UNIVERSITAT POLITÈCNICA DE VALÈNCIA

School of Informatics

Web Data Scraper

End of Degree Project

Bachelor's Degree in Informatics Engineering

AUTHOR: Prieto Roig, Ausiàs

Tutor: Hurtado Oliver, Lluis Felip

Cotutor: Segarra Soriano, Encarnación

ACADEMIC YEAR: 2022/2023

Resum

La recol·lecció de dades, també coneguda com web scraping, és el procés de recolectar informació de pàgines web de manera automàtica. Les pàgines web dinàmiques, caracteritzades per la seua capacitat d'actualitzar el contingut el temps real (AJAX), presenten reptes únics per a la recol·lecció de dades.
La importància d'aquesta informació per a les persones, empreses i investigadors és fundamental, ja que s'hi pot gastar en una gran varietat de propòsits com en l'anàlisi de negòcis, l'investigació i l'ús personal, i fins i tot per a l'entrenament de Models del Llenguatge Grans (LLM).
Describim l'arquitectura, el disseny i l'implementació de una ferramenta modular, extensible i open-source que hem creat amb l'ús de Python, Cython i la llibreria "Playwright", entre moltes altres llibreries de Processament del llenguatge natural (NLP), específicament feta per abordar aquestos reptes, i evaluem el seu rendiment a través d'experiments i estudis de casos.
Com a part de la reflexió, concluïrem discutint les possibles aplicacions de les dades recolectades en aquestes pàgines web dinàmiques i les tendències a futur d'aquest camp.

Paraules clau: Extracció d'informació, Scraping, Crawling, Pàgines web dinàmiques, AJAX, Playwright, Data harvesting, Open-source, Python, NLP

Resumen

La recolección de datos, también conocida como web scraping, es el proceso de recolectar información de sitios web de manera automática. Los sitios web dinámicos, caracterizados por su capacidad de actualizar el contenido en tiempo real (AJAX), presentan desafíos únicos para la recolección de datos.
La importancia de esta información para las personas, empresas e investigadores es fundamental, ya que se puede utilizar para una variedad de propósitos como el análisis de negocios, la investigación y para el uso personal, e incluso para el entrenamiento de Modelos del Lenguaje Grandes (LLM).
Describimos la arquitectura, el diseño e implementación de una herramienta modular, extensible y open-source que hemos creado usando Python, Cython y la librería "Playwright", entre otras muchas librerías de Procesamiento del lenguaje natural (NLP), específicamente hecha para abordar estos retos, y evaluamos su rendimiento a través de experimentos y estudios de casos.
Como parte de la reflexión, se concluirá discutiendo las posibles aplicaciones de los datos recolectados en estos sitios web dinámicos y las tendencias a futuro en este campo.

Palabras clave: Extracción de información, Scraping, Crawling, Páginas web dinámicas, AJAX, Playwright, Data harvesting, Open-source, Python, NLP

Abstract

Data harvesting, also known as web scraping, is the process of collecting information from websites automatically. Dynamic websites, which are characterized by their ability to update content in real-time (AJAX), present unique challenges

for data harvesting.
The importance of this information for people, companies and researchers is paramount, as it can be used for a variety of purposes such as business intelligence, research and personal usage, and even for training Large Language Models (LLM).
We describe the architecture, design, and implementation of a modular, extensible and open-source data harvesting tool that we have created using Python, Cython and the library "Playwright", among many other Natural Language Processing (NLP) libraries, specifically made to address these challenges, and whose performance we will evaluate through experiments and case studies.
As part of the reflection about this tool, we will conclude by discussing the potential applications of data harvested from these dynamic websites and the future trends in this field.

Key words: Information Extraction, Scraping, Crawling, Dynamic websites, AJAX, Playwright, Data harvesting, Open-Source, Python, NLP

Contents

List of Figures

List of Tables

List of algorithms

CHAPTER 1
Introduction

Ever since the dawn of humanity, keeping track of the information available was of uttermost importance.

The act of writing was the solution of a basic problem: How do I store this information so that if I forget or I am not available, this information can still be used.
After writing became paper, then printing, then magnetic storage and finally digitization. Take a look nowadays, the Internet contains an unimaginable amount of information of all kinds. However, too many information can be a problem for regular people who just have no time to read all texts, articles or websites that appear whenever you do even the simplest search.
But how can we access and use data more effectively?
How can we find relevant and reliable data sources among the vast amount of information available on the web?
How can we extract and organize data in a way that suits our needs and goals?
These are some of the challenges that data gathering tools aim to address. However, not all data gathering tools are created equal.

In this work, I will introduce Dynamic Data Gathering Tool (GDGTool), a data harvesting tool[1] that addresses the three main aspects of this discipline. Our tool allows users to search, filter, and download data from various websites in a fast and easy way. It also provides a modular and configurable architecture that enables all kind of users to customize their data gathering tasks according to their preferences and requirements. Moreover, it leverages containerization technology to enhance its scalability and portability across different platforms and environments.
I will explain how GDGTool approaches these issues, and compare it with other existing tools in the market. For this, our tool is based on three techniques that work together to perform the basic tasks required to collect data from dynamic sources.

We introduce the concept and the utility of Web Data Harvesting, composed by Web Crawling[2], Web Scraping[1] and Data Gathering[3].
Web crawling[2] is the process of iteratively finding and fetching web links starting from a list of seed URLs. It is what search engines[3][4][5] do to index web pages and discover new content. Web crawlers have to scrape the web pages they

visit to extract the links and other information.
Web Scraping is the act of accessing the internet as a human would do, with the intent to gather information autonomously. This implies that any program capable of doing information gathering should be capable of understanding language at least in a basic level. Thanks to recent developments in Natural Language Processing (NLP)[6][7], several options of pre-made multi-dimensional relationship word embeddings[8] or lexical network databases[9] (like Wordnet[10]) have become easily available, which allows the creation of intelligent tools of data scrapping[11] with the aforementioned knowledge.
Data gathering is more of a general term that refers to collecting any kind of data from any source. It can be done online or offline, with web scraping or other methods. Data gathering is often the first step in data analysis or data mining.

1.1 Motivation

Data harvesting using a scraper[1] is a powerful and widely used technique used to extract information from the internet. It has become increasingly popular as businesses and individuals recognize the benefits of quickly and efficiently gathering large amounts of data[11][12]. It involves extracting data from websites and converting it into a format that can be easily analyzed and used[13]. This information can help improve decision-making, identify trends and patterns, and provide new insights into topics of interest[12]. However, data scraping also carries risks and potential liability if not handled with care. Personal data posted online can expose users to identity fraud, harassment, and other threats[14].
Developing new data scraping tools also provides opportunities for research and innovation in the field. With the increasing complexity of websites and security measures implemented to block bots[15], there is a need for new techniques, methods, and tools that can navigate these challenges.

Existing data scraping tools can be challenging to use, and often require advanced technical skills. As a result, there is a growing demand for more user-friendly and accessible tools that can handle large amounts of data and extract data from a wide range of websites and formats. Such tools could enable more people to benefit from data scraping and improve their work or research.
Additionally, the overwhelming amount of information online makes finding reliable information a time-consuming and intensive task[16]. A computer can tirelessly and objectively search for the most relevant topics, rate sources of information, and search for information in multiple languages, among thousands of human-read-hours in mere seconds. For example, these tools can help users to manage the overwhelming amount of data is by providing advanced analytical capabilities, or by providing access to a wide range of data sources.
Data scraping tools can perform complex statistical analysis, natural language processing, and machine learning algorithms to identify patterns and trends in the data. This can help users to uncover insights and make informed decisions based on the data and gather a comprehensive view of a particular topic or issue.
Finally, the data collection could be enhanced by providing real-time data updates. This can be really useful in industries such as finance or marketing, where

access to real-time data is essential for decision-making.

It is clear that this kind of tools can help many people, ease their lives and improve their knowledge. More knowledgeable people means further advances in science and technology, and big amounts of treated data can also help technologies like the , in their creation or in the information gathering.

1.2 Objectives

In this work, we will focus on three objective users, ranked by their expertise, and try to focus the development based on their needs.
We will define the 'Non-expert user' as a non-programmer that wants to search things on the internet, and any added capability must be presented in a simple way.
Next, we have the 'Programmer user', a user capable of programming and building programs themselves, but that doesn't know about crawling or scraping processes. This user can use APIs to build their own tools with their specific needs.
Finally we take into account the 'Expert user', that is capable of programming, but it also has enough knowledge to understand the processes that happen in the tool under the hood, and so it is capable of improving it.

The main objectives of this work are:

1. To develop a fast, modular and reliable framework for data scraping, scalable that is able to efficiently extract data from a wide range of websites and formats.
   This could be achieved by ensuring a modular architecture, capable of perform crawling and data gathering fast, efficiently and reliably. Also, an enhancement could be to compile the source code into Cython[17]. The tool should ensure:

   - Performing base crawling is as fast as possible, to make sure that it does not block more complex and slower workflows.
   - The tool can handle various scenarios where data gathering would be useful.
   - Expert users can add optimized functionalities that make use of the whole tool's potential, or even extend it.
   - Programmers easily can build generic tools that make use of crawling and scraping.

2. To develop solutions to ensure users of different expertise can use the tool to better fit their needs, without spending too much time.
   For example, lesser expert users could make use of a graphical user interface, with easy buttons and controls to access some of the tool's capabilities, and programmers can have an API set to make using the program easier

3. To ensure repeatability and security in the tool. This is necessary because at some point this tool may make automated work that can't be supervised by the user.

1.3 Sustainable Development Goals

The Sustainable Development Goals (SDGs) [18] are a set of global objectives established by the United Nations to address the most pressing social, economic, and environmental challenges faced by humanity. Adopted in 2015, the SDGs provide a comprehensive framework for achieving sustainable development worldwide. In line with the principles of the SDGs, our focus within the field of study and research has centered on three specific goals:

4. **Quality Education**: By providing users with easy access to information and data through our tool, we aim to empower individuals, particularly those with limited resources to pursue learning, intellectual growth and broaden their knowledge horizons. We believe our tool can help to bridge the educational divide and contribute to a more equitable and inclusive education system.

8. **Decent Work and Economic Growth**: Our tool seeks to facilitate productivity and efficiency, allowing researchers and professionals to streamline their tasks and focus on higher-value activities. By automating repetitive processes and providing data insights, we aim to achieve greater job satisfaction in jobs where data extraction is imperative and repetitive, improve productivity, and reduce research time spending, fostering economic growth.

10. **Reduced Inequalities**: We strive to create an open tool where individuals from diverse backgrounds and contexts can harness the power of data without barriers or discrimination. By embracing inclusivity and promoting equitable access to resources, our tool contributes to breaking down information silos and narrowing the knowledge gap among different communities. With its ability to gather and analyze data from diverse sources, it empowers individuals, particularly marginalized communities, to access accurate and timely information. By providing a means to combat misinformation, such as contrasting xenophobic articles, our tool enables minorities to stay informed about news and events that directly impact them. This increased access to relevant information can foster a greater sense of inclusion and empowerment, ultimately working towards reducing inequalities and promoting a more equitable society.

As an extended goal, by automating repetitive tasks and streamlining data gathering processes, the tool can enhance researchers' productivity and efficiency. Researchers play a vital role in advancing knowledge and addressing societal challenges. By equipping them with effective tools and resources, this research strives to facilitate their work and ultimately contribute to the betterment of society.
Lastly, not only researchers but research itself can directly help society. Research

and technological advances that require large amounts of data can highly benefit from this kind of tools. Trendy data hungry deep learning and in particular Large Language Models, have recently proved the impact that they can have in society. For example, GPT-3 required around 600B tokens and LLaMA 1.4T tokens, most of which were extracted from websites using technologies such as crawling and data harvesting [19] [20].

1.4 Related Coursework

The "related coursework" section provides an opportunity to highlight the specific subjects taken during college years that have contributed to the development of the tool and the completion of this work. These courses have provided us with foundational knowledge and skills that are directly applicable to the project:

- Information storage and retrieval systems: This subject played a crucial role in making the entire project possible. It provided us with a theoretical understanding of how information can be stored and retrieved efficiently. It also equipped us with practical knowledge on various techniques and technologies used in modern information retrieval systems, which directly influenced the design and implementation of the tool.

- Statistics: Our knowledge of statistics has been instrumental in understanding data processing and incorporating statistical analysis into our project. This understanding has allowed us to effectively handle and interpret data, enabling informed decision-making throughout the development process.

- Intelligent systems: This subject provided us with a solid foundation in machine learning and natural language processing (NLP). The concepts and techniques learned in this course have empowered us to leverage ML and NLP algorithms and methodologies in our tool, enhancing its capabilities in areas such as data extraction and understanding of textual information.

- Concurrency and distributed systems: The concepts learned in this course have been valuable in understanding and implementing asynchronous processing and resource management within our tool. We have been able to design and orchestrate the efficient utilization of computing resources, enabling concurrent operations and improving overall performance.

- Data Structures and Algorithms: Our knowledge of data structures and algorithms has been pivotal in the creation of our tool. It has allowed us to make informed decisions regarding the choice of appropriate data structures, ensuring optimal performance while accommodating various use cases and requirements.

- Automata theory and formal languages: This course has inspired us to explore graph-based approaches and motivated us to try to incorporate graph structures and workflows into the project.

- Computability and complexity: Understanding computability and complexity theory has been beneficial in terms of performance considerations. Our knowledge in this area has guided us in developing algorithms that avoid performance issues, ensuring that our tool operates efficiently without encountering exponential complexity problems.

1.5 Structure of the report

Through the whole report, we provide an overview of our decisions in the making of the Data Harvesting tool.

In the Literature review, we start by reviewing and comparing the literature that previous researchers have performed, the current state of the research about data crawling, data scraping and data harvesting tools and techniques, and what parts of the research have not been addressed properly or that could be explored further, to then chain the discussion with the strengths and weaknesses of this kind of tools.
We also take advantage of the knowledge gathered for this last section to explain how this weaknesses can be mostly overcome, how did other researchers try to overcome them, and how our tool is designed around them.

In the Design and implementation chapter, we make use of that knowledge to define the design and decisions taken, the architecture that makes everything possible and the technologies used, to end with a description of how the tool is used and how can it be useful for many people.
This empathises the modularity oriented design that guides the project, which allows expert users to adapt and improve the project, for the benefit of everyone.

In the Evaluation and testing we then present the results of the evaluation of our tool, and how it compares to other tools available for this purpose. We also identify the most prominent challenges faced during the development of the tool, how they affected the tool, and how we overcame them.

Finally, we conclude with the Conclusions, where we report the accomplishments of the project, the lessons learned, and the expected improvements for future versions of the tool. This report provides a comprehensive overview of the design, implementation, and evaluation of our Data Harvesting tool and offers insights for researchers and practitioners in this field.

CHAPTER 2
Literature review

The literature review provides context and background information on the topic being studied. It helps to establish the relevance and significance of the research, and shows how the study fits into the existing body of knowledge on the subject. It is a critical component of the research methodology, as it provides the foundation and justification for the research design and methods used in the study. It allows the researcher to draw on the work of others, to build upon their findings, and to avoid duplicating previous research.

2.1 Review of the existing research on data scraping tools and techniques

Data scraping, also known as web scraping[1], is a process of extracting structured information from unstructured sources, such as web pages, documents, and many other formats, both computer-oriented and human-oriented. Data scraping can enable various applications, such as querying, organizing, and analyzing data, integrating data from multiple sources, and creating knowledge bases[12]. However, data scraping is also a challenging task that requires dealing with noisy, heterogeneous, and dynamic data.

Several competitions and studies have been conducted in the development of this field, yet many of these studies tend to focus on specific fields or particular web pages, which limits their generalizability and applicability to other domains. When the research is not focused on one specific website, the existing literature often classifies data scraping tools into two main categories:
Either they use Machine Learning, classical template "wrappers" or a combination of both, using Machine Learning to classify websites into the wrappers provided to the algorithm.

Machine learning methods rely on statistical models that learn from labeled or unlabeled data to identify and extract relevant information they are known to provide more generalization, at a much higher computational cost, both for training and analysis. Template wrappers are instead rules that specify how to locate and extract information from a given web page based on its structure and layout. This approach is less computationally expensive, but won't work unless the website is similar to the wrappers generated. Some hybrid methods combine machine learning and template wrappers to leverage both approaches, training the models to either create the wrappers or most commonly to classify the given website into one of the available patterns.

In her book, S.Sarawagi presents a detailed research on all types of data extraction tools that is appraised by most of the literature[21]. She provides a taxonomy of information extraction methods along various dimensions, such as the input sources, the output format, the extraction techniques, and the management issues. She also discusses the challenges and opportunities for future research in this field.

Online scraping can be split into three interwoven primary processes: Website analysis, website crawling, and data organizing[22]. Website crawling refers to systematically navigating through web pages to extract relevant data. It involves following links, handling different page layouts, and managing the traversal of websites. Website analysis involves examining the structure, content, and behavior of websites to determine the most effective scraping approach. Data organizing focuses on structuring and storing the collected data in a meaningful and usable format for further analysis and processing.

While web scraping is a vital component of data mining, it is important to differentiate between the two. Web scraping is primarily concerned with collecting data from various sources, such as web pages, documents, and other formats, regardless of the data's analysis. On the other hand, data mining goes beyond data collection and involves analyzing and interpreting the already collected and formatted data to discover patterns, relationships, and insights.

Data mining encompasses a wide range of sophisticated statistical techniques and algorithms. These techniques include Statistical/Machine Learning approaches such as classification, clustering, regression of missing data, and anomaly detection, or more classical algorithms like Decision trees and Association Rules, which are among the popular methods utilized in data mining [23] [24].

2.2 Identification of gaps in the existing literature and areas for further research

While there has been a considerable amount of research on data scraping, there are still several gaps in the literature. One area that has not been given enough attention is modularity, specifically the use of plugins and extensions in data scraping tools. These plugins can enhance the functionality of the scraping tools by providing additional features and capabilities that are not available in the core tool. This is related to one of the forementioned points that we noticed during the literature review: The fact that most researchers focus these kind of tools to only one field. This explains why there is not much research that takes into account the modularity, but also shows that a modular tool could unify efforts and provide researchers with an abstraction layer that handles all crawling and orchestration.

We have also found out that some research use of containerization tools like Docker[25] to package the application and its dependencies in a single deployable unit, but more importantly because it is crucial to ensure the repeatability of data scraping experiments. This is an important issue because the accuracy and reliability of the results obtained from data scraping experiments can be affected by a variety of factors, including the operating system and software environment used to run the experiments, and more often than not, scraping is a critical process where changes in the system like package or OS updates can not become a point of failure. Containerization tools can help ensure that these factors are standardized across different experiments, adding to the whole research process.

Another gap in the literature is in the integration of data scraping tools with web browsers to make them more accessible to non-experts. This is an important issue because many potential users of data scraping tools may not have the required technical expertise needed to use them effectively. By integrating data scraping tools directly into web browsers, these tools can be made more user-friendly and accessible to a wider audience. A downside of this approach is that testing must be performed on humans, who are not always consistent in their criteria for determining relevance or accuracy. This introduces a level of subjectivity and variability in evaluating the usability of the tools, as human users may have different preferences and opinions for evaluating the effectiveness of the tools. Nonetheless, the benefits of making data scraping tools more accessible to non-experts could outweigh this challenge, as it could empower users to gather and analyze data more efficiently and effectively.

Furthermore, little attention has been given to graph-storage structures, which can be useful in storing and analyzing data relationships and connections. Since data scraping tools find many kinds of data from multiple sources, we believe that heterogeneous graphs can be a very valid data structure. For one, writing graph-structured data is not slower than formatting the data into other more common formats like CSV or JSON. Graphs allow by definition a dynamic number of nodes of any kind in a non-euclidean space, and can be a great and easy way to visualize the data gathered. However, the most significant advantage of using graph-storage structures is the potential to bring applying graph machine learning techniques closer to research

.

PDF reading is another area that has been overlooked in the current literature on data scraping. PDFs are a widely used document format, and extracting information from them can be a valuable source of data. However, extracting information from PDFs is really challenging. This is really useful because the most cutting-edge, state of the art information is usually only packaged as PDF, which are important to users that need the most updated accurate information. Although there are several packages capable of reading PDFs, it is sometimes overlooked the integration with data scraping tools in a way that makes it seem like they are not relevant.

In terms of areas for further research, one promising area is graph structure and graph ML. Graphs can be used to represent complex relationships between data points, and machine learning techniques can be used to extract useful insights from graph data. We already mentioned this as a gap in the literature, but we firmly believe that this requires further research on how to effectively use graph ML techniques in the context of data scraping.

Another area where more research is needed is the integration of data scraping tools with object detection and segmentation models like YOLO. These models can be used to identify and extract specific objects or regions of interest from images or other visual data, which can be useful in many data scraping applications, but most importantly, it can learn common sense knowledge required to successfully navigate websites, a point that we will discuss in the discussion about the strengths and limitations.

Another area for further research is chat-like access to information. This involves developing natural language processing algorithms that can interpret and respond to user queries in a conversational manner. This would allow non-technical users to easily access and extract raw or aggregated information from a variety of data sources once the data gathering process has finished possibly even allowing the user to get a better insight from the data.

2.3 Discussion on the strengths and limitations of these tools

Data harvesting tools have many strengths that make it an interesting topic for the research of its many applications. For one, the baseline of comparison is human performance, which although peak in many aspects, lags behind roughly any computer in reading, memorizing, and navigating through large volumes of data, identifying relationships between concepts, recognizing patterns and keeping focus in one task, aspects at which computers excel, and that coincidentally are core to the process of harvesting data, which makes them a promising tool for many day to day uses.

However, there are certain limitations to data harvesting tools that must be taken into consideration. One of the most significant limitations is the lack of outside-world knowledge and pattern recognition skills that humans possess. While humans can interpret and rely on symbols and icons on websites to navigate with ease, this also explain why computers often struggle to explore websites and extract relevant information. This can be a significant limitation, especially since websites are created and designed for humans, which can also be worsened by the included bot-evasion measures. This implies that some websites are easier to access than others, which would mean for the tool to unknowingly prioritize information for the ones that are more bot-accessible, even if the information is less useful or less reliable

Another significant challenge is the accuracy of the information being harvested. Aggregation problems may arise when the majority of the harvested data is incorrect or unreliable, and while there algorithms can help approximate trustworthy information sources by their relationships with other websites, computers still lack the contextual understanding and the ability to interpret information in the same way that humans do. This can make it challenging for computers to distinguish between reliable and unreliable sources, particularly in cases where the data is complex or ambiguous.

This two limitations if combined can become even worse, for inaccuracies in data (which can be seen as error in the process of harvesting), can be compounded by navigation and extraction problems, if for example "unreliable" or hoax websites are easier to access or have fewer security measures in place, leading to inaccurate, flawed or biased insights and incorrect conclusions.
Attackers could potentially exploit these limitations to deliberately manipulate the data collected by data harvesting tools, further compromising the accuracy and reliability of the harvested data. For example, an attacker may create fake websites specifically designed to mislead data harvesting tools.

Mitigating the risks associated with the limitations of data harvesting tools is necessary to ensure that the insights generated from the data are valid and reliable. However, it also adds complexity to the tools, as it requires a combination of automated and manual techniques to verify the accuracy and reliability of the data being collected. This involves using techniques such as sentiment analysis, fact-checking, and manual review to detect misleading information prior to the generation of the final conclusions.

Although the strengths of the tools exceed their limitations, it is clear that their limitations are more important because they directly affect the quality of output of the tools, in some cases this implying that the tool is rendered useless after the process has completed.

2.4 Approaches to the overcoming of the weaknesses

Addressing the weaknesses related to navigation in data scraping, we found a backtracking exploratory algorithm that analyzes interactable elements within the Document Object Model (DOM) of web pages, as discussed in the paper "Crawling Ajax-Based Applications"[26]. This algorithm provides a potential solution to navigate through web pages that heavily rely on dynamic content and Ajax requests.

However, we want to note that the presence of bot traps, designed to catch automated crawlers, can pose challenges to this kind of navigation approaches. We believe that this algorithm is a good way to approach this problem, and it could potentially be improved by screenshot hashing techniques and the use of ML (see ALGORITHM 1). However, they may not suffice in scenarios involving videos or animations, where the content is dynamic and constantly changing in random periods of time.

ALGORITHM 1: Dynamic crawling using ML algorithm built upon an algorithm found in the literature

```
Require: Array $URLs$ of strings
Require: Float $minRank$
Require: ML Model $WebsiteEncoder$
Require: ML Model $VisionCNN$

  Procedure MAIN(URLs)
    begin
    $browser \leftarrow newBrowser()$
    $sortedStates \leftarrow newSortedStateMachine()$
    $sortedStates.URLs \leftarrow \text{sort}(sortedStates.URLs \cup URLs)$

    $browser.crawl(sortedStates, minRank)$
  end procedure

  Procedure BROWSER.CRAWL(sortedStates, minRank)
    begin
    while $sortedStates.length > 0$ do
      $currentState \leftarrow sortedStates \setminus \{0\}$
      $currentURLScore \leftarrow browser.getUrlScore(currentState.URL)$
      if $currentURLScore \leq minRank$ then
        break
      end if
      $browser.loadState(currentState)$
      $browser.explore()$
      $encodedState \leftarrow WebsiteEncoder.encode(currentState)$
      if $sortedStates.encodings.contains(encodedState)$ then
        continue
      else
        $sortedStates.encodings \leftarrow \text{sort}(sortedStates.encodings \cup \{encodedState,\})$
      end if
      $Clickables \leftarrow VisionCNN.GetCandidateClickables(encodedState)$
      for $c \in Clickables$ do
        if $c.targetURL \neq currentState.URL$ then
          $sortedStates.URLs \leftarrow \text{sort}(sortedStates \cup c.targetURL)$
        else
          $newState \leftarrow browser.click(c)$
          $sortedStates \leftarrow \text{sort}(sortedStates \cup newState)$
          $BACKTRACK(currentState)$
        end if
      end for
    end while
  end procedure
```

While our tool does not extensively employ Machine Learning, we recognize that AI is crucial to overcome these limitations effectively. We propose that incorporating AI models capable of gathering external world information could be a valuable solution. Although such models may lack the millions of years of evolutionary knowledge in human brains, we believe that a model capable of performing both real-world navigation tasks and website interactions tasks can be performant in generalizing and comprehend new, unfamiliar icons and elements in common websites.

To ensure the use of reputable sources we believe that the best option is relying on well-established industry-standard algorithms such as Google's Hummingbird, which can be improved by advanced Natural Language Processing (NLP) techniques to significantly improve the quality prediction of the information source. By incorporating these algorithms into our tool, we can be more future-proof as these Large Language Models (LLM) grow rapidly.

To address data accuracy potential issues, we propose performing data mining on processed data represented as triplets. We would utilize lexical relationship databases like WordNet[10] to generate dynamic entities based on attribute variance and frequency, creating a flexible window of information. The relevance and weight of these entities would be determined by the predicted trustworthiness of the respective websites, as well as the ratio of affirmations to negations concerning the statements. Although we are aware that an algorithm capable of reaching pure truth is impossible, we believe that aggregations of reliable sources can average out some errors

Finally, we advocate for the application of Reinforcement Learning in Heterogeneous Graph Machine Learning (GML) [27] [28] to capture and utilize all available information to make predictions across multiple domains simultaneously. By training one model in different predictions, we can get it to have a broader understanding of the problem in hand to tackle the weaknesses of data scraping in a unified manner, enabling more accurate and robust results.

CHAPTER 3
Design and implementation

The high loads of data, expected reliability and the limitations in performance of the most critical part of the scraping implies that the resources that are available in the computer must be orchestrated and balanced as best as possible. Modularity also plays a big role, since the best tool is but text if no one can comprehend it and hence use it. At last, it must be taken into account all possible use cases that any user might have.

We know that the whole process will be bound by I/O operations, mostly TCP/IP HTTP connections to the websites to retrieve the website and its data, done during the scrapping process.
We can optimize this by using asynchrony and by blocking some non-useful resources. As so, all processes that the scraper performs must be asynchronous. The data storage does require somewhat faster I/O processes, them being disk writing.
Then, the data extraction an processing requires mostly CPU computation. Although those have remarkably faster frequencies, the data extraction may be really heavy computationally-wise, so we can optimize this by dividing the data in processes among all processors in the computer.
Finally, we can optimize the plain Python code by compiling it into C++ using Cython[17], an Python Module that performs this compilation to speed up run time.
All this can be added in a docker container to help with reliability, repeatability and security during the use of the tool.

3.1 Description of the design and the architecture

The design of the data harvesting tool is based on a modular and extensible approach, which allows for easy customization and integration of new features. It is divided into several main components (See Figure 3.1):

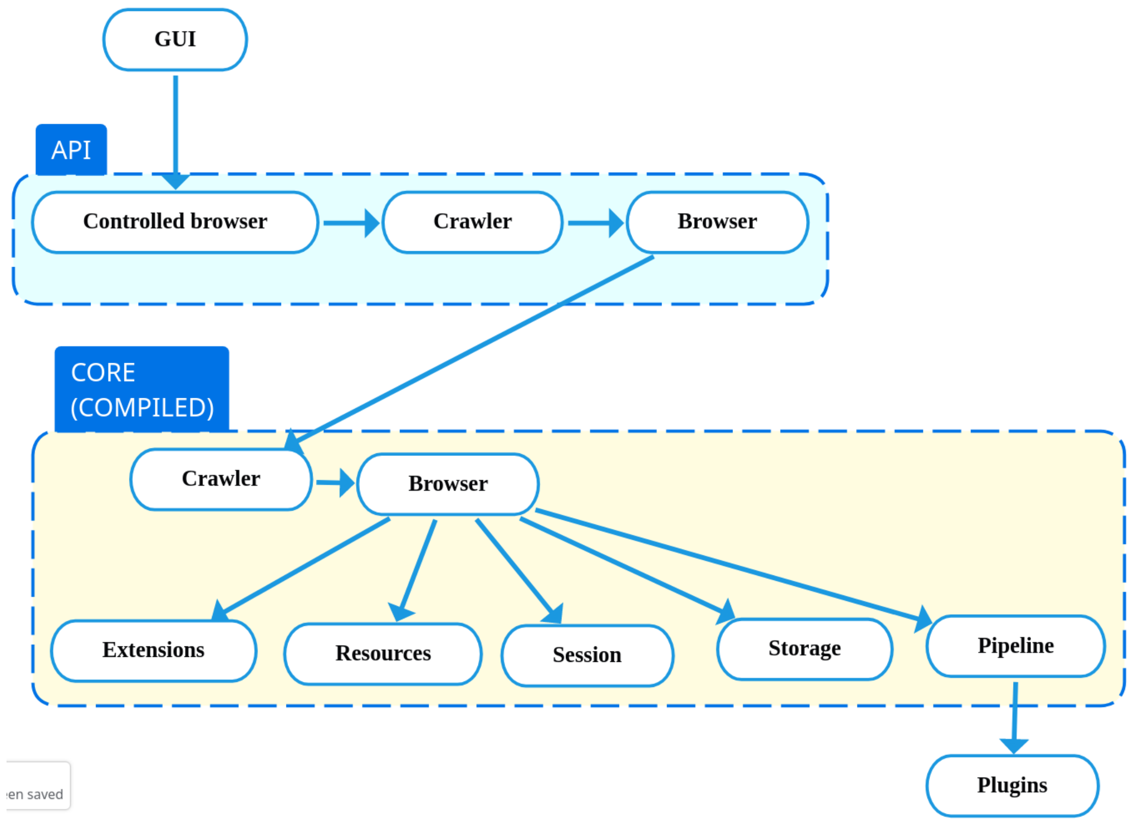


Figure 3.1: The dependency flow of the GDGTool

- GUI: The Guest User Interface allows non-expert users to use this tool by providing an easy and familiar interface and only showing simple buttons that invoke complex pre-made macros that make use of the API section.
- API: The API section provides a slow but safe interface for interacting with the tool. It includes type and value checking to ensure that the input data is valid and conforms to the expected format. This is the intended approach for users that want to extend the tool to his/her own needs but do not require a fast interaction with it.
- Core: The core section contains the main bulk of the code and is responsible for the fast and efficient execution of the scraping process. It is compiled code that handles the web crawling, data extraction, and data storage.
  - Extensions: The extensions component contains optional code that must be integrated to the core component manually due to performance or compatibility. These extensions provide additional utilities such as rotating proxies, which can be enabled or disabled as needed.
  - Resources: This component manages the shared resources for all the project to ensure that these resources are not unnecessarily instantiated multiple times throughout the program, which can lead to inefficiencies and unnecessary overhead.
  - Session: The session component is responsible for managing the storage and retrieval of the program's state when the program needs to pause or stop. This enables the tool to resume from where it left off,

preserving the progress made and avoiding redundant computations, and allows users to resume if issues are encountered.

- Storage: This component is responsible for managing the dynamic storage requirements of the collected data. Relying on a data storage library, this component temporarily holds the gathered information in RAM as it continues to accumulate. Once certain conditions are met, such as a specified memory limit or a predetermined data size, the Storage component initiates the process of dumping the accumulated data into the chosen storage format.
- Pipeline: The pipeline component manages all data handling and processing in the tool. By calling plugins at certain points during the execution of the tools, we get a fully modular tool that is both adaptable and performant.

- Plugins: The plugins component contains optional code that can be added dynamically at different stages of the scraping process. These plugins are independent and run during the harvest, but are not integrated into the core component. An example of a plugin is the video harvesting.
- Tests: The tests component contains a set of automated tests to ensure that the tool works correctly and that the different components are functioning as expected.

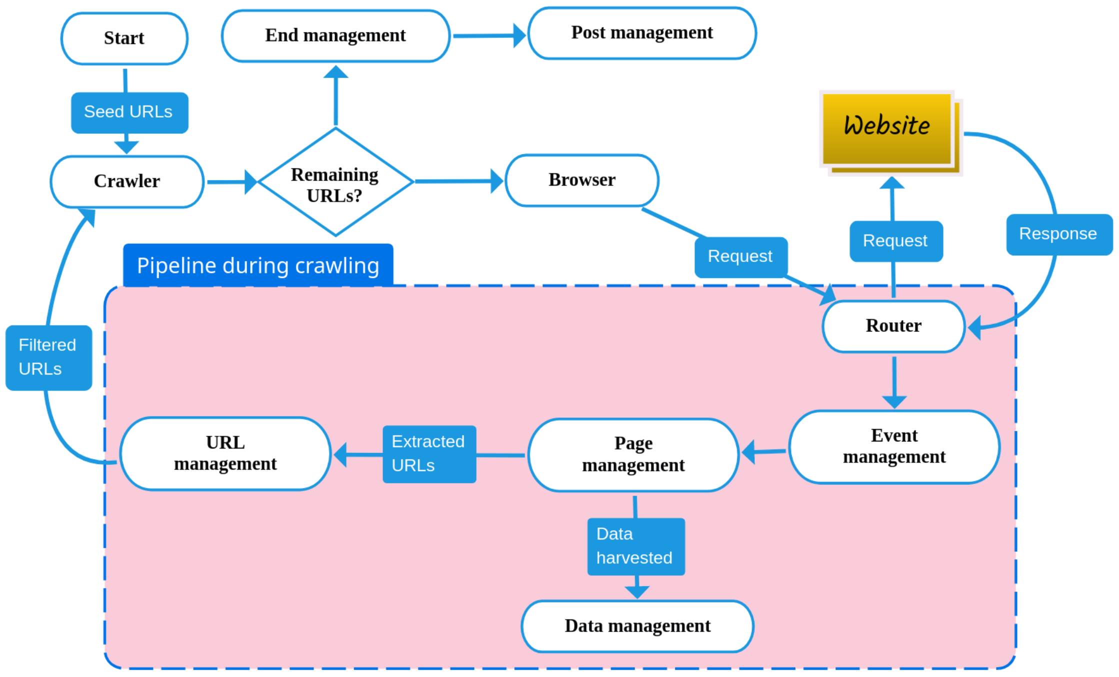


Figure 3.2: The data management steps invoked by the pipeline

This architecture allows for flexibility and scalability, as different parts of the tool can be added, removed or modified as necessary. Additionally, the use of asynchronous operations ensure that it can handle large amounts of data and high loads of requests. By using optional modules such as rotating proxies and other kind of plugins, it is more versatile and able to adapt to different personal situations. Furthermore, the use of a user interface makes it easier to use than any API-based tool.

One of the main features for the tool to be modular and extensible is the 'Plugins' component. It makes use of the pipeline component, which is a static number of steps that are executed during the crawling step. The steps are fixed to improve the performance of the tool by making it easier to optimize them, to manage memory, and to test the whole environment (See Figure 3.2):

1. Router: This step allows the plugins to run once for every request that the browser sends/receives when it matches a defined pattern. This could be used to block unwanted, heavy resources, not useful for the scraping process, or to keep track of the resources consumed during the scraping process.

2. Event management: This step allows the plugins to run when a specific event is detected. This can be used to detect popups, or to handle errors when the website loading fails.

3. Start: This step allows the plugins to run before the actual crawling process starts, this could be used for example to setup the environment, set specific configurations or pre-process data. For example, to set up a connection to a database where the scraped data will be stored.

4. Page management: This step allows the plugins to run once the website is loaded. This can be used for example to extract all wanted data individually per website, without performing any modifications. A plugin that extracts structured data such as prices, product names, and ratings could be added to the page management step.

5. Data management: This step allows the plugins to run once all page management is done. This can be used for example to generate data using all the extracted data from the previous step, for example to do some data analysis, data extraction, etc. The plugin could check the extracted data to change the search behaviour of the tool, focusing more on the products/websites with lower prices.

6. URL management: This step allows the plugins to run for every URL visited during the crawling process. This can be used for example to block unwanted URLs, remove parameters or perform other URL-specific operations.

7. End management: This step allows the plugin to run once the crawling process has finished, this could be used for example to close resources, create a final report or notify that the process has ended. For example, it could be created a email notification about the end of the scraping process, indicating the statistics of the execution.

8. Post management: This step allows the plugins to run once all data has been successfully stored. This is useful for running expensive methods, such as data analysis or further data processing that are not critical for the scraping

process but are crucial for the final outcome. It could be used to perform machine learning techniques to analyze the scraped data and generate insights.

3.2 Technologies used

Python is the primary programming language used in the development of the Data Harvester. Its versatility, extensive libraries, and ease of use make it an ideal choice for implementing various functionalities and components of the tool. Python with the Playwright[29] library for crawling, and industry standard NLTK[30], WordNet[10], Spacy[31], Pytorch[32], Tensorflow[33] and Scikit-learn[34] provide robust support for web scraping, text processing, and machine learning, enabling seamless integration of different modules and algorithms.

Although we initially leveraged starting the project in C / C++ or Java, but we reached the conclusion that Python is more fitting because of the number of data processing libraries available allows us to target the development of areas where the research is more present.

Due to the performance intensive tasks that we perform, Cython is used to allow us to write Python code that can be compiled to C or C++ extensions, enabling faster optimized execution speeds in critical sections of the code.

Playwright is a powerful automation library that enables browser automation and interaction. It supports multiple web browsers, including Chrome, Firefox, and Safari, allowing the Data Harvester to navigate through web pages, interact with elements, and extract desired information. However, the most important point about Playwright is that has an active development, which ensures that the library won't fall behind unsupported in the next few years.

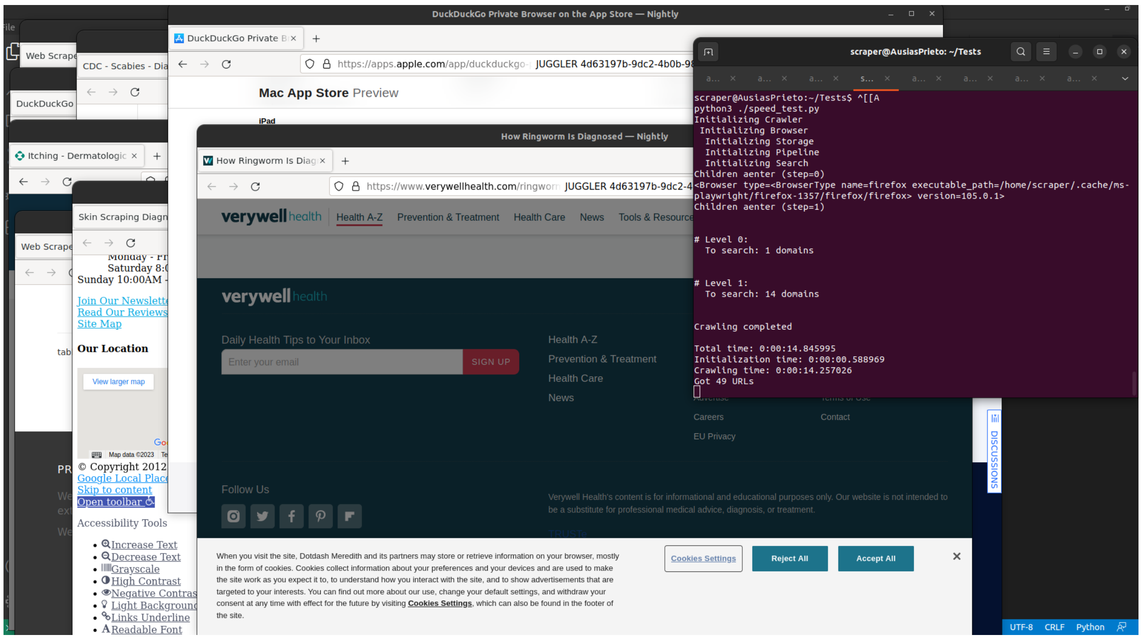
During the initial tests, we built tools using Python Scrapy and Selenium[35] to compare to Playwright's performance. However, they lacked some features we believe are key to the success of our project like we envisioned it from the beginning.
Selenium allowed to have a visual browser for non-expert users to interact with, but since Selenium was not made to be a crawling library, it lacked many features required for this purpose. The most important for us was the lack of asynchronous behaviour, that significantly slows down the process.
Scrapy[36] was at the other end, with great crawling capabilities but missing native AJAX handling. In its official documentation, Scrapy suggests using Playwright as a bridge to resolve AJAX requests. Also, with Scrapy handling the crawling, we couldn't fully control the browser during the process. We settled up in Playwright.
For data processing and specially NLP pipelines we have the NLTK, WordNet, Spacy, Pytorch, Tensorflow and Scikit-learn libraries that pretty much handle any data processing requirement that we may have. Other than that, we also have Coreferee[37], a coreference resolution[38] library that we have found to be more

performant than other libraries. Having coreference resolution allows our NLP pipelines to better handle the extracted texts.

Pretty much all libraries tested ended up in the toolbox because every one of them excel at one process best than the others. NLTK and Scikit-learn has many useful preprocessing methods, WordNet allows semantic relationships to connect entities extracted by Spacy, which also allowed for PartOfSpeech tagging. Pytorch and Tensorflow was used to try out pre-trained summarization models which ended up not giving good enough results.
Finally with Coreferee we also tried NeuralCoref[39], but not only used far more RAM, but it also had collisions with other packages.

3.3 How does the tool work

The tool uses a combination of web scraping, data parsing and data analysis techniques to automate the process of extracting data from different websites. It is designed to optimize the performance and efficiency by using asynchrony, multiprocessing, and encapsulating everything inside a Docker[25] container.



```
scraper@AusiasPrieto:~/Tests$ ^[[A
python3 ./speed_test.py
Initializing Crawler
 Initializing Browser
  Initializing Storage
  Initializing Pipeline
  Initializing Search
Children aenter (step=0)
<Browser type=<BrowserType name=firefox executable_path=/home/scraper/.cache/ms-
playwright/firefox-1357/firefox/firefox> version=105.0.1>
Children aenter (step=1)


# Level 0:
  To search: 1 domains


# Level 1:
  To search: 14 domains


Crawling completed

Total time: 0:00:14.845995
Initialization time: 0:00:00.588969
Crawling time: 0:00:14.257026
Got 49 URLs
```

Verywell Health's content is for informational and educational purposes only. Our website is not intended to be a substitute for professional medical advice, diagnosis, or treatment.

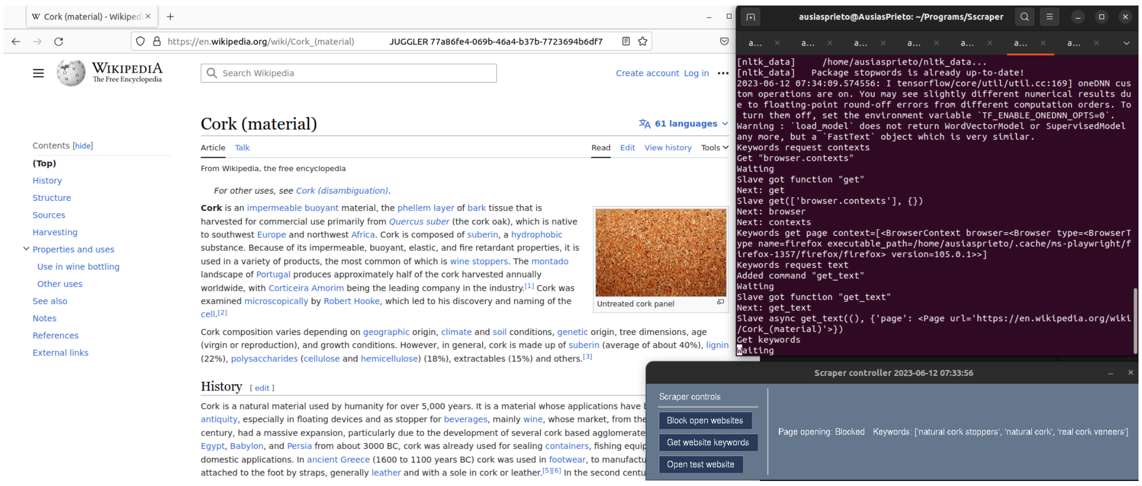
When you visit the site, Dotdash Meredith and its partners may store or retrieve information on your browser, mostly in the form of cookies. Cookies collect information about your preferences and your devices and are used to make the site work as you expect it to, to understand how you interact with the site, and to show advertisements that are targeted to your interests. You can find out more about our use, change your default settings, and withdraw your consent at any time with effect for the future by visiting **Cookies Settings**, which can also be found in the footer of the site.

Figure 3.3: The tool during the crawling process. On the left side, eight windows requesting different URLs. On the right side, the console with the logs and timing metrics

Using a web crawler, navigates through the website and follows links to find the relevant pages to scrape (See Figure 3.3). The web crawler uses a browser to retrieve the pages using asynchrony, which allows multiple requests to be sent and received simultaneously, improving the performance and speed of the scraping process. This can also be run using rotating proxies.

The data analysis step uses multiprocessing, that allows the tool to use multiple

CPUs or cores to process different parts of the data at the same time, improving the performance and speed of the data analysis process.



Cork (material)

From Wikipedia, the free encyclopedia

For other uses, see Cork (disambiguation).

Cork is an impermeable buoyant material, the phellem layer of bark tissue that is harvested for commercial use primarily from *Quercus suber* (the cork oak), which is native to southwest Europe and northwest Africa. Cork is composed of suberin, a hydrophobic substance. Because of its impermeable, buoyant, elastic, and fire retardant properties, it is used in a variety of products, the most common of which is wine stoppers. The montado landscape of Portugal produces approximately half of the cork harvested annually worldwide, with Corticeira Amorim being the leading company in the industry.[1] Cork was examined microscopically by Robert Hooke, which led to his discovery and naming of the cell.[2]

Cork composition varies depending on geographic origin, climate and soil conditions, genetic origin, tree dimensions, age (virgin or reproduction), and growth conditions. However, in general, cork is made up of suberin (average of about 40%), lignin (22%), polysaccharides (cellulose and hemicellulose) (18%), extractables (15%) and others.[3]

History [edit]

Cork is a natural material used by humanity for over 5,000 years. It is a material whose applications have b
antiquity, especially in floating devices and as stopper for beverages, mainly wine, whose market, from the
century, had a massive expansion, particularly due to the development of several cork based agglomerate
Egypt, Babylon, and Persia from about 3000 BC, cork was already used for sealing containers, fishing equip
domestic applications. In ancient Greece (1600 to 1100 years BC) cork was used in footwear, to manufactu
attached to the foot by straps, generally leather and with a sole in cork or leather.[5][6] In the second centu

```
[nltk_data]     /home/ausiasprieto/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
2023-06-12 07:34:09.574556: I tensorflow/core/util/util.cc:169] oneDNN cus
tom operations are on. You may see slightly different numerical results du
e to floating-point round-off errors from different computation orders. To
 turn them off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
Warning : `load_model` does not return WordVectorModel or SupervisedModel
any more, but a `FastText` object which is very similar.
Keywords request contexts
Get "browser.contexts"
Waiting
Slave got function "get"
Next: get
Slave get(['browser.contexts'], {})
Next: browser
Next: contexts
Keywords get page context=[<BrowserContext browser=<Browser type=<BrowserT
ype name=firefox executable_path=/home/ausiasprieto/.cache/ms-playwright/f
irefox-1357/firefox/firefox> version=105.0.1>>]
Keywords request text
Added command "get_text"
Waiting
Slave got function "get_text"
Next: get_text
Slave async get_text((), {'page': <Page url='https://en.wikipedia.org/wiki
/Cork_(material)'>})
Get keywords
Waiting
```

Page opening: Blocked   Keywords: ['natural cork stoppers', 'natural cork', 'real cork veneers']

Figure 3.4: The tool controlled by the user in GUI mode. On the left, the browser that the user sees. On the right, the console with the logs and the action buttons, showing the result of the keyword extraction functionality

For easy user control, the project may be used with a controller-UI with buttons to execute actions and relevant information displayed (See Figure 3.4).

All the processes are wrapped up inside a Docker container, which allows the tool to run isolated and separated from the host system, for security and repeatability and also make it easy to deploy and run the tool in different environments. This can also make the tool more portable.

Finally, all data is stored in a format that is easy to access and analyze, such as a text file, CSV, or database. This can be chosen by the user.

CHAPTER 4
Evaluation and testing

The evaluating and testing phase is an important step in the development of any data harvesting tool. This phase serves to ensure that the tool is working correctly and that it is able to extract the desired data accurately and efficiently.

During this phase, we run the tool against four test cases:

- Running the tool and Scrapy[36] as a baseline against the same website for crawling. The chosen website was "https://crawler-test.com/", that we choose because of its many different edge cases and loops. This was used to test for performance and resource usage.

- Running the tool against broken links. For this, we modified the tool not to remember which URLs has already visited, and we tested against "https://crawler-test.com/", which also provides with broken links.

- Running the tool data processing against Wikipedia articles. This was used to test the NLP Keyword extraction and unsuccessful summarizer.

- Running the tool against test captchas like the ones in "https://nopecha.com/demo".

The test cases' results were compared to the expected outcomes. This allowed us to identify any issues or bugs, such as missing or incorrect data, and to make any necessary adjustments that may be necessary.

It was also important to evaluate the performance of the tool by measuring how long it takes to extract the data and the amount of resources it uses, to optimize the tool and make it more efficient.

Additionally, we must test the tool against a range of different websites, including those with different structures, technologies, and levels of complexity, to ensure that the tool can handle a variety of different scenarios.

Furthermore, testing for edge cases, like very big and complex websites, 404 errors, CAPTCHAs, among others, and also testing for different use cases of the tool, can help to identify any limitations of the tool and consider them when designing the final product.

4.1 Methods and metrics

Evaluating the performance of a data harvesting tool is an important step in the development process, as it helps to ensure that the tool is able to extract useful data efficiently. We will use the following metrics to evaluate the performance of a data harvesting tool:

- Scraping speed: This metric measures the time it takes for the tool to extract data from a website. It can be measured in pages per second or in total time to extract the data.
- Resource usage: This measures the amount of resources (such as CPU and memory) that the tool uses while extracting data. This can help to identify any bottlenecks in the tool and to optimize its performance.
- Scalability: To measure the ability of the tool to extract data from large websites with long texts. It can be measured by the amount of data the tool can extract in a certain amount of time.
- Robustness: This metric measures the ability of the tool to handle errors and unexpected situations, such as broken links, CAPTCHAs, and changing website structures.
- Data Quality: This one is really important, as it measures the quality of the data that the tool extracts. It can be measured by comparing the extracted data to the expected results, or by evaluating the completeness and accuracy of the data.
- Usability: This metric tests on the non-expert users to detect shortcomings of the usability of the tool. It can be measured by providing a questionnaire to the users before and after testing the tool.

To test each metric we will set up specific scenarios oriented to one specific metric. This metric-oriented scenarios will be the base to the final analysis that we will perform and showcase about the tool.

- Scraping speed test: Test the tool extracting data in a large number of pages in the least amount of time. This will help to evaluate the tool's scraping speed and identify any bottlenecks that may be limiting its performance. A real case would be to make a search in the main search engines (Google, Bing, DuckDuckGo, Yahoo...) with a crawling depth of 2 at least and dump the data into files
- Resource usage test: Record the resource usage of the tool while performing the extraction. This can be done by measuring the CPU and memory usage of the tool during the different scenarios and compare between all the results. A hard case would be to perform the extraction of the data while performing actual analysis of the data (for example, generate a network of statements)

- Scalability test: Test the handling of large and complex websites by extracting data from a website with a large number of pages and a complex structure. This will help to evaluate the tool's scalability and identify any limitations that may be preventing it from handling large websites. A real case would be to set the tool to make the extraction in dynamic websites with infinite scrolling shuch as Reddit among others.
- Robustness test: Test the tool under unexpected situations, such as broken links, CAPTCHAs, and changing website structures. This can be done by scraping a website with known issues and evaluating the tool's ability to handle the errors and unexpected situations that may arise. We can find broken links or blocked websites and check how the tool handles this situations
- Data quality test: Test the tool's ability to extract high-quality data by comparing the extracted data to the expected results. This can be done by extracting data from a known website and comparing it to a known dataset or by evaluating the completeness and accuracy of the data. An example would be to pick a known topic from wikipedia and compare the extracted information to our personal knowledge and the one available on wikipedia.
- Usability test: Ask non-technical users for evaluation of the tool's usage, if the tool is easy to use and understand for the intended user. This can include testing the tool's user interface, the ease of configuring and running the tool, and the ease of interpreting the results. This must be performed on real users, and can include many of the features that the tool provides, and also with a free, more broad test giving the user total freedom

4.2 Results of the evaluation

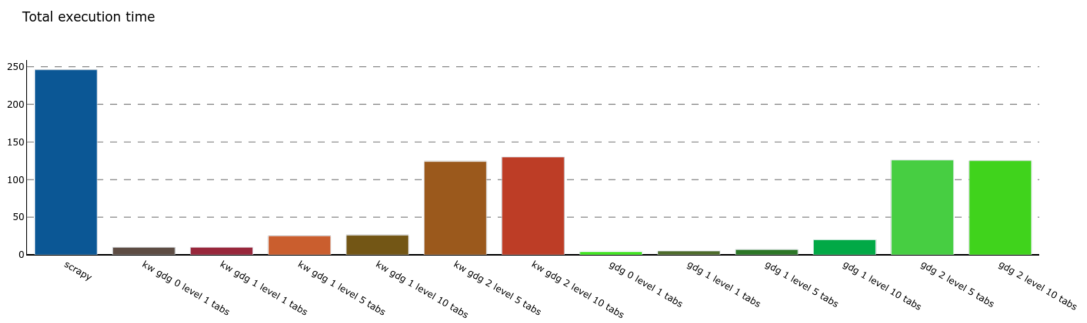


Figure 4.1: The tool's different runs doing crawling and scraping (green), crawling, scraping and keyword extraction (red) versus the Scrapy baseline time (blue)

The Figure 4.1 plots the execution times of many tests of the GDGTool.

The first group, shown in blue, represents the Scrapy baseline performance. This run involved crawling and scraping data at depth level 2. Compared to the other groups, the Scrapy baseline took the longest time to complete the task.

The second group, depicted in red, represents multiple runs of the GDGTool performing crawling, scraping, and keyword extraction. Each bar in this group corresponds to a specific configuration, denoting the depth level and the number of tabs used. Notably, as the number of tabs increases, the execution time also increases, which can be attributed to the additional time required for tab initialization. Similarly, the execution time tends to be higher for greater depth levels, as the number of URLs to explore increases exponentially.

The third group, shown in green, comprises the same runs as the red group but without keyword extraction. These runs are noticeably faster compared to the corresponding runs in the red group. This demonstrates that keyword extraction adds a significant amount of time to the overall execution process.

This shows the clear relationship between the number of tabs, depth levels, and execution time. It indicates that the GDGTool's execution time increases with a higher number of tabs and greater depth levels, aligning with the expected behavior considering the increased workload and exploration complexity.

We humbly acknowledge that Scrapy, being a more mature and widely adopted software, may have certain considerations that contribute to its longer execution time compared to our GDGTool. We believe that the timing difference observed is likely due to factors that we might not have fully considered or optimized for in our tool's current implementation. It is possible that Scrapy incorporates certain mechanisms or functionalities necessary for ensuring accurate and reliable results, which might require additional computational time. Alternatively, the integration of Cython compilation in our tool may contribute to its relatively faster performance.

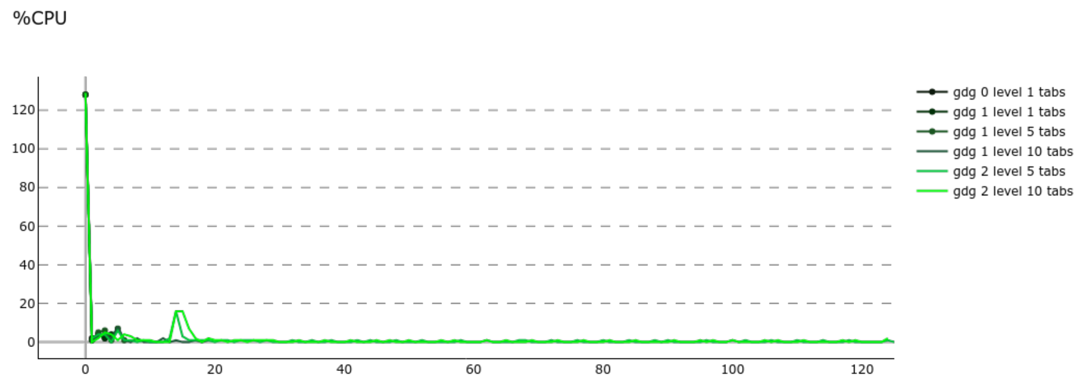


Figure 4.2: The tool's CPU usage of the multiple runs doing crawling

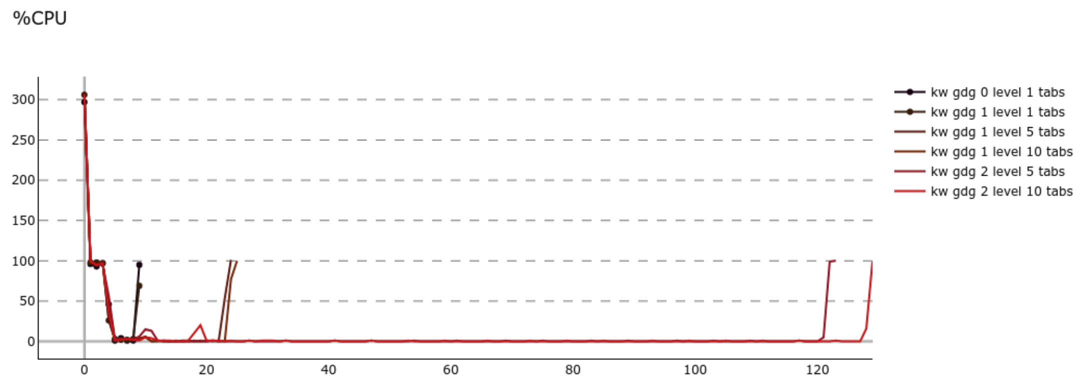


Figure 4.3: The tool's CPU usage of the multiple runs performing crawling and keyword extraction from the data gathered

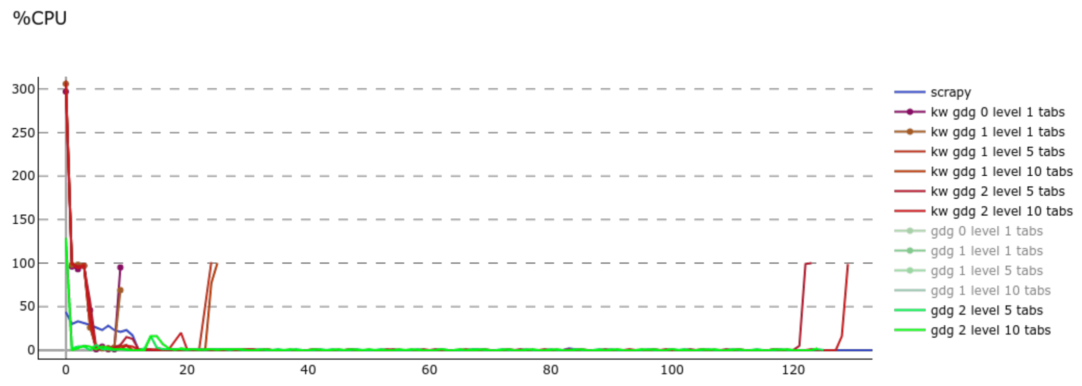


Figure 4.4: The tool's CPU usage of the base crawling and data scraping (green) and keyword extraction (red) compared to the Scrapy CPU usage during the same task(blue)

Figure 4.2 illustrates the CPU usage of the tool during multiple runs focused on crawling and scraping data. The graph shows that the CPU usage remains relatively low, except for an initial spike where the CPU cost temporarily exceeds 100%. This spike can be attributed to the concurrent execution of multiple threads during the tool's initialization process. Additionally, two minor bumps in CPU usage can be observed, which likely correspond to the periods when the tool analyzes the URLs.

Figure 4.3 presents the CPU usage of the tool during multiple runs involving crawling and keyword extraction from the gathered data. Similar to the previous figure, the CPU usage exhibits a similar pattern. However, in this case, the initial CPU usage period is longer, likely due to the initialization of machine learning models used for keyword extraction. The graph ends with a low bump, indicating the tool's completion of the keyword extraction process.

Figure 4.4 compares the CPU usage of the GDGTool (green) and keyword extraction (red) with the CPU usage of Scrapy (blue) during the base crawling and data scraping task. Using Scrapy as a baseline, the same test was conducted in both tools. The graph demonstrates that the GDGTool generally exhibits lower CPU usage at the beginning of the task, but as the task progresses, the CPU usage of both tools converges to a similar level. Additionally, it is worth noting that the GDGTool completes the task faster than Scrapy, as evidenced by the earlier completion point on the graph.

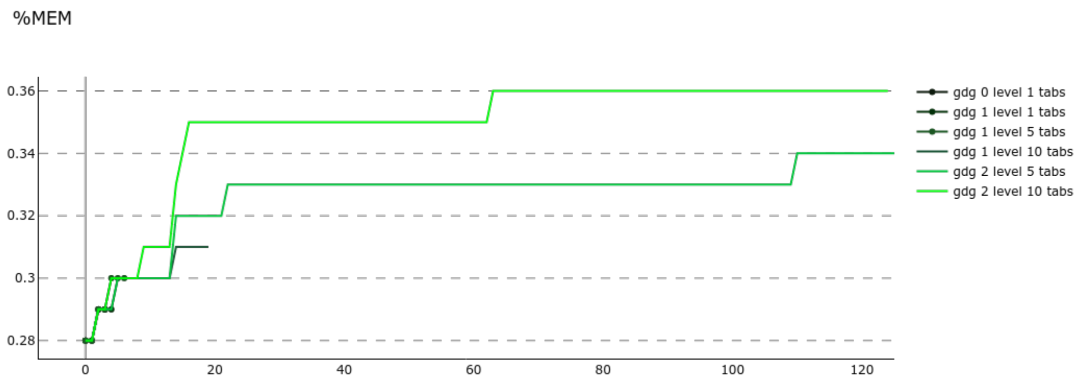


Figure 4.5: The tool's RAM usage of the multiple runs doing crawling

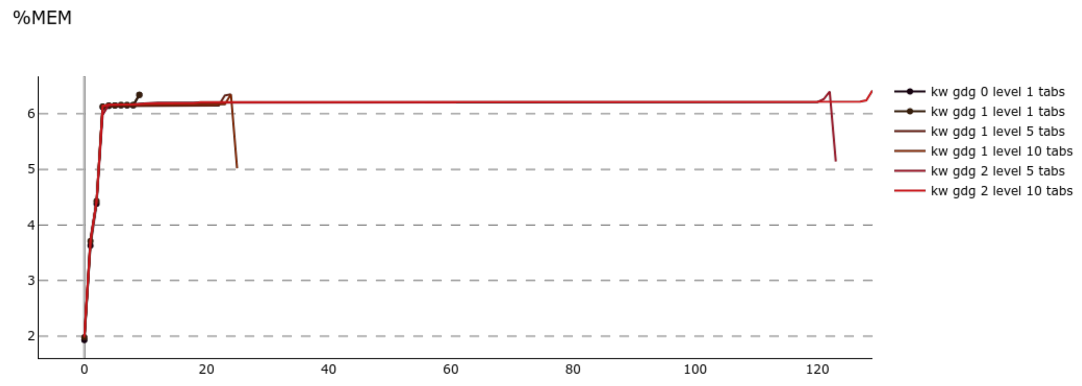


Figure 4.6: The tool's RAM usage of the multiple runs performing crawling and keyword extraction from the data gathered

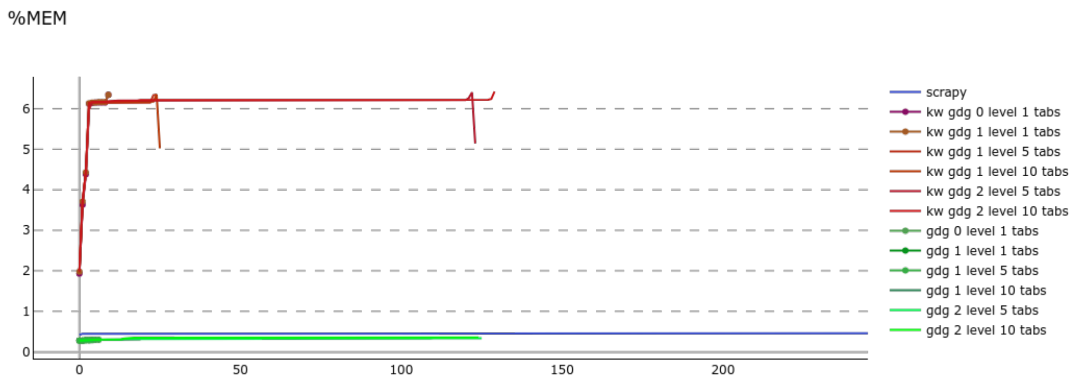


Figure 4.7: The tool's RAM usage of the base crawling and data scraping (blue) compared to the scrapy RAM usage during the same task(purple)

The RAM usage plots provide insights into the memory utilization of the system during the tool's execution.

In Figure 4.5, we observe that the memory usage is influenced by both the depth levels and the number of tabs. Notably, the runs that explore higher levels, such as (level 1, 10 tabs) and (level 2, 5 tabs) and (level 2, 10 tabs), exhibit increased memory usage. This can be attributed to the larger volume of data stored in memory as more levels are traversed and additional tabs are loaded. Interestingly, the runs at level 2 demonstrate a logarithmic-like curve, which we believe is due to the cumulative delays introduced by websites with longer loading times.

In Figure 4.6, which represents keyword extraction, a spike in memory usage is observed at the beginning, followed by a relatively stable curve. This can be attributed to the loading of NLP models, which consume a significant portion of memory. Subsequently, the memory usage remains relatively constant as the keyword extraction process proceeds, indicating that the additional memory requirements after model initialization are comparatively minor.

Figure 4.7 provides a comparison between the aforementioned runs and the Scrapy baseline. Notably, the runs that do not involve loading NLP models demonstrate lower memory usage compared to the Scrapy baseline. Conversely, the runs where NLP models are loaded exhibit higher memory usage than the Scrapy baseline, as expected due to the additional memory requirements of the models.

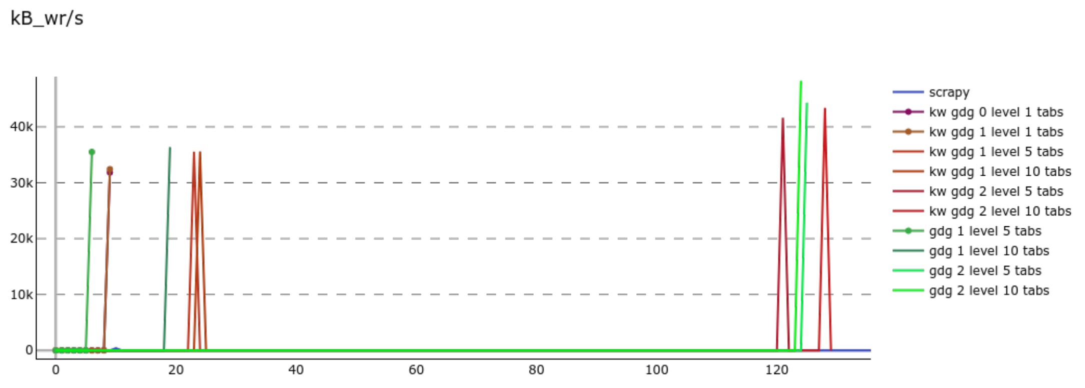


Figure 4.8: The tool's disk writing usage of the base crawling and data scraping (green), with keyword extraction (red) and compared to the scrapy disk writing usage during the same task (blue)

The Disk writing plot represents the amount of memory dumped from RAM to the disk and provides insights into the disk writing behavior of the tool.

In Figure 4.8, we observe the memory usage for data crawling and scraping (green), keyword extraction (red), and the Scrapy baseline (blue). Our tool exhibits a single spike at the end of the run, indicating the dumping of all data to the disk. In contrast, Scrapy shows a flatter line, suggesting minimal disk writing activity throughout the process.

It is important to note that while our approach of delaying the data dumping until the end reduces disk writing frequency, it also implies a higher risk in case of unexpected events such as power outages. To address this, we have implemented dynamic disk dumping capabilities that allow the tool to adaptively dump data based on the amount of memory being held. However, since this particular test scenario was not highly demanding in terms of memory usage, the data was not dumped earlier. Nonetheless, we recognize the need to enhance the tool's reliability by implementing measures to handle potential failures, such as power disruptions, and ensure the integrity of collected data.

This analysis highlights the trade-off between keeping data in memory and the cost of disk writing. While our tool optimizes disk writing by selectively dumping data, it also necessitates improvements in terms of reliability and resilience.

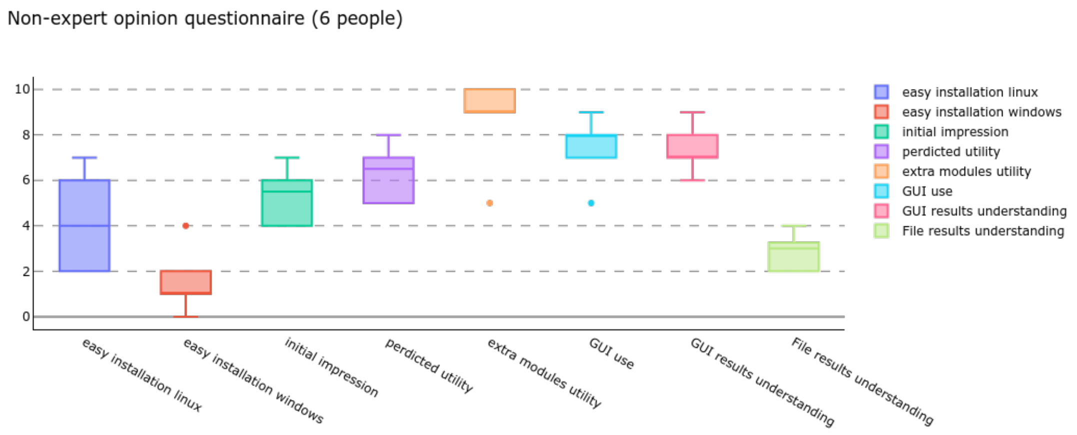


Figure 4.9: The results of the questionnaire we made for non-experts to test our tool usability (More is better)

During the evaluation phase showcased in Figure 4.9, we sought feedback from six non-expert test users regarding their experience with the tool's ease of use.
The questionnaire used to get this information had eight questions with scores from zero to ten, where zero is unusable / impossible and ten is really useful / really easy. The eight questions referenced the tool in non-expert cases that the users could find themselves in.

While the overall ratings were not exceptional, averaging around 6 out of 10, the feedback provided valuable insights into areas that required attention.

- The Linux installation score represents the subjective opinion of the users regarding the ease of installation. Users rated their experience based on their perception of the installation process, without considering the average time it took. The mean score was 4.17 out of 10, ranging from 2 to 7. We believe that this score represents how used was the user to see a console before the use of the tool and the difficulties they had to find and follow the README.

- The Windows installation score also represents how easy it was to install the tool on Windows. The mean score was 1.5 out of 10, with a range of 0 to 4. These scores indicate that users found the installation process somewhat challenging, particularly in Windows. We believe this might be a specially bad score due to the lack of guides and plug & run versions. Users had to install docker and run it, which was not intuitive.

- The initial impression scores capture the users' first perceptions of the tool before actually using it. The mean score was 5.33 out of 10, with a range of 4 to 7. This shows that the first impressions were average, but not impressive.

- The predicted utility scores shows the expectations and potential value that the user foresees for this kind of tool. The mean score was 6.33 out of 10, ranging from 5 to 8. These scores indicate a moderate level of satisfaction and positive expectations among the users.

- The extra modules utility refers to the perceived usefulness of additional modules or functionalities that could be integrated into the tool, but have not been implemented yet. Users rated this aspect with a mean score of 8.67 out of 10, suggesting that they recognized the potential value of expanding the tool's capabilities.

- The GUI use scores represents the ease of use that the users experience when using the GUI version of the tool. Users perceived this mode as a regular browser mode they are used to plus the utilities that the tool brings. The scores had a mean of 7.5 out of 10, which we can understand as an over-average score. We believe most of this is earned because of the similarities with conventional browsers.

- The GUI results understanding test shows the reported understanding of the users about the information showcased in the GUI version of the tool, about the actions that the users took in it. The scores' mean was of 7.33 out of 10, indicating that although not all users were fully comfortable, the tool is less lacking in the communication with the user in this regard.

- Regarding the file results understanding, users reported difficulties in comprehending the information presented in the comprehensive JSON-format results file generated by the tool during the whole execution. The mean score for file results understanding was 2.8 out of 10, ranging from 2 to 4. This indicates a significant challenge for users in navigating to and interpreting the data stored in the file. Improving the accessibility and clarity of the file results could enhance the overall user experience.

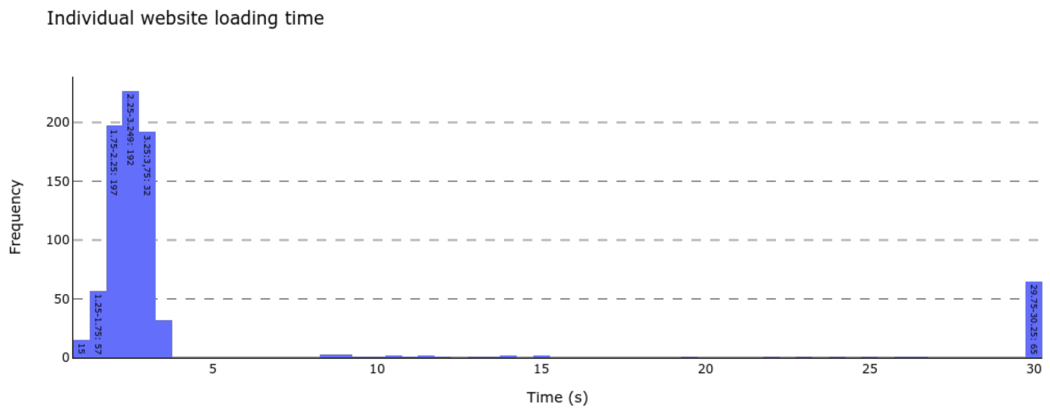


Figure 4.10: Am histogram representing the frequency of website loading time on a run with broken links

In the Figure 4.10 we see robustness test performed against a website that has known broken links. It shows that outliers aside, website loading takes about 4 seconds at maximum per tab, and outliers take about 30 seconds before a timeout is set. This is aligned with our settings that try loading the website for 30 seconds and if it is not done loading, it raises a Timeout.

Finally, the data quality test was a critical part of the analysis of the results. Initially our tool was to generate summaries of all the data gathered, but this was rendered impossible in the time we had since many issues arose. After many trial and error we set up by providing the already tested keyword extraction algorithm textrank[40] by the library PyTextRank[41], which worked fine given enough phrases.

4.3 Challenges faced

During the creation of the Data Harvester, several challenges were encountered that required careful consideration and innovative solutions. These challenges encompassed various aspects of the tool's development, ranging from technical complexities to design considerations. Let's explore each challenge in detail:

- Asynchrony: One of the primary challenges faced was dealing with asynchronous operations. As web scraping often involves interacting with multiple web pages simultaneously, managing the asynchronous nature of these operations became crucial. Ensuring proper synchronization and coordination of tasks while efficiently utilizing system resources required it, so we used the Python asyncio library.

- Modularization: Creating a modular and extensible architecture was another significant challenge. Modularization allows for easy integration of new functionalities, enhances code re-usability, and facilitates maintenance. Designing a well-structured and modular system architecture required careful planning and visualization of the general problems that the tool will tackle. It was decided that this would be used during crawling to get data, process it, and take decisions with it. This allows the tool to change its behaviour depending on the results .

- Compiling into Cython: Optimizing critical sections of the code for improved performance posed a challenge. Leveraging Cython, the Python code was compiled into C or C++ extensions to achieve faster execution speeds. This limited core classes because everything had to be defined before compilation, and a new set of '.pxd' files needed to be created.

- Storage: Efficiently storing and managing the harvested data was another challenge. As the Data Harvester collects vast amounts of information from multiple sources, designing an effective storage mechanism was crucial. Although the default is stored using JSON files in folders, this is customizable, and in the future we want to implement Graph storage in hdf5 files.

- Pipeline design: Designing an efficient and scalable data processing pipeline was a complex task. The pipeline needed to handle various stages of data extraction, transformation, and analysis seamlessly.

- Crawling algorithm: Developing an effective crawling algorithm that could navigate through websites, handle dynamic content, and avoid common pitfalls such as bot traps presented a significant challenge. Implementing intelligent algorithms that could mimic human-like behavior while efficiently exploring web pages required a deep understanding of web technologies and clever algorithmic design.

- Integration with the browser: Integrating the Data Harvester with web browsers to simulate user interactions presented unique challenges. Overcoming issues such as handling JavaScript-heavy websites, dealing with complex rendering engines, and efficiently capturing desired data required leveraging browser automation tools like Playwright and overcoming inherent limitations.

- Long processing times for NLP data: Processing natural language data can be computationally intensive and time-consuming. Challenges included handling large volumes of text, applying complex NLP algorithms, and managing resource-intensive tasks such as language parsing, entity recognition, and sentiment analysis. Optimizing the NLP pipeline to reduce processing times while maintaining accuracy and reliability was a significant challenge.

- API design: It being a key part of the project meant that this task wouldn't be easy. Taking into account all possible problems that the tool should be able to complete while taking all performance and concurrent tasks into account is not an easy task. We ended up adding four steps for data processing because it is a critical and intensive task, so the user should decide when is best to perform which analysis.

- Data analysis: Using NLP models to perform data analysis is not easy, but initially we had in mind to provide a summary of the data gathered. This turned out to be really hard, and impossible in the time we had. Language models tend to be really large, which sometimes would not fit in our 16 GB RAM. Also, extractive summarizing, whose models are not so big doesn't work that well.

CHAPTER 5
Conclusions

In conclusion, the development and implementation of the GDGTool has become a way for us to dive deeper into a whole range of new processes.
Throughout this project, we have explored and learned from various challenges, such as handling asynchrony, modularization, compiling into Cython, storage management, pipeline design, crawling algorithms, integration with browsers, and the processing of NLP data. By using technologies like Python, Cython, Playwright, NLTK, WordNet, Spacy, and Neuralcoref, we have been able to create a tool capable of efficient data extraction, analysis, and storage.

This open-source project has been developed with the aim of making advanced data scraping capabilities accessible to users with varying levels of expertise. By providing a modular and user-friendly tool, we strive to empower a wide range of users, including researchers, analysts, and professionals, to harness the power of data extraction and analysis.

Our work has successfully achieved all of the initial objectives outlined in this project.

- We have developed a fast, modular, and scalable framework for data scraping, capable of efficiently extracting data from a wide range of websites and formats. The architecture of the framework has been designed to prioritize speed, efficiency, and reliability (see Figure 4.1).
  This fulfills our objective as our tool ensures the three main required functionalities that we established in our objectives, plus the optional goal:
  - Crawling performance can be compared to other used libraries available like Scrapy[36]. This means that our tool has space to heavier processing steps to be included while not resenting its speed.
  - Our tool has successfully enabled data harvesting in various scenarios, including crawling static and dynamic websites, and gathering different types of data. This versatility makes it a valuable asset for both experts and non-experts in the field of data scraping.
  - We have established an API that facilitates the implementation of extensions for personal use cases. This API prioritizes ease of use and ensures the safety of inputs and usage. By creating separate files as a

safe bridge to the more performance oriented core files to handle common user mistakes, we have streamlined the development process and made it more accessible to a wider range of users (see Figure 3.1).

  - We have build an architecture around the tool to allow changes to the tool for the most expert users. This includes having less performant plug & play Plugins, or optimized Extensions, that enables changes to be included and turned on and off based on the needs of the task.
  - We reached the optional goal of enabling Cython compilation, to compile the code into C++, which is in many cases faster than Python.

- The integration of a built-in browser has provided non-expert users with easier access to the tool's capabilities, enhancing its usability and enabling data gathering without the need for coding skills. However, based on the feedback received from the questionnaire administered to non-expert test users, there is still room for improvement in terms of user accessibility and interface design (see Figure 4.9)
- We have also addressed security concerns by implementing measures such as Docker to ensure the integrity of the automated process, and the automatic erasure of browser data with each run. This way, we ease installation and ensure that any malicious encounter is contained within a secure area.

As an added bonus, we have identified through evaluations areas for improvement. These findings will guide us in refining the tool and enhancing its capabilities to meet the evolving needs of this kind of tool.

Moving forward, further enhancements and refinements can be made to expand its functionality and cater to evolving data harvesting requirements:

- Graphs-oriented architecture: One of our key objectives in the future is to transition towards a more graphs-oriented architecture. Graphs provide a powerful representation of the data that can be gathered through web scraping. They allow for more complex and interconnected relationships to be modeled and analyzed. By adopting a graphs-oriented approach, we can enhance the tool's capabilities in capturing and processing data from websites.
- Graph data structures as HDF5[42] files: As part of our effort to leverage graph-based representations, we would like to utilize HDF5 (Hierarchical Data Format) files to store and manipulate graph data structures. HDF5 files offer efficient storage and retrieval capabilities, enabling us to handle large-scale graphs effectively. This will facilitate tasks such as graph traversal, querying, and analysis. Furthermore, the adoption of the GraphML[27] format will enhance interoperability with other graph processing tools and libraries.
- Graph data results into SQL: Explore the possibility of processing, optimizing and converting the generated HDF5 graph files into a SQL-based format. By transforming the data into SQL, you can take advantage of optimized and efficient querying mechanisms offered by modern database sys-

tems. This can significantly improve data access speed, making it faster and easier to extract specific information from the collected data.

- Graphics card optimization: Investigate opportunities to leverage graphics card (GPU) usage as an optimization strategy. Streamline GPU utilization to prevent conflicts when multiple processes attempt to access it simultaneously. While initial tests may suggest that using the GPU could slow down the tool, explore potential optimizations that can harness the power of parallel processing offered by GPUs. This may involve optimizing data transfer between CPU and GPU, parallelizing computationally intensive tasks, or exploring GPU-accelerated libraries and frameworks that align with the tool's requirements.

- Integration of machine learning models: Machine learning techniques offer promising opportunities to enhance the capabilities of our data scraping tool. Specifically, we plan to incorporate vision-based models to identify and select clickable objects on web pages, improving the efficiency and accuracy of data extraction. Additionally, reinforcement learning (RL) algorithms can be utilized to navigate websites dynamically, adapting to changes in page layouts and structures. By leveraging machine learning models, we can automate and optimize various aspects of the scraping process (see ALGORITHM 1).

- Enhance the scheduling and resource management capabilities of your data scraping tool. Develop intelligent algorithms and strategies to optimize and limit the allocation of resources, such as bandwidth, CPU, and memory. Implement techniques like adaptive crawling, rate limiting, and distributed crawling to ensure efficient and responsible data gathering. Also, consider natively implementing schedulers to process full or partial requests at different times at specific frequencies.

We believe that applying this changes, as well as maturing the software and ensuring reliability with real use cases we can reach a tool that can very well become a industry-standard option that is able to be used and improve many people's lives.
With the GDGTool, we believe that we have reached a solution that can fit most cases, while keeping performance and reaching most users despite their expertise.

Bibliography

[1] *Web scraping*. URL: https://en.wikipedia.org/wiki/Web_scraping. (accessed: 31.03.2023).

[2] *Web crawler*. URL: https://en.wikipedia.org/wiki/Web_crawler. (accessed: 31.03.2023).

[3] *Data collection*. URL: https://en.wikipedia.org/wiki/Data_collection. (accessed: 31.03.2023).

[4] *Googlebot*. URL: https://en.wikipedia.org/wiki/Googlebot. (accessed: 31.03.2023).

[5] *Bingbot*. URL: https://en.wikipedia.org/wiki/Bingbot. (accessed: 31.03.2023).

[6] *Natural language processing*. URL: https://en.wikipedia.org/wiki/Natural_language_processing. (accessed: 31.03.2023).

[7] Stephanie Lunn, Jia Zhu, and Monique Ross. "Utilizing Web Scraping and Natural Language Processing to Better Inform Pedagogical Practice". In: Oct. 2020. DOI: 10.1109/FIE44824.2020.9274270.

[8] *Word embedding*. URL: https://en.wikipedia.org/wiki/Word_embedding. (accessed: 31.03.2023).

[9] *Lexical resource*. URL: https://en.wikipedia.org/wiki/Lexical_resource#Lexical_database. (accessed: 31.03.2023).

[10] George A. Miller. "WordNet: A Lexical Database for English". In: *Commun. ACM* 38.11 (Nov. 1995), pp. 39–41. ISSN: 0001-0782. DOI: 10.1145/219717.219748. URL: https://doi.org/10.1145/219717.219748.

[11] Elsa A. Olivetti et al. "Data-driven materials research enabled by natural language processing and information extraction". In: *Applied Physics Reviews* 7.4 (Dec. 2020). 041317. ISSN: 1931-9401. DOI: 10.1063/5.0021106. eprint: https://pubs.aip.org/aip/apr/article-pdf/doi/10.1063/5.0021106/13895961/041317_1_online.pdf. URL: https://doi.org/10.1063/5.0021106.

[12] Vidhi Singrodia, Anirban Mitra, and Subrata Paul. "A Review on Web Scrapping and its Applications". In: *2019 International Conference on Computer Communication and Informatics (ICCCI)*. 2019, pp. 1–6. DOI: 10.1109/ICCCI.2019.8821809.

[13] H.L. Shashirekha and S. Murali. “Ontology Based Structured Representation for Domain Specific Unstructured Documents”. In: *International Conference on Computational Intelligence and Multimedia Applications (ICCIMA 2007)*. Vol. 1. 2007, pp. 50–54. DOI: 10.1109/ICCIMA.2007.255.

[14] Vlad Krotov and Leiser Silva. “Legality and Ethics of Web Scraping”. In: Sept. 2018.

[15] DataDome. *Web Scraping Protection: How to Prevent Scraping & Crawler Bots*. DOI: 10.1109/ICCIMA.2007.255. URL: https://datadome.co/learning-center/scraper-crawler-bots-how-to-protect-your-website-against-intensive-scraping/. (accessed: 31.03.2023).

[16] *Writing for Success*. University of Minnesota Libraries, 2015. ISBN: 978-1-946135-28-5. DOI: 10.24926/8668.2801. URL: https://open.lib.umn.edu/writingforsuccess/chapter/11-4-strategies-for-gathering-reliable-information/. (accessed: 31.03.2023).

[17] *Cython*. URL: https://github.com/cython/cython.

[18] General Assembly. “Sustainable development goals”. In: *SDGs Transform Our World* 2030 (2015), pp. 6–28.

[19] Tom B. Brown et al. *Language Models are Few-Shot Learners*. 2020. arXiv: 2005.14165 [cs.CL].

[20] Hugo Touvron et al. *LLaMA: Open and Efficient Foundation Language Models*. 2023. arXiv: 2302.13971 [cs.CL].

[21] S. Sarawagi. *Information Extraction*. Foundations and trends in databases. Now Publishers, 2008. ISBN: 9781601981882. URL: https://books.google.es/books?id=AqHpDoYPjLQC.

[22] Plamen Milev. “Conceptual Approach for Development of Web Scraping Application for Tracking Information”. In: *Economic Alternatives* 3 (2017), pp. 475–485. URL: https://www.unwe.bg/uploads/Alternatives/10_Alt_english_br_3_2017.pdf.

[23] Shivam Agarwal. “Data Mining: Data Mining Concepts and Techniques”. In: *2013 International Conference on Machine Intelligence and Research Advancement*. 2013, pp. 203–207. DOI: 10.1109/ICMIRA.2013.45.

[24] Hussain Ahmad Madni, Zahid Anwar, and Munam Ali Shah. “Data mining techniques and applications — A decade review”. In: *2017 23rd International Conference on Automation and Computing (ICAC)*. 2017, pp. 1–7. DOI: 10.23919/IConAC.2017.8082090.

[25] Dirk Merkel. “Docker: lightweight linux containers for consistent development and deployment”. In: *Linux journal* 2014.239 (2014), p. 2.

[26] van Deursen A. Mesbah A. and Lenselink. “Crawling AJAX-Based Web Applications through Dynamic Analysis of User Interface State Changes”. In: 2012. DOI: 10.1145/2109205.2109208. URL: https://people.ece.ubc.ca/amesbah/resources/papers/tweb-final.pdf.

[27] Zhiqiang Zhong, Cheng-Te Li, and Jun Pang. “Reinforcement learning enhanced heterogeneous graph neural network”. In: *arXiv preprint arXiv:2010.13735* (2020).

[28] Shantian Yang et al. “IHG-MA: Inductive heterogeneous graph multi-agent reinforcement learning for multi-intersection traffic signal control”. In: *Neural Networks* 139 (2021), pp. 265–277. ISSN: 0893-6080. DOI: https://doi.org/10.1016/j.neunet.2021.03.015. URL: https://www.sciencedirect.com/science/article/pii/S0893608021000952.

[29] *Playwright for Python*. URL: https://github.com/microsoft/playwright-python.

[30] Steven Bird, Ewan Klein, and Edward Loper. *Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit*. O’Reilly Media, Inc., 2009.

[31] Matthew Honnibal et al. “spaCy: Industrial-strength Natural Language Processing in Python”. In: (2020). DOI: 10.5281/zenodo.1212303.

[32] Adam Paszke et al. “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach et al. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[33] Martín Abadi et al. *TensorFlow, Large-scale machine learning on heterogeneous systems*. Nov. 2015. DOI: 10.5281/zenodo.4724125.

[34] F. Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[35] *Selenium*. URL: https://github.com/SeleniumHQ/selenium.

[36] *Scrapy*. URL: https://github.com/scrapy/scrapy.

[37] R.P. Hudson. *coreferee: coreference resolution for multiple languages*. URL: https://github.com/msg-systems/coreferee.

[38] Vincent Ng and Claire Cardie. “Improving machine learning approaches to coreference resolution”. In: *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*. 2002, pp. 104–111.

[39] *NeuralCoref 4.0: Coreference Resolution in spaCy with Neural Networks*. URL: https://github.com/huggingface/neuralcoref.

[40] Rada Mihalcea and Paul Tarau. “TextRank: Bringing Order into Text”. In: *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing*. Barcelona, Spain: Association for Computational Linguistics, July 2004, pp. 404–411. URL: https://aclanthology.org/W04-3252.

[41] Paco Nathan. *PyTextRank, a Python implementation of TextRank for phrase extraction and summarization of text documents*. 2016. DOI: 10.5281/zenodo.4637885. URL: https://github.com/DerwenAI/pytextrank.

[42] The HDF Group. *Hierarchical Data Format, version 5*. https://www.hdfgroup.org/HDF5/. 1997.

APPENDIX A
Sustainable Development Goals

On 25 September 2015, world leaders adopted a set of global goals to eradicate poverty, protect the planet and ensure prosperity for all as part of a new sustainable development agenda. Each goal has specific targets to be achieved over the next 15 years.

Figure A.1: United Nations' 17 Sustainable Development Goals chart.

The Sustainable Development Goals decided to follow are the following:

1. No Poverty
2. Zero Hunger
3. Good Health and Well-Being
4. Quality Education
5. Gender Equality
6. Clean Water and Sanitation
7. Affordable and Clean Energy
8. Decent Work and Economic Growth
9. Industry, Innovation, and Infrastructure
10. Reduced Inequalities
11. Sustainable Cities and Communities

12. Responsible Consumption and Production
13. Climate Action
14. Life Below Water
15. Life on Land
16. Peace, Justice and Strong Institutions
17. Partnerships

A.1 Our work's alignment

With these sustainable goals in mind, we can create a table to show how our work aligns with this sustainable goals towards the global prosperity.

Sustainable Development Goals	**High**	**Medium**	**Low**	**Not Applicable**
SDG 1. **No Poverty**				●
SDG 2. **Zero Hunger**				●
SDG 3. **Good Health and Well-Being**				●
SDG 4. **Quality Education**	●			
SDG 5. **Gender Equality**				●
SDG 6. **Clean Water and Sanitation**				●
SDG 7. **Affordable and Clean Energy**				●
SDG 8. **Decent Work and Economic Growth**	●			
SDG 9. **Industry, Innovation, and Infrastructure**		●		
SDG 10. **Reduced Inequalities**	●			
SDG 11. **Sustainable Cities and Communities**				●
SDG 12. **Responsible Consumption and Production**				●
SDG 13. **Climate Action**				●
SDG 14. **Life Below Water**				●
SDG 15. **Life on Land**				●
SDG 16. **Peace, Justice and Strong Institutions**				●
SDG 17. **Partnerships**				●

Table A.1: This work alignment with the UN's Sustainable Development Goals

As seen in Table A.1, we believe our work best aligns with mainly four of the development goals:

4. **Quality Education**: Our tool has the potential to promote quality education by providing users with easy access to information and data. We believe that by offering a tool that enables individuals, regardless of their resources, to pursue learning and broaden their knowledge horizons, we can contribute to a more equitable and inclusive education system.
   For instance, imagine a student who wants to delve into a particular subject but lacks the necessary resources. With our tool, they can effortlessly explore a wealth of information, educational materials, and data that can enhance their understanding. We believe that by leveling the playing field and facilitating equal access to knowledge, we can promote quality education for all.

8. **Decent Work and Economic Growth**: Our tool seeks to enhance productivity and efficiency, allowing researchers and professionals to focus on high-value activities. Through automation and data insights, we believe that our tool can lead to greater job satisfaction, improved productivity, and reduced research time, thereby fostering economic growth and contributing to decent work opportunities.
   Consider a scenario where a research team can automate data extraction and analysis tasks using our tool. This enables them to dedicate more time to critical decision-making, innovation, and advancing their field. We believe that by optimizing workflows and minimizing tedious tasks, our tool can help professionals to thrive and contribute to their respective industries.

9. **Industry, Innovation and Infrastructure**: To a lesser extent, we believe that our work aligns with this goal because it is able to enhance the productivity and efficiency of jobs that require research, which will create a conducive environment for innovation to thrive. We firmly believe that research plays a fundamental role in driving innovation, and many jobs in this domain are closely tied to innovative advancements. Our tool, by automating repetitive processes and providing valuable data insights, supports researchers and professionals in their quest for new ideas and groundbreaking solutions.
   For example, by utilizing our tool's capabilities, a team of scientists can expedite their research process, gather relevant information more efficiently, and make informed decisions, leading to breakthrough innovations that contribute to the advancement of industries and overall economic growth.

10. **Reduced Inequalities**: We also believe that our tool can contribute to reducing inequalities by providing a platform that promotes inclusivity and equitable access to resources. By breaking down information silos and promoting knowledge-sharing among diverse communities, we can narrow the knowledge gap as mentioned in point 4. Quality education, help individuals from different backgrounds, and even fight malicious or careless miss-information that can promote hate towards minorities.
    Imagine a situation where our tool allows individuals to gather and analyze data from various sources, regardless of their geographical location or

social status. This democratization of information may help them to stay informed and make well-informed decisions. We believe that with this, we can address disparities, challenge misinformation, and promote a more equitable society.

APPENDIX B
System configuration

The system configuration phase of the GDGTool involves setting up the necessary environment to run the tool smoothly.
Thanks to its containerization using Docker, the system requirements are minimal and the setup process is simplified.

To run our tool, the user will only need Docker installed and 15 GB of disk space. Docker provides a lightweight and portable environment that encapsulates all the dependencies and configurations needed for the GDGTool to run effectively. This means that users only need to have Docker installed on their operating system of choice, which can be achieved by executing a single command.

B.1 Initialization phase

The initialization phase of the GDGTool streamlines the setup process from the moment Docker is installed until the visual browser becomes accessible. Once Docker is successfully installed, the GDGTool utilizes pre-configured Docker images and containers to automatically set up the required components and dependencies. These containers include all the necessary software, libraries, and configurations needed for the tool to function effectively.

For Linux users, we provide two bash scripts that run the tool. One only runs the browser GUI, but for more technical tasks and to run custom scripts we provide a second script that launches the tool but also connects to the console inside the container.

B.2 Browser GUI usage

Once the GDGTool's browser GUI is open, users can navigate websites like a regular user while also accessing the tool's macros. These macros automate common data extraction tasks, such as extracting keywords from the current page. By combining manual navigation with automated actions, users can efficiently gather data and perform advanced processing operations without complex scripting.

This user-friendly approach enhances productivity and accessibility, making data harvesting more accessible to users of varying technical expertise.

APPENDIX C
Example use case

In this appendix, we present a comprehensive use case that has served as a guiding scenario throughout the development of our data scraping tool. This use case exemplifies the practical application and benefits of our tool in a specific domain, providing a real-world context for understanding its capabilities and potential.

Throughout the development process, we have focused on addressing the needs and challenges of this particular use case, tailoring our tool's features and functionalities to meet its requirements effectively, while keeping a broader view to also help generalize the tool to other use cases.

C.1 Definition of the use case

The use case we have selected for our data scraping tool revolves around a scenario where an expert user, working in a team of non-expert users, seeks to automate data processing in their regular internet searches. This use case specifically caters to situations where a team or organization relies heavily on gathering and analyzing information from various online sources to support their decision-making processes.

In this use case, the expert user possesses in-depth knowledge and expertise in data scraping techniques, web technologies, and data processing methodologies. Their role is to lead the team and provide guidance on implementing efficient and effective data gathering and processing strategies. The expert user understands the intricacies involved in accessing and extracting relevant data from different websites, and they possess the skills to navigate the complexities of web scraping tools and technologies.

On the other hand, the non-expert users in the team are individuals who may lack the technical know-how or programming skills to implement data scraping techniques independently. However, they recognize the importance of incorporating data-driven insights into their work and understand the potential value of automating data processing in their regular internet searches. Their primary goal is to enhance their productivity and decision-making capabilities by harnessing the power of data through the use of a user-friendly and intuitive data scraping tool.

The collaboration between the expert user and the non-expert users creates a symbiotic relationship, where the expert user's knowledge and expertise are leveraged to develop a data scraping tool that caters to the specific needs and requirements of the non-expert users. The expert user serves as a facilitator, guiding the development process and ensuring that the tool is user-friendly, accessible, and capable of delivering accurate and reliable results.

C.2 Requirements and issues from the non-expert users

The non-expert users in our use case often face a common challenge: how to interact with the tool in a way that aligns with their skill set and avoids overwhelming them with complex technical details. We acknowledge that these users may not be familiar with programming concepts or the underlying structure of the tool, making it crucial to design an interface that is intuitive and user-friendly.

To address this issue, we recognize the need to simplify the menus and controls as much as possible. Instead of presenting the users with a plethora of technical options, we can provide a streamlined interface that focuses on the essential functionalities they require. For instance, using buttons that trigger pre-defined macros or actions can be an effective approach. These buttons can be labeled with descriptive and user-friendly terms, such as "Extract Data" or "Perform Analysis". By abstracting the technical complexities behind these actions, non-expert users can interact with the tool effortlessly, without the need to understand the underlying programming structure.

Additionally, providing contextual guidance within the tool can be immensely helpful for non-expert users. A partial result of each executed macro can be show next to the buttons in the interface. This way, users can gain a better understanding of how to utilize each feature without feeling overwhelmed or confused.

C.3 Requirements and issues from the expert user

While the non-expert users provide input regarding the desired functionalities, it is the expert user who will be responsible for implementing and managing these functionalities within the tool, and the one that will encounter the issues of the implemented functionalities.

One of the key requirements emphasized by the expert user is the performance of the tool. As the non-expert users may require processing large amounts of data, it is crucial that the tool is designed to handle such volumes efficiently. The expert user recognizes the importance of optimizing data scraping and processing operations to ensure timely and seamless execution, enabling the non-expert users to accomplish their tasks without unnecessary delays.

Non-expert users will require the tool to also process the data generated and format it to the format they require. The expert has the task to allow the extraction of multiples types of data, and perform a processing step on the data generated, per website crawled. At the end, the tool will be required to do an additional processing step where the data will be formatted, and the expert must take into acount that the format may change.

Another essential aspect for the expert user is the flexibility to make fast changes to the tool's functionality. While the non-expert users may provide initial requirements, it is not uncommon for additional features or modifications to be requested during the tool's usage. Therefore, the expert user needs a development environment that supports quick iterations and easy integration of new functionalities without requiring extensive relearning or rewriting of the entire tool's codebase. This flexibility enables the expert user to adapt the tool to evolving needs and incorporate user feedback efficiently.

Additionally, the non-expert users raise the importance of allowing users to enable or disable specific functionalities of the tool. Not all users may require or desire every feature provided by the tool. By incorporating a modular design and allowing users to customize the tool's functionalities, the expert user ensures that the tool remains focused and tailored to the specific needs of individual users. This customization capability enhances user experience and avoids unnecessary complexity for users who may only require a subset of the tool's capabilities.

Furthermore, the expert user acknowledges the need for deploying the tool in different systems while ensuring security. The tool should be designed to run reliably on various operating systems and environments, accommodating the preferences and requirements of different users. It should also adhere to security best practices, protecting sensitive data and preventing unauthorized access. By considering these aspects, the expert user aims to ensure that the tool is versatile, adaptable, and provides a secure environment for users to perform their data gathering tasks.

C.4 Issue addressing

To achieve the requirements and address the issues faced by both expert and non-expert users in our real case, several decisions were made during the development of the tool.

- For non-expert users, one of the key requirements is simplicity in interacting with the tool, without compromising the availability of all necessary functionalities.
  We have acknowledged that non-expert users may feel overwhelmed by an interface that resembles the programming structure of the tool. To address this, we have focused on simplifying the user interface by incorporating intuitive buttons and controls that enable easy access to the tool's capabilities.

By streamlining the user interface, non-expert users can interact with the tool in a more user-friendly manner, reducing the learning curve and enabling them to utilize the tool without requiring extensive programming knowledge. This decision caters to the non-expert users' need for simplicity and accessibility, empowering them to automate data processing tasks in their regular internet searches efficiently.

- For the expert user, several requirements were identified based on their specific needs. Firstly, the tool must exhibit high performance to handle the large amounts of data that the expert user may encounter in their tasks.
  By developing a fast, modular, and scalable framework, we have ensured that the tool can efficiently extract data from various websites and formats, while optimizing speed and execution. This performance-oriented approach allows expert users to process and analyze data swiftly, enabling them to work with extensive tasks effectively. As an added benefit, we performed Cython compilation, that makes the tool faster in some cases, which further helps to tackle the performance requirement.

- Another important requirement for the expert user is the ability to make fast changes to the functionality of the tool. We recognized that the expert user often require flexibility and adaptability in their workflows, without the need for extensive relearning of the tool. To address this, we have designed the tool with a modular architecture that facilitates the addition of new functionalities or the modification of existing ones. The expert user can easily extend the tool's capabilities, integrate their optimized functionalities, and customize its behavior to align with their specific requirements via the Extensions for critical code and Plugins, for faster developed tools.

- Additionally, we have incorporated the option for the expert user to enable or disable specific functionalities of the tool before starting a new task. This capability allows non-expert users to fine-tune the tool's behavior, selecting only the functionalities that are relevant to their current task. By enabling this level of customization in both Plugins and Extensions, the tool becomes a versatile asset, offering control over its features and ensuring that they can focus on the specific aspects of data gathering that are essential to their objectives.

- In terms of deployment and security, the expert user often need the ability to run the tool on different systems while ensuring the tool's security. To fulfill this requirement, we have implemented measures such as Docker, which allows for secure and isolated execution environments. By using Docker, the expert user can deploy the tool across various systems with confidence, knowing that the automated processes are contained within secure containers, minimizing the risk of unauthorized access or interference.

Thanks to

, I would like to take this opportunity to express my gratitude to Encarna Segarra Soriano and Lluís Felip Hurtado for their guidance and support throughout the writing of this work.

I would also like to extend my gratitude to all the wonderful teachers at Universitat Politècnica de València. Each one of them has shaped me as the person I am, instilling in me to search and learn more about our field and providing me with a really valuable knowledge and skills. I am truly fortunate to have had the privilege of being their student.

Thank you all for your tireless efforts and for making all these years so special.