

# **“AI-DRIVEN ADAPTIVE WEB DATA EXTRACTION SYSTEM”**

## **ABSTRACT**

In today's digital age, an adaptive web data extraction system powered by AI has become the need of the hour to extract useful information from the ever-evolving and vast World Wide Web in an efficient manner. The conventional web scraping techniques, which are basis-dependent on certain HTML element names and static webpage structures, are usually inflexible and inefficient. These weaknesses have a devastating impact on accuracy when webpage layouts fluctuate, which is quite frequent, particularly with respect to dynamic E-commerce websites. To surmount these difficulties, this study envisions an intelligent, adaptive web data extraction system driven by AI that is capable of operating reliably in real-world settings where web content evolves dynamically and in an unpredictable manner. In contrast to traditional approaches, this method employs a robust set of semantic keywords and contextual awareness that minimizes reliance upon brittle selectors such as XPath or element IDs. In comparison to generative AI-based approaches, the new method shows greater adaptability and consistency, facilitating smooth and accurate data extraction irrespective of front-end changes. This study not only presents a new AI-based solution for automated web data extraction but also creates a platform for future innovations, allowing companies to remain agile and competitive in a fast-changing digital environment.

## **INDEX**

<b>CHAPTER NO</b>	<b>CONTENTS</b>	<b>PAGE NO</b>
–	<b>ABSTRACT</b>	<b>i</b>
–	<b>LIST OF FIGURES</b>	<b>iv</b>
–	<b>LIST OF TABLES</b>	<b>v</b>
–	<b>LIST OF SCREENSHOTS</b>	<b>v</b>
<b>1</b>	<b>INTRODUCTION</b>	<b>1</b>
<b>1.1</b>	Purpose of the Project	<b>1</b>
<b>1.2</b>	Scope of the Project	<b>3</b>
<b>1.3</b>	Objective of the Project	<b>3</b>
<b>2</b>	<b>LITERATURE SURVEY</b>	<b>4</b>
<b>2.1</b>	Traditional Selector-Based Scraping	<b>5</b>
<b>2.2</b>	Machine Learning for Layout Understanding	<b>5</b>
<b>2.3</b>	NLP-Driven Data Extraction	<b>6</b>
<b>2.4</b>	Emerging Generative-AI Methods	<b>7</b>
<b>3</b>	<b>SYSTEM ANALYSIS</b>	<b>8</b>
<b>3.1</b>	Existing System	<b>8</b>
<b>3.2</b>	Proposed System	<b>9</b>
<b>4</b>	<b>REQUIREMENTS SPECIFICATION</b>	<b>10</b>
<b>4.1</b>	Software Requirements	<b>10</b>
<b>4.2</b>	Hardware Requirements	<b>10</b>
<b>5</b>	<b>SYSTEM DESIGN</b>	<b>11</b>
<b>5.1</b>	System Specifications	<b>11</b>

<b>CHAPTER NO</b>	<b>CONTENTS</b>	<b>PAGE NO</b>
<b>5.2</b>	System Architecture	<b>18</b>
<b>5.3</b>	System Components and Modules	<b>12</b>
<b>5.4</b>	AI-Driven Extraction Approaches	<b>20</b>
<b>5.5</b>	Machine Learning Algorithms	<b>21</b>
<b>5.6</b>	UML Diagrams	<b>24</b>
<b>6</b>	<b>SYSTEM IMPLEMENTATION</b>	<b>29</b>
<b>6.1</b>	Python Introduction	<b>29</b>
<b>6.2</b>	Source Code	<b>32</b>
<b>7</b>	<b>TESTING</b>	<b>38</b>
<b>7.1</b>	System Testing	<b>38</b>
<b>7.2</b>	Module Testing	<b>39</b>
<b>7.3</b>	Integration Testing	<b>39</b>
<b>7.4</b>	Acceptance Testing	<b>39</b>
<b>7.5</b>	Test Cases	<b>40</b>
<b>8</b>	<b>RESULTS / OUTPUTS</b>	<b>41</b>
<b>9</b>	<b>CONCLUSION &amp; FUTURE SCOPE</b>	<b>43</b>
	<b>REFERENCES</b>	<b>44</b>

### LIST OF FIGURES

FIGURE.NO	NAME OF THE DIAGRAM	PAGE.NO
5.3.1	System Architecture for AI-Driven Adaptive Web Data Extraction System	18
5.6.1	Use Case Diagram	24
5.6.2	Class Diagram	25
5.6.3	Sequence Diagram	26
5.6.4	Activity Diagram	27
5.6.5	Component Diagram	28

## LIST OF TABLES

TABLE NO	PARTICULARS	PAGE NO
7.5	Test Cases	40

# **1.INTRODUCTION**

## **1.1 PURPOSE OF THE PROJECT**

The Purpose of the AI-Driven Adaptive Web Data Extraction System is to create a smart, reliable system that can adaptively and effectively extract data from various web sources. The conventional web scraping processes, based extensively on predefined models and fixed rules, are commonly faced with important limitations. They may be useless as websites constantly change, requiring frequent adjustments to their structures and content formats. These constraints can lead to data extraction flaws and inefficiencies, especially in dynamic web contexts where layout and content are continually changed.

The AI-Driven Adaptive Web Data Extraction System was created to meet these challenges based on cutting-edge AI and machine learning technologies. The system's overall functionality is based on its constant ability to learn from the data that it extracts. This learning feature allows the system to identify and adjust to newer patterns and structures of web data in order to stay accurate and efficient in the long run. Through the use of advanced algorithms, the system can detect and adjust to website structure changes, thus ensuring high extraction accuracy levels.

Adaptability is one of the major benefits of this system. While websites are being updated and modified, the AI-Driven Adaptive Web Data Extraction System is able to dynamically adapt its extraction mechanisms to suit these updates[1]. This ability is especially beneficial for companies and researchers who rely on current and correct web data for their research and decision-making purposes. For example, companies can utilize the system to track competitor behavior, monitor market trends, and obtain customer information, all of which need timely and accurate data. Researchers can take advantage of the system's capability to gather large amounts of data from multiple sources at the same time, allowing for thorough studies and analyses.

In addition, the AI-Driven Adaptive Web Data Extraction System has been developed to extract both structured and unstructured data, making it highly adaptable to a broad spectrum of applications. It supports the processing of web data from various sources in parallel, thereby ensuring scalability and efficiency. The system further has strong error-handling capabilities to cope with data extraction failures, further improving its reliability and resilience.

Apart from its adaptability and effectiveness, the system offers insights and visualizations for users to be able to grasp the extracted information and how relevant it is. This is beneficial for companies and researchers who may need to analyse and interpret the data accordingly. The system is designed with an easy-to-use interface to make it accessible for users of diverse technical capabilities.

The AI-Driven Adaptive Web Data Extraction System also has sophisticated data preprocessing methods to sanitize and normalize the web data extracted to ensure consistency and quality[2]. This preprocessing phase is important to ensure the correctness of data analysis and interpretation. In addition, the system can be integrated with existing data analysis tools and platforms, offering a smooth workflow for the users.

Another notable characteristic of the system is its high-velocity data stream handling capability, rendering it appropriately for real-time data extraction and analysis. Such capability is crucial in applications that demand minute-by-minute data, including financial market analysis, real-time social media trend monitoring, and live monitoring of e-commerce transactions.

To conclude, the AI-Driven Adaptive Web Data Extraction System is a promising step in web data extraction technology. By integrating AI and machine learning with adaptability features, the system offers a robust platform for companies and researchers. It allows them to retrieve accurate and current web data effectively and efficiently, thus facilitating smart decision-making and in-depth analysis. The advanced features of the system, such as adaptability, scalability, error handling, and real-time processing, render it an invaluable asset for any organization or individual that depends on web data for its operations and research.

## 1.2 SCOPE OF THE PROJECT

**Data Handling:** The system is designed to manage various types of web data, including both structured and unstructured data.

**Adaptability:** It will dynamically adapt to changes in website layouts and structures, ensuring continuous and precise data extraction.

**Scalability:** The system is engineered to be highly scalable, capable of processing large volumes of data from multiple web sources simultaneously.

**Insights and Visualizations:** It will offer valuable insights and visualizations, aiding users in understanding the relevance and context of the extracted data.

**Robust Tool:** This adaptability and scalability make the system a robust tool for businesses and researchers who rely on accurate and timely web data for analysis and decision-making.

## 1.3 OBJECTIVE OF THE PROJECT

Project objectives are specific, measurable goals that define what a project aims to achieve within a given timeframe and resource constraints. They serve as a roadmap for project planning, execution, and evaluation, outlining the desired outcomes and guiding the project team towards successful completion.

### 1.3.1 Primary Objectives:

#### **Develop an Adaptive System for Web Data Extraction:**

Create a system capable of dynamically adjusting to changes in web content and structure. This involves designing algorithms that can recognize and adapt to new patterns and layouts in web data, ensuring continuous and accurate data extraction even as websites evolve.

#### **Implement Machine Learning Algorithms:**

Integrate advanced machine learning algorithms to enable the system to learn from past extraction patterns and improve its data extraction capabilities over time. This includes



selecting appropriate algorithms, training models, and continuously updating them to enhance performance and accuracy.

### **Handle Large Volumes of Data Efficiently and Accurately :**

Ensure the system can process and manage large volumes of data from multiple web sources simultaneously. This involves optimizing data extraction processes, implementing efficient data storage and retrieval mechanisms, and ensuring high levels of data accuracy and consistency.

### **Provide a User Friendly Interface :**

Develop an intuitive and easy to use interface that allows users to interact with the system effectively. This includes designing visualizations and dashboards that help users understand the extracted data and its relevance, as well as providing tools for data analysis and interpretation.

#### **1.3.2 Secondary Objectives:**

### **Integrate with Existing Data Analysis Tools and Platforms :**

Ensure the system can seamlessly integrate with existing data analysis tools and platforms, providing a cohesive workflow for users. This involves developing APIs and connectors that facilitate data exchange and interoperability with other systems.

### **Develop Robust Error Handling Mechanisms :**

Implement comprehensive error handling mechanisms to manage data extraction failures and ensure system reliability. This includes designing algorithms to detect and recover from errors, as well as providing users with feedback and support for troubleshooting.

### **Ensure Scalability and Deployment Flexibility :**

Design the system to be scalable and capable of being deployed in various environments, from small scale applications to large enterprise systems. This involves using scalable architectures, cloud based solutions, and containerization technologies to ensure flexibility and adaptability.

## 2. LITERATURE SURVEY

### 2.1 TRADITIONAL SELECTOR-BASED SCRAPING

Traditional web scraping frameworks have long relied on XPath and CSS selectors to navigate HTML documents and extract desired elements by tag names, class attributes, or fixed hierarchical paths. In essence, developers inspect a page’s Document Object Model (DOM), identify unique element paths (e.g. `//div[@class='price']` or `.product-title > a`), and hard-code these selectors into scripts using libraries like BeautifulSoup, lxml, or Selenium.[2]

While straightforward and performant in static environments, this approach exhibits several critical weaknesses in modern, dynamic web applications. First, any alteration—such as renaming a CSS class, adding an extra `<div>`, or restructuring the page hierarchy—can silently break the scraper, returning empty or incorrect data. Maintenance thus becomes a continuous, manual “whack-a-mole” exercise in selector updates. Second, frameworks based purely on selectors struggle with JavaScript-rendered content, requiring heavyweight browser automation, which increases latency and resource usage.[5] Third, these scrapers lack semantic understanding; they cannot distinguish between visually similar elements (e.g. two price displays) without further heuristics. Finally, scaling to multiple websites magnifies the burden: each new domain demands bespoke selector logic[8]. As a result, traditional selector-based scraping delivers high performance in stable settings but imposes unsustainable maintenance costs and brittle behavior in the face of frequent UI changes.

### 2.2 MACHINE-LEARNING FOR LAYOUT UNDERSTANDING

To overcome the rigidity of selector-based scraping, researchers have explored **computer vision** techniques to semantically segment web page layouts. By treating rendered web pages as images, these methods apply object-detection and region-proposal models—such as **Faster R-CNN**, **YOLO**, or lightweight **SSD** variants—to locate distinct content blocks like

product cards, navigation bars, and data tables. After segmentation, **optical character recognition** (OCR) and text-block classification assign semantic labels to each region, enabling more robust extraction pipelines. For instance, a trained CNN can learn to recognize “price label” contours across diverse templates, irrespective of underlying HTML structure. Some systems adopt **Region-of-Interest (RoI) pooling** to refine block boundaries, while others leverage **graph-based layout representations**, modelling the spatial and hierarchical relationships among detected boxes. These vision-based approaches dramatically improve resilience to DOM changes because they rely on visual appearance rather than brittle selectors. However, they introduce new challenges: the overhead of image capture and inference, sensitivity to styling variations (fonts, colors, resolution), and the need for labelled training data to fine-tune detection models. Despite these trade-offs, machine-learning-driven layout understanding represents a promising path toward adaptable, cross-domain web data extraction.

## 2.3 NLP-DRIVEN DATA EXTRACTION

Natural Language Processing (NLP) techniques bring **semantic intelligence** to data extraction by analysing the textual content of web pages rather than relying solely on structural cues. A common strategy is to first traverse the DOM, extract text snippets from all nodes, and embed these snippets using pretrained language models (e.g. **BERT**, **spaCy embeddings**, or **FastText**). By comparing these embeddings against prototype vectors representing target fields—such as “Product Name,” “Price,” or “Description”—the system can **score** and **rank** candidate nodes based on semantic similarity. This approach handles variations in label text (e.g. “Cost:” vs. “Price:”), multi-language scenarios, and unpredictable nesting. Some pipelines enhance accuracy by combining NLP with **entity recognition**: custom-trained Conditional Random Fields (CRFs) or fine-tuned transformers identify named entities like currencies, dates, or numeric amounts directly in the text flow. Advanced frameworks also use **sequence-labelling** models to tag tokens in raw HTML or post-rendered text[7]. The benefits include reduced dependence on fixed paths and improved adaptability to new templates. Yet, NLP-driven extraction requires careful handling of noisy

webpage text, removal of boilerplate (e.g. navigation menus, footers), and often a modest amount of task-specific annotated data to train field classifiers. Overall, semantic annotation and embedding-based methods represent a flexible and scalable middle ground between brittle selectors and heavy vision models.

## 2.4 EMERGING GENERATIVE-AI METHODS

Recently, **large-language models (LLMs)** such as **GPT-3/4** have been leveraged to infer extraction patterns directly from sample page snippets through **prompt engineering** or **few-shot learning**. In these generative-AI methods, the LLM is provided with a handful of labeled examples—pairs of HTML segments and desired output fields—and tasked with generating code, CSS selectors, or JSON-structured responses for new pages. This paradigm promises **rapid adaptation**: when a website updates its layout, the model can often infer new extraction logic without explicit reprogramming. Moreover, LLMs can handle ambiguous labels, infer context (e.g. “This number under ‘Rating’ is a 5-star score”), and even rephrase extracted text. However, these approaches face several limitations. First, LLMs can produce **hallucinations**, emitting plausible-looking but incorrect selectors or misidentifying content blocks. Second, inference through API calls incurs **latency** and **cost**, making large-scale scraping expensive[4]. Third, maintaining **consistency** across thousands of pages requires careful prompt management and occasional human validation. Finally, generative methods often lack transparency: debugging a malformed extraction prompt is nontrivial. Thus, while generative-AI brings exciting adaptability, its fragility and operational overhead mean it is best used in conjunction with more deterministic modules, rather than as a standalone scraper.

## 3. SYSTEM ANALYSIS

### 3.1 EXISTING SYSTEM

#### **Rigid Dependence on DOM Paths**

Most traditional web scrapers are built around hard-coded **XPath** or **CSS selectors** that pinpoint elements by their exact location in the page's DOM tree. While this yields high precision when the page structure remains unchanged, it also makes scrapers extremely brittle. A single DIV insertion, CSS-class rename, or layout tweak can invalidate one or more paths, causing the extractor to return empty values or incorrect nodes.

#### **Inefficient Maintenance Lifecycle**

Every time a target website is updated—whether through A/B testing, seasonal UI refreshes, or incremental feature rollouts—developers must manually inspect the affected pages, identify broken selectors, and craft updated XPaths or CSS rules. For organizations scraping dozens of sites, this maintenance becomes a full-time job, leading to escalating costs and slowed data pipelines.

#### **Silent Failures and Data Quality Risks**

Rigid scrapers often “fail silently.” Because most scripts do not include thorough validation checks on extracted values, broken selectors can go unnoticed for hours or days. This results in corrupted datasets—missing prices, blank product names, malformed tables—that downstream analytics or machine-learning models consume, compromising business decisions.

#### **Disadvantages:**

1. Brittleness to UI Changes
2. High Manual Maintenance Overhead
3. Silent Failures and Data Corruption
4. Inflexibility Across Domains
5. Limited Contextual Understanding
6. Poor Handling of Dynamic Content

## 3.2 PROPOSED SYSTEM:

To overcome these limitations, we propose an **AI-Driven Adaptive Framework** that dynamically adjusts its extraction strategy in response to UI changes:

### **NLP Keyword Module**

Uses pretrained language embeddings (e.g. BERT or spaCy vectors) to map domain-specific keywords (like “price,” “description,” “rating”) to textual regions in the rendered DOM.

Instead of relying on fixed paths, it scores each text node by semantic similarity to these prototypes, locating the relevant section even if nested elements shift or labels are renamed.

### **Visual Layout Analyzer**

Captures a snapshot of the fully rendered page and applies a lightweight CNN-based object detector to segment the page into logical blocks (e.g. product cards, review panels).

By correlating block coordinates back to DOM elements via browser APIs, the system remains robust to structural reordering and CSS-only changes.

### **Reinforcement-Learning Field Mapper**

Models data extraction as a sequential decision process[4]: actions include “scroll,” “expand element,” or “extract text.”

A deep Q-network receives state information—current DOM context, prior extraction success—and learns policies that maximize long-term extraction accuracy.

As the page evolves, the agent continually refines its strategy through reward feedback (correct vs. incorrect extractions), enabling **self-correction** without manual intervention.

## **4.REQUIREMENTS SPECIFICATION**

### **4.1 SOFTWARE REQUIREMENTS**

Operating system	:	Windows 10/11
Platform	:	Python 3.10, JavaScript
Libraries	:	TensorFlow/Keras, PyTorch, spaCy, OpenCV, Selenium, BeautifulSoup, Scikit-learn

### **4.2 HARDWARE REQUIREMENTS**

System	:	8-core Intel i7 or AMD Ryzen 7
GPU	:	NVIDIA RTX 3060 (for CNN inference)
RAM	:	Minimum 8GB
Storage	:	1 TB SSD

## 5.SYSTEM DESIGN

### 5.1 SYSTEM SPECIFICATIONS

The system specifications define both the **functional and non-functional** aspects necessary for implementing and deploying the AI-Driven Adaptive Web Data Extraction System. This includes details on the expected behaviour of the system and the environment in which it operates.

#### REQUIREMENT SPECIFICATIONS:

**5.1.1 Functional Requirements:** For documenting the functional requirements, the set of functionalities supported by the system are to be specified. A function can be specified by identifying the state at which data is to be input to the system, its input data domain, the output domain, and the type of processing to be carried on the input data to obtain the output data. Functional requirements define specific behaviour or function of the application.

##### **User Registration & Login**

Users (Admin or End-User) must be able to securely log in to access the scraping dashboard.

##### **Web Page Ingestion**

Accept input URLs from the user.

Load and render pages using a headless browser (e.g., Puppeteer or Selenium).

##### **Semantic Field Detection**

Identify meaningful sections (e.g., titles, prices, dates) using AI/NLP.



## **Visual Segmentation**

Break down page into logical blocks (cards, tables, rows).

## **Data Extraction & Structuring**

Pull clean and structured data in JSON/CSV format.

## **Quality Check & Validation**

Ensure extracted data meets quality benchmarks using predefined rules.

## **Feedback & Retraining**

Log incorrect extractions and enable self-correction via model retraining.

## **Output Delivery**

Allow user to preview, download, or export extracted data.

## **Dashboard for Monitoring**

Display real-time status, extraction logs, and performance metrics.

**Non-Functional Requirements:** A non-functional requirement specifies criteria that can be used to judge the operation of a system, rather than specific behaviours. Especially these are the constraints the system must work within. It specifies the quality attribute of a software system. Description of non-functional requirements is just as critical as a functional requirement. NFR include:

### **Usability**

Simple web-based interface that's intuitive for non-technical users.

### **Performance**

Page loading and extraction should be completed within 2–3 seconds per page under normal conditions.

**Scalability**

Should support scraping multiple websites concurrently (threaded or asynchronous).

**Robustness**

Capable of handling common scraping blockers like slow load, missing fields, dynamic content.

**Security**

Ensure user credentials, extracted data, and logs are securely stored.

**Portability**

Can run on cloud platforms (AWS/GCP/Azure) or local environments with minimal setup.

**Extensibility**

Designed as modular components to allow easy upgrades (e.g., switching models, adding CAPTCHA solvers).

## **5.2 SYSTEM COMPONENTS AND MODULES**

The system is composed of multiple interlinked modules, each with a distinct role. These components work together in a pipeline architecture to perform the full cycle of intelligent web data extraction — from rendering to data delivery.

**URL Input & Control Module**

This is the starting point of the system where users provide the input URLs for data extraction. It includes user-level controls and input validation.

**Key Features:**

User Input Interface: Accepts single or multiple URLs.

Validation Engine: Verifies the format and accessibility of URLs before processing.

Batch Execution Support: Allows processing multiple URLs concurrently or in queue.

Error Handling: Handles invalid or dead links gracefully with logging.

Scheduling/Automation (Optional): Future upgrades may include automated or periodic scraping based on saved jobs.

This module ensures clean and valid data input, which is essential for the accuracy and reliability of the entire system.

### **Headless Browser Rendering Engine**

Traditional scrapers often fail with modern web pages that use JavaScript to load content dynamically. This component overcomes that limitation by mimicking a real browser using tools like Selenium, Puppeteer, or Playwright.

#### **Key Features:**

Dynamic Content Support: Executes JavaScript to fully render the webpage.

Scroll & Wait Logic: Simulates scrolling and waits for content to load.

DOM Snapshot Capture: Provides a real-time view of the final rendered page structure.

Anti-Detection Features: Can spoof user agents and browser behaviour to reduce bot detection.

This component ensures that all content, including dynamically loaded elements, is available for processing.

### **Semantic Anchoring Module (AI/NLP Engine)**

This is the intelligence layer of the system. It uses natural language processing (NLP) and large language models (LLMs) to identify which sections of the page are relevant for data extraction based on semantics rather than hardcoded rules.

**Key Features:**

Contextual Understanding: Detects fields like “Price,” “Name,” “Author,” etc., even if their labels vary across websites.

Multilingual Support (Optional): Can be trained or adapted for multilingual sites.

Robust to Layout Changes: Works even if the order or structure of fields change, making it adaptable.

Domain Agnostic: Useful across domains like e-commerce, real estate, news, and job listings.

It drastically reduces the need for manual configuration and improves adaptability.

**Visual Segmentation Module**

This module logically divides the page into visual or DOM-based blocks, such as product cards, table rows, or content grids. It enables structured extraction.

**Key Features:**

Bounding Box Detection: Identifies logical groupings visually (e.g., every product card is enclosed in a div).

DOM Clustering: Uses tag hierarchy to separate blocks.

Noise Filtering: Ignores unrelated sections like banners, ads, or navigation menus.

Integration with Semantic Module: Ensures that field detection is done within the correct visual segment, not globally.

This segmentation supports clean, organized data extraction that matches real-world presentation.

## **Field Extraction Engine**

After locating segments and anchor points, this module is responsible for extracting actual data values from the page.

### **Key Features:**

Structured Extraction: Pulls key-value pairs like Name: "Samsung Galaxy", Price: "₹49,999".

Multi-format Support: Handles text, numbers, URLs, images, and dates.

Normalization: Cleans the extracted data (removing HTML tags, trimming whitespace).

Template-Free Logic: Adapts based on AI output instead of fixed selectors.

It outputs clean data in formats suitable for storage or further analysis.

## **Data Validation & Feedback Loop**

This module ensures data accuracy and quality control. It checks the extracted values against expected formats or ranges and enables adaptive learning.

### **Key Features:**

Rule-Based Validation: Example: "Price should be a numeric value."

Consistency Checks: Ensures that extracted data follows consistent structure.

Error Logging: Flags suspicious or empty extractions.

Feedback Loop: Stores failed cases to retrain models or improve logic automatically.

This module maintains long-term reliability and enhances system performance over time.

## **User Interface Module (Dashboard)**

A web-based dashboard that gives users access to control and monitor the scraping system. It provides both interaction and transparency.

### **Key Features:**

URL Submission Form

Real-Time Extraction Logs

Preview of Extracted Data

Export Options: JSON, CSV, Excel.

System Status View: CPU/Memory/Task queue overview (optional).

The UI makes the system accessible and manageable even to non-technical users.

## **Data Storage & Output Module**

Once data is validated and extracted, it must be stored securely and made available for download or integration.

### **Key Features:**

Multiple Export Formats: JSON, CSV, Excel.

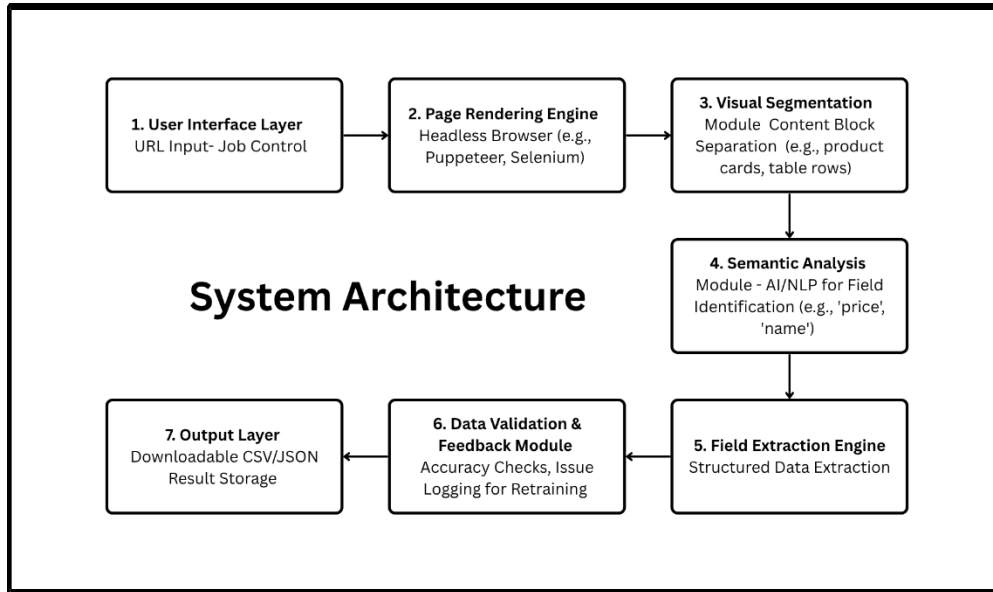
Database Integration (Optional): Can connect to MySQL, MongoDB, or Firebase.

File Management: Organizes outputs by timestamp, website, and content type.

Cloud Sync (Optional): Integration with Google Drive or AWS S3.

This module handles the final output delivery and storage of results.

## 5.3 SYSTEM ARCHITECTURE



*Fig 5.3.1 : System Architecture for AI-Driven Adaptive Web Data Extraction System*

The architecture of the AI-Driven Adaptive Web Data Extraction System is designed with a modular and pipeline-based approach to ensure scalability, robustness, and adaptability. This structure allows the system to function efficiently across a wide variety of websites—both static and dynamically rendered—by breaking down the data extraction process into well-defined, interacting components.

At the core of the architecture is the Input & Control Layer, where users provide URLs or batch inputs via a user-friendly interface. These inputs are then passed to the Page Rendering Layer, powered by headless browsers like Puppeteer or Selenium, which fully load dynamic content including JavaScript-rendered elements.

Once rendered, the page moves to the Interpretation Layer, which integrates AI-based Semantic Anchoring and Visual Segmentation. This layer identifies key data fields (e.g., price, title, ratings) by understanding the context and structure of the content, rather than

relying on static selectors. The Field Extraction Engine then pulls relevant data points and formats them into structured outputs.

Following extraction, the data is routed through the Validation & Feedback Layer, which ensures data accuracy using rule-based checks and pattern matching. If errors or anomalies are detected, they are logged and used to refine the system through a feedback loop, enhancing long-term accuracy and learning.

Finally, the Output Layer delivers clean, validated data in formats like JSON or CSV and supports integration with storage or downstream analytics. This modular flow enables the system to adapt to evolving web technologies without manual reconfiguration, ensuring high resilience and performance.

## **5.4 AI-DRIVEN ADAPTIVE WEB DATA EXTRACTION SYSTEM APPROACHES**

The process of extracting structured information from dynamic and visually complex web pages requires a combination of intelligent decision-making, adaptability, and robust content recognition. Traditional rule-based scraping fails when page layouts change. Therefore, this project introduces **AI-driven adaptive approaches** that can dynamically analyze, adapt, and extract relevant data with minimal human intervention.

### **DOM + Visual Segmentation Based Approach**

Instead of depending solely on the HTML tag hierarchy, the system interprets the layout by examining visual cues such as the size, location, and visibility of elements. It processes the DOM tree and groups related nodes into visual blocks based on their position on the screen, styling attributes, and relationship to other elements. This method mimics how humans perceive structured content on a page and is highly effective in identifying semantically significant blocks such as product listings, news items, or job posts.



## **Machine Learning-Based Block Classification**

each block is represented as a feature vector containing structural, textual, and visual attributes. A supervised learning model—such as Random Forest, Logistic Regression, or Gradient Boosted Trees—is then used to classify the blocks into categories like 'title', 'price', 'image', 'advertisement', or 'irrelevant'. This classification helps filter out noise and ensures that only the blocks containing useful information are retained for further processing.

## **Natural Language Processing (NLP)**

Models such as Named Entity Recognition (NER) and BERT are used to semantically analyze text and identify entities such as product names, prices, dates, and brand names. These models allow the system to go beyond surface-level extraction and understand the meaning and context of the content, which is particularly important when scraping text-heavy or information-rich sites like blogs, news portals, or academic repositories.

## **Reinforcement Learning (RL)**

In this model, an intelligent agent learns how to navigate through the DOM tree by performing actions like selecting elements, traversing to siblings or parents, and deciding whether to extract content. A reward mechanism is used to train the agent based on how accurately it extracts data compared to expected outputs. This allows the system to dynamically adjust to previously unseen layouts, making it resilient to minor and major changes in web design.

## **Vision-Based Deep Learning**

The entire web page is rendered as an image using a headless browser, and convolutional neural networks (cnns) or object detection models like YOLO are used to detect and extract content based on visual layout. This method is particularly useful for complex pages where traditional DOM traversal fails, such as interactive dashboards, data visualizations, or pages with embedded graphics.

## **Self-Healing Feedback Loop**

To ensure long-term reliability and adaptability. It continuously monitors the success of extractions by checking the completeness and accuracy of the output. If anomalies are detected—such as empty fields, inconsistent data, or incorrect formatting—the system logs these events and either retrains the classification models or switches to fallback strategies. These could include previously working extraction templates or rule-based patterns. This feedback-driven learning ensures that the system evolves and improves its performance over time without requiring constant manual updates.

## **5.5 ALGORITHMS**

Machine learning predictive algorithms has highly optimized estimation has to be likely outcome based on trained data. Predictive analytics is the use of data, statistical algorithms and machine learning techniques to identify the likelihood of future outcomes based on historical data. The goal is to go beyond knowing what has happened to providing a best assessment of what will happen in the future. In our system we used supervised machine learning algorithm having subcategories as classification and regression

### **Recursive DOM Traversal**

Recursive DOM Traversal works by systematically parsing the HTML document to construct a tree-based representation of all nodes. It uses a recursive approach for traversing and mapping each element, which is crucial for the classification of the document's structure. If a node contains child nodes, the algorithm will traverse each child recursively, ensuring that all elements are accounted for. This recursive method is fundamental for DOM analysis because it allows for a comprehensive exploration of the document structure, which is essential for accurate data extraction and manipulation.

**BERT (Bidirectional Encoder Representations from Transformers):**

for deeper textual understanding. BERT was used to extract semantic features from the content inside DOM blocks, enabling the system to identify entities such as prices, names, and dates with high precision. These embeddings were either directly used for content labelling or fed into other classifiers to improve performance.

**CRON-based Scheduler:**

A CRON-based scheduler is a time-based job scheduler in Unix-like operating systems that automates tasks at specified intervals. In the context of web data extraction, a CRON-based scheduler can be configured to periodically scrape data from target websites, ensuring that the system remains up-to-date with the latest information. This automation reduces the need for manual intervention, increases efficiency, and ensures that data is consistently collected at optimal times..

 **$K$  Nearest Neighbours**

$K$  Nearest Neighbours [10] method is a simplest and most popular method in machine learning which highly depends on a parameter  $k$  to determine the data classification in reference to nearby data. This approach is provided with an anomaly score which is computed for each data instance to its  $k$  NNs [24]. Here, a threshold is used to determine whether a data point is anomalous or not. This technique supports several data types such as continuous and discrete by different similarity functions in order to improve the performance of the technique.

## **5.6 UML DIAGRAMS**

UML stands for Unified Modeling Language. The Unified Modeling Language (UML) is a standardized visual language used to model and document the structure, behavior, and interactions of systems. It provides diagrams that help visualize complex systems, making them easier to design, develop, and communicate. UML diagrams are divided into structural diagrams (e.g., class, object, and component diagrams) for static aspects and behavioral diagrams (e.g., use case, activity, and sequence diagrams) for dynamic aspects.

It is widely used in software development, system engineering, and business process modeling to ensure clear communication and high-quality design. UML supports both agile and traditional methodologies, making it versatile and useful across various project stages.

### **Diagrams Classification**

#### **Structural Diagrams**

They depict a static view or structure of a system. It is widely used in the documentation of software architecture. It embraces class diagrams, composite structure diagrams, component diagrams, deployment diagrams, object diagrams, and

package diagrams.

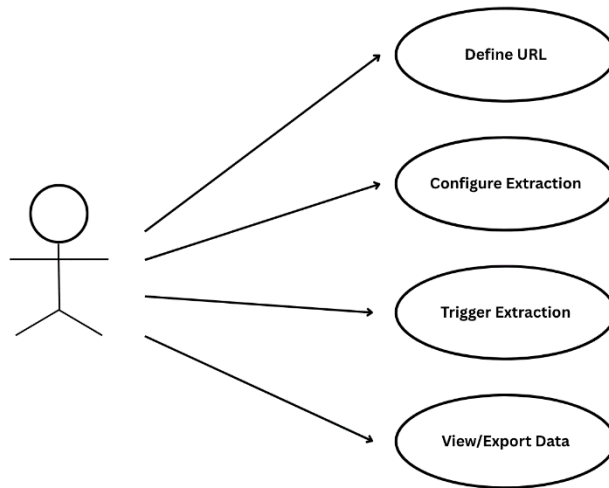
#### **Behavioural Diagrams**

They portray a dynamic view of a system or the behaviour of a system, which describes the functioning of the system. It includes use case diagrams, state diagrams, and activity diagrams.

## UML diagram

### 1. USE CASE DIAGRAM

A use case diagram at its simplest is a representation of a user's interaction with the system and depicting the specifications of a use case. A use case diagram can portray the different types of users of a system and the various ways that they interact with the system. This type of diagram is typically used in conjunction with the textual use case and will often be accompanied by other types of diagrams as well.



*Fig 5.6.1 : Use Case Diagram*

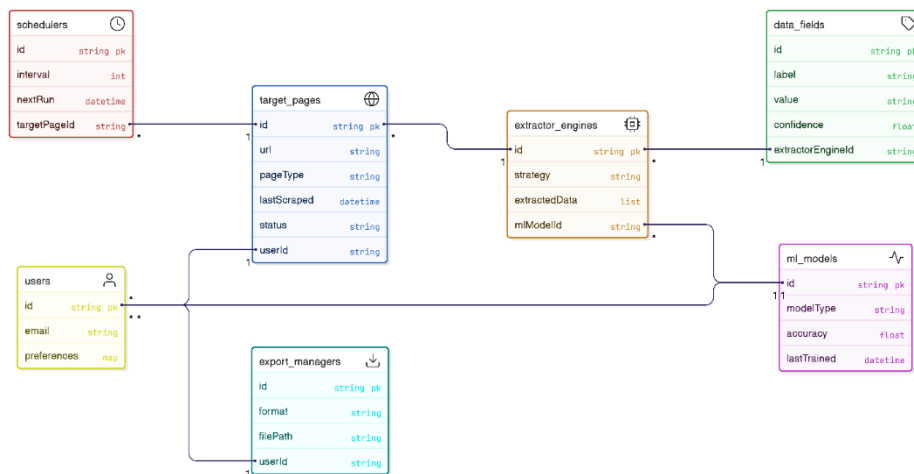
### 1.1 CLASS DIAGRAM

Class diagrams are the main building blocks of every object-oriented methods. The class diagram can be used to show the classes, relationships, interface, association, and collaboration. UML is standardized in class diagrams. Since classes are the building block of an application that is based on OOPs, so as the class diagram has appropriate structure to represent the classes, inheritance, relationships, and everything that OOPs have in its

context. It describes various kinds of objects and the static relationship in between them. The main purpose to use class diagrams are:

1. This is the only UML which can appropriately depict various aspects of OOPs concept.
2. Proper design and analysis of application can be faster and efficient.
3. It is base for deployment and component diagram.

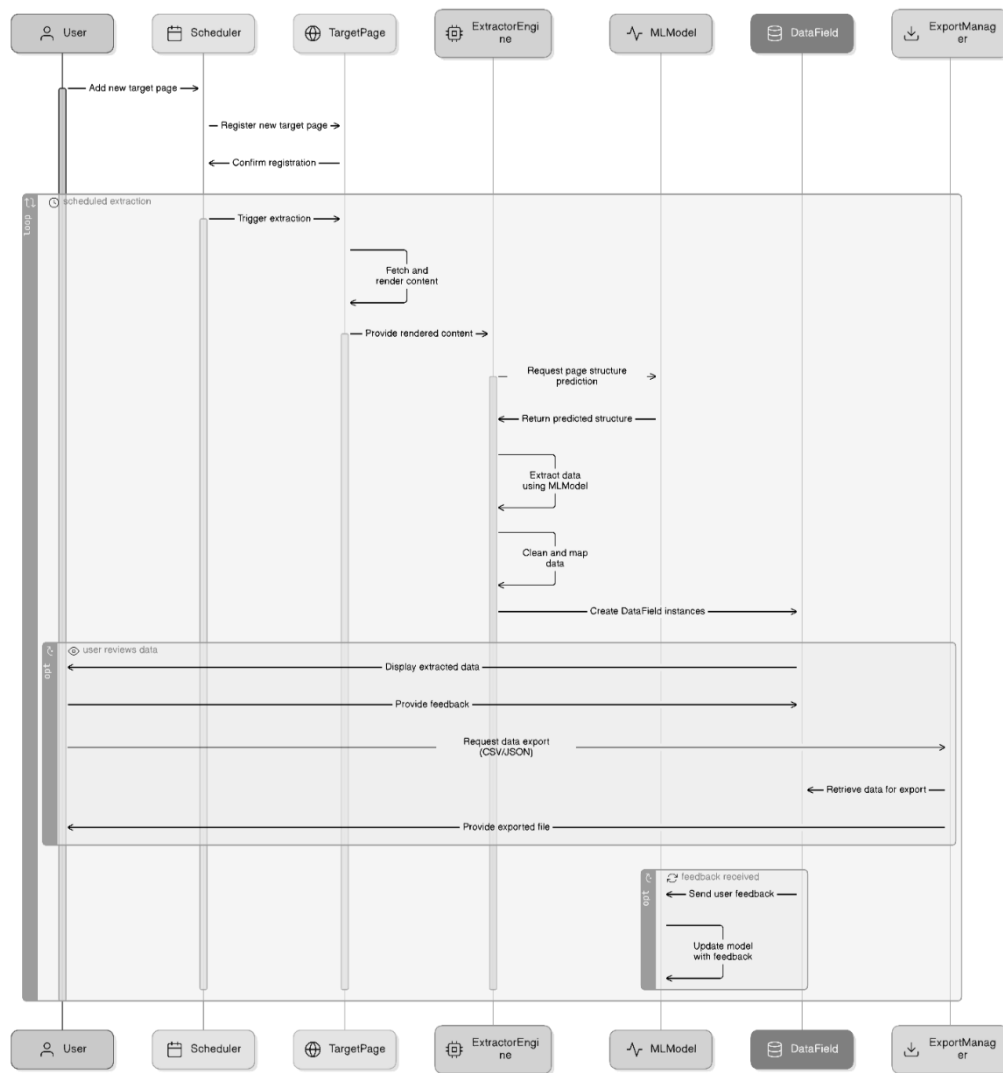
Each class is represented by a rectangle having a subdivision of three compartments name, attributes and operation



**Fig 5.6.2 : Class Diagram**

## 2. SEQUENCE DIAGRAM

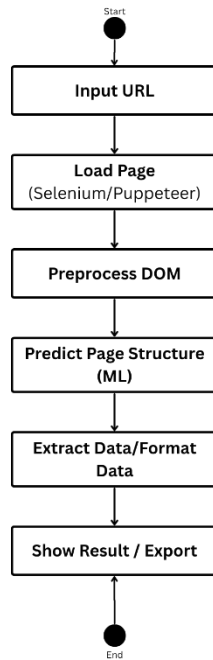
A sequence diagram is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of a message sequence chart. A sequence diagram shows object interactions arranged in time sequence. It depicts the objects and classes involved in the scenario and the sequence of message exchanged between the objects needed to carry out the functionality of the scenario.



*Fig 5.6.3 : Sequence Diagram*

### 3. ACTIVITY DIAGRAM

Activity diagram is another important diagram in UML to describe dynamic aspects of the system. It is basically a flow chart to represent the flow from one activity to another activity. The activity can be described as an operation of the system. so, the control flow is drawn from



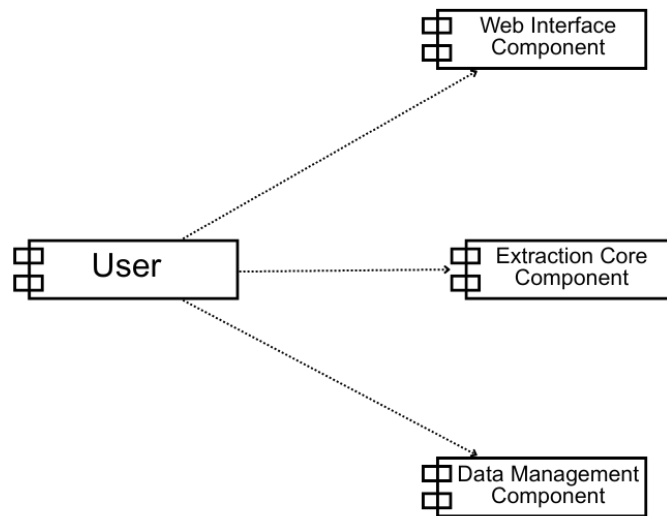
one operation to another. This flow can be sequential, branched or concurrent.

*Fig 5.6.4 : Activity Diagram*



#### 4. COMPONENT DIAGRAM

In the Unified Modelling language, a component diagram depicts how components are wired together to form larger components and are software systems. They are used to illustrate the structure of the arbitrarily complex systems. Components are wired together by using an assembly connector to connect the required interface of one component with the provided interface of another component, this illustrates the consumer service provider relationship between the two components.



*Fig 5.6.5 : Component Diagram*

## 6. SYSTEM IMPLEMENTATION

### 6.1 PYTHON INTRODUCTION

Python is a general purpose, dynamic, high level, and interpreted programming language. It supports Object Oriented programming approach to develop applications. It is simple and easy to learn and provides lots of high-level data structures. Python is easy to learn yet powerful and versatile scripting language, which makes it attractive for Application Development. Python's syntax and dynamic typing with its interpreted nature make it an ideal language for scripting and rapid application development. Python supports multiple programming pattern, including object oriented, imperative, and functional or procedural programming styles. Python is not intended to work in a particular area, such as web programming. That is why it is known as multipurpose programming language because it can be used with web, enterprise, 3D CAD, etc. We don't need to use data types to declare variable because it is dynamically typed so we can write `a=10` to assign an integer value in an integer variable. Python makes the development and debugging fast because there is no compilation step included in Python development, and edit test debug cycle is very fast.

#### Python Features

Python provides lots of features that are listed below.

- 1)Easy to Learn and Use:** Python is easy to learn and use. It is developer friendly and high-level programming language.
- 2)Expressive Language:** Python language is more expressive means that it is more understandable and readable.
- 4)Cross platform Language:** Python can run equally on different platforms such as Windows, Linux, Unix and Macintosh etc. So, we can say that Python is a portable language.
- 5)Free and Open Source:** Python language is freely available at official web address. The source code is also available. Therefore, it is open source.

**6)Object Oriented Language:** Python supports object-oriented language and concepts of classes and objects come into existence.

**7)Extensible:** It implies that other languages such as C/C++ can be used to compile the code and thus it can be used further in our python code.

**8)Large Standard Library:** Python has a large and broad library and provides rich set of module and functions for rapid application development.

**9)GUI Programming Support:** Graphical user interfaces can be developed using Python.

**10)Integrated:** It can be easily integrated with languages like C, C++, JAVA etc.

### **Python History and Versions:**

- Python laid its foundation in the late 1980s.
- The implementation of Python was started in the December 1989 by Guido Van Rossum.
- In February 1991, van Rossum published the code (labelled version 0.9.0) to outsources.
- In 1994, Python 1.0 was released with new features like: lambda, map, filter, and reduce.
- Python 2.0 added new features like: list comprehensions, garbage collection system.
- On December 3, 2008, Python 3.0 (also called "Py3K") was released. It was designed to rectify fundamental flaw of the language.
- ABC programming language is said to be the predecessor of Python language which was capable of Exception Handling and interfacing with Amoeba Operating System.
- Python programming language is being updated regularly with new features and supports. There are lots of updates in python versions, started from 1994 to current release.

## Python Applications

Python is known for its general-purpose nature that makes it applicable in almost each domain of software development. Python as a whole can be used in any sphere of development.

**1)Web Applications:** We can use Python to develop web applications. It provides libraries to handle internet protocols such as HTML and XML, JSON, Email processing, request, beautiful Soup, Feed parser etc. It also provides Frameworks such as Django, Pyramid, Flask etc to design and develop web-based applications.

**2)Desktop GUI Applications:** Python provides Tk GUI library to develop user interface in python-based application. Some other useful toolkits wxWidgets, Kivy, pyqt that are useable on several platforms. The Kivy is popular for writing multitouch applications.

**3)Software Development:** Python is helpful for software development process. It works as a support language and can be used for build control and management, testing etc.

**4)Scientific and Numeric:** Python is popular and widely used in scientific and numeric computing. Some useful library and package are SciPy, Pandas, IPython etc. SciPy is group of packages of engineering, science and mathematics.

**5)Business Applications:** Python is used to build Business applications like ERP and e commerce systems. Tryton is a high-level application platform.

**6)Console Based Application:** We can use Python to develop console-based applications. For example: IPython.

**7)Audio or Video based Applications:** Python is awesome to perform multiple tasks and can be used to develop multimedia applications. Some of real applications are: TimPlayer, cplay etc.

**8)3D CAD Applications:** To create CAD application Fandango is a real application which provides full features of CAD.

**9)Enterprise Applications:** Python can be used to create applications which can be used within an Enterprise or an Organization. Some real time applications are: Open Erp, Tryt on, Picalo etc.

## 6.2 SOURCE CODE

main.py (Revised)

```
import streamlit as st

from scrape import scrape_website, extract_body_content, clean_body_content
Removed split_dom_content import

from parse import ContentAnalyzer

import logging Added for Logging

logging.basicConfig(level=logging.INFO, format="%(asctime)s (name)s
(levelname)s %(message)s')

logger = logging.getLogger(name)

Session state initialization

if 'chat history' not in st.session_state:
    st.session_state.chat_history = []

if 'dom content' not in st.session_state:
    st.session_state.dom_content = []

if 'website url' not in st.session_state:
    st.session_state.website_url = ''

Store URL for context

Initialize Analyzer

Use a try-except block for robust initialization try:
    analyzer = ContentAnalyzer()
except:
    pass

ANALYZER_INITIALIZED = False

logger.info("ContentAnalyzer initialized successfully.")
```

```

except Exception as e:

ANALYZER INITIALIZED False

Logger.error("Failed to initialize ContentAnalyzer: (e)", exc_info=True)

#Display error prominently in the ul if initialization fails st.error(f"Fatal Error:
Could not initialize the AI Analyzer. Please check

logs and API key setup. Error: (e)", icon="")

til Configuration ---

st.set_page_config(

>

page_title="AI Web Analyst",

page_icon="",

layout="wide" # Changed to wide Layout for better content display

st.title(" AI Web Content Analyzer")

st.caption("Enter a URL, load the content, and ask questions about the page.")

# Sidebar Website Setup

with st.sidebar:

st.header(" Website Configuration")

Use text input with session state for persistence

st.session_state.website_url = st.text_input("Enter Website URL", value=st.session

state.website_url, key="website_url_input", placeholder="https://example.com")

if st.button("Load Website Content", key="load_website_btn", type="primary",

disabled=not ANALYZER_INITIALIZED):

If st.session_state.website_url and

st.session_state.website_url.startswith(('http://', 'https://'));

st.session_state.dom_content = Clear previous content st.session_state.chat_history = []

Clear history on new Load

with st.spinner("Scraping and cleaning (st.session_state.website_url)..."):

try:

```

```

logger.info("Starting scrape for:
( st.session.state.website_url)")

html = scrape_website(st.session.state.website_url)
st.session.state.dom_content = html

logger.info("Scraping done. Extracting body content...")
body_content = extract_body_content(html)

logger.info("Extracting done. Cleaning content...")

clean_body_content(body_content)
logger.info("Cleaning done. Content length:
( If st.session.state.dom_content: st.success(" Website content loaded and
cleaned!")

len(st.session.state.dom_content))")

else:

st.warning("Content loaded, but it appears empty after cleaning. The page might be
dynamic or lack text.")

scrape/init

except RuntimeError as e: Catch runtime errors from

st.error(f"Scraping Error: {e}")

logger.error(f"Scraping failed for {st.session.state.website_url}: {e}",
exc_info=True)

except Exception as e:

st.error(f"An unexpected error occurred: {str(e)}")

logger.error(f"Unexpected error during loading for {st.session.state.website_url}:
{e}", exc_info=True)

elif not st.session.state.website_url:

st.warning("Please enter a URL.")

else:

st.warning("Please enter a valid URL starting with http:// or https://")

#Display status

st.divider()

if not ANALYZER_INITIALIZED:

```

```

st.error("Analyzer Initialization Failed. Check Logs.",
icon= "")
elif st.session_state.dom_content:
st.success("Content loaded
From:\n[st.session_state.website_url]") st.info("Cleaned text length:
(len(st.session_state.dom_content)} chars")
else:
st.info("No website content loaded yet.")
Main Chat Interface
st.header("Chat with the Website Content")
Display chat history
if 'chat history' in st.session_state:
for msg in st.session_state.chat_history:
with st.chat_message(msg["role"]): st.markdown(msg["content"])
Chat Input Ensure analyzer is ready and content is loaded
chat_input_disabled not ANALYZER_INITIALIZED or not
st.session_state.dom_content
chat_placeholder "Ask about the loaded website.. chat_input_disabled else "Load a
website first..." if not
if prompt: st.chat_input(chat_placeholder, key="main_chat_input",
disabled chat_input_disabled):
Add user message to state and display it
st.session_state.chat_history.append({"role": "user", "content":
prompt))
with st.chat_message("user"):
st.markdown(prompt)
Generate and display assistant response with st.chat_message("assistant"):

```



```

message_placeholder.empty() Use placeholder for
streaming-Like effect message_placeholder.markdown(" Analyzing...")

try: with st.spinner("AI is thinking..."):
    strings)

Prepare history for the model (only content
history_for_model [msg["content"] for msg in st.session_state.chat_history if
msg["role"] != "user"] # Send

previous Assistant messages too history_for_model.extend([msg["content"] for
msg in st.session_state.chat_history if msg["role"] == "user"][:-1]) # Add previous
user messages, excluding current one.

response_analyzer.analyze_content( st.session_state.dom_content,
correctly
)

st.error
prompt,
history_for_model Poss previous messages

Display the final response
if response.startswith("A"): st.error(response) Display errors using
assistant response
else:

Do not add error messages to history as
message_placeholder.markdown (response)

Add valid assistant response to chat history
st.session_state.chat_history.append({"role":
"assistant", "content": response})

except Exception as e: logger.error("Error during analysis or display: (e)",
st.error("An unexpected error occurred during analysis:
exc_info=True)

```

(str(e)])")

but usually avoided #st.session\_state.chot\_history.append({"role":

Optionally add an error sarker to history if needed,

" assistant", "content": "Error: (str(e))"))

Content Preview Section (Optional) if st.session\_state.dom\_content:

st.divider() chars)": st.text(st.session\_state.dom\_content[:5000]

len(st.session\_state.dom\_content) > 5000 else "") ("..." if

with st.expander(" View Cleaned Text Content (First 5000

## **7. TESTING**

The implementation phase is less creative than system design. It is primarily concerned with user training, and file conversion, the system may be requiring extensive user training. The initial parameters of the system should be modifying as a result of a programming. A simple operating procedure is provided so that the user can understand the different functions clearly and quickly. The different reports can be obtained either on the inkjet or matrix printer, which is available at the disposal of the use. The proposed system is very easy to implement. In general implementation is used to mean the process of converting a new or revised system design into an operational one.

Testing is the process where the test data is prepared and is used for testing the modules individually and later the validation given for the fields. Then the system testing takes place which makes sure that all components of the system property functions as a unit. The test data should be chosen such that it passed through all possible condition. Actually, testing is the state of implementation which aimed at ensuring that the system works accurately and efficiently before the actual operation commence. The following is the description of the testing strategies, which were carried out during the testing period.

### **7.1 SYSTEM TESTING**

Testing has become an integral part of any system or project especially in the field of information technology. The importance of testing is a method of justifying, if one is ready to move further, be it to be check if one is capable to with stand the rigors of a particular situation cannot be underplayed and that is why testing before development is so critical. When the software is developed before it is given to user to use the software must be tested whether it is solving the purpose for which it is developed. This testing involves various types through which one can ensure the software is reliable. The program was tested logically and pattern of execution of the program for a set of data are repeated. Thus, the code was exhaustively checked for all possible correct data and the outcomes were also checked.

## **7.2 MODULE TESTING**

To locate errors, each module is tested individually. This enables us to detect error and correct it without affecting any other modules. Whenever the program is not satisfying the required function, it must be corrected to get the required result. Thus, all the modules are individually tested from bottom up starting with the smallest and lowest modules and proceeding to the next level. Each module in the system is tested separately. For example, the job classification module is tested separately. This module is tested with different job and its approximate execution time and the result of the test is compared with the results that are prepared manually. The comparison shows that the results proposed system works efficiently than the existing system. Each module in the system is tested separately. In this system the resource classification and job scheduling modules are tested separately and their corresponding results are obtained which reduces the process waiting time.

## **7.3 INTEGRATION TESTING**

After the module testing, the integration testing is applied. When linking the modules there may be chance for errors to occur, these errors are corrected by using this testing. In this system all modules are connected and tested. The testing results are very correct. Thus, the mapping of jobs with resources is done correctly by the system

## **8.4 ACCEPTANCE TESTING**

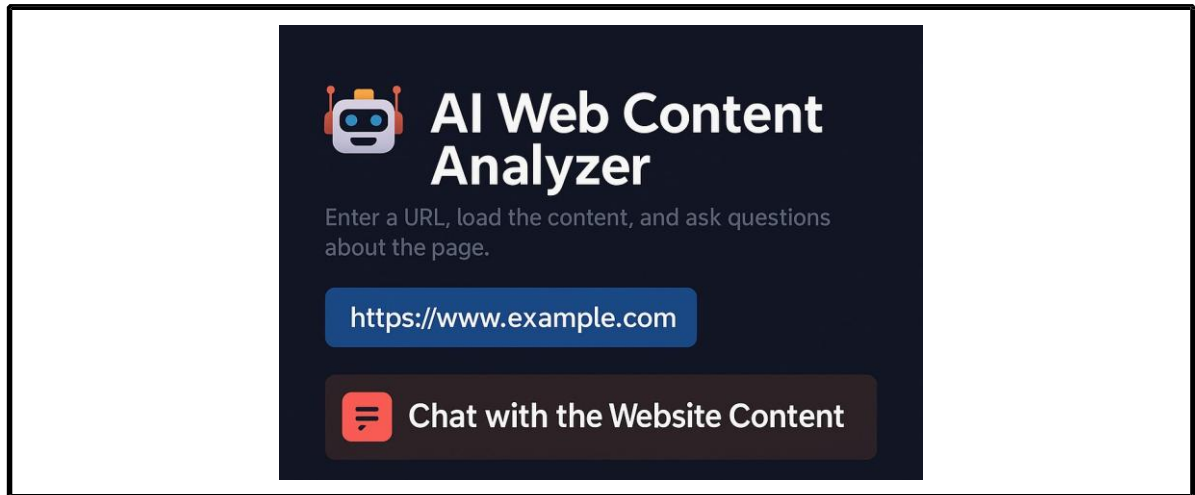
When that user find no major problems with its accuracy, the system passers through a final acceptance test. This test confirms that the system needs the original goals, objectives and requirements established during analysis without actual execution which elimination wastage of time and money acceptance tests on the shoulders of users and management, it is finally acceptable and ready for the operation.

## 7.5 TEST CASES

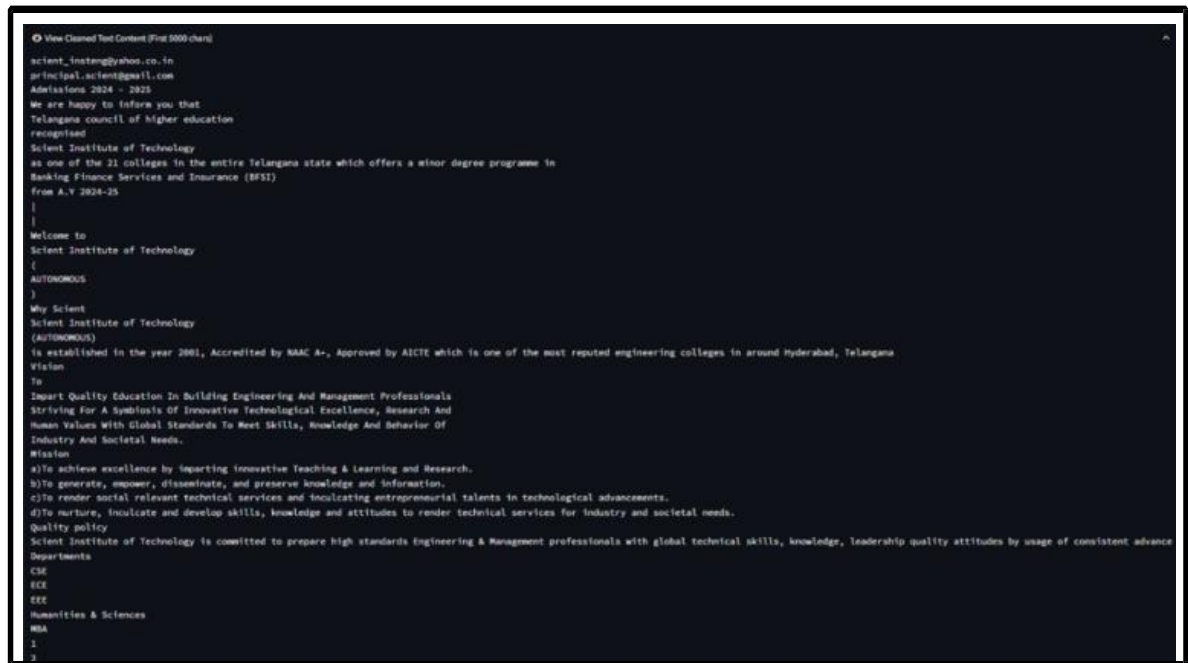
Test Case	Description	Expected Outcome
TC01	Extract price from e-commerce page	Correct numerical value
TC02	Detect CAPTCHA and respond	Skips or logs the error
TC03	Handle content inside shadow DOM	Extracts successfully
TC04	Missing field warning	Logged and flagged
TC05	HTML layout change	Auto-adjusts via LLM

## 8. DISCUSSION OF RESULTS


### SCREEN SHOTS:




*Fig 8.1: Web page interface for AI Web Content Analyzer*



*Fig 8.2: Output for the DOM data (unclean data or XML data)*


 list the main titles of this page in table format


 The provided text does not explicitly list "main titles" in a way that would allow for a clear tabular representation. However, based on the headings and prominent sections, a table approximating "main titles" can be constructed:
 

Main Title
Admissions 2024 - 2025
Welcome to Scient Institute of Technology (AUTONOMOUS)
Why Scient
Vision
Mission
Quality policy
Departments
Events
Latest News
Our Video
Our Recruiters

Note that some sections like "Why Scient" are subheadings under the main "Welcome" section. The table above attempts to represent the most prominent headings as individual "main titles".

*Fig 8.3: Output for the extracted data (clean data)*

## 9. CONCLUSION

The AI-Powered Web Scraper Development Using Python project successfully demonstrates the integration of artificial intelligence with web scraping techniques to create a more efficient and intelligent data extraction tool. By leveraging Python libraries such as Streamlit, Selenium, and LangChain, the system addresses key challenges in traditional web scraping, including handling dynamic content and bypassing website restrictions.

One of the project's significant achievements is its ability to interact with JavaScript-heavy web pages and extract relevant data intelligently. The use of Selenium for browser automation and LangChain for data interpretation enables the scraper to navigate complex websites and provide structured, meaningful results[6]. The Streamlit interface enhances user experience by allowing easy input of URLs and real-time viewing of extracted data.

The project also highlights the importance of AI in improving data quality and extraction efficiency. By incorporating machine learning models, the scraper can better understand and summarize web content, reducing the need for extensive post-processing[9]. This intelligent approach sets the foundation for future advancements in web data extraction, making the tool adaptable to various applications such as e-commerce, real estate, and academic research.

In summary, this project showcases the potential of AI-powered web scraping to revolutionize data collection. While the current prototype offers a robust solution, future work could focus on optimizing performance, enhancing AI capabilities, and expanding compatibility with diverse website structures. The successful implementation of this project underscores the transformative impact of AI in the field of data science.



## REFERENCES

- [1] Extracta.ai. (2025). AI Data Extraction Tool for Documents and Images. Retrieved from [Extracta.ai](#).
- [2] Browse AI. (n.d.). Scrape and Monitor Data from Any Website with No Code. Retrieved from [Browse AI](#).
- [3] AlgoDocs. (2024). Intelligent Document Processing - AI-Powered Document Data Extraction. Retrieved from [AlgoDocs](#).
- [4] ADASCI. (2024). AgentQL: A Hands-On Guide to AI-powered Web Data Extraction. Retrieved from [ADASCI](#).
- [5] Landing AI. (2025). Agentic Document Extraction | AI Document Intelligence. Retrieved from [Landing AI](#).
- [6] Microsoft Azure. (n.d.). Azure AI Document Intelligence. Retrieved from [Microsoft Azure](#).
- [7] Diffbot. (n.d.). Knowledge Graph, AI Web Data Extraction and Crawling. Retrieved from [Diffbot](#).
- [8] V7 Labs. (n.d.). 10 Best AI Data Extraction Tools [2024 List]. Retrieved from [V7 Labs](#).
- [9] Docsumo. (2025). Guide to Using Document AI for Data Extraction and Analysis. Retrieved from [Docsumo](#).
- [10] Taiwo Kolajo, Olawande Daramola, and Ayodele Adebisi. (2019). Big Data Stream Analysis: A Systematic Literature Review. Journal of Big Data, 1-30. Retrieved from [Journal of Big Data](#)