DOI: **10.33099/2311-7249/2024-51-3-80-88**

УДК: 004.8

Миколайчук Роман Антонович *(доктор технічних наук, доцент)* [1]
Старинський Іван Михайлович *(кандидат технічних наук)* [1]
Миколайчук Віра Романівна *(доктор філософії)*[2]

[1] ***Національний університет оборони України, Київ, Україна***
[2] ***Київський національний університет імені Тараса Шевченка, Київ, Україна***

АНАЛІЗ ТЕХНОЛОГІЧНИХ АСПЕКТІВ РЕАЛІЗАЦІЇ ВЕБ-СКРАПІНГУ СТАТИЧНИХ І ДИНАМІЧНИХ САЙТІВ

Стаття присвячена розробці рекомендацій щодо використання сучасних технологій вебскрапінгу для забезпечення ефективного збору інформації зі статичних та динамічних сайтів. У контексті зростаючого обсягу даних та їхньої складної структури, особливо на динамічних ресурсах, виникає необхідність вибору оптимальних інструментів для автоматизованого збору інформації. Традиційні підходи до вебскрапінгу часто є недостатньо гнучкими для обробки складних динамічних сайтів. Метою статті є аналіз існуючих методів вебскрапінгу та розробка практичних рекомендацій для їхнього застосування. Під час дослідження були використані бібліотеки Selenium та BeautifulSoup як окремо, так і в комбінації, що дали змогу оцінити їхню ефективність у різних умовах. Зі статичних сайтів, BeautifulSoup виявився найефективнішим, завдяки швидкості обробки, тоді як Selenium забезпечує успішний збір даних з динамічних ресурсів. Запропоновані підходи були перевірені у межах експериментального середовища, що дало змогу визначити їх переваги та обмеження. Наукова новизна дослідження зводиться до комплексного аналізу ефективності інструментів вебскрапінгу залежно від типу сайту. Теоретичною значущістю є розширення розуміння особливостей взаємодії з різними структурами вебресурсів, а практичною значущість – надання відповідних рекомендацій для розробників, аналітиків і дослідників. Викладене у статті сприяє розвитку сфери автоматизованого аналізу вебінформації з використанням запропонованих інструментів, що оптимізують процеси збору даних.

Ключові слова: *інформаційні технології, автоматизація збору даних, оцінка ефективності, вебдизайн, вебсайти, моделювання, вебскрапінг, алгоритми обробки інформації.*

Вступ

Вебскрапінг (web scraping) – це процес автоматизованого збору даних із вебсайтів за допомогою програмного забезпечення або скриптів. Основна мета – отримати структуровану інформацію з вебресурсів для подальшого аналізу чи використання. У процесі вебскрапінгу дані, представлені у вигляді HTML-коду чи динамічного контенту, перетворюються у зручний формат, такий як CSV, JSON, Excel або база даних.

В епоху цифрових трансформацій та великих даних вебскрапінг став потужним інструментом для автоматизованого збору інформації з вебресурсів. Технології вебскрапінгу успішно застосовуються у різних видах людської діяльності від аналізу ринкових тенденцій до автоматизації бізнес-процесів, таких як моніторинг цін, генерація лідів та дослідження споживчої поведінки. Особливо це актуально для фахівців, які працюють із великими обсягами неструктурованих даних.

Дана стаття присвячена аналізу технологічних аспектів вебскрапінгу з акцентом на його використання для збору інформації зі статичних та динамічних вебсторінок. Основу дослідження складають алгоритми обробки та парсингу HTML, динамічне рендеринг даних за допомогою бібліотек python, а також моделі для оцінки ефективності процесу збору інформації.

Вебскрапінг є критично важливим інструментом у таких сферах, як машинне навчання, обробка природної мови, маркетингові дослідження, аналіз соціальних мереж та інших. Проте існують виклики, пов’язані з динамічними змінами вебсторінок, технічними обмеженнями інструментів та правовими аспектами. У статті пропонуються шляхи вирішення цих проблем за допомогою комбінованих методів, що поєднують математичні підходи та сучасні інформаційні технології.

Отже, дослідження спрямовані на розробку практичних рекомендацій для автоматизації збору даних з урахуванням технічної складності та ефективності обчислень.

Постановка проблеми. У сучасному світі, де інформація стає одним із ключових факторів для прийняття обґрунтованих рішень, збір та аналіз даних із вебресурсів є важливим завданням для багатьох сфер, таких як управління військами, освіта, наука тощо. Однак традиційні методи збору інформації є трудомісткими, потребують значних часових ресурсів і можуть бути непридатними для

роботи з великими масивами даних. Крім того, зростає кількість динамічних вебсайтів, які використовують JavaScript для формування контенту, що значно ускладнює доступ до даних звичайними методами. Також виникають такі технічні та етичні виклики, як:

подолання обмежень доступу – використовуються капчі, блокування IP-адрес та інших анти-скрапінгових заходів;

обробка великих масивів даних – оптимізуються алгоритмами для зменшення часу виконання та обсягів обробки.

динамічний контент – взаємодія із сучасними вебресурсами, що формують контент на стороні клієнта.

Існуючі інструменти вебскрапінгу (BeautifulSoup, Scrapy), часто виявляються недостатньо ефективними для роботи з динамічними вебресурсами, що використовують JavaScript. У таких випадках необхідним стає використання інструментів, які забезпечують автоматизацію взаємодії з вебсторінками, наприклад, Selenium. Тому, виникає необхідність розробки ефективних методів та алгоритмів для збору й обробки даних із динамічних вебресурсів із врахуванням технічних обмежень та підвищення продуктивності процесу.

Аналіз останніх досліджень і публікацій. У науковій літературі вебскрапінг привертає все більше уваги завдяки широкому спектру його застосувань і технічних викликів, пов'язаних із динамічними вебсторінками. У [1] розглянуто прогресивні підходи до збору інформації, зокрема систему AutoScraper, яка використовує агентів для автоматизації процесу розуміння структури вебсторінок. Дослідники зосередилися на покращенні точності вилучення даних, що особливо важливо для динамічних сайтів.

У [2] автори дослідили можливості використання великих мовних моделей (далі – LLM) для автоматизації вебскрапінгу. Вони демонструють як інтеграція LLM допомагає розпізнавати складні шаблони даних і вирішувати завдання в умовах постійно змінюваних вебресурсів.

Довіра до отриманих даних є ключовим викликом, який обговорюється у [3]. Стаття аналізує якість даних, отриманих через вебскрапінг, і висвітлює проблеми, що виникають через структурні зміни вебсторінок. Автори пропонують підходи до верифікації даних для підвищення їхньої надійності.

Дослідження [4] зосереджене на використанні вебскрапінгу для географічних даних. Автори висвітлюють методи отримання просторової інформації, що має практичне значення для картографії, екологічного аналізу та урбаністичних досліджень.

У [5] наведено підхід до автоматизованої обробки навчальних корпусів для великих моделей через нейронні вебскраперів. Автори акцентують увагу на важливості чистоти даних та розробляють методики для ефективного вилучення текстової інформації. Юридичні та етичні аспекти вебскрапінгу розглядаються у [6]. Висвітлено питання, пов'язані з конфіденційністю, дотриманням авторських прав та відповідністю міжнародним нормам. Автори акцентують увагу на необхідності розробки стандартів відповідального використання технології.

Динамічні сторінки є викликом для вебскрапінгу і в [7] дослідники пропонують інструменти для подолання цих проблем, зокрема з використанням технологій headless браузерів. Їхні результати демонструють як автоматизувати вилучення даних із складних структур.

На основі аналізу згаданих джерел, можна сформулювати такі висновки: розглянуті статті підкреслюють актуальність та широкі можливості застосування вебскрапінгу. Виклики, пов'язані з юридичними, етичними аспектами та якістю даних, стимулюють дослідників до вдосконалення інструментів і методів. Це підтверджує доцільність мети даної статті, спрямованої на узагальнення технічних аспектів та методів оптимізації вебскрапінгу для подальшого наукового і практичного використання.

Разом із тим, існують невирішені питання, які потребують подальших досліджень у сфері вебскрапінгу, основними з яких є: адаптивність та гнучкість (системи мають бути здатні адаптуватися до непередбачуваних змін у структурі та поведінці вебресурсів, що вимагає високого рівня інтелектуальної автономії), обчислювальна ефективність (обмежені обчислювальні ресурси потребують ефективних алгоритмів для швидкого оброблення великих обсягів даних і реагування на зміни у вебструктурах), комунікація та координація (ефективна взаємодія між різними компонентами системи скрапінгу, такими як парсери, браузери та сховища даних, є ключовою для синхронізації дій, збирання й аналізу інформації).

Отже, розробка ефективних методів автоматизованого збору та обробки даних із динамічних вебресурсів є актуальним науковим завданням, яке може бути успішно вирішене за рахунок застосування сучасних інструментів, таких як Selenium, та розробки адаптивних алгоритмів для роботи з динамічними й ускладненими вебструктурами.

Мета статті зводиться до розробки рекомендацій використання існуючих технологій вебскрапінгу для забезпечення ефективного збору інформації зі статичних та динамічних сайтів.

Виклад основного матеріалу дослідження

Вебскрапінг є одним із ключових інструментів для автоматизованого збору даних із вебсайтів, що широко застосовується у дослідницькій, аналітичній та комерційній діяльності. Основою ефективного вебскрапінгу є розуміння структури вебсайтів, яка може варіюватися від статичних сторінок із фіксованим вмістом (наприклад ukrinform.ua) до динамічних платформ, таких як

BBC News, що активно оновлюються за допомогою серверних запитів та клієнтського рендерингу. У даній роботі розглядаються особливості організації та функціонування структури як статичних, так і динамічних сайтів, з акцентом на їхні технічні характеристики, що впливають на процес збору даних. Такий аналіз дозволяє не лише зрозуміти основні виклики вебскрапінгу, але й забезпечити основу для подальшої розробки ефективних методів автоматизації аналізу даних.

Статичний сайт, такий як Ukrinform.ua*, працює за принципом відображення заздалегідь підготовленого HTML-контенту, який зберігається на сервері. Це означає, що кожна сторінка сайту існує як окремий файл з кодом, доступний для завантаження користувачем без необхідності динамічного генерування.

Ключові етапи роботи статичного сайту:

1. Запит користувача до вебсервера. Коли користувач вводить адресу “ukrinform.ua” у браузері, створюється HTTP-запит до сервера, де розміщено сайт.

2. Передача HTML-коду на клієнтську сторону. Сервер надсилає заздалегідь підготовлений файл HTML, який містить:

структуру сторінки (розташування елементів, текст, зображення тощо);

посилання на зовнішні CSS- та JavaScript-файли, необхідні для візуального оформлення та базової інтерактивності.

3. Рендеринг у браузері. Браузер користувача інтерпретує HTML-код, завантажує пов’язані ресурси (зображення, таблиці стилів, шрифти тощо) і відображає вміст на екрані.

4. Відображення контенту. Користувач бачить статичний контент, який не змінюється безпосередньо під час його взаємодії. Наприклад:

новинні статті, що додані вручну адміністраторами та оновлюються лише за їх завантаження на сервер;

зображення, розташовані в окремих папках на сервері.

5. Навігація між сторінками. За переходу на іншу сторінку, наприклад, з новин на контакти, завантажується новий статичний HTML-файл з сервера.

HTML контент статичного сайту є зручним для перегляду у браузері (наприклад у Chrome вибір контекстного меню «Переглянути джерело сторінки»), як це зображено на рисунку 1.

```
        <li class="publication-item">
            <div class="row mt-2 mb-0">
                <div class="col-6">
                <span class="image-container">
                    <a href="/rubric-culture/3931171-foto-genocidu-vid-golodomoru-do-10ricnoi-vijni-rf-v-ukraini.html" tit
                        <img src="https://static.ukrinform.com/photos/2024_11/thumb_files/300_200_1732574948-127.jpeg" alt
                    </a>
                    <span class="gradient"></span>
                </span>
                </div>
                <div class="col-6 publication-title ps-1 align-self-center">
                    <h3><a href="/rubric-culture/3931171-foto-genocidu-vid-golodomoru-do-10ricnoi-vijni-rf-v-ukraini.html"
                    <span class="date-time">26.11.2024 12:30</span>

                </div>
                <div class="col-12 d-flex pt-1 publication-text">
                    <p>
                        <a href="/rubric-culture/3931171-foto-genocidu-vid-golodomoru-do-10ricnoi-vijni-rf-v-ukraini.html"
</a>
                    </p>
                </div>
            </div>
        </li>
```

Рисунок 1 – Приклад контенту статичного сайту

На основі рисунку 1 можемо зробити висновок про зручність вебскрапінгу для статичних сайтів (на прикладі “ukrinform.ua”). Ключові аспекти даного висновку:

1. Структурованість HTML-коду. Код має чітко визначені класи для елементів:

<li class="publication-item"> – контейнер для кожної новинної публікації.

* * – містить заголовок новини.

** – дата та час публікації.

<img src="..." – зображення, яке супроводжує новину.

Це дає змогу легко ідентифікувати необхідні блоки для отримання заголовків, дат, посилань та зображень.

2. Зрозумілі селектори. Елементи мають зрозумілі назви класів:

клас *publication-item* може бути використаний для визначення контейнера кожної статті.

клас *date-time* дає можливість зручно витягувати дату і час без зайвих маніпуляцій.

Такі селектори знижують складність коду вебскрапінгу та забезпечують точність отриманих даних.

3. Статичний характер контенту. Контент статичного сайту не змінюється динамічно під час завантаження сторінки:

URL, вказані в атрибуті *href*, є фіксованими, що полегшує побудову списку посилань.

зображення завжди доступні через статичний шлях у атрибуті *src*.

Це означає, що дані можна отримати одразу без необхідності обробляти динамічні запити або JavaScript.

4. Мінімальна потреба у післяобробці. Кожна новина відображена у чіткій структурі:
заголовок доступний у тегу <*h3*>.
посилання на статтю в <*a href="..."*>.
зображення в <*img src="..."*>.

Це дає змогу безпосередньо використовувати отримані дані у структурованому форматі, наприклад, CSV або JSON. Отже, сайт ukrinform.ua є зручним для вебскрапінгу завдяки зрозумілій та стабільній HTML-структурі, чітко визначеним класам і статичному характеру контенту. Це значно знижує час і складність реалізації скрапінгових алгоритмів, що робить його ідеальним прикладом для навчання та застосування базових інструментів вебскрапінгу, таких як Python-бібліотека BeautifulSoup.

Для збору інформації зі статичного сайту можна використати код наведений на рисунку 2.

```
url = 'https://www.ukrinform.ua'
soup = BeautifulSoup(requests.get(url).content, 'html.parser')
news_items = soup.find_all('li', class_='publication-item')

with open('news_data.csv', 'w', encoding='utf-8', newline='') as file:
    writer = csv.writer(file)
    writer.writerow(['title', 'link', 'date_time', 'image_url'])
    for item in news_items:
        try:
            title = item.find('h3').get_text(strip=True)
            link = url + item.find('a')['href']
            date_time = item.find('span', class_='date-time').get_text(strip=True)
            image_url = item.find('img')['src']
            writer.writerow([title, link, date_time, image_url])
        except:
            continue
```

Рисунок 2 – Приклад коду для веб-скрапінгу статичного сайту

Відповідно до рисунку вебскрапінг статичного сайту відбувається наступним чином:

1. Отримання HTML-контенту. Код надсилає HTTP-запит на вебсайт за вказаним URL і отримує HTML-код сторінки.

2. Парсинг HTML. Використовується обробник HTML для перетворення отриманого HTML-коду у структуру, з якою зручно працювати.

3. Пошук новин. Код шукає всі HTML-елементи на сторінці, які відповідають заданому класу `*publication-item*`. Це дає змогу виділити блоки новин.

4. Обробка даних. Для кожного блоку новин знаходяться:

заголовок новини (`*h3*`) і витягується текст;

посилання на новину (`*a*`) і додається базовий URL для створення повного посилання;

дата і час публікації (`*span*` із класом `*date-time*`) і витягується текст;

URL зображення (`*img*`) і витягується посилання на зображення.

5. Запис у файл. Отримані дані зберігаються у CSV-файл, де кожен рядок містить заголовок, посилання, дату та час, а також URL зображення.

На відміну від статичного – динамічний вебсайт створює вміст у реальному часі залежно від запиту користувача або зовнішніх даних. BBC News – це приклад динамічного сайту, що використовує складну архітектуру для надання персоналізованого та оновлюваного контенту.

Основні характеристики:

динамічний контент – новини та статті оновлюються автоматично залежно від часу, теми або регіону;

використання серверів і баз даних – контент генерується з баз даних та серверних запитів, наприклад, через API;

складна взаємодія – динамічні сайти включають інтерактивні елементи, такі як пошук, фільтри новин, відео, слайдери тощо;

динамічний рендеринг – сайт може використовувати JavaScript для завантаження частин контенту, що відбувається після рендерингу HTML.

Приклад роботи BBC News:

основна сторінка (HTML) – під час запиту користувача сервер надає основний HTML-файл із базовою структурою;

динамічний контент (JavaScript) – скрипти завантажують останні новини через API та інтегрують їх у DOM;

взаємодія – пошук, вибір категорій новин (спорт, політика) чи зміна регіону викликають нові запити до API для оновлення вмісту без перезавантаження сторінки.

Приклад HTML контенту такого сайту наведено на рисунку 3.

```
<!DOCTYPE html>
<html>
<head>
    <title>BBC News</title>
</head>
<body>
    <h1>BBC News</h1>
    <div id="top-stories">
        <!-- Контент оновлюється через JavaScript -->
    </div>
    <script>
        fetch('https://newsapi.bbc.co.uk/topstories')
            .then(response => response.json())
            .then(data => {
                const container = document.getElementById('top-stories');
                data.articles.forEach(article => {
                    const div = document.createElement('div');
                    div.innerHTML = `<h2>${article.title}</h2><p>${article.description}<
                    container.appendChild(div);
                });
            });
    </script>
</body>
</html>
```

Рисунок 3 – Приклад контенту динамічного сайту

Скрапінг такої сторінки має кілька складностей через її динамічний характер:

1. Динамічне завантаження контенту:

контент у блоці `#top-stories` завантажується через JavaScript після початкового рендерингу сторінки. Тобто, при отриманні HTML через HTTP-запит цей блок буде порожнім;

традиційні методи скрапінгу, які аналізують лише статичний HTML, не зможуть отримати ці дані.

2. Необхідність обробки JavaScript для отримання даних, які завантажуються через JavaScript, потрібно використовувати інструменти, що підтримують рендеринг JavaScript, наприклад:

Selenium (для емуляції поведінки браузера);

Playwright або Puppeteer (для роботи з сучасними вебсторінками).

3. API-запити:

сторінка використовує API для отримання списку статей у форматі JSON. Для доступу до цього API може знадобитися аутентифікація або спеціальні ключі;

якщо API доступний без обмежень, можна напряму надсилати запити до нього, що значно спрощує процес порівняно з рендерингом JavaScript.

4. Антибот-захист. Сучасні сайти часто додають захист, наприклад:

перевірки *CAPTCHA*;

блокування частих запитів із одного IP;

вимоги до заголовків запитів (наприклад, `*User-Agent*`).

5. Необхідність виконання JavaScript. У випадку, якщо API-запит не можна використовувати безпосередньо, потрібно виконувати JavaScript-код сторінки, щоб дочекатися завантаження контенту, а це значно ускладнює процес. У такому випадку застосовують такі підходи:

1. Якщо доступ до API дозволений, можливо надсилати запити до API напряму та обробляти JSON-відповідь.

2. Якщо API недоступне, використовуються інструменти рендерингу JavaScript, наприклад, Selenium чи Puppeteer, щоб дочекатися завантаження контенту.

Приклад використання бібліотеки наведено на рисунку 4. Цей код використовує бібліотеку Selenium для завантаження динамічного контенту сторінки, зокрема JavaScript-генерованих елементів. Після завантаження сторінки її HTML-код зберігається в змінній content, яка передається до BeautifulSoup для подальшого аналізу. За допомогою CSS-селекторів знаходяться потрібні елементи (заголовки та описи статей), після чого їхній текст виводиться на екран. Наприкінці браузер закривається. Отже, процес збору інформації з різних сайтів має свої унікальні особливості, які залежать від типу сайту (статичний чи динамічний) і способу генерації контенту.

```
from selenium import webdriver
from selenium.webdriver.chrome.service import Service
from webdriver_manager.chrome import ChromeDriverManager
from bs4 import BeautifulSoup

driver = webdriver.Chrome(service=Service(ChromeDriverManager().install()))
driver.get("https://example.com")
content = driver.page_source
soup = BeautifulSoup(content, "html.parser")
articles = soup.select("#top-stories div")
for article in articles:
    title = article.find("h2").text
    description = article.find("p").text
    print(f"Title: {title}\nDescription: {description}\n")
driver.quit()
```

Рисунок 4 – Приклад коду веб-скрапінгу динамічного сайту

Статичні сайти забезпечують простий доступ до контенту через стандартний HTML-код. Вебскрапінг таких сайтів виконується швидко, оскільки HTML-структура передбачувана, а JavaScript не використовується для динамічного оновлення. Бібліотеки, такі як BeautifulSoup, достатні для отримання необхідних даних.

Динамічні сайти (наприклад, BBC News) оновлюють контент за допомогою JavaScript, що ускладнює процес скрапінгу. Для таких сайтів часто використовують інструменти автоматизації браузера, як Selenium, що дають змогу завантажити сторінку повністю, включно з динамічним контентом. Однак, це потребує більших ресурсів і може бути повільнішим, ніж робота зі статичними сайтами.

Правові та етичні аспекти важливі незалежно від типу сайту. Динамічні сайти часто захищають свій контент через CAPTCHA, обмеження запитів чи блокування IP-адрес, що потребує додаткових зусиль для обходу таких механізмів. У випадку статичних сайтів ці обмеження зустрічаються рідше.

Загалом, вибір методу збору даних залежить від складності сайту, доступності даних та технічних вимог до швидкості й обробки. Динамічні сайти вимагають більших витрат часу та ресурсів, але сучасні інструменти, як Selenium у комбінації з BeautifulSoup, значно полегшують процес.

У цьому дослідженні для порівняльного аналізу ефективності двох підходів до вебскрапінгу – використання бібліотек BeautifulSoup (BS) та Selenium (S) – було розроблено методологію, що передбачає кілька етапів практичного застосування та оцінки. Головною метою дослідження є визначення, який інструмент є ефективнішим для збору даних з різних типів сайтів, зокрема статичних і динамічних.

На першому етапі дослідження було визначено критерії оцінки, що містять:

1. Швидкість виконання завдань.
2. Обчислювальні ресурси, необхідні для роботи.
3. Зручність використання та налаштування.
4. Гнучкість і здатність обробляти складні сторінки з динамічним контентом.

Для порівняння було відібрано два типи сайтів:

статичний сайт із передбачуваною структурою HTML, який не містить динамічно оновлюваного контенту ("ukrinform.ua");

динамічний сайт, який використовує JavaScript для завантаження контенту в режимі реального часу ("bbc.com").

На другому етапі кожен інструмент було застосовано для обробки цих сайтів. Для BeautifulSoup використовувалася його можливість швидкого парсингу HTML-коду після отримання сторінки за допомогою бібліотеки `requests`. Selenium, у свою чергу, застосовувався для динамічного завантаження сторінки, після чого її HTML-код передавався до BeautifulSoup для подальшого парсингу.

На третьому етапі проводився аналіз зібраних даних із кожного типу сайту. Було оцінено швидкість виконання завдання, ресурси, витрачені на виконання, і зручність роботи для розробника. Особливу увагу приділено роботі з динамічними сайтами, де BeautifulSoup безпосередньо виявився менш ефективним через відсутність підтримки JavaScript. Selenium, навпаки, забезпечив точний доступ до динамічного контенту, хоч і потребував більших обчислювальних ресурсів.

Водночас були використані такі математичні залежності:

Оцінка часу виконання завдання. Час, необхідний для отримання та обробки даних із сайту, визначається формулою:

$$T_{\text{total}} = T_{\text{load}} + T_{\text{parse}}\ , \quad (1)$$

де T_{total} – загальний час виконання завдання;

T_{load} – час завантаження сторінки;

T_{parse} – час парсингу HTML-коду.

Оцінка обчислювальних ресурсів Витрати на обчислювальні ресурси, зокрема використання оперативної пам'яті (M) і процесорного часу (C), оцінюються як:

$$R_{\text{total}} = \alpha M + \beta C, \quad (2)$$

де M – середній обсяг пам'яті, використаної під час роботи;

C – загальний час роботи процесора. під час виконання скрапінгу;

α,β – вагові коефіцієнти для приведення M і C до єдиної метрики (наприклад, до вартісного вираження в умовних одиницях або вартісного еквівалента).

Кількість успішно зібраних елементів Визначення ефективності інструмента для збору даних виконується за допомогою коефіцієнта точності:

$$A = \frac{N_{\text{collected}}}{N_{\text{total}}}, \quad (3)$$

де $N_{\text{collected}}$ – кількість успішно зібраних елементів (заголовків, текстів тощо);

N_{total} – загальна кількість елементів, доступних для збору.

Швидкість обробки сторінки Для оцінки швидкості обробки сторінки використовується середній час обробки одного елемента:

$$S = \frac{T_{\text{total}}}{N_{\text{collected}}}, \quad (4)$$

де S – швидкість обробки одного елемента (в секундах);

T_{total} – загальний час роботи;

$N_{\text{collected}}$ - кількість зібраних елементів.

Оцінка ефективності роботи з динамічним контентом Для динамічних сайтів вводиться коефіцієнт адаптивності:

$$E_{\text{dyn}} = \frac{T_{\text{selenium}}}{T_{\text{bs}}+T_{\text{js}}}, \quad (5)$$

Де E_{dyn} - коефіцієнт ефективності роботи з динамічним контентом;

T_{selenium} – час обробки сторінки Selenium;

T_{bs} – час обробки статичного HTML BeautifulSoup;

T_{js} – час виконання JavaScript на стороні клієнта.

На основі запропонованих залежностей (1–5) проведено експериментальний аналіз ефективності бібліотек, результати якого наведено у таблицях 1, 2.

Таблиця 1

Результати аналізу ефективності веб-скрапінгу статичного сайту

Показник	BeautifulSoap	Selenium
Загальний час виконання (с)	12	18
Використання пам'яті (МБ)	120	140
Час роботи CPU (с)	3	4
Успішність (%)	98	96
Швидкість обробки (с/елемент)	0,24	0.30

Таблиця 2

Результати аналізу ефективності веб-скрапінгу динамічного сайту

Показник	Selenium	Selenium + BuetifulSoap
Загальний час виконання (с)	22	24
Використання пам'яті (МБ)	170	180
Час роботи CPU (с)	6	8
Успішність (%)	95	95
Швидкість обробки (с/елемент)	0,37	0.40

На основі результатів, можливо зробити висновок, що статичні сайти BeautifulSoup демонструють високу ефективність завдяки низьким витратам ресурсів і швидкій обробці HTML-структури. Selenium, хоча і менш ефективний у цьому контексті, дає змогу взаємодіяти з базовими елементами сторінок, але його використання для статичних сайтів є надлишковим. Для динамічних сайтів – обидва підходи мають свої переваги та недоліки, залежно від цілей проєкту. Якщо важлива швидкість обробки та мінімізація ресурсів, Selenium є кращим вибором. У випадках, коли потрібна висока точність роботи з даними та складними структурами сторінок, доцільно застосовувати комбінацію Selenium та BeautifulSoup.

Результати дослідження дають можливість сформувати рекомендації щодо вибору інструменту для скрапінгу залежно від особливостей сайту та поставлених завдань. Для

ефективного вебскрапінгу необхідно враховувати тип сайту та специфіку завдань. Для статичних сайтів найкраще підходить BeautifulSoup, який забезпечує високу швидкість і низькі витрати ресурсів, працюючи виключно з HTML-контентом. У випадку з динамічними сайтами, де контент завантажується через JavaScript, Selenium є оптимальним вибором, оскільки дає змогу взаємодіяти з браузером і отримувати необхідні дані.

У складних завданнях рекомендується комбінувати Selenium та BeautifulSoup – перший забезпечує доступ до динамічних елементів, а другий оптимізує обробку HTML-структури. Для підвищення продуктивності важливо налаштувати частоту запитів та уникати блокувань з боку сайту, застосовуючи затримки між запитами чи паралельну обробку даних.

Етичність скрапінгу передбачає дотримання правил, визначених у файлі robots.txt, та уникнення збору конфіденційної інформації чи даних, що можуть порушувати права власників сайту. Постійний моніторинг змін у структурі сайту допоможе уникнути помилок у зборі даних, а збереження інформації у форматах, таких як CSV чи JSON, забезпечить зручність подальшої обробки.

Під час виконання інтенсивних завдань доцільно використовувати хмарні платформи або виділені сервери, щоб зберігати продуктивність локальних систем. Отже, дотримання цих підходів дасть змогу ефективно вирішувати задачі вебскрапінгу незалежно від складності сайту.

Висновки й перспективи подальших досліджень

На основі проведених досліджень можна зробити висновок, що мету статті – розроблення рекомендацій використання існуючих технологій вебскрапінгу для забезпечення ефективного збору інформації зі статичних та динамічних сайтів – була досягнуто. Експериментальне порівняння використання бібліотек Selenium і BeautifulSoup окремо та у поєднанні демонструє їхні переваги та обмеження у різних сценаріях. Результати свідчать, що для статичних сайтів BeautifulSoup забезпечується найвища швидкість і продуктивність, тоді як Selenium виявився незамінним для роботи з динамічними сайтами, де контент завантажується за допомогою JavaScript.

Науковою новизною дослідження є розроблення математичних залежностей порівняння продуктивності окремих бібліотек та їх комбінованого використання для оптимізації процесів збору даних у різних умовах. Теоретичною значущістю є розширення розуміння ефективності бібліотек для вирішення задач вебскрапінгу, а практичною – формування рекомендацій для розробників та дослідників щодо вибору інструментів залежно від специфіки завдань.

Подальші дослідження можуть бути спрямовані на розроблення нових підходів до роботи з динамічними сайтами, які б забезпечували ще більшу обчислювальну ефективність. Важливим напрямом є інтеграція бібліотек вебскрапінгу з методами машинного навчання для автоматичного визначення структури сайтів та адаптації до їх змін.

Список бібліографічних посилань

1. Huang W., Gu Z., Peng C., Li Z., Liang J., Xiao Y., Wen L., Chen Z. A Progressive Understanding Web Agent for Web Scraper Generation. AutoScraper. 2024. URL: https://arxiv.org/abs/2404.12753?utm_source=chatgpt.com (Accessed: 25 November 2024). **2. Ahluwalia A., Wani S.** Leveraging Large Language Models for Web Scraping. 2024. URL: https://arxiv.org/abs/2406.08246?utm_source= chatgpt.com (Accessed: 25 November 2024). **3. Foerderer J.** Should We Trust Web-Scraped Data? *arXiv preprint arXiv:2308.02231*. 2023. URL: https://arxiv.org/abs/2308.02231? utm_source =chatgpt.com (Accessed: 25 November 2024). **4. Brenning A., Henn S.** Web Scraping: A Promising Tool for Geographic Data Acquisition. 2023. URL: https://arxiv.org/abs/2305.19893 ?utm_source = chatgpt.com (Accessed: 25 November 2024). **5. Xu Z., Liu Z., Yan Y., Liu Z., Yu G., Xiong C.** Cleaner Pretraining Corpus Curation with Neural Web Scraping. 2024. URL: https://arxiv.org/abs/2402.14652?utm_source= chatgpt.com (Accessed: 25 November 2024). **6. Brown M., Gruen A., Maldoff G., Messing S., Sanderson Z., Zimmer M.** Web Scraping for Research: Legal, Ethical, Institutional, and Scientific Considerations. 2024. URL: https://arxiv.org/abs/2410.23432?utm_source=chatgpt.com (Accessed: 25 November 2024). **7. Zohaib M.** A Responsive Framework for Research Portals Data using Semantic Web Technology. 2023. URL: https://arxiv.org/abs/2306. 11642?utm_source=chatgpt.com (Accessed: 25 November 2024).

ANALYSIS OF TECHNOLOGICAL ASPECTS OF IMPLEMENTING WEB SCRAPING OF STATIC AND DYNAMIC SITES

Mykolaichuk Roman ***(Doctor of Technical Sciences, Associate Professor)*** [1]
Starynskyi Ivan ***(Candidate of Technical Sciences)*** [1]
Mykolaichuk Vira ***(Philosophy Doctor)*** [2]

[1] ***The National Defence University of Ukraine, Kyiv, Ukraine***
[2] ***Taras Shevchenko National University of Kyiv, Ukraine***

The article is devoted to the development of recommendations for the effective use of web scraping technologies to ensure efficient information collection from static and dynamic websites. In the context of advancing data collection methodologies, particularly when dealing with dynamic websites that rely heavily on JavaScript for rendering content, a significant challenge emerges. This challenge involves developing adaptable and robust approaches capable of handling

complex website structures and ensuring accurate and efficient extraction of relevant data. Traditional web scraping techniques often lack the flexibility and precision required to address these issues effectively.

Formulation of the problem in general. *The purpose of the article is to develop recommendations for the application of existing web scraping technologies to ensure efficient and accurate information collection from both static and dynamic websites.*

Analysis of recent researches and publications *In the course of the research, recent advancements in web scraping technologies were analyzed, focusing on tools and methodologies tailored for static and dynamic websites. Studies on the application of libraries like BeautifulSoup and Selenium were particularly highlighted, revealing their effectiveness in extracting structured and unstructured data. BeautifulSoup demonstrated high efficiency in parsing static HTML content, while Selenium proved indispensable for handling JavaScript-rendered dynamic content. Furthermore, research has emphasized the integration of these tools to overcome limitations associated with individual approaches, ensuring adaptability and scalability for diverse web environments. These findings underscore the importance of leveraging hybrid methodologies for enhanced efficiency and accuracy in information collection..*

Presenting the main material *The obtained results include the development of recommendations and methodologies for efficient web scraping of both static and dynamic websites, leveraging the capabilities of BeautifulSoup and Selenium. For static sites, the methodology focuses on direct HTML parsing, ensuring fast and accurate extraction of structured data. For dynamic sites, the integration of Selenium allows for effective interaction with JavaScript-rendered elements, providing comprehensive data collection. An experimental comparison was conducted to evaluate the performance of both tools independently and in combination. Static websites were processed with BeautifulSoup, showcasing high speed and minimal computational overhead. Dynamic websites were analyzed using Selenium, highlighting its ability to handle interactive content. The combined approach of Selenium with BeautifulSoup demonstrated superior performance in scenarios requiring both dynamic content handling and efficient data extraction. These experiments confirmed the adaptability and reliability of the proposed methodologies across diverse web environments.*

Elements of scientific novelty. *The scientific novelty of the research lies in the development of an integrated approach to web scraping that combines advanced tools like Selenium and BeautifulSoup for efficient data extraction from both static and dynamic websites. This approach introduces novel strategies for handling the challenges posed by JavaScript-rendered content and complex website structures, ensuring adaptability and scalability.*

Practical significance of the article *lies in enhancing the understanding of the capabilities of web scraping technologies for efficient data collection from both static and dynamic websites. This significance extends to practical applications in various fields, such as journalism, business intelligence, and education, where streamlined data extraction processes can improve decision-making and operational efficiency.*

Conclusion and the perspectives of future researches *Overall, the paper highlights critical aspects of utilizing web scraping techniques for efficient data extraction from static and dynamic websites. It provides practical recommendations for selecting appropriate tools and methods to address the challenges associated with diverse web environments. These insights pave the way for future research aimed at enhancing web scraping technologies, ensuring greater adaptability, efficiency, and accuracy in data collection processes across various domains.*

Keywords*: information technology, data collection automation, performance evaluation, web design, websites, modeling, web scraping, information processing algorithms.*

References

1. Huang, W., Gu, Z., Peng, C., Li, Z., Liang, J., Xiao, Y., Wen, L., & Chen, Z., (2024). AutoScraper: A Progressive Understanding Web Agent for Web Scraper Generation [online]. Available at: https://arxiv.org/abs/2404.12753?utm_source=chatgpt.com [Accessed: 25 November 2024]. **2. Ahluwalia, A. & Wani, S., (2024).** Leveraging Large Language Models for Web Scraping [online]. Available at: https://arxiv.org/abs/2406.08246?utm_source = chatgpt.com [Accessed: 25 November 2024]. **3. Foerderer, J., (2023).** Should We Trust Web-Scraped Data? [online]. *arXiv preprint arXiv:2308.02231*. Available at: https://arxiv.org/abs/2308.02231?utm_source =chatgpt.com [Accessed: 25 November 2024]. **4. Brenning, A. & Henn, S., (2023).** Web Scraping: A Promising Tool for Geographic Data Acquisition [online]. Available at: https://arxiv.org/abs/2305.19893 ?utm_source = chatgpt.com [Accessed: 25 November 2024]. **5. Xu, Z., Liu, Z., Yan, Y., Liu, Z., Yu, G., & Xiong, C., (2024).** Cleaner Pretraining Corpus Curation with Neural Web Scraping [online]. Available at: https://arxiv.org/abs/2402.14652?utm_source=chatgpt.com [Accessed: 25 November 2024]. **6. Brown, M., Gruen, A., Maldoff, G., Messing, S., Sanderson, Z., & Zimmer, M., (2024).** Web Scraping for Research: Legal, Ethical, Institutional, and Scientific Considerations [онлайн]. Available at: https://arxiv.org/abs/2410.23432?utm_source =chatgpt.com [Accessed: 25 November 2024]. **7. Zohaib, M., (2023).** A Responsive Framework for Research Portals Data using Semantic Web Technology. [online]. Available at: https://arxiv.org/abs/2306.11642?utm_source=chatgpt.com [Accessed: 25 November 2024].