JIT Mahasarakham University, Thailand
http://jit.it.msu.ac.th

Journal of Applied Informatics and Technology, 7(2), 346-362.

Research Article

Web Scraping-based System for E-commerce Price Comparison and Similar Product Segmentation

Pongsin Jankaew[1], Wachirawut Thamviset[1,*]

[1] Department of Computer Science and Information Technology, College of Computing, Khon Kaen University, Khon Kaen 40002, Thailand

* Corresponding author: Wachirawut Thamviset, twachi@kku.ac.th

Received:
4 November 2023
Revised:
6 April 2024
Accepted:
13 April 2024

Keywords:

Agglomerative Clustering, E-commerce, Product Identification, Web Scraping

Abstract: With the booming growth of e-commerce, finding the best deals amid a multitude of online shopping websites has become a challenge. Consumers often spend a considerable amount of time manually sifting and comparing data, leading to uncertainty in decision-making. To address this issue, our research proposes a system that utilizes web scraping techniques to identify top deals from multiple e-commerce sites. We have developed Python-based web scraping scripts and incorporated a configuration file for customization, enabling users to extract product data from diverse websites. The system scrapes data and displays result each time the user enters a query, ensuring that the scraped data is up to date. Furthermore, our system enhances the user experience by incorporating product model datasets for product identification, enabling specific searches based on product specifications, and offering recommendations for similar product models. Finally, in cases where products remain unidentified, we introduce a feature for grouping similar products through an agglomerative clustering method. This method utilizes product name and image features extracted by TF-IDF and Convolutional Neural Networks (CNN), allowing for price comparisons among similar products and enhancing the overall shopping experience. Preliminary evaluations show that our system successfully extracts data from target websites with proper customizations. The evaluations of similar

product clustering demonstrate that using a combined feature of product names and images significantly improves clustering performance, surpassing the use of product names or images alone, with a 9 percent increase and 18 percent increase, respectively.

1. Introduction

In the current digital era, the internet has become an integral part of our daily lives. Its widespread accessibility has enabled businesses and organizations worldwide to leverage its power to improve work efficiency and achieve better results. E-commerce, one of the most significant benefits of the internet, has witnessed exponential growth in recent years. Online shopping platforms have become increasingly popular, providing people with a convenient and easy way to access goods and services from the comfort of their homes (Kemp, 2022). Despite the convenience offered by online shopping, finding the best deals can be a time-consuming and frustrating process. Consumers often have to navigate multiple websites to compare prices, which can be inconvenient and inefficient (Asawa *et al*., 2022; Mehak *et al*., 2019). To address this issue, price comparison websites have emerged, allowing users to compare prices across various e-commerce platforms. In Thailand, there are existing websites perform price comparisons, including iPrice and Priceza. iPrice compares products within leading e-commerce websites like Lazada, Shopee, and Central Online, whereas Priceza stores product data in its database and compares categories like mobiles, computers, electric appliances, household goods, and fashion items. However, these websites sometimes encounter problems such as mismatched product prices on the price comparison website and the original website, or missing product information that appears on the price comparison website but not on the original website. The primary reason for these discrepancies may be the lack of regular updates on the product data in these websites, leading to inaccurate product comparisons.

In this paper, we propose a web scraping-based system designed for comparing products across multiple e-commerce websites. Our system offers configurability, allowing users to specify extraction rules for each selected target website. It operates by extracting data from the target website in real-time as the user enters a search query, ensuring that the displayed product information is up to date. This design addresses issues present in prior price comparison websites, particularly those related to mismatched or missing product data. To enhance the user experience, we are developing a feature that recommends similar products within search results. This system utilizes product names and images as features and employs an Agglomerative Clustering method for grouping similar products. Moreover, our system incorporates a product model identification feature, enabling the recognition and grouping of products sharing the same model. This feature can be used with any product dataset as a reference. The goal is to enable consumers to make informed

decisions when shopping online, saving both time and money while elevating their overall online shopping experience.

2. Related Work

2.1 Web Scraping Approaches

Web scraping stands as a prevalent data mining technology employed for extracting unstructured data from diverse online sources. The acquired data can subsequently undergo restructuring and transformation into a structured format. The process of web scraping can be executed through manual techniques or automated programs. Presently, websites are commonly categorized into two principal formats: static and dynamic. Each of these web formats necessitates distinct data extraction methodologies (Lan *et al*., 2021).

2.1.1 Static Webpages Scraping

Static webpages have a fixed data structure and display the same content stored on the web server side when HTML data is loaded on the client's web browser. Retrieving HTML data from a static page is typically done by sending an HTTP request. Subsequently, an HTML Parser is employed to extract the pertinent information from HTML documents, thereby constructing a hierarchical tree structure known as the Document Object Model (DOM Tree). Within this context, users specify the HTML tags housing the desired information to facilitate the data extraction process (Lan *et al*., 2021).

2.1.2 Dynamic Webpages Scraping

Web scraping of dynamic webpages poses challenges due to the dynamic loading of data using JavaScript. Dynamic pages cannot be accessed in the requested HTML of the target webpage by simply sending an HTTP request. To overcome this issue, the web driver is used, which is a web automation framework which allows user to execute test across various browsers (Gheorghe *et al*., 2018). Web scraping methodologies incorporate the use of a web driver, enabling the automation of actions across different web browsers. Employing a web driver to interact with a website enables the system to execute JavaScript on the target webpage, thereby ensuring the retrieval of the necessary information. Subsequently, HTML Parser are employed to transform the HTML documents retrieved from the webpage into a structured tree format, facilitating the extraction of the desired information (Lan *et al*., 2021).

The researcher conducted a review of pertinent literature concerning data extraction methodologies applied to diverse e-commerce websites. A study by Ambre *et al*. (2019) outlined the development of a web application for comparing product prices across multiple websites, utilizing static web scraping methodologies with the Python package. This involved incorporating the Request library for sending HTTP requests and the Beautiful Soup 4 library for HTML parsing. In a separate study, Alam *et al*. (2020) introduced a price comparison application tailored for

Bangladeshi e-commerce platforms, utilizing static webpage scraping methodologies via the Scrapy framework for website access and automated data extraction. In contrast, Asawa *et al*. (2022) engineered a price comparison web application involved the implementation of dynamic web scraping techniques, utilizing Selenium as a web driver, available in various languages including Python, and Beautiful Soup 4 as an HTML parser. Additionally, Mehak *et al*. (2019) developed a web application for product price comparison, employing a hybrid approach that encompassed both HTTP request transmission using Request library and web driver techniques using Selenium to access various websites, with Beautiful Soup 4 serving as the HTML parser for this application.

The mentioned studies employed diverse methods for data extraction, with some focusing on static web data extraction using tools like the Request library or Scrapy framework, while others utilized the Selenium library for websites with dynamic content. However, the latter approach often incurred longer processing times compared to tools designed for static websites. Notably, all the mentioned research fixed the target websites for their web scraping systems, limiting flexibility when users wished to search for products on different websites. Recognizing this limitation, the researcher identified an opportunity to enhance the efficiency of web data extraction by developing a system capable of customization for scraping any desired target website. This system is designed to select extraction tools based on the nature of the target website, providing increased flexibility for users.

2.2 Product Matching Approaches

The process of product matching is accomplished through the application of machine learning methodologies. Various studies have addressed the Product Matching process using distinct approaches. Li *et al*. (2020) developed a method for finding products of the same type, employing an artificial neural network model that considers two types of product descriptions: title and attribute. It consists of a model for processing product title data and a model for processing product attributes data together. Addagarla & Amalanathan (2020) have developed an image-based search for similar product recommendation system by processing product image data by performing Principal Component Analysis (PCA) using Singular Value Decomposition (SVD) method. Next, the data will be grouped using the K-means++ method to group similar product data. The input image is then measured for Manhattan distance to the target clusters set, fetching the top-N similar products with low distance measures. Kannan (2021) developed a system to classify whether products are identical, utilizing product name and image information from e-commerce websites. The model is a combination of TF-IDF for product name data and ResNet-18, a Convolutional Neural Networks (CNN) architecture, for product image data.

This research aims to develop an efficient process for grouping similar products

without relying on high processing power, ensuring the prompt presentation of grouping results to users. Despite the effective outcomes produced by the artificial neural network model, its resurce-intensive nature led the researchers to explore alternatives. While the K-means++ clustering method is robust, it requires prior determination of the number of clusters, posing a challenge when dealing with products of an unknown grouping structure. Additionally, the algorithm in the last-mentioned research focuses on classifying whether a product is similar to others, whereas our research aims to group similar products. Consequently, the researchers opted for the TF-IDF method to process product names and a pre-trained CNN model for processing product images, extracting features with less processing power. These features are then employed in an Agglomerative Clustering approach, a hierarchical clustering algorithm. Unlike partitioning methods like K-means++, agglomerative clustering builds a hierarchy of clusters without predefining the number of clusters.

3. Materials and Methods

In this research, a web application has been designed as the system's form. The system was developed using the Django framework, implemented in the Python programming language. Django allows developers to create web applications quickly by providing many pre-built components and templates, thus reducing development time and effort. An overview of the system architecture is presented in Figure 1. The proposed system comprises three main layers:

- Web Extraction Layer: This layer focuses on extracting product data from the target websites, initiated by receiving search queries from the user interface.
- Data Categorization Layer: Following the extraction of product data in the Web Extraction Layer, this layer categorizes the data. It consists of two modules: one for product type or model identification and another for clustering unidentified products.
- Application Layer: This layer provides a user interface, allowing users to enter search queries for their desired products, which are then sent to the Web Extraction Layer. Additionally, it displays the product data generated by the preceding layers.

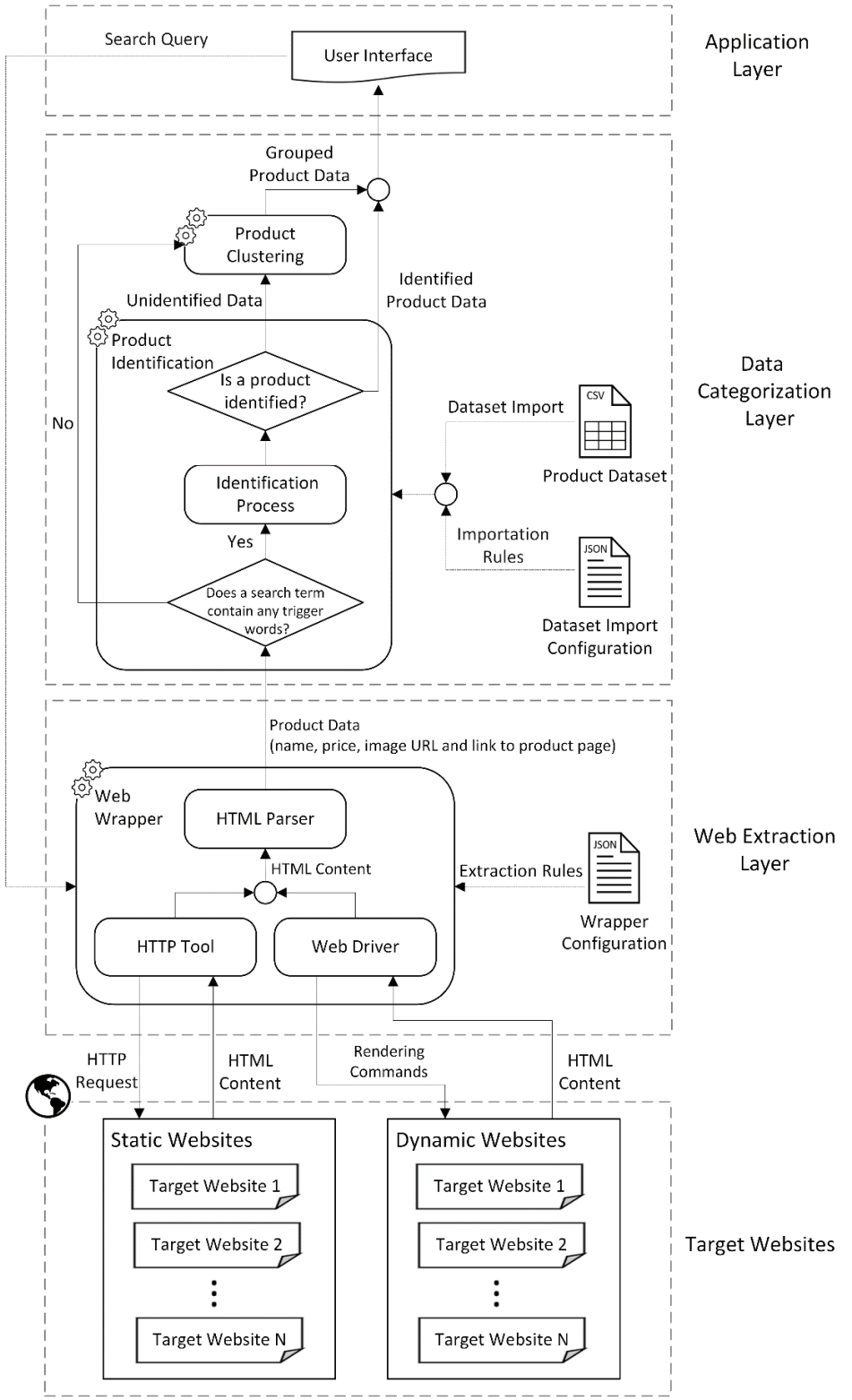

Figure 1. System Architecture

3.1 Implementation of the Web Extraction System

The web extraction process takes place within the Web Extraction Layer of this system. This implementation involves configuring extraction rules in the wrapper configuration, providing directives to the web wrapper for the accurate extraction of product data from designated target websites. When a user enters a search query in the user interface, the web wrapper extracts data from the target website based on the search query and extraction rules. This approach ensures the freshness of the extracted data, addressing issues prevalent in existing price comparison websites, such as missing or mismatched information. Subsequently, the extracted data is sent to the data categorization layer for further processing. The components involved in web extraction are detailed as follows:

3.1.1 Wrapper Configuration

Wrapper configuration allows users to customize the web wrapper in the Web Extraction Layer. It provides rules for web accessibility tools, offering two options: Requests, for making HTTP requests, and Selenium WebDriver, for automating web browsers. In the initial step, users are tasked with determining the dynamism of the target webpage by disabling JavaScript. If the desired data persists, the Requests option is used for its faster processing time; otherwise, Selenium is employed for dynamic pages. Subsequently, users need to inspect the website's HTML structure to identify the HTML elements containing required product data, such as titles, prices, images, and URLs. This information serves as a prerequisite for the configuration of a dedicated file in subsequent steps.

The wrapper configuration file template, as depicted in Figure 2, is formatted in JSON. It requires users to define values for each key. For elements requiring HTML values, users specify the tag name, class attribute, or ID of the HTML elements identified during the earlier inspection. For instance, if the product name is in " ," it's defined in the "title" key as "div[class='product-name']". Users can configure additional settings, such as the number of pages to scrape and the access method (sequential or parallel). An example of the wrapper configuration for each target website is provided in Figure 3. Furthermore, users can customize web access tools, including 'Requests' and 'Selenium.', with the flexibility to customize HTTP headers for Requests and browser actions, waiting times, and scrolling preferences for Selenium. Users can tailor web accessibility tools for each site, enabling specific actions for different sites, or use default settings if not specified.

3.1.2 Web Wrapper

Web Wrapper is responsible for automatically extract product information from the target website. To achieve successful web data extraction, the system follows a defined process. When a user enters a search term in the user interface, the system reads

```
"configs": [
    {
        "method": "[Specify the web access method: 'requests' or 'selenium']",
        "base_url": "[Enter the domain name of the target website]",
        "search_url": "[Specify the URL for embedding search queries]",
        "num_of_pages": [Number of search results pages on the target website to be scraped.],
        "multipage_processing_method":"[Specify method for accessing multiple pages: 'sequential' or 'parallel']",
        "items": "[Specify the HTML element containing the block of product data]",
        "title": "[Specify the HTML element containing the product name]",
        "price": "[Specify the HTML element containing the product price]",
        "link": "[Specify the HTML element containing the product link; leave blank if not applicable]",
        "image_url": "[Specify the HTML element containing the product image URL]",
        "source": "[User-defined name for the target website to be displayed in the application result page]"
    },
    {[Configurations of other target websites]}
],
"configs_request": {
    "headers": {"[Specify browser HTTP headers]"}
},
"configs_selenium": {
    "headless": "[Specify headless browser mode: yes ('y') or no ('n')]",
    "wait_time": [Specify wait time before parsing HTML content (in seconds)],
    "zoom":"[Apply webpage zoom: yes ('y') or no ('n')]",
    "zoom_percentage": [Specify webpage zoom percentage],
    "scrolling": "[Apply webpage scrolling: yes ('y') or no ('n')]",
    "scrolling_setting": {
        "scrolling_step": [Specify webpage scrolling step (in pixels)],
        "scrolling_pause_time": [Specify webpage scrolling pause time in each step (in seconds)]
        "max_step": [Specify the maximum number of steps for page scrolling, used if the page has infinite
```

Figure 2. Template of Wrapper Configuration File

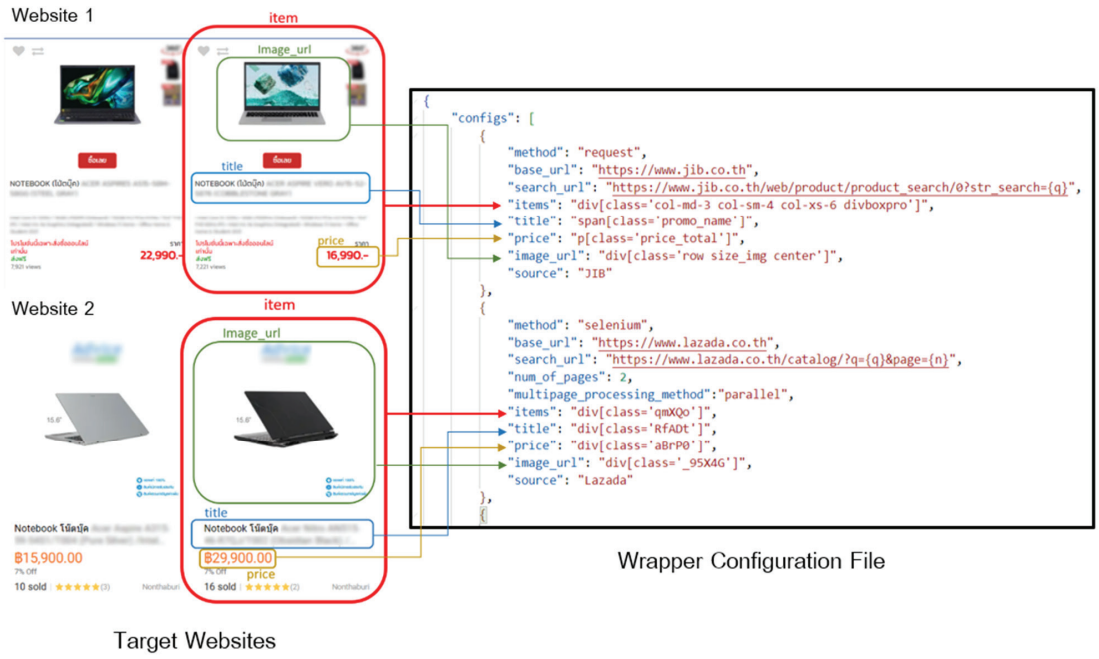

Figure 3. Example of Wrapper Configuration

the wrapper configuration file to import the specified data extraction rules. Subsequently, the search term is incorporated into the URL for product searches on the target website, and the system accesses the website using the chosen web access tool (either Requests or Selenium WebDriver). Once access to the target website is established, the Beautiful Soup 4 library is invoked to parse the HTML content, converting it into a structured tree for data extraction. Beautiful Soup 4 then extracts the data based on the rules defined in the wrapper configuration file.

3.2 Product Type Identification Process

Once the Web Wrapper extracts data from target websites, this extracted data is directed into the Product Identification module within the Data Categorization Layer. The core concept behind this process is to utilize product datasets as metadata, describing the product specifications for each item, to identify the product types or models of the extracted items. These datasets serve as a reference to make such identifications. To achieve this, the system employs TF-IDF (Term Frequency-Inverse Document Frequency) for vectorization of product names from both the extracted products and the product dataset. TF-IDF is a numerical statistic that reflects the importance of a term within a document relative to a collection of documents. The algorithm consists of two components: Term Frequency (TF) and Inverse Document Frequency (IDF). TF measures the frequency of a term within a document, while IDF evaluates the rarity of a term across the entire

corpus. The computations for TF and IDF are expressed in Equations 1 and 2, respectively. The TF-IDF score for a term in a document is then derived by multiplying its TF by its IDF, as depicted in Equation 3.

$$TF(t,d) = \frac{Numberoftimestermtappearsindocumentd}{Totalnumberoftermsindocumentd} \quad (1)$$

$$IDF(t,D) = log\left(\frac{Totalnumberofdocumentsinthecorpus D}{Numberofdocumentscontainingtermt}\right) \quad (2)$$

$$TF - IDF(t,d,D) = TF(t,d) \times IDF(t,D) \quad (3)$$

The vectorization process involves converting textual data, in this case, product names, into numerical vectors. Specifically, TF-IDF assigns weightings to each term based on its frequency within the product name and its rarity across the entire dataset. The system then evaluates which product models in the dataset exhibit the highest cosine similarity values to the extracted products. The cosine similarity is a measure of the cosine of the angle between two vectors. In the context of TF-IDF vectors, it indicates how closely aligned the vectors are in the high-dimensional space defined by the terms. The formula for cosine similarity between two vectors A and B is given by:

$$cos(A,B) = \frac{A \cdot B}{\|A\| \times \|B\|} \quad (4)$$

where A•B is the dot product of vectors A and B. ||A|| and ||B|| are the magnitude of vectors A and B, respectively. The result is a value between -1 and 1. A value of 1 indicates perfect similarity, 0 indicates no similarity, and -1 indicates perfect dissimilarity. In this scenario, the model in the dataset with the highest similarity to the scraped product under consideration is identified, but if the maximum cosine similarity value does not meet the specified threshold (defaulting to 0.5, adjustable in the dataset configuration file), the system does not identify the model for the scraped product, considering that the model is not in the dataset. Subsequently, the system checks for any extracted products identified as the same model, grouping them into the same product category, and prepares to transmit the results to the user interface. Finally, the unidentified products are forwarded to the Product Clustering module for the grouping of similar products in the subsequent step.

3.2.1 Configuring the Import of Product Datasets

Users have the option to import a product dataset into the system. This dataset can be obtained from various online sources or extracted from specific websites. The imported dataset essentially acts as a guide to assist in the identification of product types or models from the target website's data. Users can configure dataset imports through the system's JSON-formatted configuration file, with a provided template, as shown in Figure 4. In the context of this research, two datasets have been incorporated into the system, one focusing on laptop computers extracted from www.notebookspec.com and

another on mobile phones extracted from www.specphone.com. These datasets were created through web data extraction by the researcher using the Scrapy Framework, which is proficient in swiftly extracting numerous static webpages containing detailed product specifications. An example of a laptop dataset used in this system is shown in Figure 5. A crucial step involves specifying trigger words, which serve as keywords to match against the user's search queries. The system will import and process the dataset if any of these trigger words are found in the user's search term. For example, if the dataset pertains to laptop computers, the user can define trigger words like "Notebook" or "Laptop." If the user's search query includes any of these words, e.g., "Laptop for Students", the system will import the corresponding laptop dataset for further processing.

Additionally, the user must indicate the name of the column in the dataset containing product titles to ensure correct processing. Other settings allow users to customize the display of product properties on the search results page. For instance, users can choose to display information such as Central Processing Unit (CPU), Graphics Processing Unit (GPU), Random Access Memory (RAM), and Solid State Drive (SSD) storage sizes, enhancing the convenience of product searches. The example result of product type identification is presented in Figure 6.

3.2.2 Filtering Products Using Specification Data

Once the entire process is complete, users can filter products on the web application's results page using product specification data, utilizing the product dataset as metadata for filtering. For instance, when employing a laptop computer dataset, users may search for specific laptop models with particular CPU or GPU types, as illustrated in Figure 7.

```
"dataset_configs": [
  {
      "trigger_words": ["Specify a set of trigger words for dataset importation"],
      "file_path": "Specify the file path of the dataset",
      "title_column_name": "Specify the column name in the dataset containing product model
names",
      "columns_to_be_shown": ["Specify names of columns in the dataset with specification data
to display in the result page"],
      "similarity_threshold": "Specify the minimum cosine similarity threshold to consider a product
match (default is 0.5)"
  },
  {"Additional configurations for other dataset importations"}
  ]
```

Figure 4. Template of Dataset Configuration File

	A	B	C	D	E	F	G	H	I	J
1	title	cpu: CPU	cpu: Core	cpu: Threa	cpu: Cache	cpu: TDP	display: Pa	display: Si	display: Re	display: Re
2	Acer Nitro	AMD Ryzer	8 (P-Core)	16	4MB (L2 C	54 W	IPS	16 inch	16.0 inch \	165 Hz
3	Lenovo V1	Intel Core	10 (P-Core	12	18.5MB (L	-	IPS	15.6 inch	15.6 inch (	60 Hz
4	MSI Vector	Intel Core	8 (P-Core)	32	32MB (L2	55 W	IPS	16 inch	16 inch (2	240 Hz
5	MSI Cybor	Intel Core	4 (P-Core)	12	13MB (L2	45 W	IPS	15.6 inch	15.6 inch (	144 Hz
6	Apple Macl	Apple M1 I	10 (P-Core	10	24MB (L2	-	IPS	16.2 inch	16.2 inch (	120 Hz
7	Lenovo Le	Intel Core	6 (P-Core)	20	11.5MB (L	45 W	IPS	16 inch	16 inch W	240 Hz
8	Asus ROG	AMD Ryzer	8 (P-Core)	16	8MB (L2 C	9-30 W	IPS	7 inch	7 inch (19	120 Hz
9	DELL Latitu	Intel Core	4 (P-Core)	16	10MB (L2	28 W	IPS	15.6 inch	15.6 inch (	60 Hz
10	Asus ROG	Intel Core	8 (P-Core)	32	32MB (L2	55 W	IPS	16 inch	16.0 inch \	165 Hz
11	Hp Pavilior	AMD Ryzer	6 (P-Core)	12	3MB (L2 C	28 W	OLED	13.3 inch	13.3 inch \	90 Hz
12	Hp 15-fc0(	AMD Ryzer	6 (P-Core)	12	3MB (L2 C	15 W	IPS	15.6 inch	15.6 inch (	60 Hz
13	Asus Vivot	AMD Ryzer	4 (P-Core)	4	2MB (L2 C	15 W	IPS	14 inch	14 inch (1	60 Hz
14	MSI Moder	Intel Core	2 (P-Core)	4	2.5MB (L2	-	IPS	15.6 inch	15.6 inch (	60 Hz
15	DELL Inspi	Intel Core	2 (P-Core)	8	9.5MB (L2	12 W	IPS	14 inch	14 inch (1	60 Hz
16	DELL G15	Intel Core	6 (P-Core)	16	20MB (L2	55 W	IPS	15.6 inch	15.6 inch (	120 Hz
17	Asus Vivot	Intel Core	4 (P-Core)	12	13MB (L2	45 W	IPS	16 inch	16.0 inch \	60 Hz
18	MSI Moder	Intel Core	2 (P-Core)	4	2.5MB (L2	-	IPS	15.6 inch	15.6 inch (	60 Hz
19	DELL G15	Intel Core	6 (P-Core)	20	24MB (L2	55 W	IPS	15.6 inch	15.6 inch (	165 Hz
20	DELL Inspi	Intel Core	4 (P-Core)	16	10MB (L2	28 W	IPS	16 inch	16 inch (1	60 Hz
21	DELL Inspi	Intel Core	4 (P-Core)	16	21MB (L2	28 W	IPS	16 inch	16 inch (1	60 Hz
22	DELL Alien	Intel Core	6 (P-Core)	20	11.5MB (L	45 W	IPS	16 inch	16 inch (2	240 Hz

Figure 5. Example of Laptop Dataset

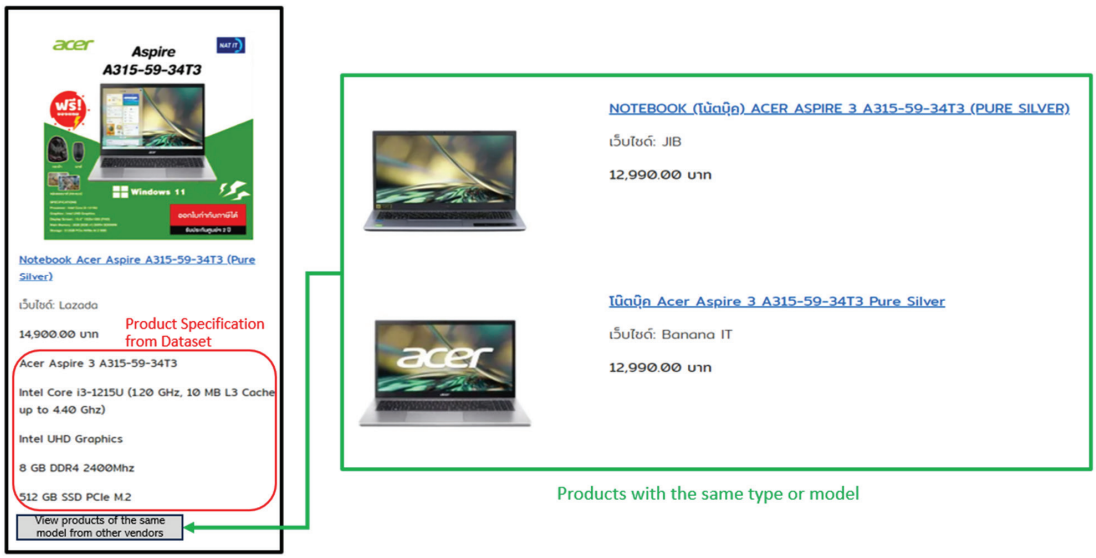

Figure 6. Example of Product Identification Result

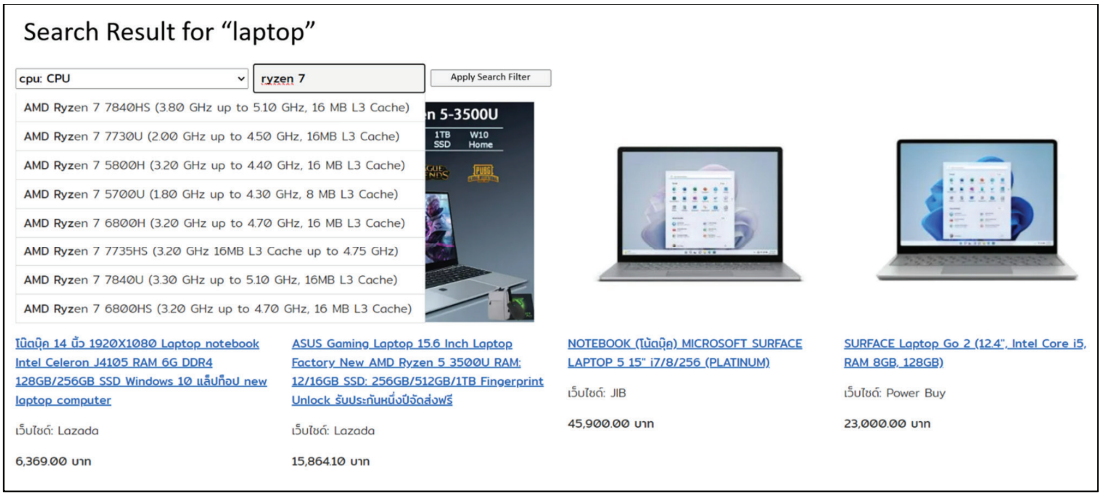

Figure 7. Search Filtering

3.3 Similar Product Grouping Process

This phase, executed within the Data Categorization Layer similar to the previous process, categorizes unclassified data from either the Product Identification module or directly from the Web Wrapper in instances where dataset importation is not performed. The Agglomerative Clustering method is employed, utilizing product names and images as features for grouping. Agglomerative Clustering is a hierarchical clustering algorithm used for grouping similar data points into clusters. The process starts with each data point considered as an individual cluster and, in each iteration, merges the closest pair of clusters until only one cluster remains. The key idea is to build a hierarchy of clusters, represented as a tree or dendrogram, where the leaves of the tree are the individual data points, and the root is the ultimate single cluster containing all data points.

Initially, product name data undergoes preprocessing, including punctuation removal, tokenization, and stop-word elimination. Then, the data is vectorized using TF-IDF. For the product images, features are extracted using a Convolutional Neural Network (CNN) model. Image features represent patterns, shapes, and structures within images, providing numerical representations of content. The CNN model, pretrained on the ImageNet-1k dataset containing 1,281,167 categorized images (Russakovsky *et al*., 2015), learns filters and patterns useful for discriminating between objects or features within images. This process is facilitated by the PyTorch library, providing image preprocessing tools such as resizing, central cropping, and normalization. After extracting both product name and image features, the system concatenates these features and employs them for data grouping via Agglomerative Clustering, utilizing the scikit-learn library and the “cosine” metric for grouping. This process requires defining

a Distance Threshold, determining when the system combines groups based on cosine distance, which can be specified in the product grouping configuration file. Subsequently, all grouped product data is collected and transmitted to the frontend system, allowing users to search for similar products, as illustrated in Figure 8. Furthermore, the system stores feature data extracted from product images in its cache to expedite future searches for the same products. Since image feature extraction is computationally intensive and time-consuming, cached feature data enables quicker processing. This optimization reduces overall processing time.

4. Results and Discussion

We conducted a preliminary evaluation of our system in two aspects. Firstly, the web extraction system underwent evaluation through experiments involving the extraction of product data from various e-commerce websites. These experiments aimed to verify whether the extracted results correctly contained the product data from the target websites. Specifically, we selected Lazada (lazada.co.th), Power Buy (powerbuy.co.th), Banana IT (bnn.in.th), and JIB (jib.co.th) as target websites. The web extraction experiments encompassed ten different search queries. The results presented in Table 1 demonstrate that our system, with a correctly configured Web Wrapper, achieved a 100 percent accuracy rate in extracting product data from target websites. This indicates that our system extracted data without any missing or mismatched information.

The second aspect of our evaluation focused on the performance of our similar products grouping system. Given the availability of numerous CNN models, researchers conducted experiments to determine the most suitable model for extracting features from images and classifying them using Agglomerative Clustering. The experiments utilized product dataset sourced from the Shopee

Figure 8. Example of Similar Product Grouping Result

website, specifically a dataset designed for a similar product classification competition on www.kaggle.com. This dataset encompasses specifications for various types of products, including fashion items, electronics, and consumer goods. The dataset, comprising product name and image information necessary for the product grouping process, was downloaded from www.kaggle.com and imported into the system for experimentation. To assess the efficiency of these models, the researchers calculated the F1 score for each product and subsequently averaged these scores. This process determined the distance threshold value that yielded the highest F1 score, guiding model selection.

Researchers opted for several CNN models that are computationally efficient, pretrained on the ImageNet dataset. To expedite the research, a subset of 1,000 products was employed in the experiments. The results, presented in Table 2, demonstrated that the EfficientNet-B0 model (Tan & Le, 2019) outperformed other models without significantly increasing processing time for feature extraction and product grouping. Consequently, this model was selected as the default for extracting image features within our system. Subsequently, when the features obtained from TF-IDF were combined with features from EfficientNet-B0 for clustering, they exhibited superior performance compared to using TF-IDF features or the EfficientNet-B0 model in isolation.

Table 1. Web extraction experimental result

Target Website	Extracting Method	Total Product Data Appeared on the Website	Total Valid Product Data Extracted
Lazada	Dynamic	487	487
Power Buy	Static	287	287
Banana IT	Static	377	377
JIB	Static	604	604

Table 2. Products grouping experimental result

Method	F1 Score	Processing Time (second)	Best Distance Threshold
MobileNetV3-Small	0.7862	6.046	0.23
MobileNetV3-Large	0.7643	6.512	0.33
Resnet18	0.7503	6.9090	0.18
EfficientNet-B0	0.7990	8.1388	0.48
TF-IDF	0.8631	0.5142	0.82
TF-IDF + EfficientNet-B0	0.9428	8.6026	0.68

5. Conclusion

In this study, we have introduced a Data Scraping-based System designed to extract product data from various E-commerce websites. This system incorporates features for product identification and grouping of similar products, enhancing the browsing experience for users. Users can customize the system by defining extraction rules for web scraping and importing data for product identification. The effectiveness of the system is underlined by its ability to provide updated and tailored search results to users, reducing data mismatch and missing issues and thereby improving their online shopping experience. We anticipate continuous improvements to our system, focusing on optimizing product data feature extraction, enhancing clustering methods, and refining the data scraping process. By continually refining and expanding the system's capabilities, we aim to bridge the gap between consumers and e-commerce platforms, making online shopping a more streamlined and rewarding experience for all parties involved.

Reference

Addagarla, S. K., & Amalanathan, A. (2020). Probabilistic unsupervised machine learning approach for a similar image recommender system for E-commerce. *Symmetry, 12*(11), 1783. https://doi.org/10.3390/sym12111783

Alam, A., Anjum, A. A., Tasin, F. S., Reyad, M. R., Sinthee, S. A., & Hossain, N. (2020). Upoma: A dynamic online price comparison tool for Bangladeshi E-commerce websites. *2020 IEEE Region 10 Symposium (TENSYMP)*, 194–197. https://doi.org/10.1109/tensymp50017.2020.9230862

Ambre, A., Gaikwad, P., Pawar, K., & Patil, V. (2019). Web and android application for comparison of E-commerce products. *International Journal of Advanced Engineering, Management and Science, 5*(4), 266–268. https://doi.org/10.22161/ijaems.5.4.5

Asawa, A., Dabre, S., Rahise, S., Bansode, M., Talele, K. T., & Chimurkar, P. (2022). Co-Mart - A daily necessity price comparison application. *2022 International Conference on Applied Artificial Intelligence and Computing (ICAAIC)*, 1076–1080. https://doi.org/10.1109/icaaic53929.2022.9792935

Gheorghe, M., Mihai, F.-C., & Dârdală, M. (2018). Modern techniques of web scraping for data scientists. *International Journal of User-System Interaction, 11*(1), 63–75. https://rochi.utcluj.ro/rrioc/articole/RRIOC-11-1-Gheorghe.pdf

Kannan, H. K. (2021). *E-commerce product similarity match detection using product text and images* [Master's thesis, National College of Ireland]. https://norma.ncirl.ie/5171/

Kemp, S. (2022). Digital 2022: Another year of bumper growth. We are social. Retrieved August 24, 2023, from https://wearesocial.com/us/blog/2022/01/digital-2022-another-year-of-bumper-growth-2/

Lan, H., Sha, D., Malarvizhi, A. S., Liu, Y., Li, Y., Meister, N., Liu, Q., Wang, Z., Yang, J., & Yang, C. P. (2021). COVID-Scraper: An open-source toolset for automatically scraping and processing global multi-scale spatiotemporal COVID-19 records. *IEEE Access, 9*, 84783–84798. https://doi.org/10.1109/access.2021.3085682

Li, J., Dou, Z., Zhu, Y., Zuo, X., & Wen, J.-R. (2019). Deep cross-platform product matching in E-commerce. *Information Retrieval Journal, 23*(2), 136–158. https://doi.org/10.1007/s10791-019-09360-1

Mehak, S., Zafar, R., Aslam, S., & Bhatti, S. M. (2019). Exploiting filtering approach with web scrapping for smart online shopping : Penny Wise: A wise tool for online shopping. *2019 2nd International Conference on Computing, Mathematics and Engineering Technologies (ICoMET)*, 1–5. https://doi.org/10.1109/icomet.2019.8673399

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., & Fei-Fei, L. (2015). ImageNet large scale visual recognition challenge. *International Journal of Computer Vision, 115*(3), 211–252. https://doi.org/10.1007/s11263-015-0816-y

Tan, M., & Le, Q. (2019). EfficientNet: Rethinking model scaling for convolutional neural networks. *Proceedings of the 36th International Conference on Machine Learning (Vol. 97)*, 6105–6114. https://proceedings.mlr.press/v97/tan19a.html