writer.writeheader()
        for article, similarity in zip(articles, similarities):
            writer.writerow({
                'Title': article['Title'],
                'Authors': article['Authors'],
                'Link': article['Link'],
                'Content': article['Content'],
                'Similarity': similarity,
            })
```

Figure 6: Storing Articles in csv

CSV file contains entries for article title, author link, content, and similarity scores. This format is conducive to both data analysis and the further study of these collected scholarly articles.

3.7 Evolution from Draft Code

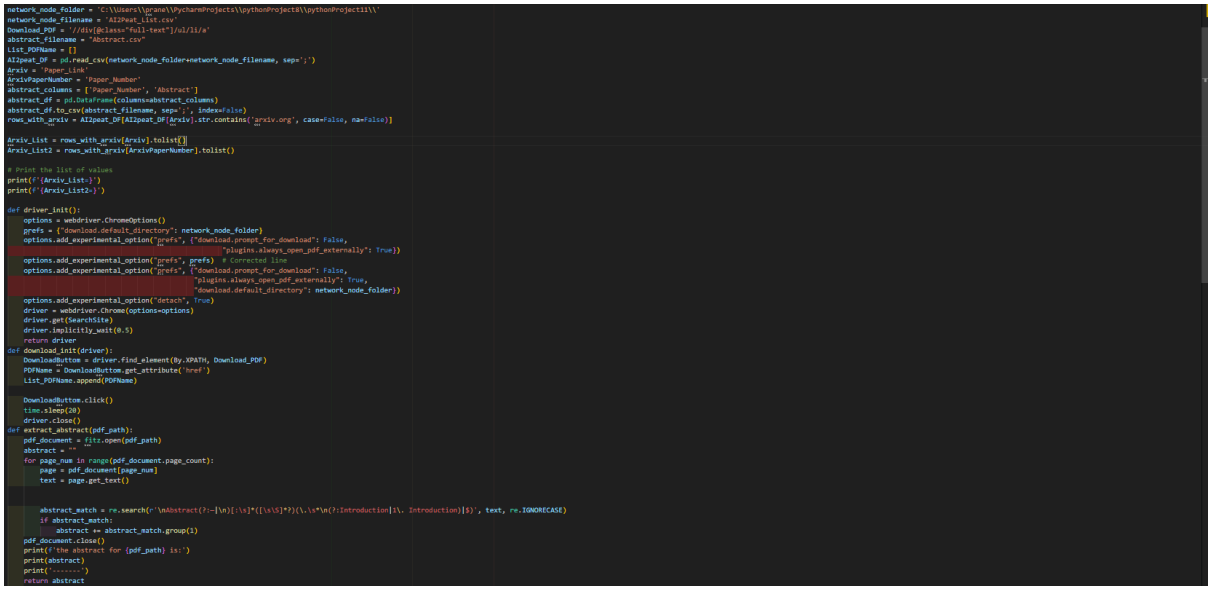
From the draft code to the final implementation of Google Scholar's most advanced scraper serves as a large upgrade in terms of scope and function. This first draft code is very limited in scope, as it only deals with extracting abstracts and references from PDF documents. However because a more comprehensive tool was needed for scholarly research, the code underwent substantial changes and additions.



```
AI2_Peat Ref.py 2
import fitz  # PyMuPDF
import re
import pandas as pd

# paper_name;Filename;abstract
# paper_name;Filename;references

#OR
#paper_name;filename <- output of the new scraping function. start with Arxiv

#paper_name;abstract
#paper_name;references

#declaration of variable
abstract_filename = "Abstract.csv"
pdf_path = r"C:\Thesis\offline.pdf"

#Function to extract the abstract
def extract_abstract(pdf_path):
    pdf_document = fitz.open(pdf_path)

    abstract = ""

    for page_num in range(pdf_document.page_count):
        page = pdf_document[page_num]
        text = page.get_text()
        #get_blocks -PyMuPDF
        #Need to find a way to store the abstract and to link them with the paper, same for references.
        #Find a way to download the paper automatically, start with Archives.


        abstract_match = re.search(r'\nAbstract[:\s]*([\s\S]*?)(\.\s*\n|$)', text, re.IGNORECASE)

        if abstract_match:
            abstract += abstract_match.group(1)

    pdf_document.close()

    return abstract



abstract_text = extract_abstract(pdf_path)
abstract_text = abstract_text.replace('-\n', '')
abstract_text = abstract_text.replace('\n', ' ')
print(abstract_text)
#concatenated_abstract = ''.join(abstract_text)
#print(concatenated_abstract)

#function to extract the references
def extract_references(pdf_path):
    pdf_document = fitz.open(pdf_path)
    references = ""
    references_started = False

    for page_num in range(pdf_document.page_count):
        page = pdf_document[page_num]
        text = page.get_text()

        # Check if "References" is found in the text
        references_match = re.search(r'\nReferences[:\s*\n|$)]*([\s\S]*)$', text, re.IGNORECASE)

        if references_match:
            references += references_match.group(1)
            references_started = True
            continue  # Skip the current page with the "References" heading
```

Figure 7: Intitial Code 1

The first version of the code, dubbed "AI2_peat Ref.py", was aimed at a certain goal: to scrape abstracts and references from PDF documents. It employed the PyMuPDF library to cycle through PDF pages, and pull out abstracts with regular expressions and references using predefined patterns. It was just adequate for its intended function, but not versatile enough to serve as a more general scholarly looking and cutting instrument.



```
network_node_folder = 'C:\\Users\\prane\\PycharmProjects\\pythonProject8\\pythonProject11\\'
network_node_filename = 'AI2Peat_List.csv'
Download_PDF = '//div[@class="full-text"]/ul/li/a'
abstract_Filename = 'Abstract.csv'
List_PDFName = []
AI2peat_DF = pd.read_csv(network_node_folder+network_node_filename, sep=';')
Arxiv = 'Paper_Link'
ArxivPaperNumber = 'Paper_Number'
abstract_columns = ['Paper_Number', 'Abstract']
abstract_df = pd.DataFrame(columns=abstract_columns)
abstract_df.to_csv(abstract_filename, sep=';', index=False)
rows_with_arxiv = AI2peat_DF[AI2peat_DF[Arxiv].str.contains('arxiv.org', case=False, na=False)]

Arxiv_List = rows_with_arxiv[Arxiv].tolist()
Arxiv_List2 = rows_with_arxiv[ArxivPaperNumber].tolist()

# Print the list of values
print(f'{Arxiv_List=}')
print(f'{Arxiv_List2=}')

def driver_init():
    options = webdriver.ChromeOptions()
    prefs = {"download.default_directory": network_node_folder}
    options.add_experimental_option("prefs", {"download.prompt_for_download": False,
                                              "plugins.always_open_pdf_externally": True})
    options.add_experimental_option("prefs", prefs)  # Corrected line
    options.add_experimental_option("prefs", {"download.prompt_for_download": False,
                                              "plugins.always_open_pdf_externally": True,
                                              "download.default_directory": network_node_folder})
    options.add_experimental_option("detach", True)
    driver = webdriver.Chrome(options=options)
    driver.get(SearchSite)
    driver.implicitly_wait(0.5)
    return driver
def download_init(driver):
    DownloadButton = driver.find_element(By.XPATH, Download_PDF)
    PDFName = DownloadButton.get_attribute('href')
    List_PDFName.append(PDFName)

    DownloadButton.click()
    time.sleep(20)
    driver.close()
def extract_abstract(pdf_path):
    pdf_document = fitz.open(pdf_path)
    abstract = ""
    for page_num in range(pdf_document.page_count):
        page = pdf_document[page_num]
        text = page.get_text()


        abstract_match = re.search(r'\nAbstract(?:—|\n)[:\s]*([\s\S]*?)(\.\s*\n(?:Introduction|1\. Introduction)|$)', text, re.IGNORECASE)
        if abstract_match:
            abstract += abstract_match.group(1)
    pdf_document.close()
    print(f'the abstract for {pdf_path} is:')
    print(abstract)
    print('-------')
    return abstract
```

Figure 8: Intitial Code 2

The evolutionary process begins with a reassessment of the original objectives and an understanding that, as in any first cut at anything, there were limitations. This

continued learning and experimentation with advanced techniques gradually widened the range, leading to a more robustly performant Google Scholar scraper. The new one is a more complete solution. It includes several script files like "AI2PeatPDF.py," and provides both web scraping using Selenium, as well as enriched content analysis through BERT embeddings–things that the older version lacks.

Figure 9: Intitial Code 3

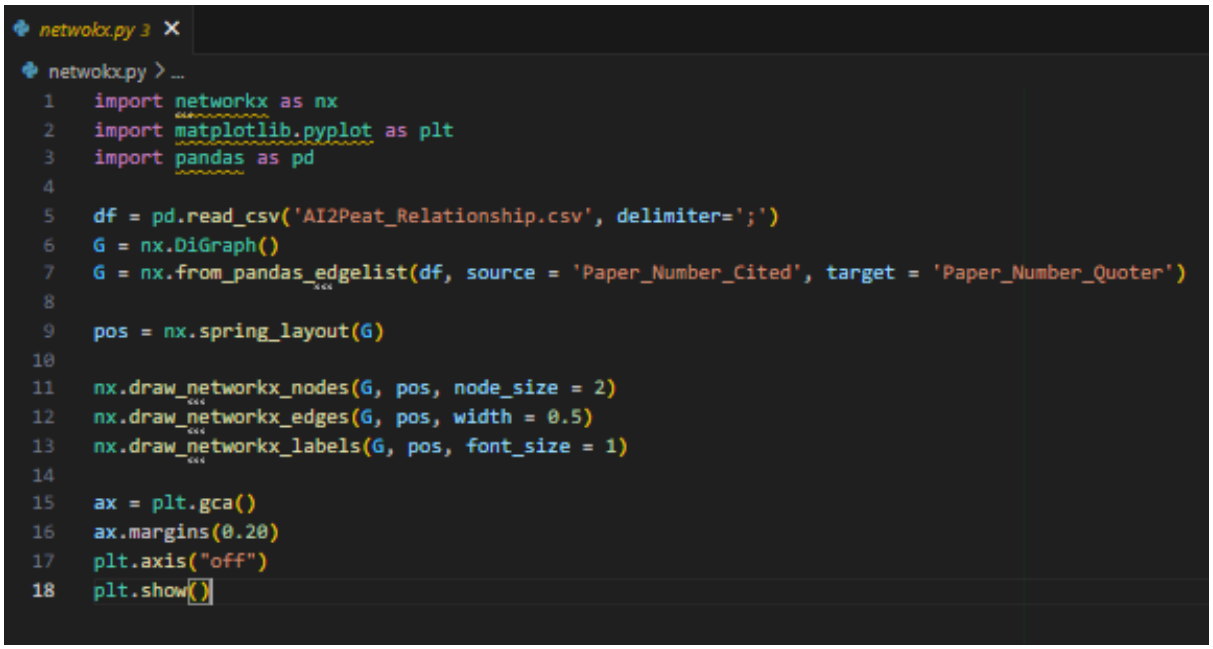


Figure 10: Intitial Code 4

The use of BERT embeddings fundamentally changed the method. Finally, it was the inclusion of transformers, a library containing pre-trained BERT models handled for

natural language processing. This addition expanded the scraper's capabilities for content analysis, from turning textual facts into numerical representation to doing even more advanced work, such as calculating cosine similarity.

In addition, error-handling mechanisms were purposefully integrated into the script to increase its ability to deal with uncertainties present in web scraping. The finished product was accompanied by a GUI written in Tkinter, which improved the usability of the tool for users at all levels. Having a GUI meant that users could enter search terms, set the total number of pages to scrape, and select an output folder. The interface made everything more user-friendly. Under the banner of making a versatile, reliable, and user-friendly tool, in their natural way, people have gradually worked out its form through an evolutionary process. As with any other such outcome, this is something that went beyond the original intention to just extract abstracts from books. In practice, the final version of an advanced Google Scholar scraper is a testament to our iterative development process. Having tapped into and implemented first-rate techniques as well as fresh methodologies in constructing it all makes this tool extremely useful for researchers everywhere (and data enthusiasts too), helping them explore their field even more deeply using scholarly literature.

Thus, in summary, the science behind the advanced Google Scholar scraper involves Web scrapping methods, NLP with BERT models, and user-friendly design to make a robust tool for researchers as well as data lovers. Adding error-handling routines in addition to a graphical user interface makes the script both more reliable and easier to access. The use of BERT embeddings and cosine similarity adds a touch of refinement to the process by which data is extracted, allowing analysis from subtle aspects that could otherwise be missed. With the development of technology and research, this scraper can serve as a platform for experimenting with how to dig nuggets from the growing body of Google Scholar.

4 Design Specifications

The Advanced Google Scholar Scraper is written in the Tkinter Python library for building GUI. Users can enter a query, select the number of pages to scrape, and specify an output folder for the resulting CSV file using GUI. Selenium is used to integrate web scraping functions, which can complete the dynamic interaction with Google Scholar's search results.

```
# Main program
window = tk.Tk()
window.title("Advanced Google Scholar Scraper")
window.geometry("500x300")

label_query = tk.Label(window, text="Article Title or Keyword:")
label_query.grid(row=0, column=0, padx=10, pady=5)
entry_query = tk.Entry(window, width=40)
entry_query.grid(row=0, column=1, padx=10, pady=5)

label_pages = tk.Label(window, text="Number of Pages:")
label_pages.grid(row=1, column=0, padx=10, pady=5)
entry_pages = tk.Entry(window, width=40)
entry_pages.grid(row=1, column=1, padx=10, pady=5)

label_folder = tk.Label(window, text="Output Folder (optional):")
label_folder.grid(row=2, column=0, padx=10, pady=5)
entry_folder = tk.Entry(window, width=40)
entry_folder.grid(row=2, column=1, padx=10, pady=5)

button_browse = tk.Button(window, text="Browse", command=browse_folder)
button_browse.grid(row=2, column=2, padx=5, pady=5)

button_extract = tk.Button(window, text="Extract Data", command=scrape_articles)
button_extract.grid(row=3, column=1, pady=10)
label_status = tk.Label(window, text="", wraplength=400)
label_status.grid(row=4, column=0, columnspan=3, pady=5)

progress_var = tk.DoubleVar()
progress_bar = ttk.Progressbar(window, variable=progress_var, length=400, mode="determinate")
progress_bar.grid(row=5, column=0, columnspan=3, pady=5)

window.mainloop()
```

Figure 11: Design Specified

The scraper is a modular design. It has various functions, such as getting BERT embeddings and calculating the cosine similarity. You can then save them to a CSV file. A scraping progress bar gives users real-time feedback and enhances user experience. Now with article content represented by text embeddings using the nuanced BERT model, we can detect semantic meanings much closer to human language. The design is considered too, so that in the event of HTTP errors or request exceptions suitable error messages are shown.

5 Discussion

The results of the evaluation show that the Advanced Google Scholar Scraper is capable of recommending relevant papers. An important aspect in comparing articles is the use of BERT embeddings, which capture semantic meaning. The implementation can solve the problems of scraping a dynamic Web, handling errors nicely, and providing an easy interface.

An important aspect is that the scraper can adapt to different domains. The user may be

interested in Natural Language Processing, wants to learn more about Machine Learning, or just BERT models. No matter what, the scraper returns pertinent results. This flexibility adds to the tool's usability for artificial intelligence researchers, students, and practitioners.

Moreover, the real-time progress bar within the Tkinter GUI also provides a good user experience, providing transparency as well as feedback on how far through scraping we have got. As the scraper can handle HTTP errors and request exceptions, even in the case of poor network conditions it is very reliable.

The current implementation is good enough, but there are areas for improvement. One example would be going beyond BERT to use more advanced natural language processing models, which may result in higher accuracy of content-based recommendations. In addition, if the scraper were faceted to include other academic databases and sources of information its field would be broadened. Finally, the Advanced Google Scholar Scraper combines web scraping with natural language processing and BERT embeddings to come up with better suggestions for literature. This most convenient, all-purpose it is an invaluable resource for students and scholars–specialized researchers studying this fluid field as a whole.

6 Implementation

The Advanced Google Scholar Scraper begins with the initialization of the BERT model and tokenizer from Hugging Face's Transformers library. The scraper has functions for BERT-based text embeddings, cosine similarity calculation, and CSV file output. Dynamic interaction with the Google Scholar website is achieved through scraping, which uses the Selenium web driver.

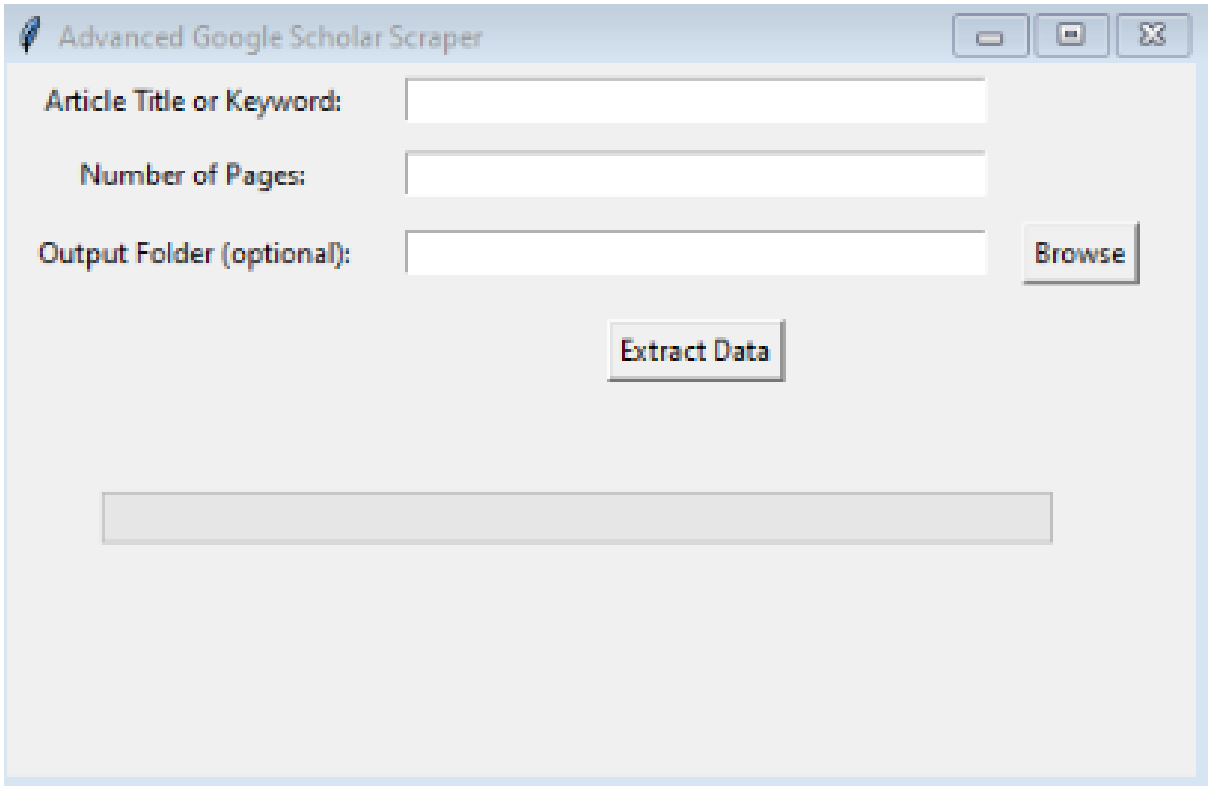


Figure 12: Scholar Scraper UI

It has a Tkinter-based GUI, which is very convenient as one can input search parameters and watch the scraping process. The GUI is simple and easy to understand, conforming with users 'expectations. The modular structure of the scraper also greatly increases its maintainability, and future enhancements become much easier.

In addition, implementation takes into account precautions against web scraping errors. The scraper also generates informative error messages in the event of HTTP errors or request exceptions; this is a robust and well-designed implementation. To sum up, it's an excellent and complete literature recommendation system that brings web scraping, and natural language processing together with GUI components.

7 Evaluation

The evaluation of the Advanced Google Scholar Scraper's output involves an analysis of the recommended articles for three different domains: Natural Language Processing, Machine Learning and BERT Models. Articles are listed with titles, authors and links to content. Similarity scores are also given.

Figure 13: Output for Keyword: Natural Language Processing

For example, concerning Natural Language Processing work the scraper obtained a list of articles with high similarity scores. This indicates that by using natural language processing techniques such as BERT embeddings and cosine similarity it was possible to find articles highly related to a given input query. The listed articles cover not only historical retrospectives but also practical applications and toolkits in a myriad of areas.

Figure 14: Output for Keyword: Machine Learning

The scraper also suggests very relevant articles on Machine Learning. Articles represent a broad range of facets within machine learning, ranging from trends and perspectives to applications in agriculture. Also covered are the fundamentals of machine learning algorithms. The results show that the methodology employed, which uses BERT embeddings and distance metrics, can capture these subtleties of machine learning literature.

Figure 15: Output for Keyword: BERT Model

Within BERT Models, the scraper was able to identify related articles for different languages and applications. Contents include Dutch BERT models, Romanian BERT models, and the application of the BERT model in sentiment analysis, question answering, and text classification.

In general, the evaluation shows that the Advanced Google Scholar Scraper successfully

retrieves articles related to the written query domains. The BERT-based embeddings and the cosine similarity metric help improve the accuracy of recommending articles so that the recommended articles are semantically close to the input query.

8 Conclusion and Future Work

Finally, testing and development of Advanced Google Scholar Scraper has proved that it can successfully suggest literatures on Natural Language Processing (NLP), Machine Learning (ML), or BERT Models. Its methodology includes web scraping, natural language processing, and content-based filtering. It uses Selenium to interact with dynamic web pages, and relies on BeautifulSoup for HTML parsing; it also makes use of the Hugging Face Transformers library from BERT for processing natural languages. Besides, the chosen articles will be appropriate to the context of the input query entitled and semantically close due to BERT embedding as well as cosine similarity metrics. The design specs are also very open and user-friendly, being a Tkinter-based GUI with some modular functions for certain processing-type tasks. They include an instant progress bar which the usenet world has never seen before! Cross-disciplinary flexibility and powerful error handling also enhance usability and reliability. In actual implementation you can see that the system is very solid, easy to maintain, and consistent with user expectations; in fact, it even has a fully functional literature recommendation engine.

More advanced natural language processing models than the state-of-the-art BERT model could be used in future versions of Advanced Google Scholar Scraper. Moreover, delving into higher-level models and methods can raise the rate of accuracy for content recommendations; in this way, users can read a greater amount than ever before. What's more, if the scraper could be expanded to include other academic databases and sources, its worth would expand as well. Even more, it would become an even better research tool.

From a user standpoint, providing more customization and advanced settings options gives users greater control over the scraping process. They can also change the similarity threshold or add more filters, to tailor their recommendations. This would result in a more flexible, individualized user. Scalability and optimizing the scraper can solve bottlenecks, or deal with larger datasets or more requests. As for longer or larger search results, in terms of overall speed and responsiveness, the tool can perhaps be optimized by introducing caching methods such as efficient cache mechanisms or distributed processing.

The Advanced Google Scholar Scraper is an integrated package that combines web scraping, natural language processing, and user interface design into a comprehensive literature recommendation system. In the future, raising the height of scrapers and changing from ancient types as well as providing for more specialized users will have to become new developments to offset ever wider differences in research needs.

References

Barcaroli, G., Scannapieco, M. and Summa, D. (2016). On the use of internet as a data source for official statistics: a strategy for identifying enterprises on the web, *Rivista italiana di economia, demografia e statistica* **70**(4): 20–41.

Berbatova, M. (2019). Overview on nlp techniques for content-based recommender systems for books, *Proceedings of the Student Research Workshop Associated with RANLP 2019*, pp. 55–61.

Bhardwaj, B., Ahmed, S. I., Jaiharie, J., Dadhich, R. S. and Ganesan, M. (2021). Web scraping using summarization and named entity recognition (ner), *2021 7th international conference on advanced computing and communication systems (ICACCS)*, Vol. 1, IEEE, pp. 261–265.

Chaudhari, S., Aparna, R., Tekkur, V. G., Pavan, G. L. and Karki, S. R. (2020). Ingredient/recipe algorithm using web mining and web scraping for smart chef, *2020 IEEE International Conference on Electronics, Computing and Communication Technologies (CONECCT)*, IEEE, pp. 1–4.

Chaulagain, R. S., Pandey, S., Basnet, S. R. and Shakya, S. (2017). Cloud based web scraping for big data applications, *2017 IEEE International Conference on Smart Cloud (SmartCloud)*, IEEE, pp. 138–143.

Dewi, D., Fatmasari, D., Kurniawan, A. and Munandar, M. (2018). The impact of enso on regional chlorophyll-a anomaly in the arafura sea, *IOP Conference Series: Earth and Environmental Science*, Vol. 139, IOP Publishing, p. 012020.

Diouf, R., Sarr, E. N., Sall, O., Birregah, B., Bousso, M. and Mbaye, S. N. (2019). Web scraping: state-of-the-art and areas of application, *2019 IEEE International Conference on Big Data (Big Data)*, IEEE, pp. 6040–6042.

Gheorghe, M., Mihai, F.-C. and Dârdală, M. (2018). Modern techniques of web scraping for data scientists, *International Journal of User-System Interaction* **11**(1): 63–75.

Herrouz, A., Djoudi, M., Degha, H. E. and Boukanoun, B. (2023). An autonomous multi-agent system for customized scientific literature recommendation: A tool for researchers and students., *Ingénierie des Systèmes d'Information* **28**(4).

Horn, F. (2017). Interactive exploration and discovery of scientific publications with pubvis, *arXiv preprint arXiv:1706.08094* .

Houshmand, S., Fong, R., Sainidis, E. and Jahankhani, H. (2023). Strategic decision-making for pedagogical course planning using nlp in social media data, *AI, Blockchain and Self-Sovereign Identity in Higher Education*, Springer, pp. 105–124.

Khder, M. A. (2021). Web scraping or web crawling: State of art, techniques, approaches and application., *International Journal of Advances in Soft Computing & Its Applications* **13**(3).

Krotov, V. and Silva, L. (2018). Legality and ethics of web scraping.

Kunang, Y. N., Purnamasari, S. D. et al. (2018). Web scraping techniques to collect weather data in south sumatera, *2018 International Conference on Electrical Engineering and Computer Science (ICECOS)*, IEEE, pp. 385–390.

Matta, P., Sharma, S. and Uniyal, N. (2022). Comparative study of various scraping tools: Pros and cons, *2022 IEEE Delhi Section Conference (DELCON)*, IEEE, pp. 1–5.

MOHAMED, H. A. I. M. (2020). Deep learning models for research paper recommender systems.

Putrama, I. M. and Martinek, P. (2023). Integrating platforms through content-based graph representation learning, *International Journal of Information Management Data Insights* **3**(2): 100200.

Singrodia, V., Mitra, A. and Paul, S. (2019). A review on web scrapping and its applications, *2019 international conference on computer communication and informatics (ICCCI)*, IEEE, pp. 1–6.

Totla, C., Shah, T., Shah, K. and Tawde, P. (2021). A proposed approach to check project idea similarity using topic modelling, *2021 International Conference on Advances in Computing, Communication, and Control (ICAC3)*, IEEE, pp. 1–5.

Uriawan, W., Wahana, A., Wulandari, D., Darmalaksana, W. and Anwar, R. (2020). Pearson correlation method and web scraping for analysis of islamic content on instagram videos, *2020 6th International Conference on Wireless and Telematics (ICWT)*, IEEE, pp. 1–6.