Multimedia Tools and Applications (2024) 83:60343–60364
https://doi.org/10.1007/s11042-023-17925-y

A novel combining method of dynamic and static web crawler with parallel computing

Qingyang Liu[1] · Ramin Yahyapour[1] · Hongjiu Liu[2] · Yanrong Hu[2]

Received: 14 June 2022 / Revised: 23 October 2023 / Accepted: 17 December 2023 /
Published online: 5 January 2024

Abstract
Recovering information from a targeted website that undergoes dynamic changes is a complicated undertaking. It necessitates the use of a highly efficient web crawler by search engines. In this study, we merged two web crawlers: *Selenium* with parallel computing capabilities and *Scrapy*, to gather electron molecular collision cross-section data from the *National Fusion Research Institute* (*NFRI*) database. The method effectively combines static and dynamic web crawling. The primary challenges lie in the time-consuming nature of dynamic web crawling using *Selenium* and that *Scrapy*'s limited support for parallel computing within the "download middleware". Nevertheless, this combined approach proves exceptionally well-suited for the task of data extraction from an online database, which comprises multiple web pages with unchanging URLs when specific keywords are submitted. We applied natural language processing techniques to identify species and dissect reaction formulas into various states. Employing these methodologies, we extracted a total of 76,893 data points pertaining to 112 species. These data pieces offer intricate insights into the processes unfolding within the plasma, all collected within a span of ten minutes. When compared to traditional web crawling methods, our approach boasts a speed advantage of roughly 100 times faster than dynamic web crawlers and exhibits greater flexibility than static web crawlers. In this report, we present the retrieved results, encompassing reaction formulas, reference information, species metadata, and time comparison among various methods.

Keywords Web crawling · Dynamic · Static · Natural language processing · Parallel computing

1 Introduction

In today's digital landscape, web crawling has become an indispensable tool for gathering data of all walks of life from the vast expanse of the Internet [1]. For example, the semiconductor industry's relentless pursuit of cutting-edge devices, which demands increasingly complex models endowed with predictive abilities. To comprehend the intricacies of plasma processes and their vital role in this industry, it becomes imperative to access comprehensive information [2].

Extended author information available on the last page of the article

Recovering information from a targeted database is a complicated endeavour that involves the utilization of a web crawler by a search engine. The search engine provides an interface for accessing and retrieving extensive data stored on the web. The primary role of web crawlers is to employ various algorithms to uncover and retrieve relevant web data [3].

There are primarily four types of web crawlers: focused crawler [4, 5]; general purpose web crawler; incremental web crawler [6, 7]; and deep web crawler [8]. In the realm of the Internet, web pages can be categorized as surface pages and deep pages based on their existence. A surface page refers to a static page that can be accessed using static links without submitting a form. Conversely, deep web pages are concealed behind forms and cannot be directly obtained through static links; they require the submission of specific keywords to access [9]. The first three crawlers exclusively utilize surface pages, falling under the category of static crawling, while the deep web crawler employs both surface and deep pages, falling under the domain of dynamic crawling.

Absolutely, the dynamic nature of websites in the contemporary digital landscape presents a considerable challenge. This dynamism is characterized by the inclusion of dynamic content such as images and tables that can change in real-time based on user interactions or various keywords [10]. Many modern websites heavily rely on client-side scripting languages like *JavaScript* to dynamically load content. As a result, the critical data on these websites is often generated or modified after the initial *HTML* page has loaded. Traditional static web crawlers find it challenging to access and extract data from these dynamic sources effectively. Furthermore, with the proliferation of technologies like *Single-Page Applications* (*SPAs*) and *AJAX* (*Asynchronous JavaScript and XML*) [11], web contents are now frequently loaded asynchronously, making it even more daunting for static crawlers to navigate through web pages and capture information accurately and efficiently.

In addition to these technical complexities, web crawling also faces other persistent challenges such as handling authentication mechanisms, navigating rate-limiting restrictions imposed by websites, and mitigating the risk of IP blocking.

Indeed, dynamic web crawling does offer the capability to handle a wide range of dynamic content on websites. However, one of its drawbacks is that it tends to be significantly slower compared to static web crawling. In some cases, the crawling speed in dynamic web crawling can be as much as a hundred times slower than that of static web crawling. This reduced speed is often a consequence of the intricate process involved in rendering and interacting with web pages as a human user would. The web crawler, using tools like *Selenium*, must load the page, execute *JavaScript*, and simulate user interactions, which consumes more time and resources than simply parsing static *HTML* content.

To tackle the challenges mentioned above, we have put forth an innovative approach that seeks to combine the advantages of both dynamic and static web crawling methods. amalgamate the strengths of both dynamic and static web crawling methods. By combing the convenience and efficiency of two approaches, our method can adapt to the varying nature of content across the Internet, striking a harmonious balance between adaptability and crawling speed.

The purpose of this paper is to employ web crawlers to enrich the data within the expanding Quantemol database, also referred to as QDB. as a comprehensive resource, catering to the requirements of plasma modelers across various dimensions. It bridges the gaps present in other databases from the United States and Japan. This endeavor aids in the exploration of plasma discharge intricacies and the fundamental understanding of interactions between distinct species [12]. In doing so, it addresses issues stemming from the utilization of feed gases in plasma processes, which have the potential to generate waste and contribute to global warming [13]. To enhance the QDB database, the collection of data from additional sources, such as the *National Fusion Research Institute* database, known as *NFRI*, is imperative [14].

The *NFRI* database is a website that exhibits a distinctive characteristic by blending surface and deep pages. To effectively extract data from this database, we harnessed a synergy of both static and dynamic web crawling techniques. In our paper, we implemented *Python* to execute these dynamic and static web crawling methods. Notably, we introduced a novel approach by incorporating *Selenium* with *parallel computing* and utilizing *Scrapy* for scalable data retrieval from both databases and websites, leveraging their unique strengths.

The key advantages of this methodology lie in its capacity to significantly reduce the execution time of *Selenium*. This was made possible through the implementation of *Multiprocessing* to enable *p*arallel computing. In addition, to expedite the process, we employed *Scrapy* to crawl data from surface pages. This approach not only minimizes the volume of requests sent but also alleviates the burden on the website server. Consequently, this combined methodology proves particularly well-suited for tackling the challenge of extracting data from online databases that encompass multiple web pages and maintain consistent URLs even after keyword submission.

The following section conducts literature reviews in multiple domains: web crawlers, web extraction techniques, and natural language processing. In Section 3, we describe our methodology, where we employ two distinct web crawling approaches where one incorporating *parallel computing*. We also use natural language processing techniques to extract data from *NFRI*. Section 4 entails a comparative analysis of the three methods employed for data retrieval from *NFRI*, elucidating the structure of the obtained information. Finally, the concluding section outlines potential avenues for future research and presents our overarching conclusions.

2 Literature review

2.1 Web crawlers

Web crawlers, also known as web spiders, are automated programs that browse the Internet in a systematic and methodical manner to gather information from websites. These crawlers play a crucial role in indexing and cataloging web content, which makes it more accessible for search engines and users. There are several popular web crawlers used for various purposed. For example, *Googlebot* is responsible for indexing websites for the Google search engine [15]; *Splash* is a headless web browser with an HTTP API, often used for rendering *JavaScript*-heavy websites when scraping with *Scrapy* [16].

As highlighted in Section 1, two primary types of web crawlers exist: dynamic and static. In this paper, we have opted to employ Python for the implementation of our web crawling programs. Numerous libraries and frameworks are available for web crawling and web scraping tasks, including but not limited to *Scrapy* [17], *Beautiful Soup* [18], *Requests* [19], *Selenium* [20], *LXML* [21], and others.

The representation of dynamic web crawlers is *Selenium* in *Python*, a web automation tool that allows you to control a web browser programmatically [22]. The most significant benefit is that it can wait for *JavaScript* to execute and respond to dynamic content changes, making it suitable for crawling and scraping web applications that heavily rely on *JavaScript* [23]. Moreover, it supports multiple web browsers, including *Chrome*, *Firefox*, *Safari*, *Edge*, and more, enabling to test and automate web inter-actions on different browsers, ensuring cross-browser compatibility [24]. While *Selenium* has these advantages, it's important to consider its limitations as well. *Selenium* can be slower compared to specialized web crawling libraries

like Scrapy for large-scale data scraping tasks. It may not be the best choice for tasks requiring high-speed data extraction from numerous websites [25]. Moreover, running *Selenium* tests or crawlers can be resource-intensive, especially when dealing with multiple browser instances simultaneously. This can lead to high memory and CPU usage [22].

To address the execution speed issue mentioned earlier, we have adopted a hybrid approach that combines the strengths of both dynamic and static web crawlers.

Static web crawlers excel in swiftly retrieving data from websites, contributing to improved overall performance. A prominent library in *Python* is *Scrapy*, a widely-used open-source web crawling framework. It provides a high-level API for building web spiders and supports various features like request/response handling, data extraction, and more [26]. *Scrapy* offers several notable advantages, including: (a) asynchronous capability and flexible concurrency; (b) use of readable *XPath*; (c) shell mode for independent debugging [27]. However, *Scrapy* may encounter limitations when dealing with web pages that require the execution of *JavaScript* to obtain data. Additionally, due to its reliance on the Twisted framework, if a running exception occurs, it won't necessarily terminate the reactor, and thus errors in asynchronous tasks might be challenging to detect [28].

Indeed, when using *Selenium* in conjunction with *Scrapy*, *Scrapy* can handle dynamic web crawling. However, it's essential to be aware that with *Selenium*, *Chrome* typically processes one page at a time. Regarding *Scrapy-Redis*, it enables distributed web crawling, enhancing the scalability of the process. Nevertheless, it's a valid concern that if *Redis* experiences an issue or failure, mission-critical information could be lost. This highlights a potential vulnerability in terms of data persistence and emphasizes the importance of implementing robust mechanisms for data backup and recovery to mitigate such risks from a security perspective [29].

2.2 Web extraction techniques

To enhance the QDB database, we leveraged web data extraction techniques, often referred to as web crawling. This field encompasses a fusion of web crawling and natural language processing methodologies, which collectively enable the automatic extraction of data from websites and web pages [30]. This data can include text, images, structured information, and more.

Web data extraction concentrates on the retrieval of extensive and cross-domain data, a need that has become increasingly critical due to the exponential growth of the web and the proliferation of data sources on a large scale [31]. Web data extraction represents a fundamental challenge that has been investigated using various scientific methodologies and has given rise to a multitude of products and approaches designed to address specific issues and cater to ad-hoc domains. The success of these endeavours largely hinges on the effectiveness of the techniques employed in web information extraction [32]. As an example, Bazeer introduced a distinctive approach involving tree-based pattern matching to represent page generation in 2022. This methodology has proven effective for web data extraction, particularly when dealing with a substantial number of online pages [33]. Subsequently, Yuan Zhou presented an enhanced information extraction method that utilizes mixed text density grids to improve the effectiveness of web page information extraction [34].

While the extraction process can indeed be complex and challenging, methods and algorithms for wrapper induction can be categorized into three primary types: page labelling, document structure analysis, and knowledge-based wrap-pers. When considering the need

for manual examples, these approaches can be broadly grouped into two classes: supervised and unsupervised. The majority of them are highly automated and involve minimal user interaction [35]. This paper is primarily focused on document structure analysis, a technique that involves studying the logical structure of websites to obtain a wrapper.

It's important to note that our study does not explore the use of unsupervised learning wrappers, which are designed to handle incomplete data and noise assumptions when retrieving data from websites with structures resembling databases. Instead, we employ supervised *Xpath*-based wrapper induction, where supervisors play a role in determining the wrapper templates [36]. Some relevant approaches that deal with string alignment are referenced in this paper [37, 38], although they do not provide an exhaustive description of the *XPath* grammar.

2.3 Natural language processing

Natural Language Processing, known as *NLP*, constitutes a domain within *Artificial Intelligence* (*AI*) that focuses on the interaction between computers and human language [39]. This field encompasses a diverse array of tasks, encompassing speech recognition [40], language translation [41], sentiment analysis [42], and so on [43]. This paper specifically addresses two branches of *Natural Language Processing*.

One of them is word segmentation, the process of dividing a continuous sequence of text, such as a sentence or a paragraph, into individual words or tokens [44]. It is a fundamental preprocessing step, as it forms the basis for various language-related tasks, including text analysis, information retrieval, and machine learning [45]. The process of identifying individual punctuation marks bears similarities to the task of identifying single words, a topic frequently discussed in the context of Chinese word segmentation [46]. Word segmentation methods can be categorized into four primary types: rule-based segmentation [47], statistical segmentation [48], machine learning-based segmentation [49], and hybrid methods [50]. In this study, we utilized the positive maximum match method, which is categorized as a rule-based segmentation technique. This method serves as a fundamental approach for word segmentation in languages where word boundaries are not explicitly marked. Essentially, it dissects a given sentence into all possible divisions and chooses the arrangement with the fewest words that reconstruct the sentence [51]. This approach contributes to enhancements in both efficiency and accuracy of word segmentation [52].

Fig. 1 Search page

	NO	GRAPH	REACTION FORMULA
1	B201200015		e + H
2	B201206148		e + H

Fig. 2 Result page

Another *Natural Language Processing* that we plan to employ is information extraction, which revolves around the automatic extraction of structured information from unstructured text [53]. The primary objective of information extraction is to convert textual data into a structured format, thereby facilitating ease of analysis, querying, and utilization for various applications [54]. It's clear that information extraction methods have a wide range of applications in various domains, including biomedical fields [55] such as molecular biology [56] and clinical records [57]. These methods are valuable for extracting structured information from unstructured text data, which is prevalent

Prop. No. B201200015

Referece Link	2011000118
Categorize	Collision Processes
Collision data categoriz e	Cross Section
Collision process	Scattering
Sub process	Total
Collision type	Electron Impact
THEORY / EXPERIMENT / RECOMMENDATION	Recommended Data
Reaction Formula	e + H
XSAMS Doc.	Download

Fig. 3 Reaction information page

Fig. 4 Reference information page

Reference Type	학술지
Title	Photon and Electron Interactions with Atoms, Molecules and Ions
Author	S. J. Buckman, J. W. Cooper, M. T. Elford, M. Inokuti, Y. Itikawa, H. Tawara
Journal Name	Landolt-Bornstein
DOI	-
ISSN_P	-
ISSN_N	-
ISBN	-
NDSL	-
Volume	17
Number	A
Start page	1
End page	174
Published year	2000

in scientific literature, clinical records, and other text-heavy domains. For instance, Chaussabel used it to extract cell line profiling data from the literature and concluded that it could be used beyond genomic data analysis [58].

3 Methodology

The web crawling process is broken down into five distinct phases, meticulously executed in the sequence outlined in Part 3.1. In this endeavor, we harnessed the power of two Python packages, Selenium and Scrapy, to extract data from web sources. Subsequently, we employed the "re" package to dissect the acquired data into several discrete columns for structured analysis.

3.1 Process of web crawling

In this section, we will outline the five key steps involved in the process of data crawling from the NFRI online database.

Step 1: Obtaining Cross-Sectional URLs

The search process starts with the search page, as depicted in Fig. 1. Specific species names are entered as keywords into the search interface. Subsequently, the "search" button is clicked to initiate the search. The results page provides cross-sectional URLs that are relevant to the specified species.

Prop. No. B201200015

X value	Y value	X err	Y Max er r	Y Min er r	Ratio	Pressure
1	23	0	0	0	-	-
2	17.1	0	0	0	-	-
3	14	0	0	0	-	-
5	10.2	0	0	0	-	-
7	8.4	0	0	0	-	-
10	6.3	0	0	0	-	-
15	4.84	0	0	0	-	-
20	4.12	0	0	0	-	-
30	3.26	0	0	0	-	-
50	2.55	0	0	0	-	-
100	1.67	0	0	0	-	-
200	1.08	0	0	0	-	-
500	.55	0	0	0	-	-

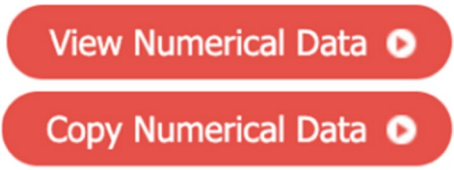
Fig. 5 Metadata page

Step 2: Retrieving Record Numbers

In the second step, the focus is on obtaining the record numbers, which serves as critical identifiers for connecting to and retrieving further information and metadata associated with the reactions. The reaction information URLs, each associated with a specific record number, are readily available on this page, as demonstrated in Fig. 2.

Step 3: Retrieving Reaction Information

Fig. 6 View numerical data page



To compare the performance and speed in data extraction between *Selenium* and *Scrapy*, a new step was initiated, where *Scrapy* was utilized to retrieve reaction information. This step encompasses the retrieval of reaction details, including reference URLs, as depicted in Fig. 3.

Step 4: Retrieving Reference Information

In the fourth section, the focus is on retrieving reference information associated with the previously obtained reaction details. Figure 4 illustrates the process of gathering reference URLs. It's important to note that reference URLs might be duplicated because different reaction can be related to the same articles. A Scrapy project was initiated to facilitate this step, allowing for the efficient extraction of reference information. The Scrapy framework can manage duplicated reference URLs effectively.

Step 5: Retrieving Metadata

In the final step, the objective is to retrieve metadata, as shown in Fig. 5, using the record numbers. Additionally, metadata can be obtained through the result page by following a sequence that involves clicking the "graph" button and subsequently the "View numerical data" button, as illustrated in Fig. 6. But this process is dynamic and relatively slow due to the simulation of opening a browser.

To overcome the slow and dynamic process, a more efficient approach was adopted. This involved using the record numbers to generate metadata URLs and utilizing *Scrapy* for metadata retrieval. One notable challenge with Scrapy is that it doesn't readily support the creation of multiple tables for different keywords within a single spider. Therefore, a new spider was used to retrieve metadata, considering the volume of data, as indicated in Fig. 5.

The pseudocode is illustrated in Algorithm 1 to describe the implementation of the proposed method with five steps mentioned above.

```
Function Web Crawling (species)
For each specie do:
  Parallel do:
    Call get_cross-section_URLs (specie);
    Return cross-section URLs list;
  For each cross-section URL do:
    Call get_record_number (cross_section URL);
    Return record numbers;
    For each record number do:
      Call get_metadata(record_number);
      Return metadata;
      Call get_DOI_and_other (record number);
      Return DOI numbers and other information;
      For each DOI number do:
        Call get_DOI_information (DOI);
        Return DOI information;
end
return "Main","Metadata","Reference" table;
```

Algorithm 1 Pseudocode of the proposed method

3.2 Get cross-section URLs for each specie

The species list utilized in this study was from the QDB database. Instead of PhantomJS, which is no longer maintained for web crawling, we opted for a headless *Chrome* browser. This approach enables us to retrieve website content without the need to open the actual page, thus saving time. In this section, we implement *Parallel Computing* to further enhance time efficiency. The code snippet for this process is depicted schematically in Algorithm 2.

input: species list
output: cross-section URL*s* list
for each species:
1 deleted "+" and "-" to get specie without charge;
2 fill the keyword with the result and click;
3 get the content of *XPath* expressions;
end
return cross-section URL*s* list;

Algorithm 2 Get each species cross-section URLs

Initially, we selected species without a charge because *NFRI* is unable to search for data involving species with charges. Thus, in our program, we needed to determine the charge of each species within the reaction formula. Prior to web crawling, we compiled a list of species without charges. Using natural language processing, we removed the "+" and "-" symbols from the original species names. Subsequently, we used this list of species without charges as keywords and applied *XPath* expressions to extract the desired fields from the websites.

This process is dynamic and involves entering several species simultaneously into the search box by using *Selenium*, followed by synchronous page openings. We configured the pool worker to be 8, resulting in the simulation browser opening 8 windows and acquiring the respective URLs. Once one step is completed, the next step is added, and this continues until all the species have been processed. We did not integrate *Scrapy* with *Parallel Computing*, *Scrapy* itself is fast enough to deal with the following four steps.

3.3 Get record numbers

While studying the website structure, we discovered that the URLs containing reaction data and formulas, along with other information, were associated with record numbers. This enabled us to locate the URLs for reaction formulas with relative ease. However, when seeking the URLs for reaction data, we encountered challenges, necessitating the use of static web crawling techniques. Even with the application of *Parallel Computing*,

this process remained slow. To address this, we decided to utilize the static web crawling method *Scrapy*, but we needed record numbers.

Acquiring record numbers can be relatively easily using *XPath* expressions. But this method requires record numbers for recommended data. Consequently, we utilized the ID of the record numbers to find the associated theory and then determined whether they belonged to the recommended data. The step-by-step procedure is outlined schematically in Algorithm 3.

input: cross-section URL*s* list
output: record numbers list
for each cross-section URL:
1 find all the record numbers;
2 get record numbers' attribute *id*;
3 achieve theories according to *id*;
4 judge whether they belong to recommended data;
end
return record numbers list;

Algorithm 3 Get record numbers

3.4 Get DOI numbers and other information

By using the record numbers, we were able to access information related to the reaction formulas, DOI numbers, collision processes, sub-processes, and collision types.

The primary objective of this section is to extract specific components from the reaction formula, including species, ionic state, initial state, initial state conf, and final state. In cases where there were no products indicated by arrows in the reaction formula, determining the final state was not possible. The initial state, initial state conf, and species could be identified based on the presence of brackets in the reaction formula, while the ionic state was determined by the presence of positive or negative symbols in the reaction formula.

Furthermore, we aimed to build connections between collision processes, sub-processes, and collision types with counterparts in the QDB database. To streamline this process and save time, we utilized *Scrapy* for static web crawling. The step-by-step procedure is schematically outlined in Algorithm 4.

input: record numbers list
output: reaction formula list
for each record number:
1 get reaction formula, collision process, sub-process, collision type, and *DOI* numbers
2 split reaction formula and define the final state.
3 use NLP to define species with charge, *initial state* and, *initial state conf*;
4 judge charge in *species* with charge;
5 combine collision process, sub-process, and collision type and connect it with *QDB*.
end
return DOI numbers and other information;

Algorithm 4 Get DOI number & other information

3.5 Get recommended data

We were able to retrieve the recommended data based on the record numbers, which we referred to as a metadata table. We established the connection between the main table and the metadata table using record numbers. For obtaining recommended data, which necessitates a 'click' operation, we utilized *Selenium*. As in other sections, parts of the website are organized around record numbers, making it convenient to retrieve data using *Scrapy*. In this particular section, when '*Y minus error*' is equal to '*Y max error*,' it was only necessary to maintain one value. The step-by-step procedure is schematically depicted in Algorithm 5.

input: record numbers list
output: *metadata* table
for each record number:
1 retrieve *x, y, x error, y min error,* and *y max error*;
2 compare *y min error* and *y max error*;
end
return *metadata* table

Algorithm 5 Get metadata

3.6 Get reference information

In this part, we were able to gather reference information, including DOI, title, authors, journal name, page number, and more. We avoided using record numbers because many records were associated with the same articles. As part of this process, we employed *NLP* methods to parse and separate the authors' names. Just as in other sections, a part of the website was organized around DOI numbers. After retrieving reference information from the website, we merged the reference information table with the reaction formula table. This procedure is visually presented in Algorithm 6.

input: reaction formula list
output: main table
for each DOI number:
1 retrieve DOI, title, author, journal name, volume, issue number, page number, and date of publication
2 split authors' names.
combine reaction formula list and reference information list into the main table;
end
return main table

Algorithm 6 Get reference information

4 Results and discussion

The following shows the structure of the information retrieved, discusses the results, and compares the three methods to retrieve data from *NFRI*.

4.1 Data structure

The results consist of three tables mentioned above: the 'Main,' 'Data,' and 'Reference' tables. The "Main" table include reaction information for each reaction formula for each species, including reaction type, process type, initial state, and more. Each reaction corresponds to a special record number and reference number. The "Data" table contains data for each reaction formula. And the "Reference" table contains research information, including author and journal name. The data relationships between them are illustrated in Fig. 7. The 'Main' table is linked to the 'Data' table through the 'Record number' in the 'Main' table and the 'ID' in the 'Data' table. Additionally, the 'Main' table is also connected to the 'Reference' table through the 'Reference number' found in both tables.

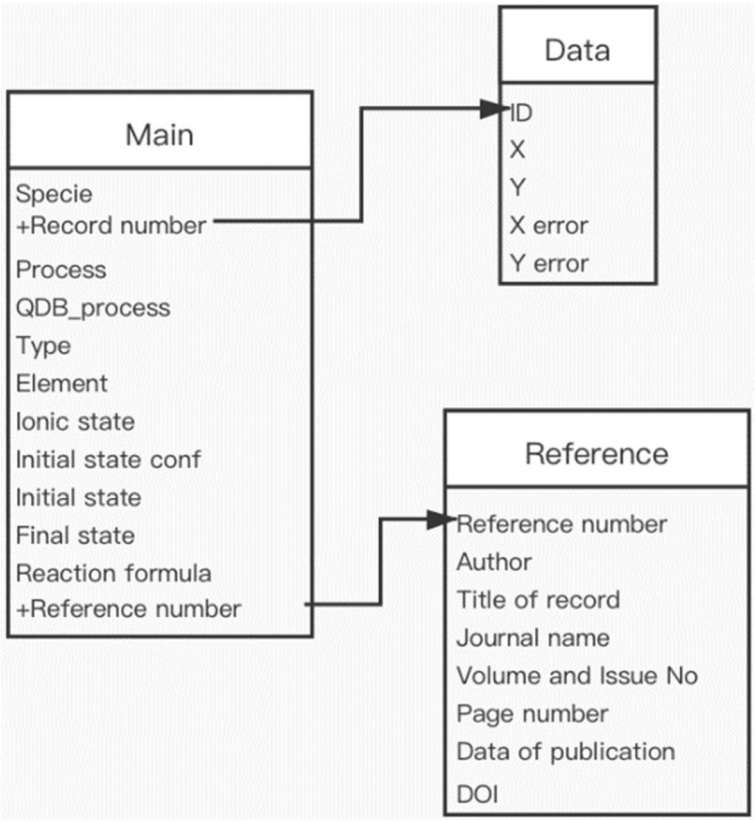


Fig. 7 An outline of the data structure

The 'Main' and 'Data' tables are exported as CSV files and managed in Microsoft Excel. After validation, the staff will upload this data into the QDB database, making it accessible to anyone. Additionally, we have recorded the three methods used in the CSV file. Statistical analysis and graphical representations were generated using both Excel and Python.

4.2 Discussion of the results

The original species encompass various states such as positive and negative electrodes. However, when entering species, it is only necessary to input molecules. Consequently, we employ *Natural Language Processing* techniques for preprocessing. As a result, we obtained a list of species without charge, totaling 255 entries. The initial list consisted of 475 species. The list of species without charge is depicted in Fig. 8.

Firstly, we inputted this list of species into the *NFRI* database to determine the number of existing species. This step aimed to reduce redundant work in subsequent processes and save execution time. However, the code only identified 80 species without charge. The species present in the *NFRI* database are illustrated in Fig. 9. The challenge encountered during this process was the extended execution time due to the use of a dynamic web crawling library. To address this, we implemented *Parallel Computing* to simultaneously open multiple Chrome pages.

NF	CH2F	Br	Si5H11	SiCl2	SF	CHF 3.00	CaF	O3	C3H52	CF3	C3H5	CCl3	C4F8	H7O3	BCl
CH3	SiO2	N2O2	Si6H13	SOF3	C3H8	HONO2	C4F4	Xe2	H13O6	NH	H3	BeH2	H4NO3	C3F6	CNH
Si7H16	SF3	C4H	CF32	SOF4	C10H2	C2F4	F	C2H3	Ar2	Si5H12	HCHO	C3H	C2F	C4F6	HN03
Si6H14	SiF	H22	C3H42	CH4	C3H3	N2H4	CF2I	C8H4	C3H32	H4	SiCl4	NF3	H2O	BeH	H2F
Si2H4	C3F3	C	Si	C3H6	Si9H20	CO22	Cu	H11O5	HCl	N4	C12H2	hv	SO2	SiF4	H6NO4
C2	SiH	C4H2	SF4	COF2	C2H	N2H	CF4	NH4	Si2H5	SF2	C2OH6	Ne	H	SO2F2	C3F5
C4H10	BCl3	C3N	C2H6	N3	Ne2	HBr	N2O4	Si2H6	C2H5	CCl22	C6H4	N2	Kr	Si4H10	HO2NO2
ClO	N2O5	CF3I	HCP	H15O7	CF22	Ar	CHF 2.00	NO3	C3H4	CH3F	HNO4	CF3O2	COS	C2F5	C3H22
S	H2O2	C3F	SiO	C4F5	Cl	e	SOF	C4F7	B	CH	C2H2	C8H2	C3F4	HCN	N2H2
N2O	CH2	Si3H8	CCl4	C10H	C3	C2F3	H2	F4	Al	SF5	SiH3	SiCl3	H2O3	F3	HONO
CCl32	SOF2	Ti	H9O4	O4	C10H6	O	C6H2	N	C8H6	CN	SiCl	HO2	SF6	HN02	F2O
C3F8	OH	N2H3	He	C12H6	H2CN	CF2	C8H	Xe	PH3	Cl2	C3H2	SiH3O	SiF3	C2F2	H3O
NH2	Si8H18	HNO	Br2	HF	H2S	BF	CO	CONH3	He2	CO2	C3F2	I	SiF2	ArH	NH3
CH2F2	CH5	NO	C2F6	C4H3	FNO	C6H3	C2H4	C3H7	BF3	02	Ar3	C6H	H5O2	NO2	Si3H7
BCl2	Hg	CaF2	Si8H17	FO	C12H	C2H22	Si2H2	H4O2	COF	SiH2O	SiHO	CCl	CHF	CCl2	BF2
C2H52	C3F7	SiH2	H2NO2	CF	NF2	Si7H15	F2	SiH3N	M	N2O3	CS	SiH4	Si4H9	SiH4N	

Fig. 8 Species without charge

Ti	BeH	Br	Hg	SiH	Si2H6	CO	SO2F2	O3	SiH4
CH	CH2F2	Cl	SiCl4	C2F6	SiF4	Cu	Ne	CN	CH3
H2O	F	C3F6	Ar	NO	N2O	HCl	CF4	Al	C2F4
NH3	CO2	COS	CF3	BCl3	Cl2	HF	O2	C2H2	CH2
PH3	HCHO	CHF	HCN	CCl4	Xe	SF6	CH4	CH3F	BF
SF4	BeH2	H2S	C3H6	NF3	O	S	CF	BF3	SO2
C4H10	CF3I	B	N	SiF	H2	C	Si	Kr	N2
C3F8	NO2	C2H4	H	CH2F	BCl	He	CF2	CHF3	C2H6

Fig. 9 Species in the *NFRI* database

To illustrate the entire web retrieval process within the *NFRI* database, we'll use the example of the 'Cl2' species. We successfully retrieved 16 records of reaction formula information and stored them in the 'Main' table, as depicted in Fig. 10a and b. As shown in the figures, some references were lacking DOI information and volume details, which prompted subsequent searches to ensure reliable data sources. The challenge we encountered in retrieving details of reaction formulas and reference information, same in retrieving data for each reaction formula, was that, even with the combined use of Selenium and Parallel Computing, the average execution time only shortened to a tenth of the original duration. To address this challenge and fully accommodate the database's structure, we opted for the static web crawling technique using Scrapy to retrieve information.

In this illustrative example, the ninth record number in the table was 'rowB201100329.' Within this example, there were ten data records, as presented in Fig. 11. These records were initially disordered in the original 'Data' table but would be sorted into a specific sequence at a later stage. In this table, the value of 'X_error' was consistently zero, while the value of 'Y_error' equaled 'Y_Max_error' if 'Y_Max_error' was the same as 'Y_Min_error'.

specie	record_number	process	QDB_process	type	element	ionic_state	initial_state_conf	initial_state	final_state	reaction_formula	x_unit	y_unit
Cl2	rowB201100326	Electron Impact Total Scattering	ETS	R	Cl2	0		Cl2		e + Cl2	eV	1E-16 cm^2
Cl2	rowB201213072	Electron Impact Total Scattering	ETS	R	Cl2	0		Cl2		e + Cl2	eV	1E-16 cm^2
Cl2	rowB201213624	Electron Impact Total Scattering	ETS	R	Cl2	0		Cl2		e + Cl2	eV	1E-16 cm^2
Cl2	rowB201100327	Electron Impact Elastic Scattering	EEL	R	Cl2	0		Cl2		e + Cl2	eV	1E-16 cm^2
Cl2	rowB201213625	Electron Impact Elastic Scattering	EEL	R	Cl2	0		Cl2		e + Cl2	eV	1E-16 cm^2
Cl2	rowB201100328	Electron Impact Total Ionization	ETI	R	Cl2	0		Cl2		e + Cl2	eV	1E-16 cm^2
Cl2	rowB201212645	Electron Impact Total Ionization	ETI	R	Cl2	0		Cl2		e + Cl2	eV	1E-16 cm^2
Cl2	rowB201213626	Electron Impact Total Ionization	ETI	R	Cl2	0		Cl2		e + Cl2	eV	1E-16 cm^2
Cl2	rowB201100329	Electron Impact Total Neutral Fragments Dissociation	EDS	R	Cl2	0		Cl2		e + Cl2	eV	1E-16 cm^2
Cl2	rowB201213627	Electron Impact Neutral Product Dissociation Dissociation	EDS	R	Cl2	0		Cl2		e + Cl2	eV	1E-16 cm^2
Cl2	rowB201213701	Electron Impact Total Attachment	EDA	R	Cl2	0		Cl2		e + Cl2	eV	1E-16 cm^2
Cl2	rowB201213703	Electron Impact Total Attachment	EDA	R	Cl2	0		Cl2		e + Cl2	eV	1E-16 cm^2
Cl2	rowB201212972	Electron Impact Dissociative Attachment	EDA	R	Cl2	0		Cl2	Cl- + Cl	e + Cl2 -> Cl- + Cl	eV	1E-16 cm^2
Cl2	rowB201213628	Electron Impact Dissociative Attachment	EDA	R	Cl2	0		Cl2		e + Cl2	eV	1E-16 cm^2
Cl2	rowB201213629	Electron Impact Ion Pair Production Reaction		R	Cl2	0		Cl2		e + Cl2	eV	1E-16 cm^2
Cl2	rowB201216633	Electron Impact Ion Pair Production Reaction		R	Cl2	0		Cl2	Cl+ + Cl- + e	e + Cl2 -> Cl+ + Cl- + e	eV	1E-16 cm^2

a

reference_number	author	title_of_record	journal_name	volume_and_issue_No	page_number	date_of_publication	DOI
2011000126	L. G. Christophorou J. K. Olthoff	Electron interaction with Cl2	J. Phys. Chem. Ref. Data	28	131/169	1999	10.1063/1.556036
2011000255	G. P. Karwasz R. S. Brusa A. Zecca	Photon and Electron Interactions with Atoms, Molecules and Ions	Landolt-Bornstein	17C	6-1/6-51	2003	
2012000709	Loucas G. Christophorou James K. Olthoff	Fundamental Electron Interactions with Plasma Processing Gases	BK		1/-	2004	
2011000126	L. G. Christophorou J. K. Olthoff	Electron interaction with Cl2	J. Phys. Chem. Ref. Data	28	131/169	1999	10.1063/1.556036
2012000709	Loucas G. Christophorou James K. Olthoff	Fundamental Electron Interactions with Plasma Processing Gases	BK		1/-	2004	
2011000126	L. G. Christophorou J. K. Olthoff	Electron interaction with Cl2	J. Phys. Chem. Ref. Data	28	131/169	1999	10.1063/1.556036
2011000253	B.G. Lindsay M.A. Mangan	Photon and Electron Interactions with Atoms, Molecules and Ions	Landolt-Bornstein	17C	5-1/5-77	2003	
2012000709	Loucas G. Christophorou James K. Olthoff	Fundamental Electron Interactions with Plasma Processing Gases	BK		1/-	2004	
2011000126	L. G. Christophorou J. K. Olthoff	Electron interaction with Cl2	J. Phys. Chem. Ref. Data	28	131/169	1999	10.1063/1.556036
2012000709	Loucas G. Christophorou James K. Olthoff	Fundamental Electron Interactions with Plasma Processing Gases	BK		1/-	2004	
2012000709	Loucas G. Christophorou James K. Olthoff	Fundamental Electron Interactions with Plasma Processing Gases	BK		1/-	2004	
2012000709	Loucas G. Christophorou James K. Olthoff	Fundamental Electron Interactions with Plasma Processing Gases	BK		1/-	2004	
2011000254	Y. Itikawa	Photon and Electron Interactions with Atoms, Molecules and Ions	Landolt-Bornstein	17C	5-78/5-114	2003	
2012000709	Loucas G. Christophorou James K. Olthoff	Fundamental Electron Interactions with Plasma Processing Gases	BK		1/-	2004	
2012000709	Loucas G. Christophorou James K. Olthoff	Fundamental Electron Interactions with Plasma Processing Gases	BK		1/-	2004	
2011000126	L. G. Christophorou J. K. Olthoff	Electron interaction with Cl2	J. Phys. Chem. Ref. Data	28	131/169	1999	10.1063/1.556036

b

Fig. 10 **a** "Cl2" is an example in the main table. **b** "Cl2" is an example in the main table

Fig. 11 Data table example-"rowB201100329"

ID	X	Y	X_error	Y_error
rowB201100329	8.4	0.48	0	0.14
rowB201100329	9.9	1.04	0	0.31
rowB201100329	12.4	1.36	0	0.41
rowB201100329	14.9	2.07	0	0.62
rowB201100329	17.4	1.51	0	0.45
rowB201100329	19.9	1.52	0	0.46
rowB201100329	22.4	1.19	0	0.36
rowB201100329	27.4	0.96	0	0.29
rowB201100329	47.4	0.52	0	0.16
rowB201100329	97.4	0.24	0	0.07

Table 1 Time comparison

No.	Normal	Parallel	Scrapy
1	46m23s	6m2s	/
2	38m5s	4m55s	19s
3	3h16m6s	23m28s	1m10s
4	6m33s	53s	3s
5	4h34m13s	1h19m58s	1m36s
Total	9h21m20s	1h55m16s	3m8s

4.3 Comparison of the time of three methods

In this section, we compared the time required for five steps among three methods: standard *Selenium*, *Selenium* with *Parallel Computing*, and *Scrapy*. The elapsed time for each method was recorded in seconds, and each method was executed ten times. The average execution time and the total running time for all three methods for each step are presented in Table 1. The programs were run in a CPU environment.

As indicated by the table, Selenium with Parallel Computing is notably faster than standard Selenium, roughly five times quicker. Although Scrapy cannot be applied in the initial step, it excels in the subsequent four steps, being approximately 34 times faster than Selenium with Parallel Computing, underscoring its suitability for those steps. If we employ Selenium with Parallel Computing in the first step and switch to Scrapy in the following four steps, the total execution time amounts to around 9 min.

In the initial step of our process, we obtained cross-section URLs for each species. To achieve this, we employed both *Parallel Computing* and *Normal Computing* within *Selenium*. The comparison of the time taken for these two approaches is presented in Fig. 12. The results indicate that *Normal Computing* required an average web crawling time of approximately 2783 s. In contrast, *Parallel Computing* significantly outperformed *Normal Computing*, with an average web crawling time of about 362 s. This demonstrates that *Parallel Computing* is notably more efficient than the traditional approach when it comes to dynamic web crawling, as it is approximately eight times faster.

Consequently, the utilization of *Selenium* with *Parallel Computing* is highly re-commended for web crawling tasks that involve retrieving data with a moderate data volume

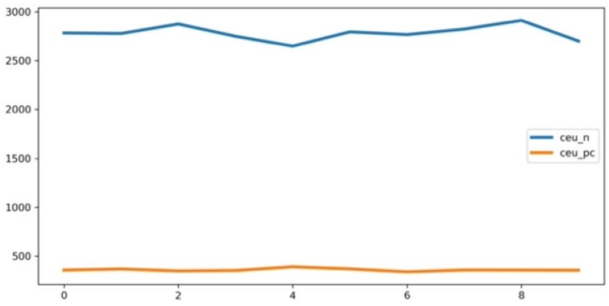


Fig. 12 Compare time used in *Parallel Computing* and *Normal Computing* to retrieve cross-section URLs

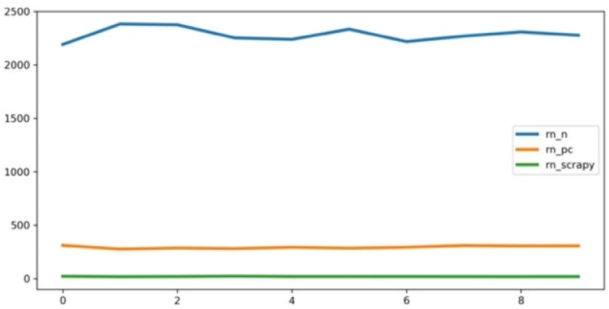


Fig. 13 Compare time used in *Scrapy* and *Parallel Computing* and *Normal Computing* in *Selenium* to retrieve record number

from websites. This approach is particularly effective when dealing with websites that require the submission of keywords and maintain consistent URLs, which do not change during the data retrieval process.

In the second phase of our study, we obtained record numbers for each cross-section URL. This process involved *Scrapy*, as well as a combination of both *Parallel Computing* and *Normal Computing* using *Selenium*. The time comparison results are illustrated in Fig. 13. On average, *Normal Computing* required approximately 2285 s for web crawling. In contrast, the use of *Parallel Computing* in *Selenium* significantly improved efficiency, with an average time of around 295 s. However, *Scrapy* was approximately 15 times faster than *Parallel Computing* in *Selenium*, with an average web crawling time of around 19 s. *Scrapy* is recognized as a highly effective tool for static web crawling, particularly for websites with regular and predictable URLs.

In the third phase of our research, we retrieved DOI numbers and other information related to the reaction formula, and the time comparison results are depicted in Fig. 14. On average, *Normal Computing* in *Selenium* required around 11,766 s for web crawling, while *Parallel Computing* in *Selenium* reduced the time to approximately 1,408 s. Notably, *Scrapy* spent around 70 s, 20 times faster than *Parallel Computing* in *Selenium*.

In the fourth phase of our research, we retrieved reference information based on DOI numbers, and the time comparison results are presented in Fig. 15. On average, *Normal Computing* in *Selenium* required approximately 393 s for web crawling, while *Parallel Computing* in *Selenium* reduced the time to around 53 s. Notably, *Scrapy* took only about 3 s, approximately 131 times faster than *Parallel Computing in Selenium*.

In the final part of our research, we obtained metadata based on record numbers, and the time comparison results are illustrated in Fig. 16. On average, *Normal Computing* in

Selenium required approximately 16,453 s for web crawling, while *Parallel Computing* in *Selenium* reduced the time to around 4,798 s. Nevertheless, *Scrapy* was roughly 171 times faster than *Parallel Computing* in *Selenium*, with an average web crawling time of only about 96 s.

Notably, as the data volume increased, *Scrapy* demonstrated more pronounced pros. With an escalating demand for web crawling, the time required in both Scrapy and Selenium increased; however, the gap in the time spent between the two methods also continued to grow. This underscores the scalability and efficiency offered by *Scrapy* when dealing with larger data-sets and more extensive web crawling tasks. The choice of the method should be aligned with the specific requirements of the data volume and the complexity of the task at hand.

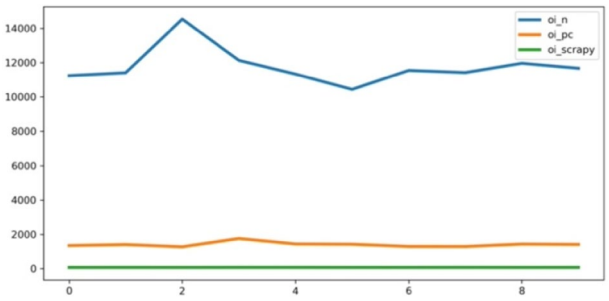


Fig. 14 Compare the time used in *Scrapy* and *Parallel Computing* and *Normal Computing* in *Selenium* to retrieve DOI numbers and other information

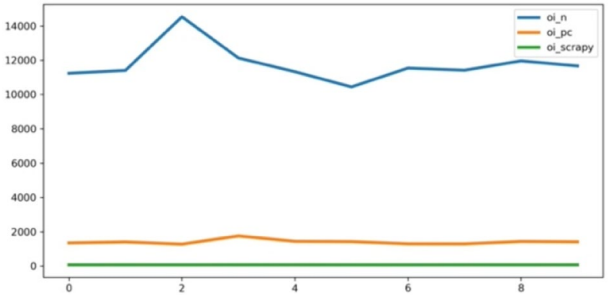


Fig. 15 Compare time used in *Scrapy* and *Parallel Computing* and *Normal Computing* in *Selenium to* retrieve reference information

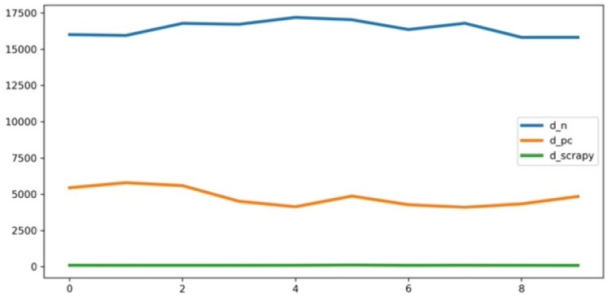


Fig. 16 Compare time used in *Scrapy* and *Parallel Computing* and *Normal Computing* in *Selenium* to retrieve metadata

5 Conclusion

In our research, we applied *Web Information Extraction*, *Natural Language Processing*, and *Parallel Computing* techniques to retrieve reference information, basic data, and metadata related to various species from *NFRI* online database. Through an in-depth analysis of the web structure of *NFRI* and the desired data structure, we were able to simplify the code structure and implement *Parallel Computing* in *Python* to save time. This approach addressed a part of the research gap mentioned in Section 1.

In total, we successfully retrieved 76,893 data points for 112 species, specifically focusing on electron-molecular collision cross-sections. We conducted a comparison of the time required for three different methods and identified that a combined approach for web crawling, involving the integration of *Selenium* with *Parallel Computing* and *Scrapy*, proved to be the most efficient. This method allowed us to significantly reduce the web crawling time, from 12 h down to just ten minutes. Notably, this technique can be applied to dynamic web crawling scenarios, even when the website's URL remains unchanged after sending keywords. The data we collected are instrumental in constructing complex models for plasma processing, and our efforts in generating reliable and accessible electron-molecular collision cross-section data have yielded substantial success.

Overall, these methods and approaches through the combination of *Selenium*, *Scrapy*, and *Parallel Computing* can be applied to a wide range of data collection tasks beyond this specific research. This includes areas such as market research, competitive intelligence, and data aggregation for business purposes, where large-scale data collection and processing are required. This can also improve the efficiency of data-driven processes in industries ranging from e-commerce to healthcare.

However, it's important to acknowledge that both *Selenium* and *Scrapy* have their limitations. *Scrapy*, in particular, faces challenges when attempting dynamic web crawling with *Parallel Computing* within the program due to *Python*'s Global Interpreter Lock (GIL), which restricts parallelism. Overcoming these limitations and exploring potential solutions is a topic that warrants consideration in future work.

Acknowledgements We thank Professor Jonathan Tennyson in University College London and the QDB company for providing the species list during this study. This study was funded by the China Scholarship Council.

Data availability The data in this article are available from National Fusion Research Institute (NFRI) database (https://dcpp.kfe.re.kr/index.do).

Declarations

Competing interests The authors have declared that no competing interests exist.

References

1. Subramani N et al (2022) An automated word embedding with parameter tuned model for web crawling. Intell Autom Soft Comput 32:1617–1632
2. Ayilaran A et al (2019) Reduced chemistries with the Quantemol database (QDB). Plasma Sci Technol 21(6):064006
3. Sharma A et al (2020) Experimental performance analysis of web crawlers using single and Multi-Threaded web crawling and indexing algorithm for the application of smart web contents. Mater Today: Proc 37: 1403–1408

4. Shrivastava G et al (2022) An efficient focused crawler using LSTM-CNN based deep learning. Int J Syst Assur Eng Manag 14(1):391–407
5. Mohd Nain FN et al (2023) Focus web crawler on drug herbs Interaction patterns. Informatica 46:531–542
6. Pavai G, Geetha TV (2017) Improving the freshness of the search engines by a probabilistic approach based incremental crawler. Information Systems Frontiers 19(5): 1013–1028
7. Aru O et al (2021) Development of an intelligent web based dynamic. News aggregator integrating infospider and incremental web crawling technology. Inform Syst Front 15(1):11–22
8. Basaligheh P (2020) Mining of deep web interfaces using Multi Stage web crawler. Int J New Practices Manage Eng 9:11–16
9. Zhang Z et al (2011) A framework for incremental deep web crawler based on URL classification. Web Information Systems and Mining, pp 302–310
10. Bal S et al (2021) IHWC: intelligent hidden web crawler for harvesting data in urban domains. Complex Intell Syst 4:3635–3653
11. Odirichukwu J, Nnamdi R (2023) Web-based igbo thesaurus with real-time retrieval. J Comput SciEng Soft Test 9(1):1–8
12. Christophorou LG, Olthoff JK (2001) Electron collision data for plasma-processing gases. In: Kimura M, Itikawa Y (eds) Advances in atomic, molecular, and optical physics. Academic Press, pp 59–98
13. Christophorou LG, Olthoff JK (2001) Electron collision data for plasma-processing gases. Elsevier Science & Technology 44:59–98
14. Park J-H et al (2020) A new version of the plasma database for plasma physics in the data center for plasma properties. Appl Sci Convergence Technol 29(1):5–9
15. Algiriyage N et al (2018) Distinguishing real web crawlers from fakes. 2018 Moratuwa Engineering Research Conference, pp 13–18
16. Navarrete R et al (2023) Evaluating embedded semantics for accessibility description of web crawl data. AHFE (2023) International Conference, p 94
17. Gao L, Meng Q (2023) Design of crawler and visual interactive interface based on scrapy framework.2023 IEEE 3rd International Conference on Electronic Technology, Communication and Information (ICETCI), pp 1840–1844
18. Adekunle G (2023) Automating data retention from a website using an application programming interface 2:220–226
19. Lapin K (2023) Improving the usability of requests for consent to use cookies. Digital Interaction and Machine Intelligence 191–201
20. Tihi N, Zorjan D (2023) Selenium web driver for javascript and its application in the subject of software engineering. Konferencija sa međunarodnim učešćem napredne tehnologije u obrazovanju i privredi
21. Uzun E et al (2018) Comparison of Python Libraries used for Web Data Extraction 24:87–92
22. Neethidevan V, Chandrasekaran G (2019) Web automation using selenium web driver python. Int J Recent Technol Eng 7:845–847
23. Krishna V, Gopinath G (2021) Test automation of web application Login Page by using selenium ide in a web browser. Webology 18:713–732
24. Kusumo S (2022) Penerapan web scraping Deskripsi Produk Menggunakan Selenium Python Dan Framework Laravel. JATISI (Jurnal Teknik Informatika Dan Sistem Informasi) 9:3426–3435
25. Bhutani N (2023) A review for automating a website using selenium and Java. Int Sci J Eng 2(4):2583–6129
26. Orrequia-Barea A, Marín-Honor C (2020) Scrapy: methodology in extracting user-generated content to compile a Corpus from the internet. In: Current trends in corpus linguistics, pp 119–135
27. Liu C, Tang Y (2023) Research on Chinese content monitoring technology of darknet based on Scrapy. In: International conference on computer application and information security (ICCAIS 2022), pp 455–462
28. Asikri M et al (2020) Using web scraping in a knowledge environment to build ontologies using python and scrapy. Eur J Transl Clin Med 7:433–442
29. Bal S, Ganesan G (2020) SIMHAR - Smart distributed web crawler for the hidden web using SIM + hash and redis server. IEEE Access 8:17582–117592
30. Chen Z et al (2023) Web record extraction with Invariants. Proc VLDB Endowment 16:959–972
31. Li Z et al (2023) WIERT: web information extraction via render tree. Proc AAAI Conf Artif Intell 37:13166–13173
32. Ferrara E et al (2014) Web data extraction, applications and techniques: a survey. Knowl Based Syst 70(C):301–323
33. Bagrudeen BA et al (2022) An efficient mechanism for deep web data extraction based on tree-structured web pattern matching. Wirel Commun Mob Comput 2022:1–10

34. Zhou Y et al (2023) An information extraction method based on improved mixed text density web pages. Expert Systems e13267
35. Patnaik S, Babu C (2021) Trends in web data extraction using machine learning. Web Intell 19:1–22
36. Reddy B et al (2023) Strategies and approaches for Generating identical extensive XML tree instances. Int J Recent Innov Trends Comput Commun 11:559–564
37. Akram Abdulrazzaq A et al (2023) Parallel processing of E-Atheer algorithm using pthread paradigm. Indonesian J Electr Eng Comput Sci 30:1624–1633
38. Gupta C et al (2023) Secure XML parsing pattern for prevention of XML attacks. Information and communication technology for competitive strategies (ICTCS 2022): Intelligent Strategies for ICT 615:759–770
39. Ratana P (2023) Natural language processing and digital literacy in Cambodia. International conference on earth resources and geo-environmental technology 2023
40. Wasiuk P et al (2023) Predicting speech-in-speech recognition: short-term audibility and spatial separation. J Acoust Soc Am 154:1827–1837
41. Sainin M et al (2023) The application of computer-aided under-resourced language translation for Malay into Kadazandusun. Ann Emerg Technol Comput 7:11–24
42. Bharadwaj L (2023) Sentiment analysis in online product reviews: mining customer opinions for sentiment classification. Int J Multidiscip Res 5(5)
43. K, M., et al. (2023) A Survey (NLP) Natural language processing and transactions on (NNL) Neural networks and learning systems. E3S Web of Conferences 430:01148
44. Sreedevi I et al (2022) Word segmentation by component tracing and association (CTA) technique. J Eng Res. https://doi.org/10.36909/jer.15207
45. Chay-intr T et al (2023) Character-based Thai Word segmentation with multiple attentions. J Nat Lang Process 30:372–400
46. Guo S et al (2023) CWSXLNet: a sentiment analysis Model based on Chinese Word Segmentation Information Enhancement. Appl Sci 13:4056
47. Madireddy I, Wu T (2022) Rule and neural network-based image segmentation of mice vertebrae images. Cureus 14(7):e27247
48. Magotra S et al (2023) Takri touching text segmentation using statistical approach. Sādhanā 48(3):0448:103–118
49. Manikandan G et al (2023) Enhanced Ai-Based machine learning model for an accurate segmentation and classification methods. Int J Recent Innov Trends Comput Commun 11:11–18
50. Lei Y et al (2023) CFHA-Net: a polyp segmentation method with cross-scale fusion strategy and hybrid attention. Comput Biol Med 164:107301
51. Tum P (2007) Information retrieval for Khmer documents: Challenges and approaches to word segmentation. In: Monge A (ed) ProQuest Dissertations Publishing
52. Ye J et al (2011) The prefix and suffix query of Chinese word segmentation algorithm for maximum matching. International Conference on Image Analysis & Signal Processing, pp. 74–77
53. Fang H et al (2023) A system review on bootstrapping information extraction. Multimed Tools Appl 1–25
54. Li X et al (2023) Spatio-temporal information extraction and geoparsing for public Chinese resumes. ISPRS Int J Geo-Inf 12:377
55. David EJ et al (2014) Automatic extraction of nanoparticle properties using natural language processing: NanoSifter an application to acquire PAMAM dendrimer properties. PLoS ONE 9(1):83932
56. Libbus B, Rindflesch TC (2002) NLP-based information extraction for managing the molecular biology literature. Proceedings. In: AMIA Symposium. AMIA, pp 445–449
57. Han J et al (2019) Improving the efficacy of the data entry process for clinical research with a natural language processing–driven medical information extraction system: quantitative field research. JMIR Med Inform 7(3):e13331
58. Chaussabel D (2004) Biomedical literature mining: challenges and solutions in the 'omics' era. American Journal of Pharmacogenomics: Genomics-related Research in Drug Development and Clinical Practice 4:383–393

Authors and Affiliations

Qingyang Liu[1] · Ramin Yahyapour[1] · Hongjiu Liu[2] · Yanrong Hu[2]

✉ Qingyang Liu
qingyang.liu@stud.uni-goettingen.de

Ramin Yahyapour
ramin.yahyapour@gwdg.de

Hongjiu Liu
joe_hunter@zafu.edu.cn

Yanrong Hu
yanrong_hu@zafu.edu.cn

[1] Institute of Informatics, Georg-August-Universität Göttingen, Wilhelmsplatz 1 (Aula), 37077 Göttingen, Germany

[2] College of Mathematics and Computer Science, Zhejiang A&F University, Lin'an, Hangzhou 311300, China