IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2024.0429000

Web scraping by end users

ALEX TACURI[1], SERGIO FIRMENICH[2], ALEJANDRO FERNÁNDEZ[1], FLORENCIA RIVA[1], MATÍAS URBIETA[1], AND GUSTAVO ROSSI [1]

[1]LIFIA, Facultad de Informática, UNLP , La Plata, Buenos Aires, 1900, Argentina(e-mail: alex.tacuriu@info.unlp.edu.ar)
[2]Universidad Loyola Andalucía, España

Corresponding author: Alex Tacuri (e-mail: alex.tacuriu@info.unlp.edu.ar).

This work was supported by LIFIA

ABSTRACT Scraping is a topic studied from various perspectives, encompassing automatic and AI-based approaches, and a wide range of programming libraries that expedite development. As the volume of available web content increases, it becomes increasingly challenging to anticipate end-user requirements regarding what, how, and when to extract data from the web. This challenge is compounded when integrating data from multiple websites, particularly when websites' search engines dynamically retrieve unavailable data via permanent links. Complex scraping processes, such as these are difficult to develop using general-purpose programming languages and are challenging to automate with AI-based approaches. Controllability is a crucial aspect of scraping, that is, how end users can make decisions during the scraper specification process, understand information sources, and how the data are ultimately extracted, compiled, and formatted for output. In response, our study presents an innovative end-user approach for specifying scrapers that focuses on seamlessly integrating data from multiple sources. Through this approach and its supporting toolset, we aim to provide users with greater control and transparency over the extraction, integration, and formatting of data, thereby addressing the key concerns in web scraping. The approach and toolset were evaluated and they yielded promising results.

INDEX TERMS Web mining, End-user computing, Human computer interaction, User centered design, Web scraping, Data integration, Scraper specification, Web data extraction.

I. INTRODUCTION

WEB scraping is a well-established technique for obtaining structured information from the web. It is formally defined as "a technique to extract data from the World Wide Web and save it to a file system or database for later retrieval or analysis" [1]. Other authors define a Web scraper as "a software that simulates human browsing on the web to collect detailed information from different websites." [2], while it is also defined as "the procedure of automatic extraction of data from websites using software" [3]. These definitions, as well as others we can find in the literature, are based on the following key aspects:

- Automation of interaction: Web scraping requires a strategy for traversing web pages, usually involving some kind of automation where a software artifact visits pages, stimulating the system to generate web page content.
- Data Extraction: Web scraping involves extracting data from web pages once a web page is targeted. It gathers information from the page, including text, images, tables, and links. The data extraction process typically involves parsing the HTML structure of a webpage and identifying specific elements or patterns to be extracted.
- Output building: The final step of the scraping process consists of generating a type of output for structured data.

Web scrapers are software artifacts that can be created using various approaches. Web scrapers may be created by programming them using general-purpose programming languages, such as Python or JavaScript, commonly using a library (i.e., Selenium[1] or Cheerio[2]), which makes it easier for some parts of this development. However, end-user-driven approaches allow users without programming skills to define scrapers, usually supported by visual programming or programming using demonstration tools. Mid-way between tools for developers and tools for users without programming skills, there are approaches based on domain-specific languages (DSL) that provide a way to specify scrapers without advanced programming skills but still program them using a programming language specifically designed for these aims, which ultimately reduces requires programming skills.

[1]https://www.selenium.dev/
[2]https://cheerio.js.org/

Although AI-based approaches offer a good way to reduce the overall workload for defining scrapings, they are not easy to use by end users if they want to ''tune'' the scraping process.

The implications of using one or more of these strategies are diverse, and are discussed in depth in the following section. At this point, and for a good understanding of our motivation and the presentation of our approach, it is important to note that end-user-driven approaches make it quick and easy to define web scrapers, but at the same time reduce expressiveness (i.e., there is a trade-off between the simplicity of these approaches and the overall power of the resulting scraper). However, AI-based approaches may affect user controllability.

Although Web scraping is an old technique, new approaches are constantly emerging, and it remains a relevant technique because the Web has become the most important data source. Although the Web of APIs has been growing over the past few decades, it is still true that not every web application provides an API to obtain the information they provide [4]. In the new era of Web browsers that integrate IA, providing new approaches for powerful scraping driven by end users in the browser is an interesting way to combine automatic information processing with data designed by users.

Therefore, Web scraping remains relevant, while some authors analyze its use in different fields such as Business Intelligence, AI, Data Science, Big Data, Cloud Computing, and Cyber Security [3], Web scraping is also an important aspect in end-user oriented engineering approaches such as mashups [5] and Web Augmentation [6]. This implies that Web scraping is a technique used in several scenarios and is performed by users with different levels of technical skills.

In this study, we focus on end-user programming of web scrapers considering two main dimensions that, to the best of our knowledge, have not been considered in the existing approaches.

- Many Web applications provide information exclusively through their search engines, making it challenging for end users to access and utilize data efficiently. To address this limitation and unlock the wealth of knowledge hidden behind these search engines, a web scraping approach should be able to emulate the user interaction required to perform a search using the search engines provided by web applications. In addition, such an emulation should consider other aspects of the search engine UI, such as pagination and filters. These characteristics also differ from those of typical web crawlers, which depend on permalinks and usually do not take advantage of the dynamic content.
- The information on the web is spread in different web applications, which implies that obtaining the desired data could require the integration of information pieces from multiple sources, that is, the information objects produced as outputs are composed of properties whose values are spread among different web applications.

To illustrate these ideas with a simple example, imagine a scraper of the scientific literature domain. This scraped the Springer website to obtain articles on a specific topic. At first glance, it is clear that an automated process for emulating a search in Springer is required. Once the results page was retrieved, two things occurred: First, the pagination mechanism should be activated in order to obtain more studies. Second, for each resulting paper in the Springer search, it may be necessary to automate navigation to the paper's webpage to obtain further information that is not available on the search results page, such as the complete abstract of the paper. Thus far, the scraping process has produced an output containing information objects (papers), with properties (*title, year, abstract, authors*) and their values extracted from Springer. However, more information related to each study can be obtained from other scientific repositories. For example, the scraping process may require attaching a property *citations* obtained from Google Scholar to each paper obtained from Springer. Then, for each paper, it would be necessary to automate the search in Google Scholar using the value of the property *title* to check that a Google Scholar result has the same title, and if that is the case, extract the number of citations. Beyond this, it would also be interesting to add *papers of the same author's property* to each of the papers extracted from Springer, and someone could consider that the best place to obtain papers for a specific author is the DBLP website. In this case, searching for an author in the DBLP and extracting other papers from the same author would be necessary.

In this scenario, as the reader will notice, scraping would require emulating the user interaction with at least three websites and, more importantly, coordinating the integration process among the results of the three search engines. Finally, it is necessary to specify the final composed information object.

Certainly, a skilled developer can create this scraper using JavaScript, Python, and related technologies. However, existing end-user programming approaches for web scraping do not provide sufficient expressiveness to create a scraper. We believe that the scraping of information from the web is very important. It is necessary to create technological enablers that allow end users without programming or technical skills to define how to scrape the web. Although very interesting works exist in this respect, we consider that because the Web is clearly a heterogeneous corpus of information spread across multiple sources, new approaches are necessary.

In this paper, we present the Programmable Retrieval Interaction SIMulator (PRISM) approach, which is an evolution of our previous work called ANDES.

Our contribution in this study is based on an approach for defining web scrapers that allows end users without programming skills to define this type of artifact, considering the above-mentioned aspects. Simply knowing how to navigate the web and having a basic understanding of browser extensions would suffice for the users. In addition, our approach is structured around two main components: (1) the automation

of data extraction from websites, referred to as the PRISM spec; and (2) the composition of these specifications, called PRISM composition specs, which define the integration and data flow from one specification's output into another's input. For example, let us consider a user searching for a specific book on a review website ((such as Goodreads) who simultaneously wants to check its availability on their city library's web portal and its price/purchase options on Amazon.

Traditionally, users would have to perform search and information extraction manually and independently across all three websites. With our solution, the user can simply create the necessary PRISM specs for each site and integrate them into a single workflow. This approach drastically reduces the search time and effort, and provides a consolidated result.

To validate our approach and tools, we conducted a usability evaluation with more than 40 users without programming skills.

In Section II, we present both background and related work, reporting a brief literature review on the topic, considering an introduction to the topic and two more specific subsections. We present our approach in Section III, and the toolset supporting this approach in Sections IV and V. In Section VI, we report on a developer's experience in developing an example of a scraper. A comprehensive evaluation of the proposed approach and its tools, along with formal metrics derived from our measurements, is presented in Section VII. Finally, in Section VIII, we present our conclusions and discuss future research.

II. BACKGROUND AND RELATED WORK

Web scraping has a long history. Even before the term ''web scraping'' was coined, similar concepts were referred to by various names, for instance, ''web wrappers'' [7]. Despite this age, web scraping remains significant even within the ''API world'' [8]. It is still an active research field, and also it offers possibilities in the industry, there are now several IT product companies that rely heavily on these techniques for their value proposition, such as Octoparse and Import.io.

As it was said before, nowadays, very mature technologies for performing web scraping using general-purpose languages [9] are available, which offer great flexibility but require advanced programming skills.

The essential goal of our approach is to enable users to obtain structured information from the Web by automating the search functionalities of websites search functionalities. Our approach has three crucial stages: searching for information in websites, composing or integrating results from several websites, and determining the output format.

Currently, studies focusing on the first stage involve scraping information from a web page. In our case, this was performed semi-automatically by initially requesting the user to select the attributes to be scraped. Zhou et al. [28] created the Simpon model, which focuses on *attribute extraction* similar to our approach; the difference lies in how the data are obtained from the DOM. In our case, we requested the user to specify the attribute to retrieve, whereas Simpon employed an artificial intelligence (AI) method based on a multi-layer perceptron (MLP) for multi-class classification to obtain DOM attributes. A similar AI approach to attribute extraction from web pages is Schema-Driven Information Extraction [21], which utilizes a Large Language Model (LLM) to extract specific attributes from tables through machine learning. Another AI approach for 'attribute extraction' is DOM-LM [20], which utilizes a self-supervised pre-training algorithm to learn the textual semantics of a node locally and how it fits into its context within the DOM tree. Artificial intelligence offers a powerful mechanism for recognizing semantics in web page content. However, as we mentioned before, it is not the unique step required in the type of scraper whose creation we want to support. It is important to note the integration of properties from multiple sources and the controllability that end users may require in this process. In this sense, the free choice of DOM properties a user wants to obtain is intrinsic to a person and the scenario of use, furthermore cannot be achieved with AI algorithms because they are not aware of the likes and preferences of the end user. Although AI has undoubtedly transformed many aspects of web scraping, end-user tools continue to play a valuable role in democratizing data access and empowering users with diverse skill sets and requirements.

Some of the advantages of our approach over AI-based solutions stem from its low computational cost, as our tool runs directly in the browser, a benefit also noted by Barba et al. [35] when discussing traditional methods. Another strength of our approach is the level of control, transparency, and traceability it provides, as it is rule-based, allowing users to easily understand how each piece of data is obtained and ensuring greater oversight of the extraction process. Additionally, maintaining control over what we extract ensures that we do not violate ethical or legal principles. In contrast, when using AI-based approaches, compliance with these principles depends heavily on how the underlying algorithms are trained [37].

Currently, works such as that of Yannikos et al. [36] leverage artificial intelligence to address specific tasks, such as solving CAPTCHAs with neural networks, before proceeding to content extraction. This reinforces the idea that integrating AI with approaches, such as ours provides an effective solution for web information extraction.

A. END-USER PROGRAMMING FOR WEB SCRAPING

Web scraping libraries available for users with programming knowledge, such as Selenium [10], Scrapy [11], Beautiful Soup [12] and Puppeteer [14], are widely used. These libraries excel at retrieving and parsing web content. However, they can be complex for end-users without programming knowledge. These tools require the installation of a development environment, configuration of libraries, and handling of errors when executing the scripts. Our tool frees users from these tasks, enabling them to obtain web content without complications from programming tasks.

Kumar et al. [13] used the BeautifulSoup library within the

Jupyter Notebook environment, presenting stages similar to those through which our tool extracts data from the Web. The most common methods include web data inspection, parsing, and visualization; however, they still require programming skills.

In recent years, various efforts have been made to create tools for individuals with no programming knowledge who wish to extract content from the Web. Some tools have an approach similar to ours and focus on non-programmers, such as CoScripter [30], Vegemite [15], and Rousillon [16]. Many of these tools primarily focus on content retrieval.

Rousillon [16] focused on scraping the distributed hierarchical data. It generates two-level page abstraction loops, which are the features shared by our DOM data extraction tool. In contrast to Rousillon, our approach leverages the search capabilities of websites, that Rousillon does not offer. In addition, Rousillon does not allow integration of information objects from different sources.

ScrAPIr is an interesting tool for allowing end users to deal with information on the web [17] by consuming the integrated APIs of websites that provide this service. The advantage of this tool is that it provides an interface that non-programmer end users can access. However, this is limited to websites with integrated APIs. The potential of our approach is that it can be applied to all websites that offer a search engine to retrieve the domain objects.

Some commercial web platforms perform scraping, one of which is Octorpase [18], which shares features with our tool, such as being designed for non-programmer users. However, they do not consider the simultaneous integration of multiple search engines from different sites to gather information.

B. WEB EXTENSIONS FOR WEB SCRAPING

Web browser extensions deserve special attention because they are convenient ways to access and parse the web content. Currently, browser extensions such as Wildcard [19], web scraper [22], Data Miner [23], and Octoparse [24], perform web scraping.

Wildcard, presented by Litt et al. [19], is a web extension that enables the augmentation and manipulation of object properties obtained from the DOM. It was designed to be used by non-programmers through a technique called *spreadsheet-driven customization*. This feature distinguishes it from our approach, as we employ a lightweight interpreted language similar to JavaScript to manipulate the DOM objects.

Web Scraper [22] is a Chrome extension that allows users to extract data from web pages and save it in formats such as CSV or JSON. It provides tools to select and extract specific elements from a web page, similar to our approach. However, this differs because we can reuse the results from a website search engine to perform other queries, thereby enabling us to simultaneously build more elaborate information objects from multiple sites.

Data Miner [23] is another chrome extension that simplifies website data extraction. This allows users to create custom scrapers to extract information from complex web pages. This was based on visual programming by demonstration, which was also utilized in our tool to automate the extraction process. The difference with this tool is that in addition to scraping data, it enables data integration between websites while the scraping process is ongoing. This means that the information obtained simultaneously can serve as input data for another PRISM spec, allowing for the creation of more complex information models from different sources within the same time frame.

Another web scraping tool that offers both a desktop version and chrome extension is Octoparse [24]. It enables the extraction of structured data from various web sources and supports the programming of complex data extraction tasks. The main difference between this tool and ours is that it does not allow combining the results obtained from one site and injecting them into queries on other sites simultaneously while scraping.

ParseHub [25] is a browser extension and cloud-based data extraction tool. It allows users to extract data from dynamic websites, provides features to work with complex data, and creates custom extraction rules.

None of these tools generates object models before abstracting content, and they also do not allow for treating object properties or obtaining them from multiple sources.

C. COMPARISON WITH WEB SCRAPING TOOLS

Table 2 presents an analysis of the main commercial and research-oriented web scraping tools most relevant to the objectives of our approach, many of which feature visual programming capabilities. A detailed comparison between the proposed approach and existing commercial tools is provided below.

Required Technical Level: Similar to many of the tools presented in Table 2, our tools require only a low level of technical knowledge.

Internal Search Engines: Our tools leverage the internal search engines of the target websites through a PRISM spec, a feature shared only with the tools analyzed in Table 2.

Multi-source Integration/Pipelines: Only Apify, Bright Data, and our tools can fully utilize the data obtained from one website as input for another in a pipeline-like process

Authentication and Pagination: Our tools, like many of the tools presented in Table 2, support authentication and pagination.

Semantic Extraction: As can be observed in Table 2, most approaches do not support semantic extraction or partially support it. In our approach the semantic extraction is fully supported.

End User Control: : Similar to many of the tools in Table 2, our tool offers a very high level of end-user control.

Detail Page Navigation: Like many of the tools in Table 2, our approach offers multi-level extraction. This means you can navigate from a table of results (such as a search page) and then automatically proceed to the detailed page for each individual item to collect more in-depth information.

Polymorphic Combination: Very few tools have this property: the ability to automatically merge records from different websites into a single dataset. Most tools require programming to accomplish complex tasks. In contrast, our tool allows users to perform the same task without any programming knowledge.

Object Composition: This feature is the essence of our approach because no other tool can enrich a single data object with properties from different websites without requiring any programming knowledge.

Captcha Handling: Our tool does not have these features at the moment because the goal of this approach was to integrate data from various websites to obtain objects with properties from different sites, and for this to be possible for users without technical programming knowledge.Although CAPTCHA solving is not automated, the solution is for the user to address it manually. Under this condition, our tools perform correctly in content extraction, and future work may focus on automating the CAPTCHA resolution.

Browser Extension Support: This feature is the essence of our approach, as it allows users without programming knowledge to define web scrapers.

III. THE APPROACH IN A NUTSHELL

The main goal of our approach and supporting tool is to consider the three typical tasks of a scraper described in the Introduction section (UI automation, information extraction, and output generation), taking into consideration the integration of information obtained using multiple website search engines.

It is important to make it clear here that all the activities contemplated for the definition and use of the scraper are performed by end users in the web browser, considering that our main focus is to support users without programming skills in the definition and use of a web scraper. Nevertheless, in this section, we specifically focus on scraper definition, which involves the novelty of this study and its evaluation.

Figure 1 depicts the overall concept (the details of each step are presented in the following sections). First, we propose a way to specify what we call the ''PRISM spec'', that is, the specification of how to automate the UI interaction required to perform a search on a particular web page, and the specification of how information objects are composed using information presented on the search result page or the website specific to a resulting item. This is Step 1 in Figure 1, and must be repeated for every target website. As a result of this step, three PRISM specs, one per website, will be available. Each of these is capable of retrieving information objects under demand. During Step 2, users may compose different specifications, which means that they may specify how, for each result from one specification, others are executed to retrieve further information. Combining these is done through the properties of each specification, and we can establish the relationships between them, either one-to-one or one-to-many. Finally, in Step 3, the user sets how the output of the scraper would be composed.

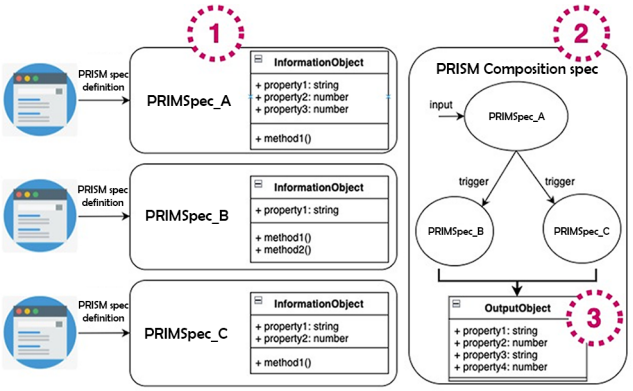

FIGURE 1. The approach in a nutshell.

We developed a suite of tools to support all these activities. On the one hand, we proposed in a previous work a PRISM spec definition tool (supporting Step 1) and was reused in this new approach to support the PRISM composition spec tool (supporting Steps 2 and 3). These tools are delivered together in a unique web extension. Our approach and tool also support simpler scraping scenarios where information objects are retrieved from different PRISM specs without any integration, that is, simply putting them together in the output.

A. UNDERLYING ARCHITECTURE

Figure 2 depicts the PRISM's architecture.

- Actor and system boundary
  - **End user** operates the system.
  - Everything runs inside the **PRISM** system (top box) with a dedicated **PRISM storage** (bottom box).
- Main tools (top layer)
  - **PRISM spec creation tool** – creates the foundational specifications (PRIM/PRISM specs).
  - **(PRISM) specs composition tool** – composes higher-level specs from existing ones; enables **include/reuse** of PRIM specs for modularity.
  - **PRISM composition specs interpreter** – executes/interprets the composition specs to perform the defined processes (e.g., scraping).
- Stored artifacts (bottom layer)
  - **PRISM specs** – base specs produced by the creation tool.
  - **PRISM composition specs** – composed specs that **include** PRISM specs.
  - **Scraping results** – outputs generated by running the interpreter.
- Usage and creation flow (dashed lines)
  - The **End user** uses all the three tools.
  - Each tool **uses** storage to read existing artifacts and **creates** new ones,
    - Creation tool → **PRISM specs**.

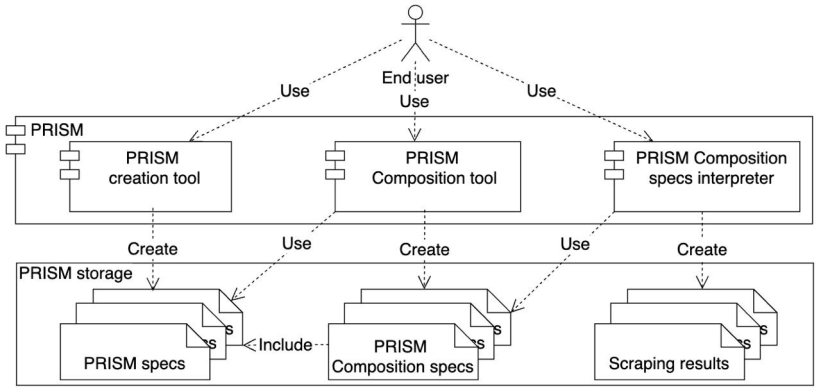

FIGURE 2. PRISM Architecture.

  - ∗ Composition tool → **PRISM composition specs** (including PRIM specs).
  - ∗ Interpreter → **Scraping results**.
- Key idea
  - -- The architecture cleanly separates **base specification authoring**, **composition/reuse**, and **execution**, all backed by a common repository to ensure **traceability and versioning** of specs and results.

IV. CREATING PRISM SPECS FOR SCRAPING

This section primarily focuses on the first dimension, which involves obtaining information from websites using a search engine. To achieve this, we enable users without programming knowledge to create a PRISM spec for a website by annotating the UI elements. We subsequently employ these PRISM specs to address the second aspect of our approach: the PRISM composition specs.

In this study we introduce the evolution of the tool presented in [26] to automate web searches on websites. We further developed it by, changing the scraping technique, generating a new method of searching within DOM elements, and changing the technology with which we request web servers and retrieve responses. Additionally, the previously used method and algorithm took several seconds to scrape a site, and the user did not know whether it was working on the scraping or not working correctly. We addressed a new method and algorithm, that reduces this time by 50% compared with the previous implementation of our PRISM spec tool. In addition, to reduce the time required to generate the PRISM spec and decrease the likelihood of errors occurring during the creation of the query service, we reduced the number of steps required to generate the PRISM spec. The main feature added to our previous tool is multilevel scraping, which essentially allows the creation of scraping cycles to enter the next level of a search, that is, for each search result, the next level of each result can be automatically scraped.

Now we are going to review the necessary steps to create a PRISM spec.

- First, to create a scraping-based PRISM spec, the user needs to access the website and perform a search using the site's search engine, as illustrated by an example on the Springer website. Once the search results are loaded, the next step is to open the web extension, as shown in Figure 3(Step 1).
- This tool provides a menu for creating PRISM specs and PRISM composition specs. In this scenario, we select ''Services'', as shown in Figure 3 (Step 2).
- The tool displays all defined PRISM specs and a button to create a new service, as shown in Figure 3 (Step 3). The user must click this button to create a new PRISM spec for the current web site.
- Next, the tool asks for a name that serves as an identifier for the PRISM spec, as shown in Figure 4 (Step 4).
- When clicking the 'Next' button, the tool searches for DOM elements corresponding to the input fields on the page and adds functionality to these elements, enabling the user to select one of them, Figure 4 (Step 5). In this way, our tool determines which DOM element allows the input of the search text. This will be useful for our tool to change the search text and execute new queries on the PRISM spec.
- Continuing with the construction of the PRISM spec, the tool performs a DOM search to find elements that contain the results presented by the search engine. This enables the user to select one of these elements and to input the number of occurrences. The user also assigs a semantic tag to the object displayed in the search results. In this case, the object is named *Articles* with 20 occurrences, and the corresponding semantic tag is assigned as an *Article* as shown in Figure 5 (Step 6).
- Finally, from a search container element, users can choose the property they wish to abstract from DOM. Once they have finished selecting all the properties, as depicted in Figure 6 (Step 7).
- The construction of the PRISM spec is concluded, as shown in Figure 6 (Step 8). It is worth mentioning that our tool can also abstract the properties that may be present on a subsequent page displayed upon selecting

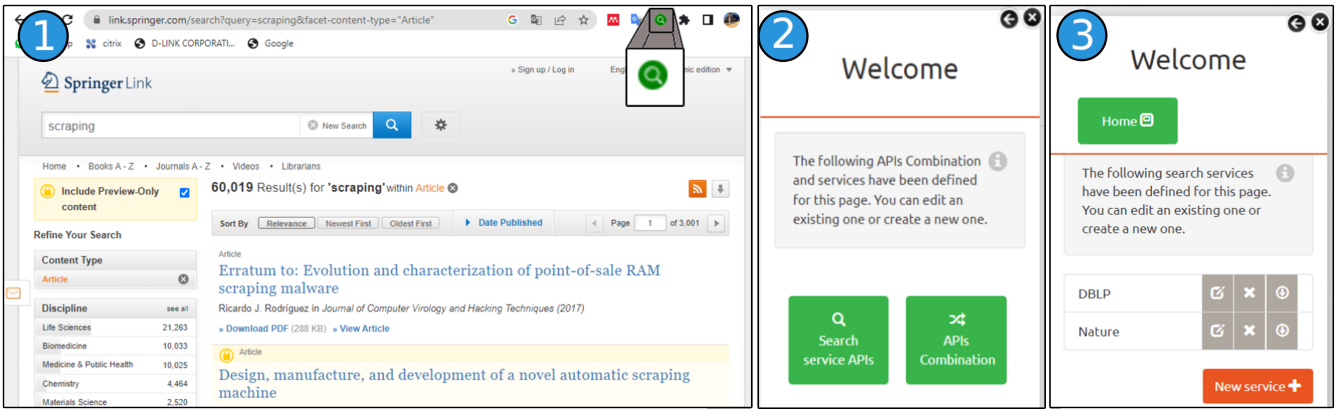

FIGURE 3. **Steps 1,2 and 3 for creating a PRISM spec.**

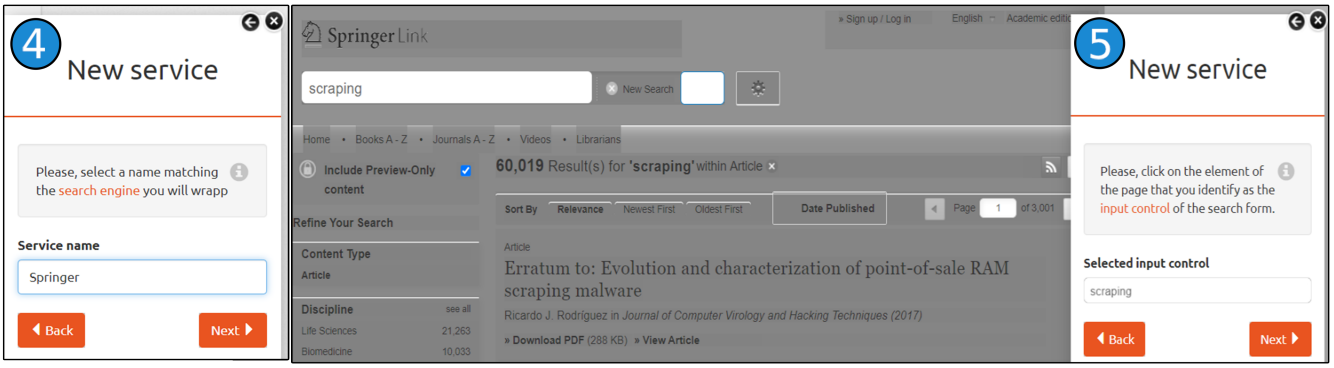

FIGURE 4. **Steps 4 and 5 for creating a PRISM spec.**

FIGURE 5. **Steps 6 for creating a PRISM spec.**

an element from the search results.

V. COMBINING PRISM SPECS FOR CREATING SCRAPING MODELS

To this point, whether a user possesses programming knowledge or not, they can create a PRISM spec for any website of interest that has a search engine. Now, we will take a step further by addressing the PRISM composition specs. The primary goal is to construct more robust and intricate structures of information originating from various sites without the need to write a single line of code.

The PRISM composition specs is carried out through the properties of objects, essentially linking one with another, similar to an entity-relationship model. This composition can be edited and configured according to the needs of the end-user. All PRISM specs that the user intends to combine must be constructed previously to initiate the composition process. Once inside our tool, we accessed the section for PRISM composition specs. The tool presents a menu containing various options for managing composition, as illustrated in Figure 7.

To create a new PRISM composition specs, one must follow three steps: i) assign a name to the composition, ii) design the combination, and iii) test the combination.

To illustrate this process clearly, let us create a combination called ''Investigation'' using three PRISM specs: Springer, DBLP, and Springer Nature Citation. In the combination design, the tool displays the available PRISM specs in the browser (Figure 8 displays the four PRISM specs available within the browser: Springer, Nature, Google Scholar, and DBLP), allowing the user to add them to a canvas where they can be linked and configured.

In this specific example, we initiated a search for articles on Springer. For each article discovered in Springer, we further searched for articles associated with the first author of that article in the DBLP database, as illustrated in Figure 9. This constituted the first combination.

In Addition, we will establish another combination of Springer and DBLP. In this scenario, for each article retrieved from Springer, we look for the corresponding number of citations in Springer Nature Citation,as depicted in Figure 10. Thus, we created a broader information relationship.

Finally, we executed searches within this composition and observed the results from each search engine abstracted by our tool, as depicted in Figure 13.

In Addition, we can configure scraping strategies, allowing us to set up the result of each PRISM spec to be added directly to the output JSON of the model, as shown in Figure 11. Alternatively, we can enable the integration of PRISM specs, resulting in a combined JSON output of the PRISM specs, as illustrated in Figure 12.

This approach enables users to create specific and customized PRISM composition specs to meet their research needs efficiently.

A. UNDERLYING TECHNICAL ISSUES

Among the various technical aspects involved in retrieving and parsing existing web content, two are particularly important in our approach and tool: (i) managing website upgrades that break DOM selectors, and (ii) handling anti-scraping measures applied by websites.

Regarding the former, we have enhanced our previous version of the tool for creating and executing PRISM specs to support selector redundancy. That is, when a PRISM spec is defined, for each DOM element selected by the user via point-and-click, multiple selectors are stored to increase resilience. However interesting studies have been conducted on heuristics for automatic repairing selectors, these approaches are mainly valuable in scenarios requiring greater autonomy. In our end-user-driven approach, executed in a web-browser context where the user is continually interacting, if a website changes and the redundant selectors are insufficient to keep the PRISM specs working, the tool notifies the user and immediately prompts them to fix the broken selector by selecting a new one via point-and-click.

Regarding the latter, avoiding CAPTCHAs and other anti-scraping measures is crucial for any commercial tool and relevant in this research as well. Nevertheless, even though this paper focuses on proposing a visual programming editor to create scraping composition models and assess its usefulness for end users without programming skills, our approach is advantageous in this regard for one key reason: the tool operates directly in the user's web browser. This means that the requests sent to the web servers closely resemble those generated by real user interactions with the web application. In several examples, including the experiment reported later in this paper, we did not encounter this problem. Although our goal was to develop a research prototype, rather than a commercial product, we foresee strategies to mitigate these barriers when a scraping model is executed. The first strategy aims to prevent CAPTCHAs from appearing: the tool can be easily adapted to issue requests at the same cadence as a typical user, and because the requests are made from the user's browser, it reduces the likelihood of triggering bot-detection mechanisms. A second strategy that can be used when a website presents a CAPTCHA during model execution leverages the browser context again: the tool detects the CAPTCHA and asks the user to complete it so that the process can continue.

VI. QUICK COMPARISON WITH TRADITIONAL WEB SCRAPING

To compare our tool with traditional scraping, we used Selenium, a Python library that has been used in various publications on web scraping [27]. Our tools were developed using JavaScript and function within the web browser as a browser extension, whereas web scraping performed with Selenium was performed with Python using execution scripts in an existing programming environment, which inherently makes it difficult for an end user to use selenium without technical programming knowledge.

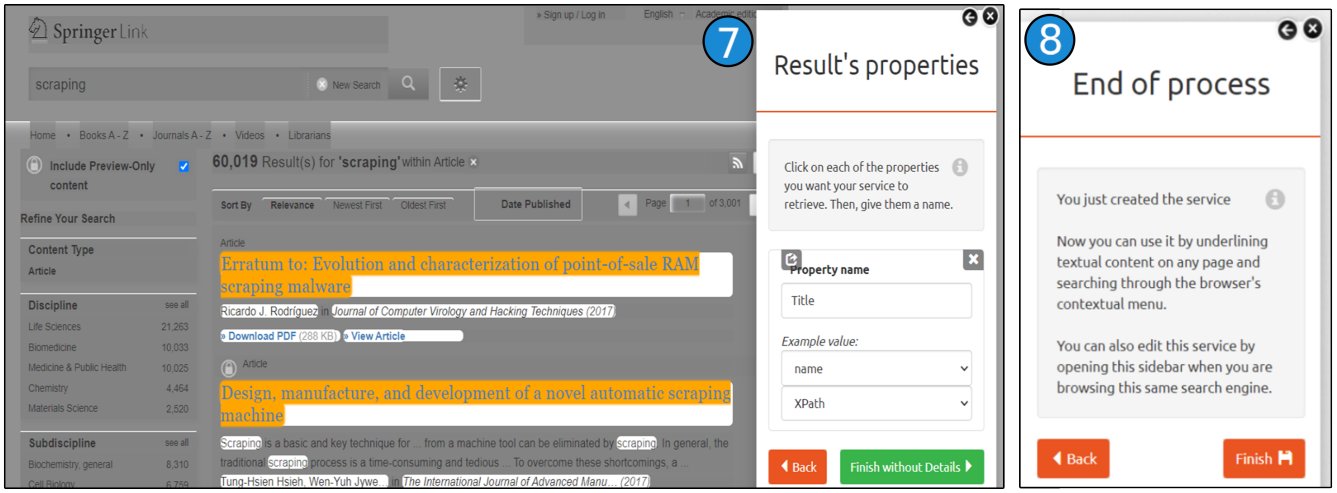

FIGURE 6. **Steps 7 and 8 for creating a PRISM spec.**

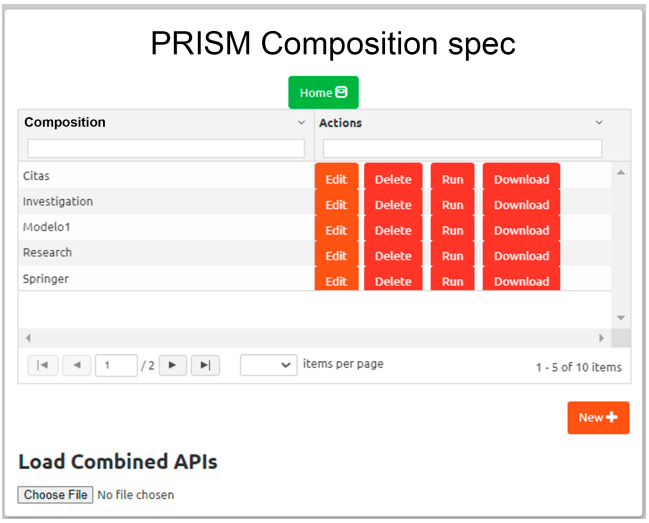

FIGURE 7. **PRISM composition spec.**

For this comparison, an author of this document with 18 years of programming experience recreated the example from Section IV, where the following data were taken for the generation of each of the search and integration services:

First, we analyzed the effort made to create a query service on a scale of 1 to 5, where 1 is low effort and 5 is high effort. To generate the Selenium script for the query service of the DBLP site, the effort level would be 4, and for Springer and Nature Citation, the effort level would be 5. In contrast, with our tool, the effort required to generate services is between one and two.

In terms of time, creating a service with Selenium for the DBLP site took 45 min, and for the Springer and Nature Citation sites, it took 60 min each, totalling 165 min. In contrast, our tool required 10 min to generate all three PRISM specs.

Let us now compare the efforts required to integrate the query services of various websites using the previously mentioned scale. For the Selenium script, it would be an effort level of five, and for our tool, it would be a level of one. In terms of time, integrating the three services took 50 min of work, whereas with the use of our tool, it took only three min.

An interesting criterion to analyze is the time required to execute queries in integrated services. In this regard, in tests performed with the example from Section IV with Selenium, a query on integration takes 10 min, whereas with our tool, it takes no more than 10 s.

The complications encountered during the construction of services with both selenium and our tool were confirmed as accepting or rejecting the use of cookies, which increased the time required to capture information from websites. Additionally, both environments are sensitive to changes in source websites; however, but the amount of effort required for maintenance is much lower with our tool.

A. ANALYSIS OF IMPACT SCENARIOS ON SELENIUM AND OUR APPROACH.

This approach is highly effective across a wide range of complexities because it is specifically designed to extract content from dynamic websites. It focuses on the environments in which information is returned in response to a search function. In Selenium has a medium-to-high degree of complexity. The impacts of changes to websites or updates to PRISM composition specs are reviewed in Table 1.

VII. EVALUATION

In this study, quantitative and qualitative analyses were carried out to provide a comprehensive understanding of the user experience with the tool. Combining both types of data enabled triangulation and validation of the information, contrasting usability metrics with subjective perceptions to achieve a more robust analysis.

TABLE 1. Impact and maintenance effort in web scraping scripts.

Scenery	Complexity	Steps to resolve the Selenium script	Steps to solve in our approach
In a PRISM composition specs, a property on the original site updates its CSS class, which causes the scraper to stop working.	Low	1) Locate the application's scraping script. 2) Read the code and identify the lines responsible for scraping (programming knowledge is required). 3) Find an *XPath* or a CSS selector that allows you to get the updated property. 4) Update the code with the new *XPath* or CSS selector. 5) Test the solution and, if the error persists, repeat the procedure.	1) Locate the PRISM spec that was affected by the change. 2) Run the property editing process by visually selecting the updated property (no programming knowledge is required). 3) Test the solution and, if the error persists, repeat the process.
A final user wants to add a new property available on the original site.	Low	1) Locate the application's scraping script. 2) Read the code and identify where to "add the code" to include the new property. 3) Find an *XPath* or a CSS selector that allows you to get the new property. 4) Add the code using the new *XPath* or CSS selector. 5) Test the solution. If the new property is not displayed, repeat the procedure.	1) Locate the PRISM spec where you want to add the new property. 2) Run the property editing process by visually selecting the new property. 3) Test the solution and, if the new property is not displayed, repeat the process.
In a composition between two PRISM specs, where a single result from the second service is obtained for each result from the first, the user wants to change the configuration to return not just the first result, but all the results from the second service.	Medium	1) Locate the application's scraping script. 2) Read the code and identify where to add the code to include a cycle of repetition to collect all results. 3) Create a new algorithm that iterates through the elements of the list resulting from the original site's query and obtains their properties. 4) Add the new algorithm. 5) Test the solution. If all the properties are not displayed, repeat the procedure.	1) Locate the link between the two PRISM specs. 2) Edit the link between the two PRISM specs to change the output from a single result to all results. 3) Test the solution. If all the properties are not displayed, repeat the procedure.
Given a composition between two PRISM specs, I want to add one or more additional PRISM specs and interconnect with them.	High	1) Locate the application's scraping script. 2) For each new service, a script must be created to scrape the new site. 3) Verify its functionality for each of the added sites. 4) Integrate the different scripts so they can communicate with each other. 5) Verify their functionality; otherwise, repeat the procedure.	1) Create the PRISM specs. 2) Edit the composition model, adding the new PRISM specs and editing the connection between the PRISM specs. 3) Verify their functionality; otherwise, repeat the procedure.

TABLE 2. Comparison with Web Scraping Tools

Tool	Required Tech Level	Internal Search Engines	Multi-source Integration / Pipelines	Authentication	Semantic Extraction	End User Control	Pagination	Detail Page Navigation	Polymorphic Combination	Object Composition	CAPTCHA Handling	Visual Programming
Octoparse	Very Low (no-code)	Limited	No, separate tasks	Basic	No	Very High	Yes	Yes	Partial (external)	No (external)	Basic (limited)	Yes, point-and-click visual interface
Apify	Medium-High (JS/Node)	Very Flexible	Yes, orchestration with Actors	Complete	Partial (programmable)	Medium	Yes	Yes	Yes (homogeneous datasets)	Yes (enriched objects)	Yes (with external resolvers)	Partial, visual interface + scriptable "Actors"
Mozenda	Low-Medium (visual)	Limited	Partial (chain agents)	Yes	No	High	Yes	Yes	Partial	Not real (external)	Limited	Yes, workflow-based visual designer
Import.io	Low-Medium (web)	Limited	Partial (merge extractors)	Yes	Partial (simple structures)	High	Yes	Yes	Yes (dashboard)	No	Partial	Yes, fully visual and automated
Bright Data	High (IDE, scripting)	Very Flexible	Yes, arbitrary flows	Complete	Partial (depends on your code)	Low	Yes	Yes	Yes, programmable	Yes, programmable	Very High (solvers + proxies)	Partial, requires scripts or API setup
Wildcard	Low-Medium (visual demo)	No	No	No	No	Very High	Yes	Yes	No	No	No	Yes, AI-based no-code visual interface
Web Scraper	Low (no-code, visual)	Yes	Yes, orchestration with Actors	Yes	No	Very High	Yes	Yes	No	No	Yes (solvers + proxies)	Yes, visual sitemap structure (click & node)
Data Miner	Low (no-code, visual)	No	No	Basic (session)	No	Very High	Yes	Yes	No	No	No	Yes, spreadsheet-style visual interface
ParseHub	Low (no-code, visual)	Yes	No	Basic (session)	No	Very High	Yes	Yes	Yes	No	Yes	Yes, guided step-by-step visual interface

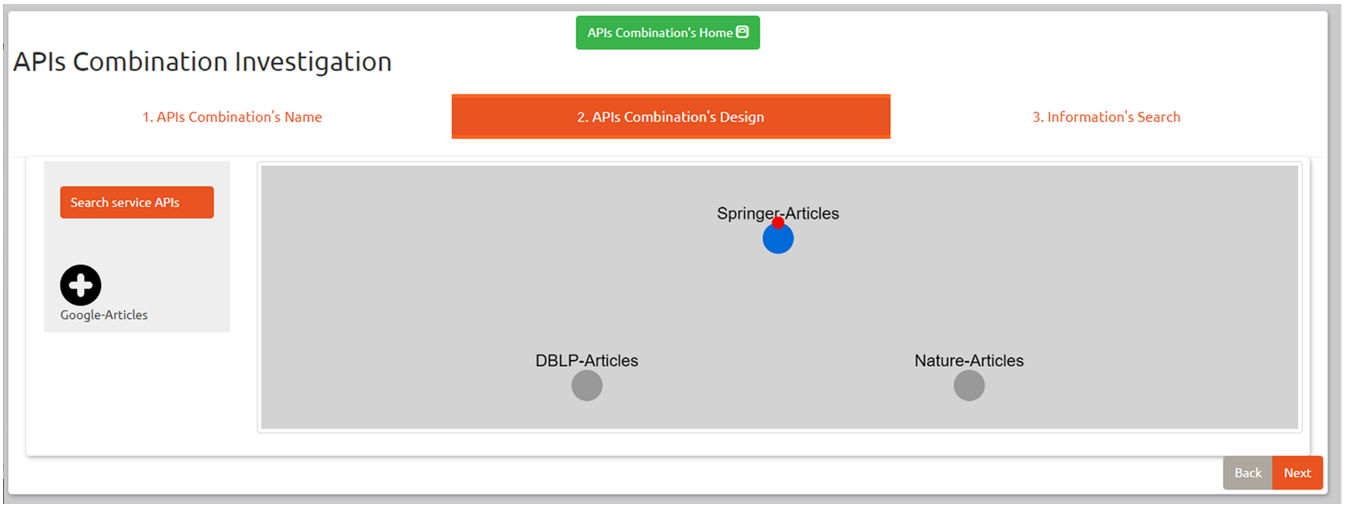

FIGURE 8. **The PRISM specs in the browser include Springer, Springer Nature Citation, Google Scholar, and DBLP. Three of these will be included in the combination.**

FIGURE 9. **Creating a relationship within the Springer and DBLP PRISM specs**

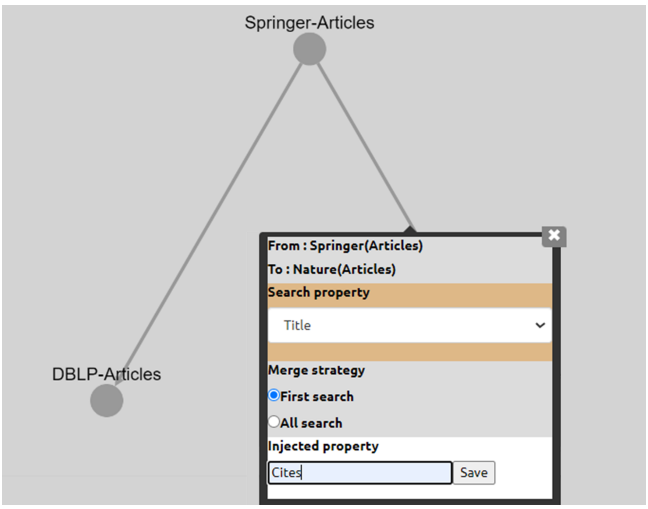

FIGURE 10. **Creating a relationship within the Springer and Nature PRISM specs.**

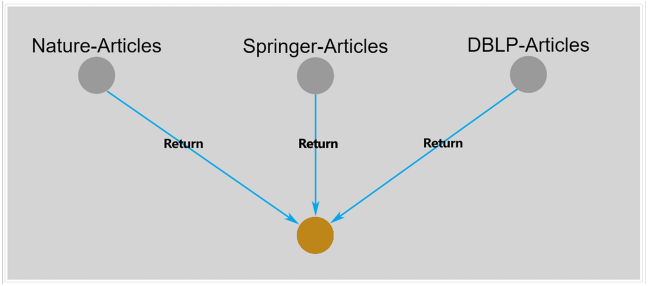

FIGURE 11. **Direct output from the PRISM spec to the JSON response.**

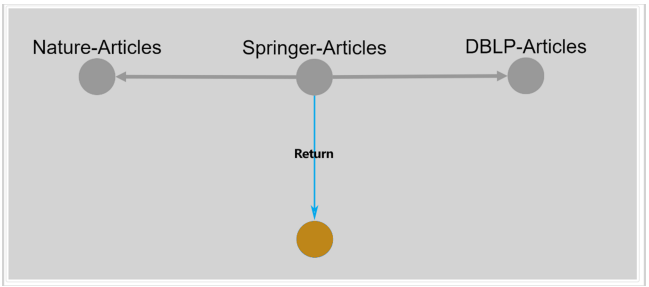

FIGURE 12. **Combining results from multiple PRISM specs and returning the integrated JSON response.**

A. INSTRUMENTATION

A call for participation in the experiment was made to undergraduate students enrolled in the B.S. Degree in Electronics and B.S Degree in Nutrition and Dietetics, with the selection criteria being students lacking programming skills. The experiment was conducted in the computing laboratory of the Faculty of Public Health at Polytechnic School of Chimborazo, Ecuador. In this case, the independent variable was the participants' field of study, comprising two groups: undergraduate students in Nutrition and in Electronics.

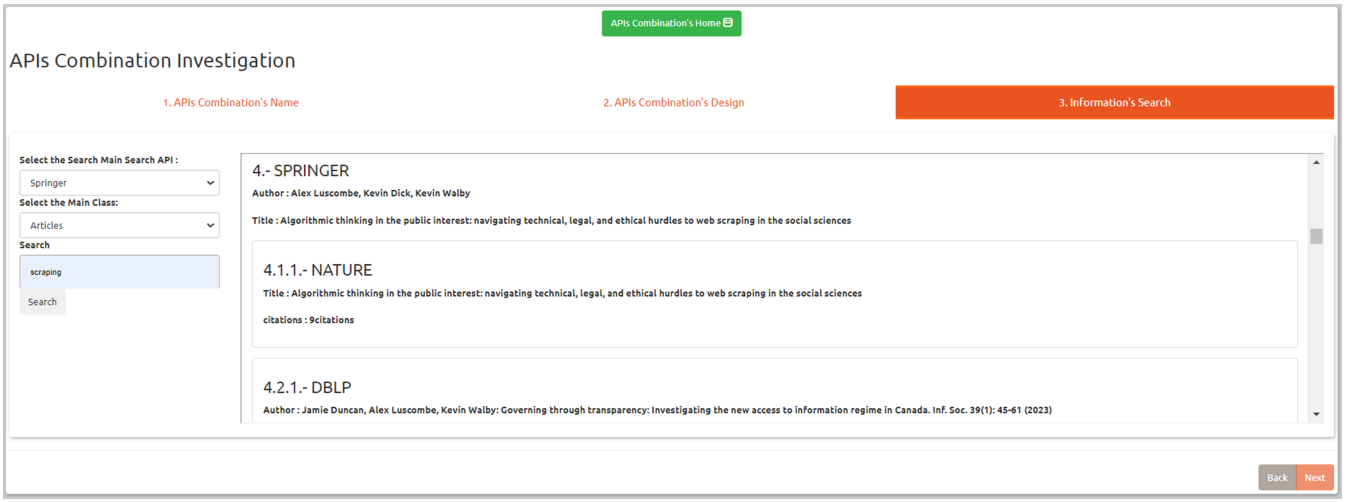

FIGURE 13. Results of the search conducted within the PRISM composition specs.

Among the data collection instruments, a pre-experiment survey was required to be filled out to gather demographic information (age, self-identified gender, nationality, educational level, and occupation) and aspects related to technology use, such as the frequency of web usage and familiarity with browser extensions. In addition, the SUS questionnaire was employed, a [31] standardized and widely recognized metric for usability assessment , which allows comparison of the results obtained with at benchmark score of 68 points.

Then, a post-experiment survey was administered, which included both the standardized SUS questionnaire items and open-ended questions about the user experience with the tool. The purpose of this post-experiment survey was to identify and understand participants' perceptions, highlighting positive and negative aspects, as well as opportunities for improvement of the tool. Open-ended responses were analyzed through inductive thematic categorization, which allowed for the identification of emerging patterns without imposing predefined categories.

The computers used for the experiment had the same specifications, including an Intel Core i7-12700 processor at 2.10 GHz, 16 GB RAM, and a 500 GB hard drive.

Upon entering the experiment, an introduction was given to the activities. We observed and supervised the process.

The tools for measuring times, click numbers, and other statistics were embedded within the same tools that stored the data in the browser's storage in the JSON format. After the experiment, these data were extracted from computers for the analyses presented below.

B. PARTICIPANTS

The tool is intended to be used by any user with or without any sort of specialization. Therefore, we ran the experiment with two types of subjects so that we could identify whether their profile could affect tool adoption.

Fifty participants participated in the study. One sample consisted of 16 undergraduate students from the Nutrition and Dietetics group, while the other sample consisted of 34 undergraduate students from Electronics. Both groups belonged to the Escuela Superior Politécnica de Chimborazo, Ecuador. The gender and academic program distributions are shown in Table 3.

It is important to note the gender composition of each degree at the institutional level. In Electronics, the majority of students were male (approximately 84.5%), whereas in Nutrition, most students were female (approximately 76.7%). These figures help explain the sex distribution observed in our experimental samples for each group, where male participants predominated in the Electronics sample and female participants predominated in the Nutrition sample.

It should be noted that all participants took part voluntarily and completed the same set of tasks under the same experimental conditions. Both groups were categorized as non-programmers, which makes them representative users for evaluating the tool in contexts in which prior training in computer science is required.

With regard to the decision to employ students from the Nutrition and Electronics programs without a programming background, this choice is supported by prior discussions on the validity of using students in software engineering experiments [32]. The literature emphasizes that what matters most is not the academic status of participants but their experience in relation to the tasks under study. In our case, since the goal was to examine usability perceptions among non-programmer users, students without programming expertise were considered an appropriate and representative sample of the intended population.

C. GOAL AND HYPOTHESIS

The goal of this experiment was to assess its usability. A system usability scale (SUS) was used for this purpose.

To evaluate whether there were statistically significant differences between the SUS scores of the Nutrition and Electronics student groups and the reference value of 68 on

TABLE 3. Sample distribution by program and gender

Program	Female	Male	Total
Nutrition	10	6	16
Electronics	8	26	34
Total	18	32	50

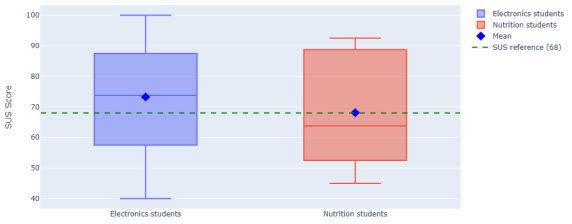

FIGURE 14. SUS scores obtained by the sample vs. average SUS score.

the SUS scale, a one-sample t-test with a one-sided alternative hypothesis (greater) was conducted. Because the variable followed a normal distribution, the use of this parametric test was considered appropriate. The following hypotheses were established:

- $\mathbf{H_0}$ (Electronics): The mean SUS score of the electronics student group equals the benchmark value of 68, i.e., $\mu_{\text{Electronics}} = 68$.
- $\mathbf{H_1}$ (Electronics): The mean SUS score of the electronics student group is greater than the benchmark value of 68, i.e., $\mu_{\text{Electronics}} > 68$.
- $\mathbf{H_0}$ (Nutrition): The mean SUS score of the nutrition student group equals the benchmark value of 68, i.e., $\mu_{\text{Nutrition}} = 68$.
- $\mathbf{H_1}$ (Nutrition): The mean SUS score of the nutrition student group is greater than the benchmark value of 68, i.e., $\mu_{\text{Nutrition}} > 68$.

Based on the one-sample t-test, it was observed that in the Electronics student group, the mean SUS score was significantly higher than the reference value of 68 ($t(33) = 1.856$, $p = 0.036$), leading to the rejection of the null hypothesis. As shown in the boxplot 14, the mean score of the Electronics students was 73.75, i.e., 5.75 points above the benchmark[3]. These results suggest that the Electronics students perceived the tool as having usability superior to the reference standard.

In the Nutrition group, the one-sample t-test did not show a statistically significant difference from the reference value ($t(15) = 0.029$, $p = 0.489$), so the null hypothesis could not be rejected. This indicated that the scores of this group were consistent with the reference values of 68. As illustrated in the Graph 14, the mean SUS score of the Nutrition students was 68.12, which is practically equal to the benchmark.

[3]https://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html

D. CORRELATION ANALYSIS

Cramér's V analysis was conducted to evaluate the potential associations between SUS score categories and participants' gender in both groups. In the Nutrition group, most participants were female (10 out of 16), whereas in the Electronics group most were male (26 out of 34). Despite these unbalanced distributions, the analysis did not reveal any association between the SUS scores and sex in either group. These preliminary findings suggest that, within the analyzed samples, gender was not related to the perceived usability of the system. It is worth noting that age homogeneity (all participants were young) and the observed sex distribution may limit the generalizability of this finding.

Considering the relevance of user diversity in understanding how individuals interact with systems and generate content, future work should extend this analysis to broader and more diverse populations, in line with recent discussions in the literature [33], [34].

On the other hand, the correlation between the number of clicks and task completion time was analyzed using Spearman's rank correlation coefficient (ρ), since the data did not follow a normal distribution and a non-parametric method was therefore more appropriate. Analyses were conducted separately for each group. In the case of the Electronics group, as shown in Figure 15, the results indicate that a higher number of clicks per task corresponded to a longer completion time for the students.

For example, the correlation value between the clicks in Tasks 1 and 2 and the time required to complete them was (ρ)=0.66, indicating a moderate to strong association.

The same situation can be observed between the clicks performed during Tasks 3 and 4 and the time allocated to these tasks. In this case, the correlation value is 0.74, indicating a significant association between the two variables. Regarding the correlation between the clicks made in tasks 5 and 6 and the time spent, we also found a moderate positive relationship, with a value of (ρ)=0.67.

Finally, it is worth highlighting the correlation value between the clicks executed in Task 7 and the time devoted to this task, which was 0.8. Thus, similar to previous cases, it represents a moderate to strong positive association.

Notably, the total time required to complete the tasks in the experiment was positively correlated with the clicks made throughout all activities. This implies that the total time required increases as the number of clicks increases.

In the Nutrition group, the Spearman correlation analysis between task completion time and the number of clicks revealed similar strong associations between these variables. As shown in the heatmap (Figure 16), the correlation coefficient between Tasks 1 and 2 and the number of clicks was (ρ)=0.86, indicating a strong relationship between both variables. Similarly, the association between the number of clicks and time for Tasks 3 and 4 was moderately high, with a coefficient of 0.59.

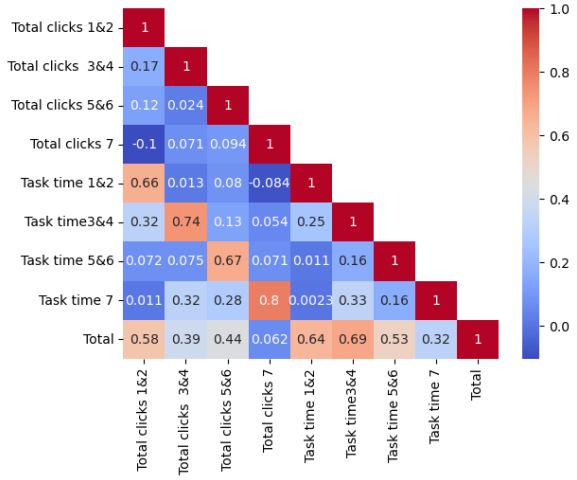

FIGURE 15. Correlation between Total clicks and Task time. Electronic Group

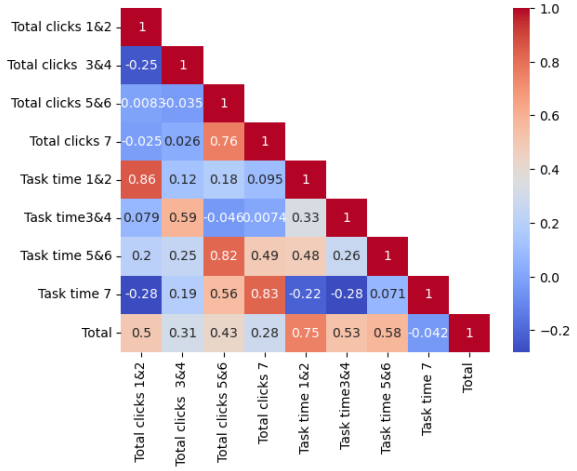

FIGURE 16. Correlation between Total clicks and Task time. Nutrition Group

Similar correlation patterns were observed in the other tasks, with coefficients indicating high correlations between these variables (0.82 for tasks 5 and 6, and 0.83 for task 7). These values indicate that, overall, a longer completion time is associated with a higher number of clicks.

E. QUALITATIVE ANALYSIS

1) Performance analysis

A relevant dimension for understanding participants' experiences relates to their degree of continuity in completing the tasks. As shown in Figure 17, in the Electronics group, almost all the students (97%) completed the first six activities. However, in the seventh task, the number of completions dropped to 25 (73%), mainly because some participants chose to leave sessions early because of academic commitment. This suggests that the lower completion rate was not associated with the difficulties inherent to the tool, but rather with the external conditions of the experiment.

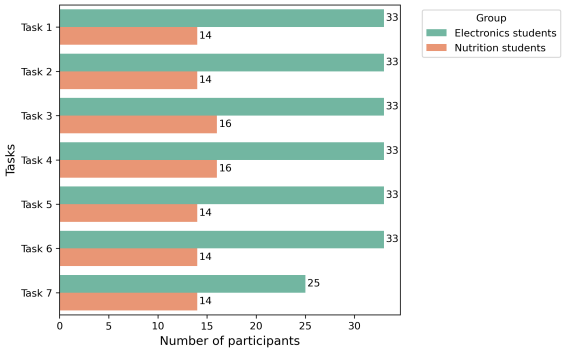

FIGURE 17. Number of participants who completed each task by group

In the Nutrition group, participation remained more stable across different activities, with an average completion rate of 87.5%. It is worth noting that Tasks 3 (model generation) and 4 (model search) were completed by all students, which could indicate greater accessibility or a lower perceived level of complexity in these activities.

Tasks one to six focused on the development of three models with progressively higher complexity, determined by the number of PRISM specs used. Analysis of the results indicated that, even as task complexity increased, both Nutrition and Electronics students did not have difficulties completing them successfully.

2) User's feedback

In the post-experiment survey, in addition to usability questions (SUS), open-ended questions were included to explore and better understand participants' perceptions of the tool. The analysis of these responses showed that the majority of students (55%) indicated that they would prefer to use it for academic purposes, such as writing scientific articles and theses. Another 30% stated that they would use it for various everyday inquiries, while the remaining 15% reported that they would use it for completing assignments and coursework. It should be noted that, for all participants, this was their first experience in a usability evaluation.

Among the main strengths, participants highlighted ease of use, speed of obtaining results, and the usefulness of integrating multiple source factors that reinforce the usability scores previously presented. Some responses illustrate these positive perceptions:

"*The ability to link search engines, edit relationships, and create different models*"

"*It shows precise, fast, and necessary results*"

"*Easy to use, intuitive, very useful and functional*"

On the other hand, participants also reported weaknesses, mainly related to technical issues (e.g., unresponsive buttons, errors in zooming or information display), as well as limitations of the interface, which in some cases was described as

"*rigid*" or "*not very intuitive.*" Regarding possible improvements, several participants suggested the inclusion of tutorials and preconfigured models, as reflected in comments such as:

"*The user experience could be improved by incorporating tutorials and preconfigured models, as well as optimizing the visual design and system stability*"

"*The extension should be better explained to make its use more user-friendly.*"

Taken together, these qualitative observations complement the usability metrics: showing that the tool was perceived as efficient, easy to use, and valuable for both academic and everyday purposes, as also supported by an average SUS score above the standard benchmark, while at the same time highlighting the need for technical and design adjustments to encourage broader adoption.

F. THREATS TO VALIDITY

In this section, we address the main challenges to validity and how we attempted to mitigate them during the design and execution of the experiment. For this purpose, we used the classification proposed by Wohlin [29].

1) Conclusion validity

In the context of the reported experiment, validity refers to the relationship between the treatment and outcome, ensuring that a statistical relationship exists with a certain level of significance.

To avoid low statistical power, we worked with a total sample of 50 participants, divided into two groups (16 from Nutrition and Dietetics and 34 from Electronics), ensuring adequate sample size and power. Additionally, to address the threat of violated assumptions of statistical tests, the one sample T test was used to compare the mean SUS scores obtained by the participants with the average reference value from this survey. Furthermore, additional statistical analyses were conducted to examine correlations (Cramer's V and Spearman correlation), after verifying the measurement scales and probability distributions of all variables involved.

Regarding fishing and error rates, the experiment focused on objective measures, and no data points were excluded from the dataset to guarantee precise results. The reliability of the measurements was ensured by collecting measurements automatically to avoid errors that may occur during manual data processing. Additionally, to maintain reliability in treatment implementation, the experiment utilized documents and questionnaires to ensure uniform execution of tasks across participants.

An optimal experimental environment was provided for random irrelevancies. Finally, participants with similar profiles and academic affiliations (undergraduate students in Electronics and Nutrition at the Escuela Superior Politécnica de Chimborazo, Ecuador) were selected to minimize the threat of random heterogeneity.

2) Internal validity

Threats to internal validity can influence the independent variable regarding causality without the researcher's knowledge.

Because the experiment was conducted with a single group, we will only discuss the threats identified by the author for this situation. First, regarding history, the experiment was conducted on the same day of the week, without specific events. To address the threat of maturation, the experiment was designed to allow participants to complete successive tasks within approximately an hour and a half, with the aim of preventing any effects of fatigue or boredom.

Considering the single-group design and measurement of participants' performance across different tasks, the threat of testing was minimized by planning tasks in a way that did not promote unintended learning. It is worth mentioning that participants had no prior experience with the tools used in the experiment.

Regarding instrumentation, pre and post-experiment questionnaires and documents detailing each task were carefully designed to be clear and understandable to the participants. An introductory video of the PRISM specs was also provided. Importantly, the participants were not involved in any of the previous experiments conducted by the researchers, thus reducing the risk of statistical regression.

For participant selection, undergraduate students in Electronics and Nutrition were selected, given that they constitute a representative non-programmer population. Participation in the study was voluntary. While working with volunteers may introduce motivational bias, this threat was considered when analyzing the effects of the independent variable on the dependent variables. Finally, there were no dropouts in participation (mortality), because all participants completed the experiment on the same day and at the same location.

3) Construct validity

Construct validity focuses on the relationship between theory and observation, that is, the ability to generalize an experiment's results to the underlying concept or theory. This type of threat pertains to social factors and experimental design.

The experiment investigated the possibility of repurposing web PRISM specs to create models that integrate information retrieval. Participants' performance was evaluated in seven model creation and editing tasks. During these tasks, the participants conducted searches within the models. There were no indications that participants were more sensitive or responsive to treatment; hence, no interaction between testing and treatment was observed.

Furthermore, considering that all subjects participated in all conditions and were not concurrently involved in a similar study at the time of recruitment, the interaction of different treatments would not pose a threat in this case.

To prevent participants from guessing the purpose of the study (hypothesis guessing), they were not informed of the research objectives or the hypotheses. Only general information regarding the experiment was provided. In addition

to measuring the participants' performance (task completion time, number of tasks completed, and number of clicks), an SUS survey was included to compare all measures used, thereby avoiding mono-method bias.

4) External validity

Threats to external validity refer to the ability to generalize experimental results, which are influenced by both the design of the experiment and the selected subjects and objects.

As mentioned earlier, the experimental subjects were undergraduate students in Electronics and Nutrition, aged between 20 and 28 years, with similar experience using the Internet and web search engines, making it a homogeneous group The participants in both samples share a non-programmer profile, which is relevant given that the evaluated tool is intended for general-purpose use. Considering that the subjects fall within a young age range, the results have limited generalizability and are primarily applicable to young non-programmer populations. Therefore, the data could be generalized beyond this study, but only to populations sharing similar contexts and characteristics. To generalize the results to a broader population, an experiment should be conducted with new samples, including participants with different profiles, such as industry employees or individuals from various age ranges.

Regarding the threat of interaction between setting and treatment, the experiment was conducted in a university environment, which, while providing the necessary elements. Finally, the threat of the interaction of history and treatment is not considered relevant because no events or circumstances could have affected the results before or during the experiment.

VIII. CONCLUSIONS AND FUTURE WORKS

The accessibility of vast amounts of data on the Web underscores its significance and importance for end users across diverse fields. However effectively harnessing these data presents challenges that can be addressed through approaches, such as web scraping.

The role of AI in modern web-scraping techniques cannot be overlook. AI algorithms facilitate automated data extraction and offer a high accuracy and speed. Their adaptability to dynamic web environments enhances the effectiveness of scraping processes, thereby ensuring dependable results for end users. However, the emphasis on controllability, as discussed in the previous sections, sets apart the end-user approach to web scraping. However, the end-user approach for web scraping brings another benefit: a) customization, which allows users to customize their scraping processes according to their specific requirements, as they can define the data fields they need, the websites to scrape, and the frequency of scraping; b) ease of use: not everyone is familiar with programming or AI-based techniques; and c) quick deployment: for small-scale scraping tasks or one-time data extraction needs, using an end-user tool can be quicker and more convenient than developing custom AI solutions.

End users, empowered by AI tools and web browsers, play a pivotal role in shaping the scraping process. They can control parameters such as data selection criteria, extraction methods, and frequency, ensuring precision and relevance in data extraction. This level of control enhances the quality of the extracted data and enables users to tailor scraping processes to their specific needs and preferences beyond professional use.

Furthermore, leveraging scraping techniques to develop web extensions enhances user experience and interaction on the web because scraped data may be used as input for other tools to customize the online experience, access relevant information, and effectively streamline workflows.

With the surfaced LLM-based tools that are appearing nowadays, it is clear that end users may be required to use them by "feeding" them using specific information that has recently been collected or scraped from the web. This is a very interesting and convenient approach for web scraping that works in a web browser because, at any moment, it can also use the collected data as input In LLM-based applications, defining interesting workflows for end users.

Therefore, in future work, we plan to use AI-based techniques to support the specification of PRISM specs. Although the ANDES tool has been proven to be suitable for end users without programming skills, we are aware of the benefits of using a more automated approach for PRISM specs. In this sense we plan to use AI to make it easier for the annotation process required to define the PRISM specs, mainly by extracting a common metadata structure from web search result pages, which would reduce the workload of the end user just to tasks related to maintaining the controllability that our approach currently offers.

Similarly, we plan to incorporate AI uses to support conversational interaction for composing models; that is, to use conversational AI to compose models automatically as a response to a user prompt based on natural language. The main idea rests on the fact that having semi-structured natural language specifications about both the PRISM specs and how these PRISM specs are combined, it would be possible (by using prompt engineering and pipelines) to transform end users intents into specifications that stablish how to create the composition model. With this in mind, it would be possible to define composition models on the fly in response to end user requirements such as "Search about 'Web Scraping' in Sringer and for each resulting article obtain the citation amount from Google Scholar", something that could be used to define the composition model or just for their direct use in front of more volatile need about information scraping.

We are also designing tools for in-browser use of scraped information to improve the overall user experience while surfacing the web, improving navigation, and adding concern-sensitive helpers. Additionally, future work will extend this analysis to broader and more diverse populations, considering the relevance of user diversity in understanding how individuals interact with systems and generate content. This aligns with recent discussions in the literature showing that

TABLE 4. SUS Values, Age, and Gender of Participants

ID	SUS Value	Age	Gender[a]
1	52.5	28	0
2	70.0	21	0
3	50.0	25	0
4	65.0	27	1
5	45.0	22	1
6	62.5	27	1
7	87.5	23	0
8	90.0	24	0
9	92.5	23	0
10	52.5	21	1
11	62.5	25	0
12	90.0	22	1
13	57.5	22	0
14	47.5	24	0
15	75.0	23	0
16	90.0	21	1
17	72.5	22	0
18	75.0	26	1
19	67.5	23	0
20	97.5	24	1
21	90.0	22	1
22	65.0	21	0
23	40.0	22	1
24	72.5	23	1
25	75.0	22	1
26	70.0	28	1
27	57.5	26	1
28	90.0	24	0
29	92.5	24	1
30	85.0	25	0
31	50.0	22	1
32	52.5	22	1
33	50.0	22	1
34	100.0	22	1
35	80.0	20	0
36	75.0	21	1
37	67.5	21	1
38	62.5	22	1
39	50.0	23	1
40	85.0	22	1
41	85.0	23	0
42	100.0	21	1
43	65.0	25	1
44	57.5	23	0
45	87.5	21	1
46	82.5	22	1
47	57.5	22	1
48	52.5	22	1
49	90.0	23	1
50	90.0	22	1

[a]Gender is encoded as 0 for female and 1 for male.

user characteristics such as gender and age can affect the nature and dynamics of online content [33], [34]. Finally, new experiments must be designed and implemented using these approaches and tools.

APPENDIX A
SUS SAMPLES

Next, the SUS values from the experiment are presented in the Table 4:

CONFLICT OF INTEREST

The authors declare that they have no competing interests.

ACKNOWLEDGMENT

This research was partially funded by FONCYT Argentina, project PICT-2020-SERIEA-03378, and by LIFIA, Faculty of Informatics, National University of La Plata.

REFERENCES

[1] B. Zhao, "Web scraping," in *Encyclopedia of Big Data*, L. A. Schintler and C. L. McNeely, Eds. Cham, Switzerland: Springer, 2017, pp. 1–3. doi: 10.1007/978-3-319-32001-4_483-1.

[2] R. Diouf, E. N. Sarr, O. Sall, B. Birregah, M. Bousso, and S. N. Mbaye, "Web scraping: state-of-the-art and areas of application," in *Proc. IEEE Int. Conf. Big Data (Big Data)*, Los Angeles, CA, USA, Dec. 2019, pp. 6040–6042. doi: 10.1109/BigData47090.2019.9006496.

[3] M. A. Khder, "Web scraping or web crawling: State of art, techniques, approaches and application," *Int. J. Adv. Soft Comput. Appl.*, vol. 13, no. 3, pp. 1–12, 2021.

[4] D. Glez-Peña, A. Lourenço, H. López-Fernández, M. Reboiro-Jato, and F. Fdez-Riverola, "Web scraping technologies in an API world," *Brief. Bioinform.*, vol. 15, no. 5, pp. 788–797, Sep. 2014. doi: 10.1093/bib/bbt026.

[5] A. Schulz, J. Lässig, and M. Gaedke, "Practical web data extraction: Are we there yet?—A short survey," in *Proc. IEEE/WIC/ACM Int. Conf. Web Intell. (WI)*, Omaha, NE, USA, Oct. 2016, pp. 562–567. doi: 10.1109/WI-ICT.2016.0094.

[6] G. Bosetti, S. Firmenich, M. Winckler, G. Rossi, U. C. Fandos, and E. Egyed-Zsigmond, "An end-user pipeline for scraping and visualizing semi-structured data over the Web," in *Web Engineering (ICWE 2019)*, Daejeon, South Korea: Springer, Jun. 2019, pp. 223–237. doi: 10.1007/978-3-030-19274-7_14.

[7] J. Gruser, L. Raschid, M.-E. Vidal, and L. Bright, "Wrapper generation for Web accessible data sources," in *Proc. 3rd IFCIS Int. Conf. Cooperative Inf. Syst. (CoopIS)*, New York, NY, USA, Aug. 1998, pp. 14–23. doi: 10.1109/COOPIS.1998.706273.

[8] D. Glez-Peña, A. Lourenço, H. López-Fernández, M. Reboiro-Jato, and F. Fdez-Riverola, "Web scraping technologies in an API world," *Brief. Bioinform.*, vol. 15, no. 5, pp. 788–797, Sep. 2014. doi: 10.1093/bib/bbt026.

[9] R. R. NR and M. Vijayalakshmi, "Web scraping tools and techniques: A brief survey," in *Proc. 4th Int. Conf. Innovative Trends Inf. Technol. (ICITIIT)*, Kochi, India, Feb. 2023, pp. 1–4. doi: 10.1109/ICI-TIIT57168.2023.10136084.

[10] K. U. Manjari, S. Rousha, D. Sumanth, and J. S. Devi, "Extractive text summarization from web pages using Selenium and TF-IDF algorithm," in *Proc. 4th Int. Conf. Trends Electron. Informat. (ICOEI)*, Tirunelveli, India, Jun. 2020, pp. 648–652. doi: 10.1109/ICOEI48184.2020.9142991.

[11] R. Alshammari, R. Alrashed, A. Almutiri, M. Alwalah, W. Al-Marrai, D. Alqahtani, and A. Alzahrani, "Data extraction based on Web Scrapy," in *Innovations in Bio-Inspired Computing and Applications*, A. Abraham, H. Sasaki, R. Rios, N. Gandhi, U. Singh, and K. Ma, Eds. Cham, Switzerland: Springer, 2021, pp. 506–514. doi: 10.1007/978-3-030-73603-3_51.

[12] J. M. Patel, "Web scraping in Python using Beautiful Soup library," in *Getting Structured Data from the Internet*, Berkeley, CA, USA: Apress, 2020, pp. 31–84. [Online]. Available: https://link.springer.com/chapter/10.1007/978-1-4842-6576-5_2

[13] S. Kumar and U. B. Roy, "2 - a technique of data collection: web scraping with python," in *Statistical Modeling in Machine Learning*, T. Goswami and G. R. Sinha, Eds. Massachusetts: Academic Press, 2023, pp. 23–36. doi: 10.1016/B978-0-323-91776-6.00011-7

[14] N. Dinh and L. Ogiela, "Human-artificial intelligence approaches for secure analysis in CAPTCHA codes," *EURASIP J. Inf. Secur.*, vol. 2022, no. 1, pp. 1–20, Dec. 2022. doi: 10.1186/s13635-022-00128-3.

[15] J. Lin, J. Wong, J. Nichols, A. Cypher, and T. A. Lau, "End-user programming of mashups with Vegemite," in *Proc. Int. Conf. Intell. User Interfaces (IUI)*, Sanibel Island, FL, USA, 2009, pp. 97–106.

[16] S. E. Chasins, M. Mueller, and R. Bodik, "Rousillon: Scraping distributed hierarchical web data," in *Proc. 31st Annu. ACM Symp. User Interface Softw. Technol. (UIST)*, Berlin, Germany, Oct. 2018, pp. 963–975. [Online]. Available: https://doi.org/10.1145/3242587.3242661

[17] T. Alrashed, J. Almahmoud, A. X. Zhang, and D. R. Karger, "ScrAPIr: Making web data APIs accessible to end users," in *Proc. CHI Conf. Human Factors Comput. Syst.*, Honolulu, HI, USA, Apr. 2020. [Online]. Available: https://dl.acm.org/doi/10.1145/3313831.3376691

[18] H. Kühnemann, "Anwendungen des Web Scraping in der amtlichen Statistik," *AStA Wirtschafts- Sozialstatistisches Arch.*, vol. 15, no. 1, pp. 5–25, Mar. 2021. doi: 10.1007/s11943-020-00269-3.

[19] G. Litt and D. Jackson, "Wildcard: Spreadsheet-driven customization of web applications," in *Proc. ACM Int. Conf. Interactive Surfaces and Spaces (ISS)*, Lisbon, Portugal, Mar. 2020, pp. 126–135. [Online]. Available: https://dl.acm.org/doi/10.1145/3397537.3397541

[20] X. Deng, P. Shiralkar, C. Lockard, B. Huang, and H. Sun, "DOM-LM: Learning generalizable representations for HTML documents," *ArXiv*, vol. abs/2201.10608, 2022. [Online]. Available: https://api.semanticscholar.org/CorpusID:246285527

[21] F. Bai, J. Kang, G. Stanovsky, D. Freitag, M. Dredze, and A. Ritter, "Schema-driven information extraction from heterogeneous tables," in *Findings of the Association for Computational Linguistics: EMNLP 2024*, Miami, Florida, USA, Nov. 2024, pp. 10252–10273. Publisher: Association for Computational Linguistics. [Online]. Available: https://aclanthology.org/2024.findings-emnlp.600/ doi: 10.18653/v1/2024.findings-emnlp.600

[22] "Web Scraper - The #1 web scraping extension." [Online]. Available: https://webscraper.io/

[23] "Scrape data from any website with 1 Click | Data Miner." [Online]. Available: https://dataminer.io/

[24] D. Ahamad, A. Mahmoud, M. Mahmoud, and M. Akhtar, "Strategy and implementation of web mining tools," *Int. J. Innov. Res. Adv. Eng.*, vol. 12, pp. 1–7, 2017. [Online]. Available: https://www.ijirae.com

[25] I. S. H. Almaqbali, F. M. A. A. Khufairi, M. S. Khan, A. Z. Bhat, and I. Ahmed, "Web scrapping: Data extraction from websites," *J. Student Res.*, Jul. 2020. [Online]. Available: https://www.jsr.org/index.php/path/article/view/942

[26] G. Bosetti, A. Tacuri, I. Gambo, S. Firmenich, G. Rossi, M. Winckler, and A. Fernandez, "Andes: An approach to embed PRISM specsvices on the web browser," *Comput. Stand. Interfaces*, vol. 82, 2022. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85126970757

[27] H. So, M. Chang, S. K. Lee, and S. M. Yoo, "Automation of estimation of urban railway station location using web scraping for establishment of facility database," *J. Korean Soc. Railway*, vol. 26, no. 168, pp. 401–408, 2023. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85168941345

[28] Y. Zhou, Y. Sheng, N. Vo, N. Edmonds, S. Tata, and J. K. Rowling, "Simplified DOM trees for transferable attribute extraction from the web," *arXiv preprint arXiv:2101.02415*, 2021. [Online]. Available: https://doi.org/10.48550/arXiv.2101.02415 doi: 10.48550/arXiv.2101.02415

[29] C. Wohlin, P. Runeson, M. Höst, M. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in Software Engineering*, ser. Computer Science. Berlin, Germany: Springer Berlin Heidelberg, 2012.

[30] G. Leshed, E. M. Haber, T. Matthews, and T. A. Lau, "CoScripter: automating & sharing how-to knowledge in the enterprise," in *Proc. 2008 Conf. on Human Factors in Computing Systems (CHI)*, Florence, Italy, Apr. 2008, pp. 1719–1728, doi: 10.1145/1357054.1357323.

[31] A. Bangor, P. T. Kortum, and J. T. Miller, "An empirical evaluation of the System Usability Scale," in *International Journal of Human–Computer Interaction*, vol. 24, no. 6, pp. 574–594, 2008, doi: 10.1080/10447310802205776.

[32] D. Falessi, N. Juristo, C. Wohlin, B. Turhan, J. Münch, A. Jedlitschka, and M. Oivo, "Empirical software engineering experts on the use of students and professionals in experiments," in *Empirical Software Engineering*, vol. 23, no. 1, pp. 452–489, 2018, doi: 10.1007/s10664-017-9523-3.

[33] L. Hudders and S. De Jans, "Gender effects in influencer marketing: an experimental study on the efficacy of endorsements by same- vs. other-gender social media influencers on Instagram," in *International Journal of Advertising*, vol. 41, no. 1, pp. 128–149, 2022, doi: 10.1080/02650487.2021.1997455.

[34] N. Thakur, S. Cui, K. Khanna, V. Knieling, Y. N. Duggal, and M. Shao, "Investigation of the gender-specific discourse about online learning during COVID-19 on Twitter using sentiment analysis, subjectivity analysis, and toxicity analysis," in *Computers*, vol. 12, no. 11, p. 221, 2023, doi: 10.3390/computers12110221.

[35] G. Barba, M. Lezzi, M. Lazoi, and A. Corallo, "Combined use of web scraping and AI-based models for business applications: research evolution and future trends," in *Management Review Quarterly*, Springer, Sep. 2025. doi: 10.1007/s11301-025-00551-3.

[36] Y. Yannikos, J. Heeger, and M. Steinebach, "Scraping and Analyzing Data of a Large Darknet Marketplace," *Journal of Cyber Security and Mobility*, vol. 12, no. 2, pp. 161–186, 2023, doi: 10.13052/jcsm2245-1439.1222.

[37] A. S. Kazmali and A. Sayar, "Web Scraping: Legal and Ethical Considerations in General and Local Context — A Review," *Procedia Computer Science*, vol. 259, pp. 1563–1572, 2025. Sixth International Conference on Futuristic Trends in Networks and Computing Technologies (FTNCT06), Uttarakhand, India. doi: https://doi.org/10.1016/j.procs.2025.04.111.

ALEX TACURI earned his degree in Systems Engineering in 2005 in Ecuador. In 2010, he obtained a Master's degree in Network Interconnectivity. He is currently a Ph.D. candidate in Computer Science at the National University of La Plata (UNLP), Argentina. He works as an Information Systems Development Analyst at the Escuela Superior Politécnica de Chimborazo (ESPOCH), and he is currently completing his doctoral internship at LIFIA (UNLP), conducting research on end-user programming, web augmentation, and web scraping.

SERGIO FIRMENICH received his degree in Computer Science in 2008. In 2009, he joined LIFIA (UNLP) to begin his Ph.D. with a scholarship from CONICET Argentina, that he completed in April 2013. His doctoral thesis focuses on an approach to support user tasks on the Web through client-side adaptation. He is currently researching various aspects of Web Engineering, including methodological aspects, empirical software engineering, end-user programming, and web augmentation, among others. He is currently a professor and researcher at the Loyola University, Andalusia, Spain.

ALEJANDRO FERNÁNDEZ is the Director of the LIFIA Research Center at the National University of La Plata (UNLP), Argentina. He is a Full Professor at the UNLP School of Computer Science and an Independent Researcher at the Scientific Research Commission of the Province of Buenos Aires (CICPBA). He holds a Ph.D. in Computer Science from FernUniversität in Hagen, Germany. His research spans Human-Centered Software Engineering, Web Data Extraction, and End-User Development. He has led projects involving the design of tools for data integration and end-user interactions with complex software systems. He also coordinates with the Quantum Computing Working Group of CLEI and is a founding member of the Ibero-American Network for Advancing Quantum Software Engineering.

FLORENCIA RIVA holds a bachelor's degree in Sociology from the National University of La Plata (UNLP). She is currently working as a professional assistant in the Research Support Staff Program of the Scientific Research Commission of the Province of Buenos Aires (CIC-PBA), based at the LIFIA Research Center (Faculty of Computer Science, UNLP, Argentina). Her research interests include empirical software engineering, with a particular focus on the planning, design, and execution of experiments.

MATÍAS URBIETA was born in Formosa, Argentina, in 1981. He received the B.S. and Ph.D. degrees in computer science from the National University of La Plata (UNLP). Since 2008, he has been a Research Assistant at the Laboratory for Research and Training in Advanced Informatics (LIFIA), La Plata, Buenos Aires, Argentina. He taught object-oriented programming concepts and web engineering at undergraduate and Ph.D. levels. His research interests include separation of concerns in web applications, agile requirements engineering, and web augmentation.

GUSTAVO ROSSI received the Ph.D. from PUC-Rio, Brazil. He is a Full Professor with Universidad Nacional de La Plata, Argentina and a researcher with CONICET. His current research interests include web engineering, requirements engineering, and human–computer interaction. He is a member of the Editorial Board of IEEE INTERNET COMPUTING, IEEE IT PROFESSIONAL, and World Wide Web journal.