# 1. INTRODUCTION

In today's digital age, the extraction of data from websites has become an integral component of various industries and fields, ranging from business analytics to academic research. As the volume of online information continues to grow exponentially, the ability to efficiently gather and utilize web data has become increasingly important. Traditionally, this task has been tackled through a process known as web scraping. Web scraping involves the creation of intricate scripts designed to navigate the complexities of website structures and retrieve desired data points. These scripts must be meticulously crafted to handle the diverse and often unpredictable nature of web page designs, making web scraping a highly technical and specialized skill. For many individuals without extensive programming expertise, web scraping can be both daunting and inaccessible.

WebQuery is a groundbreaking tool that revolutionizes the landscape of web data extraction. Unlike traditional scraping methods that require detailed programming knowledge and technical skills, WebQuery employs a query-based approach. This approach is similar to the intuitive searches performed on popular search engines, allowing users to articulate their data requirements in plain language. This innovative technique eliminates the need for complex coding and technical know-how, making web data extraction accessible to a much broader audience. By leveraging advanced algorithms, WebQuery interprets these plain language queries, navigates through web page structures, and efficiently extracts the relevant data elements.

The shift towards query-based extraction not only democratizes access to web data but also significantly streamlines the entire process. Users can now obtain the data they need without the steep learning curve associated with traditional web scraping methods. WebQuery's user-friendly interface and intuitive design make it an ideal tool for individuals across various fields who need to gather web data quickly and accurately.
Moreover, WebQuery's capabilities extend beyond simple data extraction. It is designed to handle dynamically generated content and complex web layouts, which are common on modern websites. This means that WebQuery can effectively extract data from websites that use advanced technologies such as JavaScript and AJAX to generate content

dynamically. This versatility enhances WebQuery's applicability in diverse scenarios, from monitoring e-commerce prices and gathering competitive intelligence to conducting academic research and performing data-driven journalism.

WebQuery represents a significant leap forward in the field of web data extraction. By providing a more accessible, efficient, and versatile solution, WebQuery empowers users to unlock valuable insights and information from the vast expanse of the internet with unprecedented ease. This innovative tool is poised to transform how individuals and organizations interact with web data, driving advancements in various industries and enabling more informed decision-making processes. As the digital landscape continues to evolve, WebQuery stands out as a powerful ally in the quest for knowledge and data-driven insights.

WebQuery removes the barriers associated with traditional web scraping by allowing users to specify their data needs in plain language. This means that individuals who are not proficient in programming languages such as Python, JavaScript, or Ruby can still efficiently extract web data. This democratization of access opens up opportunities for a wide range of users, from small business owners to academic researchers, to leverage web data without needing to hire technical experts.

Traditional web scraping can be time-consuming, requiring significant effort to create and maintain scripts, especially when dealing with websites that frequently change their structure or use complex designs. WebQuery, on the other hand, uses advanced algorithms to interpret user queries and adapt to different web page structures, thereby significantly reducing the time and effort required to obtain the necessary data. This leads to faster data collection and the ability to quickly respond to changing information needs.

Modern websites often use JavaScript and AJAX to load content dynamically, which poses a challenge for traditional web scraping techniques that rely on static HTML structures. WebQuery is equipped to handle these dynamic elements, ensuring that users can extract data from a wider range of websites, including those with interactive and real-time content. This capability is crucial for applications such as real-time market analysis,

news aggregation, and monitoring social media trends. While WebQuery simplifies the process of data extraction, it also offers advanced features for users who need more control over the extraction process. Users can customize queries to target specific data points, filter out irrelevant information, and combine data from multiple sources. This flexibility makes WebQuery suitable for a variety of use cases, from detailed competitive intelligence reports to broad-based market research.

# 2.LITERATURE SURVEY

"Web Scraping using Natural Language Processing: Exploiting Unstructured Text for Data Extraction and Analysis," Pichiyan et al. (2023) explore the synergy between web scraping and NLP. The authors discuss how these technologies can be harnessed to extract meaningful data from the vast amount of unstructured text available online. By doing so, they provide insights into data processing and analysis techniques that can be beneficial for various applications, including education, research, and industry. the limitations and challenges of web scraping and NLP, including data completeness, processing time, and potential biases. Despite these challenges, the authors suggest that by leveraging these technologies and adhering to scientific principles, educators can enhance the relevance and effectiveness of computer science education, ultimately improving graduate employability and contributing to the advancement of the field [1].

Web scraping is a method used to extract large amounts of data from websites, while NLP involves the processing and analysis of human language data by machines. The combination of these technologies can automate the extraction and analysis of educational content, providing educators with valuable insights and resources. Lunn et al. (2020) highlight the role of web scraping and NLP in expanding knowledge within computer science education. By automating data collection from various online sources, these technologies help educators stay updated with the latest trends and information. For example, scraping job market data can reveal current demands and skills needed in the industry, which can then be integrated into the curriculum to better prepare students for their careers. Lunn et al. (2020) apply connectivism to demonstrate how web scraping and NLP can inform and improve pedagogy. By leveraging these technologies, educators can create more connected and interactive learning environments. For instance, they can curate and personalize educational content based on real-time data, fostering a more engaging and relevant learning experience for students [2].

Fernandez and Williams (2020) in their paper discuss the transformative impact of automating data extraction and analysis. By leveraging web scraping techniques, vast amounts of textual data can be efficiently harvested from websites. This automation not

only accelerates the data collection process but also ensures that educators and researchers can dedicate more time to higher-level analytical and educational activities. In the realm of computer science education, such automation is invaluable, as it allows for the rapid assembly of relevant educational materials and resources [3].

Gupta and Patel underscore the growing relevance of machine learning and its significance in providing students with a competitive advantage in the job market. They argue that educators should recognize the importance of integrating machine learning concepts into their courses to prepare students for the evolving demands of various industries. With machine learning playing a pivotal role in numerous sectors, including natural language processing, image recognition, and predictive analytics, students equipped with knowledge in this area are better positioned to meet industry expectations. The authors highlight the necessity of aligning higher education curricula with industry expectations to address the demand for practical skills, particularly in programming and testing. They emphasize the role of Python as a widely-used language in the context of machine learning and programming education. Given Python's versatility and popularity in data science, machine learning, and web development, integrating Python-based coursework can provide students with hands-on experience and relevant skills sought by employers [4].

The theoretical framework adopted by Rajan and Paul is rooted in connectivism, a learning theory that emphasizes the importance of networks and connections in learning. Through the lens of connectivism, the authors elucidate how web scraping serves as a powerful tool for extracting data from web pages, while NLP techniques enable the derivation of meaningful insights and information from this data. One of the key contributions of the paper lies in its elucidation of how the amalgamation of web scraping and NLP can inform and enrich various facets of computer science education. For instance, the authors discuss how these technologies facilitate the identification of emerging trends in the job market for computer science students. By analyzing job postings and requirements scraped from online platforms, educators can tailor their curriculum to align with industry demands, thereby equipping students with the practical skills necessary for success in their careers[5].

The paper "CETR: Content Extraction via Tag Ratios" by Tim Weninger, William H. Hsu, and Jiawei Han presents a significant advancement in the field of web content extraction. The method proposed, known as Content Extraction via Tag Ratios (CETR), is designed to efficiently and effectively differentiate between the main content of web pages and the surrounding noise, such as advertisements and navigation bars, using the structural information provided by HTML tags. Wrapper-based techniques, for instance, require custom rules for each website, making them inflexible and labor-intensive. Vision-based approaches and text density methods, while effective in some scenarios, can be computationally intensive and less accurate on cluttered pages. CETR introduces a novel technique that uses the ratio of text content to HTML tags within different segments of a web page. This method is based on the observation that meaningful content areas generally have a higher proportion of text compared to noise regions, which are often rich in HTML tags but sparse in actual content [6].

# 3.METHODOLOGY

## 3.1 WEB SCRAPING

Web scraping is an automated method used to extract large amounts of data from websites quickly. It involves fetching and parsing the HTML of a webpage to extract the desired information, which could include text, images, links, or any other data embedded in the HTML structure.

**Procedure**

- Web scraping starts with sending HTTP requests to the target website's server to retrieve the HTML content of the desired webpage. Python provides libraries like Requests to make HTTP requests easily.
- Once the HTML content is fetched, it needs to be parsed to extract the relevant data. BeautifulSoup is a popular Python library for parsing HTML and navigating the HTML DOM (Document Object Model) tree.
- After parsing the HTML, specific data elements can be extracted using various methods provided by BeautifulSoup. This could involve finding elements by tag names, class names, IDs, or other attributes.
- Often, the extracted data contains unnecessary HTML tags, extra whitespace, or other unwanted characters. Data cleaning involves removing these artifacts to obtain clean and structured data.
-

Python offers several libraries for web scraping, such as Requests for making HTTP requests, BeautifulSoup for parsing HTML, and re for regular expressions.
In the project code , the scrape_website function takes a URL as input, retrieves the webpage content using the Requests library, parses the HTML using BeautifulSoup, and extracts text from <p> (paragraph) tags. It then joins the extracted text into a single string, removes extra whitespace using regular expressions, and returns the cleaned text.

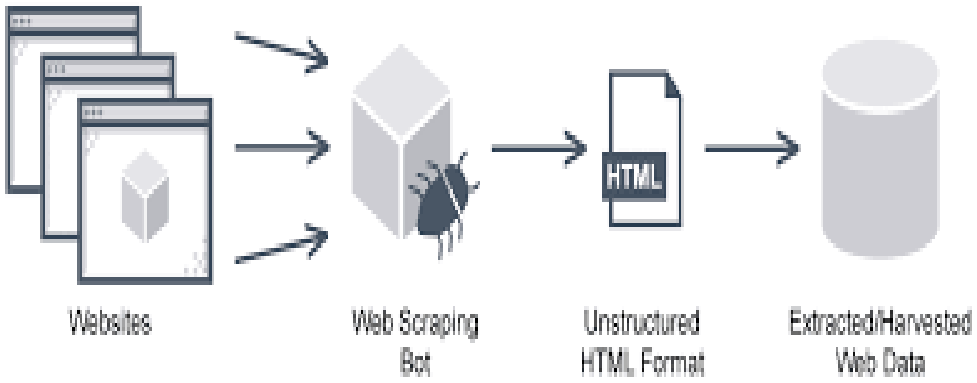


**Fig-1**:**Web Scraping Architecture**

## 3.2 NATURAL LANGUAGE PROCESSING

Natural Language Processing is one of the most important branches of artificial intelligence. It helps the computer to interpret and reciprocate human language. Millions of data are flowing around in the form of speech, text, and audio. This data can be channelized in a particular way to extract valuable information.

### 3.2.1 Text preprocessing

Text preprocessing in NLP is the process by which we clean the raw text data by removing the noise, such as punctuations, emojis, and common words, to make it ready for our model to train. It is very important to remove unhelpful data or parts from our text. This technique comes in very handy when you want to work with user-influenced data such as tweets. The sole purpose of text preprocessing in NLP is to improve the efficiency as well as the accuracy of our machine learning model. Raw text data is unstructured. Text analytics helps in converting this data into a quantitative one to extract meaningful insights and information. Natural Language Tool Kit(NLTK) is the most widely used platform for building Python programs while working with the human language. This

library simplifies the various text preprocessing steps to a whole new level. It provides a set of manifold algorithms used for Natural Language. The code snippet in the project does a process of splitting the text into sentences using regular expressions, which detect punctuation marks such as periods and question marks.

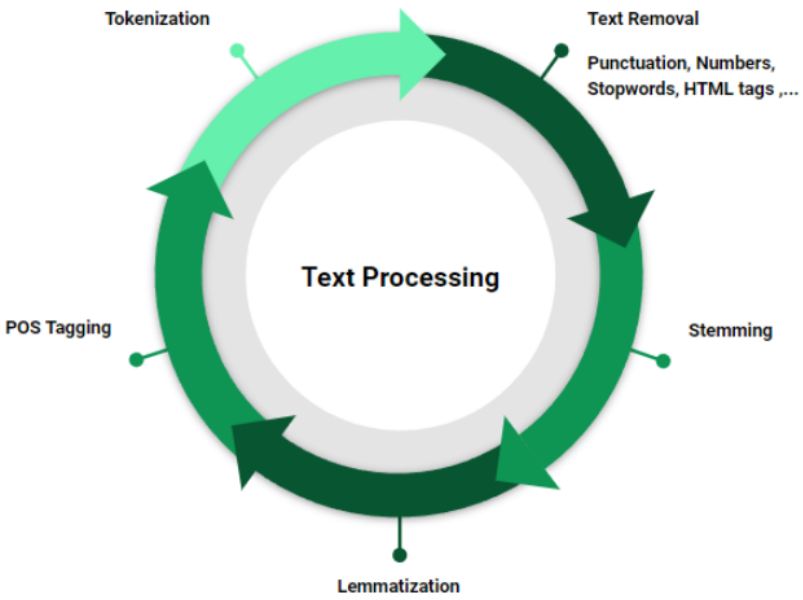


**Fig-2:Text Processing Overview**

### 3.2.2 Vectorization

Vectorization is jargon for a classic approach of converting input data from its raw format (i.e. text ) into vectors of real numbers which is the format that ML models support. This approach has been there ever since computers were first built, it has worked wonderfully across various domains, and it's now used in NLP. In Machine Learning, vectorization is a step in feature extraction. The idea is to get some distinct features out of the text for the model to train on, by converting text to numerical vectors. The vectorization technique used in the project is "TF-IDF Vectorization".

Term Frequency-Inverse Document Frequency (TF-IDF) is a statistical measure used to evaluate the importance of a word in a document relative to a collection of documents (or

corpus). The importance increases proportionally with the number of times a word appears in the document but is offset by the frequency of the word in the corpus.

Term Frequency (TF):

- Measures how frequently a term appears in a document.
- Formula:

$$\text{TF}(t , d) = \frac{number\ of\ times\ term\ t\ appears\ in\ document\ d}{total\ number\ of\ terms\ in\ document\ d}$$

Inverse Document Frequency (IDF):

- Measures how important a term is within the entire corpus.
- Formula:

$$\text{IDF}(t , D) = \log \frac{total\ number\ of\ documents}{number\ of\ documents\ containing\ terms\ t}$$

TF-IDF Score:

- Combines TF and IDF to give a balanced measure of the importance of a term within a document and the entire corpus.
- Formula : TF-IDF(t , d ,D) = TF(t , d) * IDF( t, D)

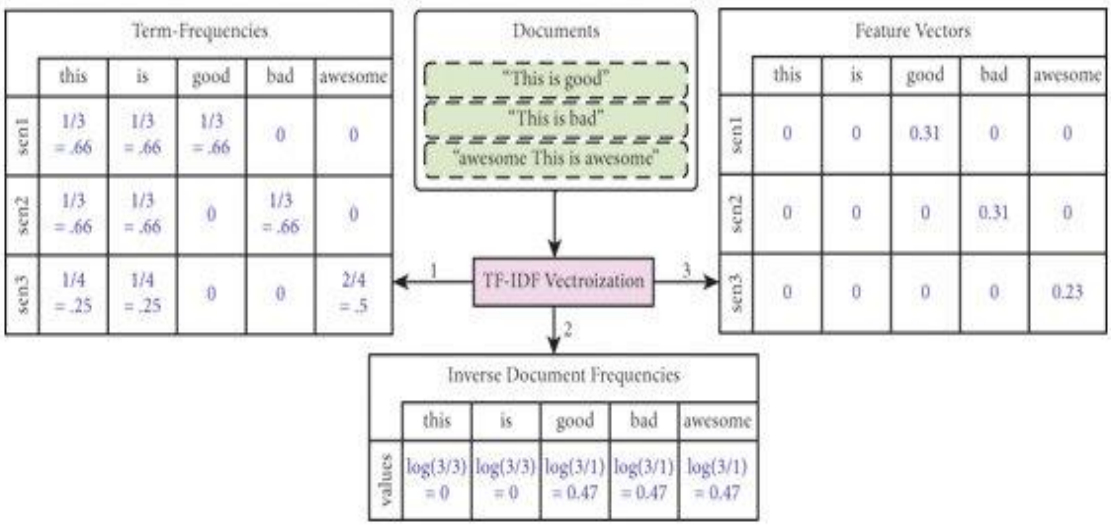


**Fig-3**: **Illustration of TF-IDF vectorization.**

### 3.3.3 Semantic Search

Semantic search is a search engine technology that interprets the meaning of words and phrases. The results of a semantic search will return content matching the meaning of a query, as opposed to content that literally matches words in the query. Semantic search is a set of search engine capabilities, which includes understanding words from the searcher's intent and their search context.

This type of search is intended to improve the quality of search results by interpreting natural language more accurately and in context. Semantic search achieves this by matching search intent to semantic meaning with the help of technologies such as machine learning and artificial intelligence.

Key Components and Steps:

- Transform the Question into a Vector: The question is converted into a TF-IDF vector using the previously trained TfidfVectorizer.
- Retrieve Stored Vectors: The function connects to an SQLite database where TF-IDF vectors of sentences are stored. It retrieves these vectors and deserializes them for comparison.
- Calculate Cosine Similarity: Cosine similarity is calculated between the question vector and each stored vector to measure how similar they are.
- Identify the Most Similar Sentence: The function identifies the stored sentence with the highest similarity score. If the highest similarity score is below a specified threshold, it returns a default response indicating no relevant match was found. Otherwise, it returns the most similar sentence.

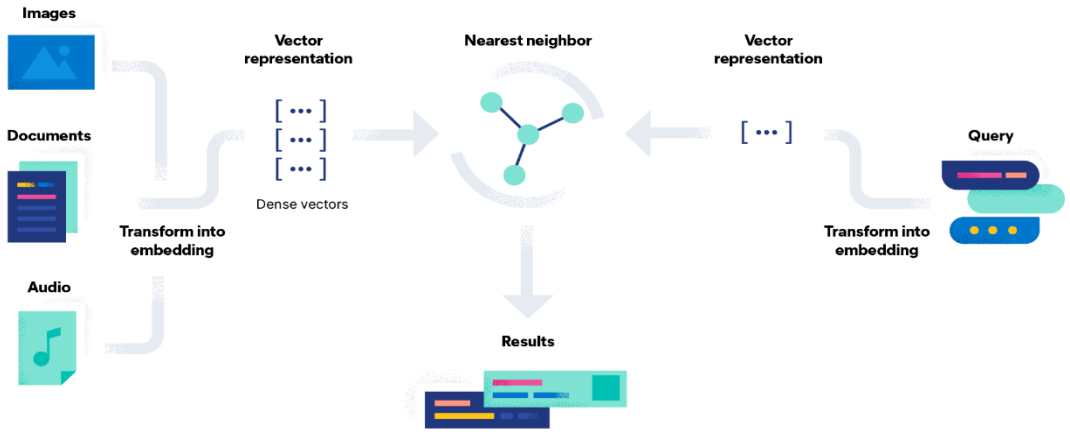


**Fig-4** : **Semantic Search Architecture**

## 3.3 Large Language Model

GPT-2 is a large transformer-based language model with 1.5 billion parameters, trained on a dataset[1] of 8 million web pages. GPT-2 is trained with a simple objective: predict the next word, given all of the previous words within some text. The diversity of the dataset causes this simple goal to contain naturally occurring demonstrations of many tasks across diverse domains. GPT-2 is a direct scale-up of GPT, with more than 10X the parameters and trained on more than 10X the amount of data.

The llm models used leverages a pre-trained language model, specifically GPT-2, to generate a response to a user's query. This function is part of the Webquery application and is responsible for providing coherent and contextually appropriate answers based on the input question. This helps in providing more elaborate and human-like answers compared to basic keyword matching or semantic search techniques.

Loading the Pre-trained Model and Tokenizer:

- Transformers Library:

The transformers library by Hugging Face provides easy access to pre-trained models and tokenizers for natural language processing tasks.

GPT-2 Tokenizer:

- Tokenizer:

Converts text into numerical tokens that the model can process.

Model: GPT-2, a generative language model that predicts the next word in a sequence of words, thereby generating coherent text.

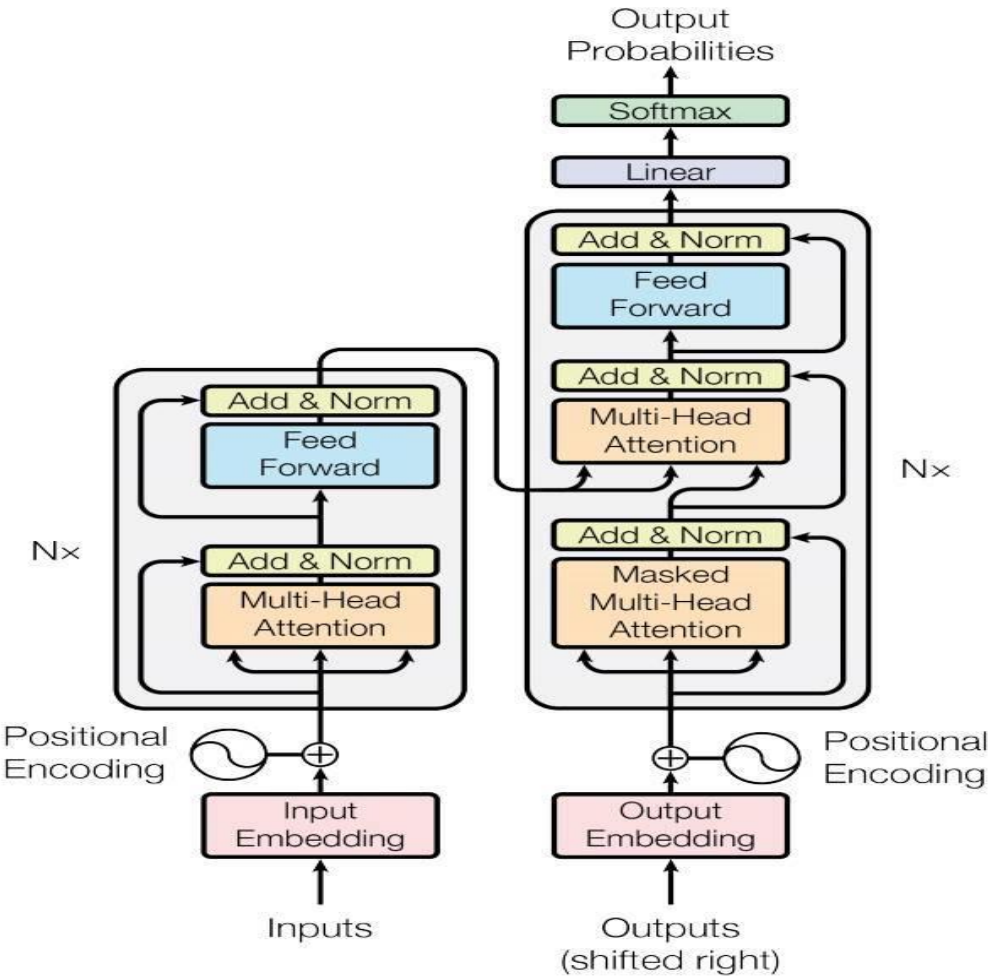


**Fig-5** : **GPT2 LMHead Model Architecture**

## 3.4 Streamlit

Streamlit is an open-source Python library that makes it easy to create web applications for machine learning, data science, and other data-related tasks. With Streamlit, you can build interactive web apps quickly and easily using simple Python scripts.

Streamlit allows you to create web applications directly from Python scripts without having to write HTML, CSS, or JavaScript code. You can use Streamlit to create interactive data visualizations, dashboards, and other web interfaces for your data analysis projects. In the context of the Webquery application, capturing the website URL and user queries, and displaying the conversation history in a chat-like interface, are crucial steps for facilitating user interaction.

Process Steps:

1. User Input for Website URL:

- The application uses Streamlit's sidebar to provide a text input field for the user to enter the website URL.
- A submit button is used to trigger the scraping and vectorization processes once the URL is entered.

2. User Input for Queries:

- A text input field is provided within the main interface of the application where users can type their questions.

## 4.1 WORK FLOW:

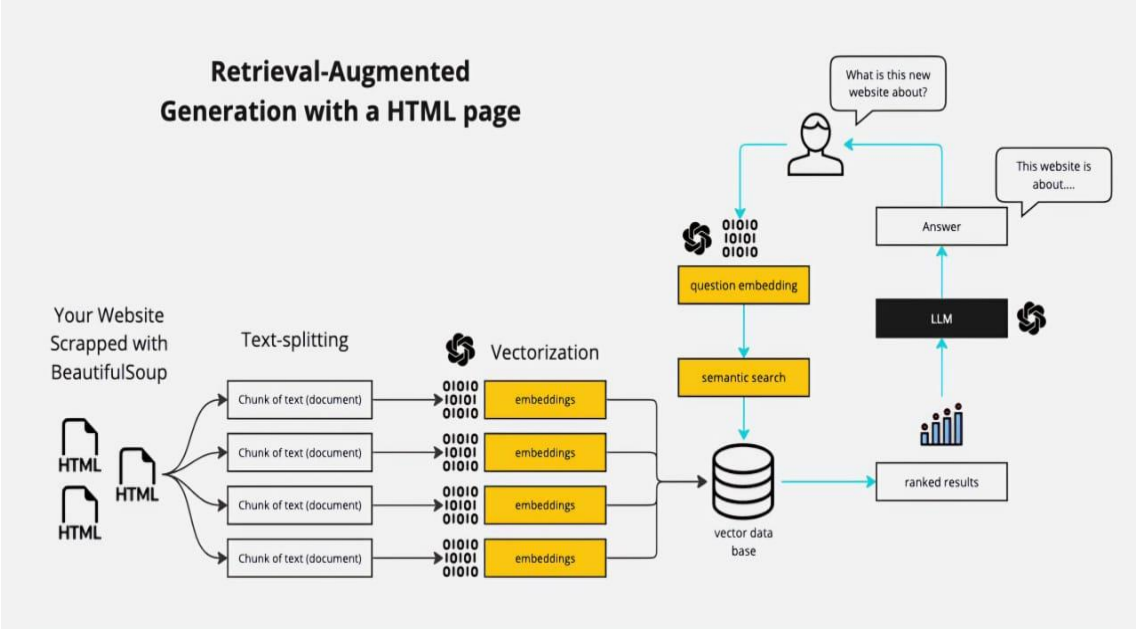


**Fig-6: workflow of the project**

The workflow of the Retrieval-Augmented Generation (RAG) system is a multi-step process designed to extract information from a website and generate relevant responses to user queries. Each step plays a crucial role in ensuring that the system can provide accurate and contextually relevant answers. Below is a detailed elaboration of the workflow:

- Web Scraping with BeautifulSoup: The process begins by scraping the content of a website using BeautifulSoup. This involves extracting HTML content and converting it into raw text data.

- Text Splitting: The scraped text is then divided into manageable chunks or documents. Each chunk represents a portion of the original content, facilitating easier processing in subsequent steps.

- Vectorization: Each chunk of text is transformed into a vector representation through a process known as vectorization. This step involves converting the text into numerical embeddings that capture the semantic meaning of the content.

- Storage in Vector Database: The generated embeddings are stored in a vector database. This database allows for efficient retrieval and comparison of vectors based on their semantic similarity.

- User Query and Question Embedding: When a user poses a question, it is also converted into an embedding. This question embedding captures the semantic meaning of the user's query in a similar vector format.

- Semantic Search: The question embedding is used to perform a semantic search against the embeddings stored in the vector database. This search identifies the most relevant chunks of text that match the semantic content of the user’s query.

- Ranked Results: The results of the semantic search are ranked based on their similarity to the question embedding. This ranking helps in selecting the most pertinent pieces of information from the database.

- Language Model (LLM) Processing: The ranked results are then fed into a large language model (LLM), which processes the information to generate a coherent and contextually relevant answer.

- Answer Generation: Finally, the generated answer is presented to the user in response to their query. This answer is constructed based on the most relevant information retrieved from the vector database and refined by the language model.

  Finally, the workflow integrates web scraping, text processing, vectorization, semantic search, and advanced language model capabilities to provide accurate and relevant answers to user queries based on the content of a given website.

## Code Implimentation

**# Import all the packages**

```
import os
import requests
from bs4 import BeautifulSoup
import re
from sklearn.feature_extraction.text import TfidfVectorizer
import sqlite3
import pickle
from sklearn.metrics.pairwise import cosine_similarity
import numpy as np
from transformers import GPT2Tokenizer, GPT2LMHeadModel, AutoTokenizer,
AutoModelForSeq2SeqLM
import streamlit as st
from langchain_core.messages import AIMessage, HumanMessage
```

**#Web Scraping**

```
def scrape_website(url):
    try:
        response = requests.get(url)
        soup = BeautifulSoup(response.content, 'html.parser')
        paragraphs = soup.find_all('p')
        extracted_text = [paragraph.get_text() for paragraph in paragraphs]

        joined_text = " ".join(extracted_text)
        cleaned_text = re.sub(r'\s+', ' ', joined_text).strip()
        return cleaned_text
    except Exception as e:
        st.error(f"An error occurred while scraping the website: {e}")
        return None
```

```
# Text Processing
def split_text(text):
    sentences = re.split(r'(?<!\w\.\w.)(?<![A-Z][a-z]\.)(?<=\.|\?)\s', text)
    return sentences

# Vectorization
def vectorize_data(sentences):
    try:
        vectorizer = TfidfVectorizer()
        tfidf_vectors = vectorizer.fit_transform(sentences)
        return tfidf_vectors, vectorizer
    except Exception as e:
        st.error(f"An error occurred during vectorization: {e}")
        return None, None

# Database Management
def store_vectors_in_database(tfidf_vectors, sentences):
    try:
        with sqlite3.connect('vector_database.db') as conn:
            cursor = conn.cursor()
            cursor.execute('''DROP TABLE IF EXISTS tfidf_vectors''')
            cursor.execute('''CREATE TABLE IF NOT EXISTS tfidf_vectors (
                                id INTEGER PRIMARY KEY,
                                sentence TEXT,
                                vector BLOB
                            )''')
            for i, vector in enumerate(tfidf_vectors):
                sentence = sentences[i]
                vector_serialized = pickle.dumps(vector)
                cursor.execute('''INSERT INTO tfidf_vectors (sentence, vector)
                                  VALUES (?, ?)''', (sentence, vector_serialized))
        st.write("TF-IDF vectors stored in database successfully.")
```

```
    except Exception as e:
        st.error(f"An error occurred while storing vectors in the database: {e}")

# Keyword Matching
def keyword_matching(question, sentences):
    matching_sentences = []
    for sentence in sentences:
        if re.search(r'\b{}\b'.format(re.escape(question)), sentence, re.IGNORECASE):
            matching_sentences.append(sentence)
    return matching_sentences

# Semantic Search
def semantic_search(question, vectorizer):
    try:
        question_vector = vectorizer.transform([question])
        with sqlite3.connect('vector_database.db') as conn:
            cursor = conn.cursor()
            cursor.execute('''SELECT * FROM tfidf_vectors''')
            rows = cursor.fetchall()
            tfidf_vectors = [pickle.loads(row[2]) for row in rows]

        similarities = [cosine_similarity(question_vector, v)[0][0] for v in      tfidf_vectors]
        max_similarity_index = np.argmax(similarities)
        most_similar_sentence = rows[max_similarity_index][1]

        return most_similar_sentence
    except Exception as e:
        st.error(f"An error occurred during semantic search: {e}")
        return None

# Large Language Model
def use_llm(question, max_response_length):
```

```
    try:
        tokenizer = GPT2Tokenizer.from_pretrained("gpt2")
        model = GPT2LMHeadModel.from_pretrained("gpt2")

        inputs=tokenizer.encode(question,return_tensors="pt",
            max_length=tokenizer.model_max_length, truncation=True)
        max_length = min(tokenizer.model_max_length, len(inputs[0]) +
            max_response_length)

        output = model.generate(inputs, max_length=max_length,
            pad_token_id=tokenizer.eos_token_id)
        response = tokenizer.decode(output[0], skip_special_tokens=True)

        return response
    except Exception as e:
        st.error(f"An error occurred during GPT-2 response generation: {e}")
        return None
```

**# Summerization of response**

```
def give_answer(answer):
    tokenizer = AutoTokenizer.from_pretrained("Mr-Vicky-01/Bart-Finetuned-
conversational-summarization")
    model = AutoModelForSeq2SeqLM.from_pretrained("Mr-Vicky-01/Bart-
Finetuned-conversational-summarization")

    def generate_summary(answer):
        inputs = tokenizer([answer], max_length=1024, return_tensors='pt',
            truncation=True)
        summary_ids = model.generate(inputs['input_ids'], max_new_tokens=100,
do_sample=False)
        summary = tokenizer.decode(summary_ids[0], skip_special_tokens=True)
        return summary
```

```
    summary = generate_summary(answer)
    return summary

# Streamlit Interface
st.set_page_config(page_title="Webquery: Chat with Website", page_icon=":books:")
st.title("Webquery: Chat with Website")

if "chat_history" not in st.session_state:
    st.session_state.chat_history = [
        AIMessage(content="Hello, I am a bot. How can I help you?"),
    ]

if "sentences" not in st.session_state:
    st.session_state.sentences = []

if "vectorizer" not in st.session_state:
    st.session_state.vectorizer = None

with st.sidebar:
    st.header("Website URL !!!")
    website_url = st.text_input("Enter the website URL .....")
    submit = st.button("Submit")

if submit:
    try:
        if os.path.exists('vector_database.db'):
            os.remove('vector_database.db')

        text = scrape_website(website_url)
        st.session_state.sentences = split_text(text)
```

```
        tfidf_vectors, st.session_state.vectorizer =
vectorize_data(st.session_state.sentences)
        store_vectors_in_database(tfidf_vectors, st.session_state.sentences)
    except Exception as e:
        st.error(f"An error occurred: {e}")

user_query = st.text_input("Type your question here....")

if user_query and user_query.strip() and st.session_state.vectorizer:
    matching_sentences = keyword_matching(user_query, st.session_state.sentences)
    most_similar_sentence = semantic_search(user_query, st.session_state.vectorizer)

    if most_similar_sentence:
        response = use_llm(most_similar_sentence,
max_response_length=len(user_query.split()) * 5)
        answer = give_answer(response)
        st.session_state.chat_history.append(HumanMessage(content=user_query))
        st.session_state.chat_history.append(AIMessage(content=answer))

def render_chat_message(message, icon):
    st.markdown(
        f"""
        <div style="display: flex; align-items: center; margin-bottom: 1rem;">
            <img src="{icon}" alt="icon" style="width: 50px; height: 50px; margin-right:
1rem;">
            <div>{message}</div>
        </div>
        """, unsafe_allow_html=True
    )

human_icon_url="
AADhCAMAAAAJbSJIAAAAolBMVEUALFb//// =="
```

```
ai_icon_url="https://encryptedtbn0.gstatic.com/images?q=tbn:ANd9GcRdXaPfUob7UJ
WACFD-DKjSHuxCpJ0tg-uDmw&s"

for message in st.session_state.chat_history:
    if isinstance(message, AIMessage):
        render_chat_message(message.content, ai_icon_url)
    elif isinstance(message, HumanMessage):
        render_chat_message(message.content, human_icon_url)
```

# Results & Discussions

The project involves building a web application named Webquery that allows users to interact with the content of a specified website through a chat interface. The system performs web scraping, text processing, semantic search, and generates responses using a language model.

The project successfully implemented web scraping functionality using requests and BeautifulSoup to extract textual data from a given website URL. The extracted data, which consisted of paragraphs, was cleaned and joined into a coherent string for further processing. The extracted text was effectively split into individual sentences using a regular expression-based method. This segmentation is crucial for downstream processing, such as vectorization and semantic search. The TF-IDF vectors along with their corresponding sentences were successfully stored in an SQLite database. incorporated a semantic search functionality that compared the TF-IDF vectors of the user query with the stored sentence vectors using cosine similarity. The system identified the most semantically similar sentence to the query and returned it as a potential answer.it integrated a BART model fine-tuned for conversational summarization to provide concise and relevant answers.

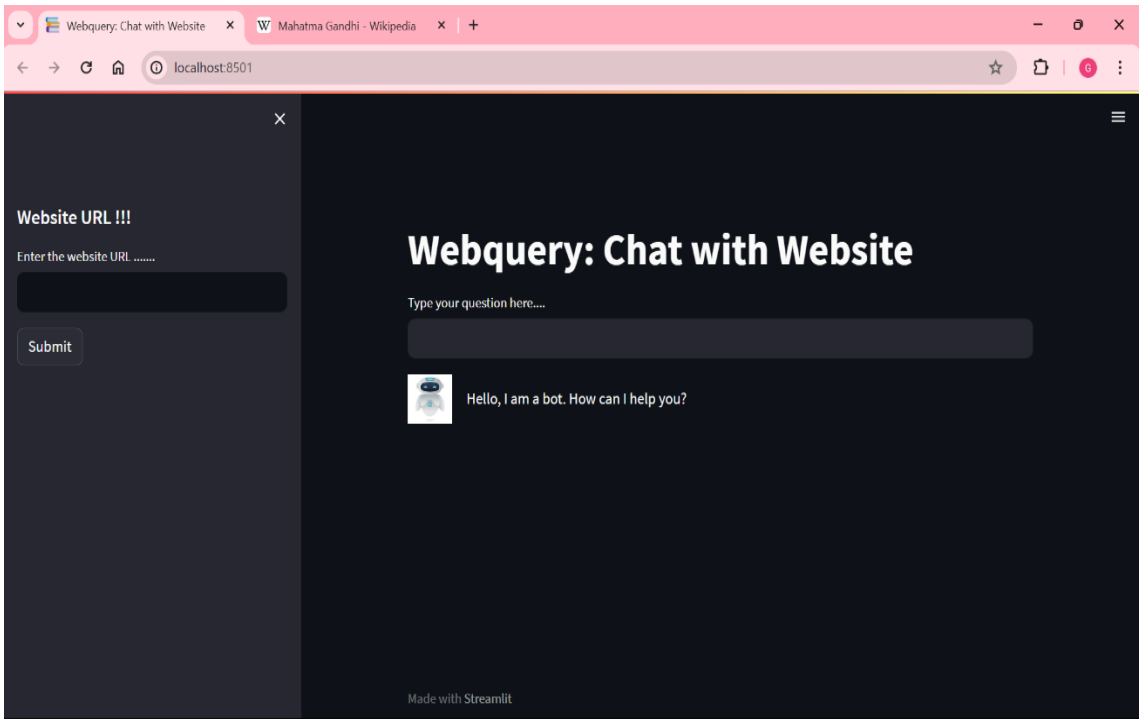


**Fig-7: Initial Interface of Project**

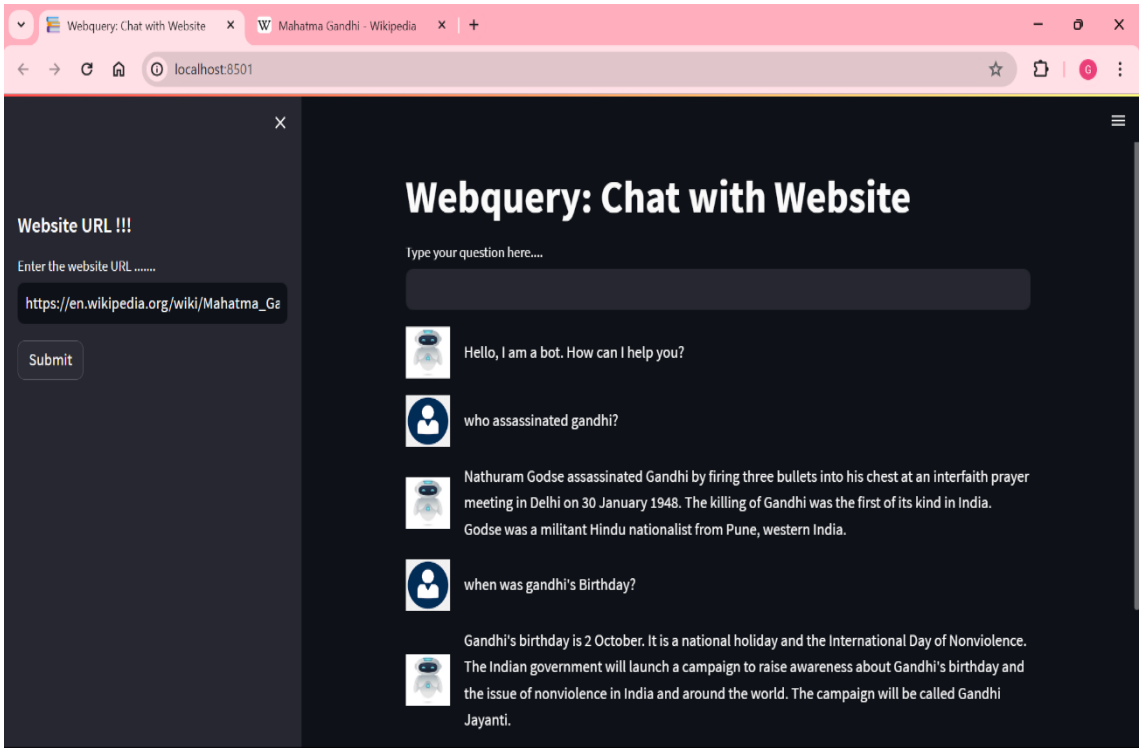


**Fig-8 : execution of the project**

- In the above figure , The user interface, developed using Streamlit, provides a clean and intuitive platform for interacting with the web scraping and question-answering system.
- Users can easily input a website URL and submit it for processing. The chat interface allows users to type their questions and receive answers in real-time.
- The interface displays both user queries and AI responses, mimicking a conversational chat experience.
- In the provided example, the user enters the URL of the Mahatma Gandhi Wikipedia page and submits it.
- The system processes the URL, scrapes the content, and prepares it for interaction.
- The AI successfully responds to user queries based on the scraped website content. For instance, when asked "who assassinated Gandhi?" the system correctly identifies Nathuram Godse as the assassin and provides additional context. Similarly, when asked "when was Gandhi's Birthday?" the AI provides the date and relevant details about the significance of the day.

## Conclusion

The "Webquery: Chat with Website" project represents a sophisticated integration of cutting-edge technologies in machine learning and natural language processing (NLP) to offer users an intuitive platform for engaging with web content. At its core, the project seamlessly blends together multiple components, including web scraping, text processing, vectorization, and language models, to construct an interactive chatbot capable of understanding and responding to user queries effectively. project successfully integrates various machine learning and natural language processing techniques to provide a user-friendly interface for interacting with web content. This project demonstrates the power of combining web scraping, text processing, vectorization, and language models to create a responsive chatbot. Users can input a URL, have the website's content processed and stored, and then ask questions about the content, receiving relevant and coherent answers

## Future Scope

Improve the web scraping function to handle a wider range of website structures and incorporate advanced techniques for dealing with dynamic content. Incorporate more advanced natural language processing techniques, such as named entity recognition (NER) and sentiment analysis, to provide richer context and insights in responses. Explore other vectorization techniques like word embeddings (Word2Vec, GloVe) or contextual embeddings (BERT, RoBERTa) to improve the accuracy of semantic search.Moreover use all modern technologies and models to increase the accuracy of getting accurate and relevant answers to the user queries.

# REFERENCES

**[1]** Pichiyan, V., Muthulingam, S., Sathar, G., Nalajala, S., Ch, A., & Das, M. N. (2023). Web Scraping using Natural Language Processing: Exploiting Unstructured Text for Data Extraction and Analysis. Procedia Computer Science, 230, 193-202..

**[2]** Lunn, S., Zhu, J., & Ross, M. (2020, October). Utilizing web scraping and natural language processing to better inform pedagogical practice. In 2020 IEEE Frontiers in Education Conference (FIE) (pp. 1-9). IEEE.

**[3]** Anisha, P. R., Nguyen, N. G., & Sreelatha, G. (2021, November). A Text Mining using Web Scraping for Meaningful Insights. In Journal of Physics: Conference Series (Vol.2089,No.1,p.012048).IOPPublishing.

**[4]** Uzun, E. (2020). A novel web scraping approach using the additional information obtained from web pages. IEEE Access, 8, 61726-61740.

**[5]** Patnaik, S. K., Babu, C. N., & Bhave, M. (2021). Intelligent and adaptive web data extraction system using convolutional and long short-term memory deep learning networks. Big Data Mining and Analytics, 4(4), 279-297.

**[6]** Ferrara, E., De Meo, P., Fiumara, G., & Baumgartner, R. (2014). Web data extraction, applications and techniques: A survey. Knowledge-based systems, 70, 301-323.

**[7]** Campos Macias, N., Düggelin, W., Ruf, Y., & Hanne, T. (2022). Building a technology recommender system using web crawling and natural language processing Technology. Algorithms, 15(8), 272.

**[8]** Bird, S. (2006, July). NLTK: the natural language toolkit. In Proceedings of the COLING/ACL 2006 Interactive Presentation Sessions (pp. 69-72).

**[9]** Ananthi, K., & Goldena, N. J. (2022). Extraction and Retrieval of Web based Content in Web Engineering.

**[10]** Weninger, T., Hsu, W. H., & Han, J. (2010, April). CETR: content extraction via tag ratios. In Proceedings of the 19th international conference on World wide web (pp. 971-980).

**[11]** Tan, Z., He, C., Fang, Y., Ge, B., & Xiao, W. (2018). -Based Extraction of

News Contents for Text Mining. IEEE Access, 6, 64085-64095.

**[12]** Fayzrakhmanov, R. R., Sallinger, E., Spencer, B., Furche, T., & Gottlob, G. (2018, April). Browserless web data extraction: challenges and opportunities. In Proceedings of the 2018 World Wide Web Conference (pp. 1095-1104).

**[13]** Gunasundari, R., & Karthikeyan, S. (2012). A study of content extraction from web pages based on links. International Journal of Data Mining & Knowledge Management Process, 2(3), 23.

**[14]** Lotfi, C., Srinivasan, S., Ertz, M., Latrous, I., & Manjushree, S. (2021). Web Scraping Techniques and Applications: A Literature Review. In *SCRS Conference Proceedings on Intelligent Systems. SCRS, India* (pp. 381-394).

**[15]** Zeleny, J., Burget, R., & Zendulka, J. (2017). Box clustering segmentation: A new method for vision-based web page preprocessing. Information Processing & Management, 53(3), 735-750..

**[16]** Yang, D., & Song, J. (2010, October). Web content information extraction approach based on removing noise and content-features. In 2010 International conference on web information systems and mining (Vol. 1, pp. 246-249). IEEE.

**[17]** Pan, D., Qiu, S., & Yin, D. (2008, October). Web page content extraction method based on link density and statistic. In 2008 4th International Conference on Wireless Communications, Networking and Mobile Computing (pp. 1-4). IEEE.

**[18]** Buttler, D., Liu, L., & Pu, C. (2001, April). A fully automated object extraction system for the World Wide Web. In Proceedings 21st International Conference on Distributed Computing Systems (pp. 361-370). IEEE.

**[19]** Cai, D., Yu, S., Wen, J. R., & Ma, W. Y. (2003). Extracting content structure for web pages based on visual representation. In Web Technologies and Applications: 5th Asia-Pacific Web Conference, APWeb 2003, Xian, China, April 23–25, 2003 Proceedings 5 (pp. 406-417). Springer Berlin Heidelberg.

**[20]** Martin, J. H. (2009). Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition.

**[21]** Pichiyan, V., Muthulingam, S., Sathar, G., Nalajala, S., Ch, A., & Das, M. N. (2023). Web Scraping using Natural Language Processing: Exploiting Unstructured Text for Data Extraction and Analysis. Procedia Computer

Science, 230, 193-202..

**[22]** Burstein, J., Sabatini, J., & Shore, J. (2014). Natural language processing for educational applications.

**[23]** Khurana, D., Koli, A., Khatter, K., & Singh, S. (2023). Natural language processing: State of the art, current trends and challenges. Multimedia tools and applications, 82(3), 3713-3744.

**[24]** Matta, P., Sharma, N., Sharma, D., Pant, B., & Sharma, S. (2020). Web scraping: Applications and scraping tools. International Journal of Advanced Trends in Computer Science and Engineering, 9(5), 8202-8206.

**[25]** B. Liu, R. L. Grossman and Y. Zhai, "Mining data records in Web pages", L. Getoor, T. Senator, P. Domingos, C. Faloutsos, Proc. Of the Int'l Conf. on Knowledge Discovery and Data Mining (KDD 2003), Washington, ACM Press, (2003), pp. 601-606.

**[26]** C. H. Chang, M. Kayed, M. R. Girgis and K. F. Shaalan, "A survey of Web information extraction systems", IEEE Trans. on Knowledge and Data Engineering, vol. 18, no. 10, (2006), pp. 1411-1428.

**[27]** G. Weikum and M. Theobald, "From information to knowledge: Harvesting entities and relationships from Web source", Paredaens J, Van Gucht D, eds. Proc. of the 29th ACM SIGMOD-SIGACT-SIGART Symp. on Principles of Database Systems (PODS 2010).ACM Press, (2010), pp. 65-76, [doi: 10.1145/1807085.1807097]

**[28]** W. Huang, L. Zhang, J. Zhang, M. Zhu, "Focused Crawling for Retrieving E-commerce Information Based on Learnable Ontology and Link Prediction", IEEC, International Symposium on Information Engineering and Electronic Commerce, (2009), pp. 574-579.

**[29]** I. Muslea, S. Minton and C. Knoblock, "Hierarchical wrapper induction for semi-structured information sources", Autonomous Agents and Multi-Agent Systems, vol. 4, no. 1-2, (2001), pp. 93-114.

**[30]** O. Etzioni, M. Cafarella, D. Downey, S. Kok, A.-M. Popescu, T. Shaked, S. Soderland, D. S. Weld and A. Yates, "Web-scale information extraction in KnowItAll (preliminaryresults)", Proceedings of the 13th World Wide Web Conference, (2004), pp. 100-109.