SMART WEB EXPLORE

(AI-Powered Chatbot Built with LangChain and Streamlit)

A project submitted to the Bharathidasan University in partial fulfillment of the requirements for the award of the Degree of

MASTER OF SCIENCE IN DATA SCIENCE

Submitted by

RAGAVI S
235229149

Under the guidance of
Mr. P. Velmurugan, M.Sc., M.Phil., B.Ed., SET., NET.,
Assistant Professor

DEPARTMENT OF DATA SCIENCE
BISHOP HEBER COLLEGE
(AUTONOMOUS)
"Nationally Reaccredited at A++ Grade with a CGPA of 3.69 out of 4 in NAAC IV Cycle"
(Recognized by UGC as "College of Excellence")
(Affiliated to Bharathidasan University)
TIRUCHIRAPPALLI-620017

March – 2025

DECLARATION

I hereby declare that the project work presented is originally done by me under the guidance of **Mr. P. Velmurugan, M.Sc., M.Phil., B.Ed., SET., NET., Assistant Professor, Department of Data Science, Bishop Heber College (Autonomous), Tiruchirappalli620 017** and has not been included in any other project submitted for any other degree.

Name of the Candidate : **RAGAVI S**

Register Number : **235229149**

Batch : **2023-2025**

Signature of the Candidate

Date:

Mr. P. Velmurugan, M.Sc., M.Phil., B.Ed., SET., NET.,
Assistant Professor,
Department of Data Science,
Bishop Heber College (Autonomous),
Tiruchirappalli – 620017

Date:

CERTIFICATE

This is to certify that the project work entitled **"SMART WEB EXPLORE" (AI-Powered Chatbot Built with LangChain and Streamlit)** is a bonafide record work done by **RAGAVI S, 235229149** in partial fulfillment of the requirements for the award of the degree of **MASTER OF SCIENCE IN DATA SCIENCE** during the period **2023 - 2025.**

Place: **Signature of the Guide**

DEPARTMENT OF DATA SCIENCE
BISHOP HEBER COLLEGE (AUTONOMOUS),
"Nationally Reaccredited at A++ Grade with a CGPA of 3.69 out of 4 in NAAC IV Cycle"
(Recognized by UGC as "College of Excellence")
(Affiliated to Bharathidasan University)
TIRUCHIRAPPALLI - 620017

Date:

Course Title: Project with Viva-voce **Course: P23DS4PJ**

The Viva-Voce examination for the candidate **RAGAVI S, 235229149** was held on _____.

Signature of the Guide **Signature of the HOD**

Examiners:

1.

2.

ACKNOWLEDGEMENTS

I take this opportunity to express my thanks to **Dr. J. Princy Merlin M.Sc., M.Phil., Ph.D.,** Principal, Bishop Heber College, Trichy.

I would like to express my sincere gratitude and deep thanks to **Dr. JEMIMA PRIYADARSINI R MCA., M.Phil., Ph.D.,** Associate Professor, Head of Department of Data Science, and Internal Guide, who has been a source of encouragement, guidance, and moral strength throughout our study period. She has been the backbone of my project, and her support helped me in completing it successfully.

My deepest appreciation goes to my internal guide, **Mr. P. Velmurugan, M.Sc., M.Phil., B.Ed., SET., NET.,** Assistant Professor, Department of Computer Science. Her expertise, patience, and constructive feedback played a crucial role in the successful completion of this project. She has been the backbone of my work, and her encouragement and moral support have been instrumental in this journey.

I am also profoundly grateful to my family members and dear friends, whose constant support, motivation, and assistance have helped me overcome challenges and successfully complete this project.

RAGAVI S

ABSTRACT

In the digital age, businesses and organizations face challenges in providing real-time customer support, personalized interactions, and scalable assistance on their websites. Traditional support methods, such as email or human-operated live chat, often lead to delays, high operational costs, and inconsistent user experiences. As customer expectations for instant responses grow, there is a need for an AI-driven chatbot that can engage users, answer queries, and enhance website interactions efficiently. Advances in Natural Language Processing (NLP) and Machine Learning (ML) have enabled the development of intelligent chatbots capable of handling complex conversations, learning from interactions, and providing contextual responses. This project is a LangChain-powered chatbot with a Streamlit GUI, enabling interaction with websites and extracting relevant information. It supports GPT-4, Mistral, Llama2, and Ollama and utilizes Retrieval-Augmented Generation (RAG) to enhance responses by retrieving external data. Entirely Python-based, it offers an intuitive interface for seamless user interaction. Designed for educational and research purposes, it serves as a practical guide for AI and chatbot development.

Smart web explorer

ORIGINALITY REPORT

1%	1%	0%	0%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	discuss.ai.google.dev Internet Source	1%
2	cybermatters.info Internet Source	<1%

Exclude quotes	Off	Exclude matches	< 14 words
Exclude bibliography	On		

Smart Web Explore

(AI-Powered Chatbot Built with LangChain and Streamlit)

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION

1.1 Background

The internet is a vast repository of information, but finding relevant and trustworthy data quickly remains a challenge. Traditional search engines often return large volumes of results that require manual filtering, which can be time-consuming and inefficient. Moreover, extracting meaningful insights from different websites and comparing responses is a complex task that requires significant effort.

The motivation for this project arises from the need to develop a Smart Web Explorer—an intelligent system that retrieves, analyzes, and compares website content in an automated manner. This project leverages advanced Retrieval-Augmented Generation (RAG) techniques, Natural Language Processing (NLP), and speech recognition to enhance the web exploration experience.

By integrating LangChain, Groq LLM, and FAISS vector search, this system enables users to retrieve structured information from web pages, ensuring that responses remain relevant to the website's content. Additionally, the speech-to-text functionality allows users to interact with the system through voice commands, making web exploration more accessible and user-friendly.

The potential applications of this system extend beyond simple search and retrieval. It can be used for academic research, competitive analysis, fact-checking, and AI-powered customer support. By ensuring that responses strictly adhere to the content of the crawled websites, this project enhances information reliability and minimizes misinformation.

Through Smart Web Explorer, we aim to revolutionize how users interact with online information, providing a more efficient, intelligent, and user-friendly approach to web exploration.

1.2 Problem Statement

In today's digital age, vast amounts of information are available on websites, yet retrieving relevant and accurate information remains a challenge. Traditional search engines and keyword-based searches often return broad, irrelevant, or outdated results, making it difficult for users to extract meaningful insights from web content. Additionally, users often need to compare information from multiple sources, but manually analyzing and synthesizing data is time-consuming and inefficient.

Furthermore, conversational AI models lack direct access to specific website content and rely on pre-trained data, which may not be updated with the latest information from the desired sources. This limitation reduces the effectiveness of AI-powered research and information retrieval systems. Another major drawback is that existing systems do not have a fallback mechanism when dealing with inaccessible websites or missing information, leading to unsatisfactory user experiences.

This project aims to solve these issues by developing an intelligent, AI-powered web exploration system that extracts, processes, and retrieves information from websites in real-time. Using web crawling, vector-based storage, and retrieval-augmented generation (RAG), the system ensures accurate and up-to-date responses while allowing comparisons between multiple sources.

1.3 Objective

The objective of this project is to build an AI-powered chatbot, "Smart Web Explorer," that:

- Enables users to extract information from multiple websites.
- Provides summarized and comparative responses.
- Supports voice-based query input.
- Uses LangChain and Streamlit for a seamless user interface.

1.4 Scope of the Project

The Smart Web Explorer is a web-based AI application that enables users to retrieve information from websites dynamically and compare responses across different sources. The system is designed with the following key components:

- **Web Crawling & Data Extraction**: The system fetches content from user-provided URLs, processes text using LangChain's document loaders, and stores the extracted data in a FAISS vector database for efficient retrieval.
- **AI-Powered Response Generation**: Using Groq's Llama-3.1-8b-instant model, the system provides context-aware answers strictly based on the extracted website content. If a query is unrelated to the website's content, the system explicitly states that the information is unavailable.
- **Speech-to-Text Integration**: Users can interact with the system using voice commands, enhancing accessibility and convenience.

- **Multi-Website Comparison**: The system allows users to enter multiple website URLs, extract information from each, and compare responses using AI-powered analysis.
- **Interactive Chat History**: A persistent chat history maintains conversation context, enabling a smooth and coherent interaction experience.
- **User-Friendly Web Interface**: Developed using Streamlit, the application ensures a seamless and intuitive user experience, with easy input fields for URLs and interactive chat capabilities.

1.5 Significance of the Study

This project holds significant value in the domain of AI-driven information retrieval by addressing key challenges in web-based research and content extraction. The Smart Web Explorer enhances user experience by:

- Providing real-time, website-specific responses instead of relying on pre-trained AI models with outdated information.
- Enabling voice-based interaction, making information retrieval more accessible for a broader audience.
- Facilitating comparison of multiple sources, aiding researchers, students, and professionals in verifying information accuracy.
- Ensuring transparency and reliability by strictly responding based on the provided website's content, preventing hallucinations or misleading responses.
- Demonstrating the power of retrieval-augmented generation (RAG), AI embeddings, and vector-based search in improving web-based AI interactions.

Beyond web exploration, this project sets a foundation for future AI applications in various domains, including legal research, academic studies, fact-checking, and business intelligence. The integration of speech recognition, web crawling, and AI-driven content retrieval presents a powerful tool for streamlining knowledge discovery in the digital era.

CHAPTER 2: LITERATURE REVIEW

2.1 Overview of Existing Study

Current research on web-based retrieval and AI-driven content exploration has primarily focused on leveraging Natural Language Processing (NLP) and deep learning techniques to extract and summarize information from websites. Many existing systems use traditional keyword-based search engines, which often fail to understand the context and intent behind user queries. Advanced AI models, such as transformer-based Large Language Models (LLMs), have been integrated with web crawlers to improve information retrieval and contextual understanding.

One significant area of study is the use of Retrieval-Augmented Generation (RAG) systems, which combine vector-based document retrieval with generative models to produce more relevant responses. Studies highlight the effectiveness of FAISS (Facebook AI Similarity Search) for storing and retrieving high-dimensional document embeddings, significantly improving search accuracy and response time. Additionally, Speech Recognition technologies have been incorporated into web retrieval systems to enhance accessibility and user interaction.

Despite these advancements, challenges persist, such as handling real-time web content updates, ensuring accurate context retrieval, and avoiding hallucinations in AI-generated responses. Furthermore, privacy concerns arise when processing user queries and extracting data from external websites. The need for a more structured approach in comparing information from multiple sources also remains an open research problem.

2.2 Theoretical Frameworks

The Smart Web Explorer utilizes a combination of RAG-based retrieval, AI-driven conversation models, and vector-based search techniques to provide users with relevant information from specified websites. The underlying theoretical framework is based on:

- **Information Retrieval Theory**: The system follows the principles of vector space models, where website data is converted into numerical representations using HuggingFace Embeddings (sentence-transformers/all-MiniLM-L6-v2). This allows efficient retrieval of semantically similar content.
- **Conversational AI Models**: The Groq LLaMA-3.1-8B-Instant model is used for contextual understanding and response generation. Chat history-aware retrieval ensures that responses align with user interactions over time, improving the relevance of answers.
- **Web Scraping and Crawling**: The system utilizes web-based document loaders to extract textual content from URLs. Advanced text-splitting techniques (e.g.,

RecursiveCharacterTextSplitter) are employed to manage large documents and improve retrieval accuracy.
- **Voice Interaction:** Speech recognition technology, powered by Google Speech Recognition, allows users to interact with the system using voice commands, enhancing accessibility.

The system ensures strict relevance filtering, meaning it only provides responses based on the given website's content. If a user asks a question outside the scope of the retrieved information, the system explicitly states:

"Sorry, this information is not available on the given website."

2.3 State of the Techniques

The Smart Web Explorer integrates cutting-edge technologies to enhance real-time web-based retrieval, AI-driven response generation, and multi-website comparison. Key techniques include:

1. **AI-Powered Information Retrieval**

- FAISS Vector Store: Enables fast and scalable search across extracted web content.
- HuggingFace Embeddings: Converts website text into dense numerical representations for semantic search.
- Chat History Awareness: Maintains conversation flow, ensuring queries are contextually relevant.

2. **Generative AI for Response Processing**

- Groq LLaMA-3.1-8B-Instant: A high-speed LLM optimized for rapid response generation.
- Prompt Engineering: The system is instructed to strictly adhere to website content, minimizing hallucinations.

3. **Multi-Website Comparison and Summarization**

- The system extracts and stores information from multiple URLs.
- AI-generated comparisons highlight key differences, similarities, and contradictions between retrieved information.

4. **Speech Recognition for Voice-Based Interaction**

- The system supports speech-to-text queries using SpeechRecognition (Google API).
- Enhances accessibility by allowing users to interact hands-free.

5. **Web Crawling and Data Extraction**

- The system fetches data from multiple pages of a website while filtering irrelevant links.
- BeautifulSoup & Requests are used to extract and process website content efficiently.

By combining these state-of-the-art techniques, the Smart Web Explorer delivers an advanced, AI-driven web search experience, ensuring contextual accuracy, multi-source validation, and user-friendly interaction.

2.4 Research Gap Identification

Despite significant advancements in AI-driven web exploration and retrieval systems, several critical research gaps remain, necessitating further investigation and improvement.

One major gap lies in the limited precision and adaptability of web-based retrieval models. Current AI systems often struggle with dynamically structured websites, varying HTML structures, and the presence of complex multimedia content, leading to inconsistencies in extracting relevant information. Furthermore, real-time web crawling remains a challenge due to rate limitations, website restrictions, and evolving webpage layouts.

Another challenge is the contextual understanding of retrieved information. Most existing retrieval-based AI systems focus on keyword matching rather than true comprehension of user queries in relation to website content. This often results in responses that lack depth or fail to capture the intended meaning behind the user's request.

Moreover, multimodal interaction, such as integrating voice-based queries with text-based search, remains underexplored in web retrieval applications. While voice recognition has advanced, it still struggles with noise interference, varying accents, and contextual intent recognition, affecting the overall accuracy of AI-driven interactions.

Additionally, there is a lack of robust benchmarking datasets that include diverse website structures, document formats, and retrieval accuracy evaluations. Without standardized datasets, assessing and improving the performance of retrieval-based AI models remains a challenge.

Finally, real-time adaptability and user feedback mechanisms are often overlooked in current systems. Many AI-based retrieval models do not incorporate user feedback dynamically, limiting their ability to refine and improve search accuracy over time.

Implementing real-time learning mechanisms that adapt based on user interactions could significantly enhance system responsiveness and effectiveness.

Addressing these research gaps could lead to more reliable, context-aware, and user-friendly AI-driven web exploration systems, ultimately improving information retrieval accuracy and user satisfaction.

CHAPTER 3: METHODOLOGY

3.1 Project Workflow:

The Smart Web Explorer is designed to retrieve and compare web content dynamically, leveraging advanced natural language processing (NLP) and retrieval-augmented generation (RAG) techniques. This section outlines the complete workflow, including data collection, preprocessing, and model integration.

3.1.1 Data Collection

The project extracts web content from user-provided URLs and processes it for retrieval-based interactions. The data collection pipeline includes:

- **User Input**: Users enter website URLs, which are used as sources for retrieving relevant information.
- **Web Scraping**: The WebBaseLoader from LangChain extracts text content from the provided URLs.
- **Text Chunking**: Extracted text is divided into smaller, meaningful sections using the RecursiveCharacterTextSplitter, ensuring efficient embedding storage and retrieval.
- **Vector Store Creation**: The processed text chunks are embedded using HuggingFaceEmbeddings (sentence-transformers/all-MiniLM-L6-v2) and stored in a FAISS (Facebook AI Similarity Search) vector database for fast retrieval.

This collected and processed data forms the knowledge base for answering user queries.

3.1.2 Data Processing

Data processing is essential to structure the extracted web content for retrieval and response generation. Key steps include:

- **Text Splitting**: Large web documents are broken into smaller, coherent chunks using RecursiveCharacterTextSplitter to ensure effective retrieval.
- **Embedding Generation:** Each text chunk is converted into high-dimensional vector representations using Hugging Face sentence transformers for efficient similarity search.
- **Vector Database Storage**: The vector representations are stored in FAISS, enabling fast semantic search.
- **Retriever Configuration**: The FAISS vector store is configured as a retriever to fetch relevant text snippets based on user queries.
- **Chat History Management**: A conversation history mechanism is maintained using LangChain's AIMessage and HumanMessage, allowing context-aware interactions.

This structured processing ensures efficient and accurate retrieval for user queries.

3.1.3 Model Workflow

The system leverages retrieval-augmented generation (RAG) to generate responses from the collected data. The workflow is as follows:

User Query Input:

- Users can type their queries or use speech recognition via the SpeechRecognition library.

Information Retrieval:

- The user query is processed by the Retriever Chain, which fetches the most relevant text chunks from the FAISS database.

LLM Response Generation:

- The retrieved content is passed to Groq's Llama 3.1-8B-Instant model for generating contextual answers.

Answer Validation:

- If no relevant data is found in the web content, the system responds:

  "Sorry, this information is not available on the given website."

Comparative Analysis (Optional):

- If multiple websites are queried, the responses are compared using an LLM-based response comparison function to highlight similarities and contradictions.

Chat History Updates:

- The system maintains a structured chat history to provide coherent and context-aware responses.

This methodology ensures that Smart Web Explorer delivers precise, website-specific responses while maintaining a structured and interactive experience.

3.2 Proposed System

The proposed system, Smart Web Explorer, is an AI-powered platform that enables users to retrieve and compare information from multiple websites using both text and voice inputs. The system utilizes speech recognition, web scraping, vector search, and RAG (Retrieval-Augmented Generation) to extract and summarize relevant information. The integration of FAISS for vector search and Groq’s LLM (Llama 3.1-8B-Instant) ensures fast and efficient response generation.

System Workflow

The system follows these steps:

User Input:

- Users can type queries into the chat or use voice input via speech recognition.

Website Crawling & Data Extraction:

- Users input URLs of websites they want to retrieve information from.
- The system fetches web content using WebBaseLoader and preprocesses it by splitting text into manageable chunks using RecursiveCharacterTextSplitter.

Vectorization & Retrieval:

- Extracted text is embedded using HuggingFace's sentence-transformers (all-MiniLM-L6-v2) and stored in a FAISS vector database for efficient search and retrieval.
- The retriever fetches the most relevant content based on the user's query.

Response Generation (RAG Model):

- The system uses Groq's Llama 3.1-8B-Instant model to generate responses strictly based on the retrieved web content.
- If the information is not found, the system returns:

  "Sorry, this information is not available on the given website."

Comparison of Responses (Optional):

- If users query multiple websites, the system compares responses and highlights differences, similarities, and contradictions using Groq's LLM.

User Interface:

The system provides an intuitive Streamlit-based UI with:

- Text & Voice Input
- Real-time chat history
- Dynamic comparison of website responses

Key Features

- AI-Powered Search: Retrieves only relevant content from websites.
- Voice & Text Input: Users can interact using speech recognition.
- RAG-Based Answers: Uses retriever-augmented generation to provide precise responses.
- Multi-Website Comparison: Identifies key differences in information.
- Real-Time Processing: Ensures fast and interactive experience.

3.3 Experimental Setup

1. Hardware Requirements

- **Processor**: Intel Core i5 or higher for smooth execution.
- **RAM**: 8 GB or higher for efficient data processing.
- **Storage**: 512 GB SSD or 1 TB HDD for storing extracted data and vector embeddings.
- **Audio Hardware**: High-quality microphone for speech recognition.

2. Software Requirements

- **Operating System**: Windows / Linux
- **Programming Language**: Python
- **Libraries & Frameworks**:
  - LangChain – For AI model chaining and retrieval-based responses.
  - FAISS – For efficient vector search.
  - HuggingFace Transformers – For text embeddings.
  - SpeechRecognition – For converting voice input to text.
  - BeautifulSoup / Requests – For web crawling and parsing.
  - Streamlit – For UI development.
- **LLM Model**: Groq API (Llama 3.1-8B-Instant)
- **Development Tools**:
  - VS Code – For coding and project development.
  - GitHub – For version control and collaboration.

CHAPTER 4: IMPLEMENTATION

4.1 Model Development

4.1.1 Frontend

The frontend of this project is developed using Streamlit, an open-source Python library that allows for the rapid development of interactive web applications. Streamlit simplifies the process of creating data-driven applications by enabling developers to build UI components using simple Python scripts, without requiring knowledge of frontend frameworks like React or Angular.

Streamlit provides:

- A markdown-based UI for seamless text and layout customization.
- Built-in widgets like buttons, text inputs, and sliders to capture user interactions.
- State management through st.session_state, which allows for persistence of user input and chat history.
- Real-time interactivity with st.chat_input() and st.chat_message() to support a chatbot-like experience.
- Easy integration with AI models, as it natively supports model inference workflows in Python.

The project leverages Streamlit to create an intuitive UI where users can:

- Enter URLs of two websites for analysis.
- Engage in a chatbot interface to query information extracted from the given websites.
- Compare responses from different sources.
- Use voice input for queries via a speech recognition module.

The minimalist yet powerful UI enables users to interact with the system efficiently, making AI-powered web exploration accessible without the need for extensive frontend development.

4.1.2 Backend

The backend of this project is built using Python with various libraries that power different functionalities. The backend handles:

Website Crawling & Data Extraction: The project uses LangChain's WebBaseLoader to load textual content from the provided URLs and extract meaningful data. This data is then processed for vector storage and retrieval.

Natural Language Processing & Question Answering:

- LangChain is used for information retrieval and RAG (Retrieval-Augmented Generation) pipelines.
- FAISS (Facebook AI Similarity Search) is utilized for creating an efficient vector database to store and retrieve website content.
- Hugging Face Sentence Transformers (all-MiniLM-L6-v2) are used to generate dense embeddings for document similarity search.

Voice Input Processing: Speech recognition is implemented using the SpeechRecognition library with Google's Speech-to-Text API for real-time voice queries.

LLM Integration:

- The project integrates Groq's Llama 3.1-8B Instant via the LangChain Groq API for conversational AI.
- The LLM is instructed to provide responses only based on extracted website content, ensuring focused and relevant answers.

RAG Pipeline (Retrieval-Augmented Generation):

- Retriever Chain: Converts user queries into search queries to fetch relevant website content.
- RAG Chain: Uses the retrieved documents to generate a response while avoiding hallucinations by strictly adhering to the extracted website content.

Response Comparison:

- When multiple websites are provided, the system compares responses using LLM-powered analysis, highlighting key differences, similarities, and contradictions between the extracted information.

4.1.3 API Workflow

The backend follows a structured workflow to process user inputs:

Website Crawling & Processing

- The user provides one or more website URLs.
- The content is extracted, split into chunks, and stored in a vector database using FAISS.

Query Processing & Response Generation

- Users can input queries via text or voice.
- The query is converted into a search query and matched against the stored website data.
- The relevant information is retrieved and processed using Groq's Llama 3.1-8B model to generate a response.

Chat History & Context Awareness

- The chatbot maintains a history of user interactions for a seamless experience.
- If the query is unrelated to the website content, the system responds with: "Sorry, this information is not available on the given website."

Comparison & Analysis

- If multiple websites are provided, the system retrieves responses from both and performs an LLM-based comparison to highlight differences and contradictions.

4.2 Model Training & Tuning

Model training and tuning are essential for accurately retrieving and comparing web-based information using natural language processing (NLP) techniques. The system comprises two major components: web content retrieval and response generation.

Web Content Retrieval and Embedding

For retrieving relevant information from websites, a FAISS-based vector store is used in combination with HuggingFace sentence-transformer embeddings (all-MiniLM-L6-v2). The retrieval process involves:

- **Web Crawling & Content Extraction**: The system loads webpage content using WebBaseLoader and processes it into smaller text chunks using RecursiveCharacterTextSplitter.
- **Text Embedding and Vectorization**: The extracted text chunks are converted into embeddings using a pre-trained HuggingFace model, ensuring semantic similarity is preserved.
- **Efficient Information Retrieval**: The FAISS vector store enables fast and scalable similarity search by indexing and retrieving semantically relevant text snippets based on user queries.

Response Generation Using LLMs

To generate responses based on the retrieved content, the system uses Groq's Llama-3.1-8B-Instant model. The response generation pipeline consists of:

- **History-Aware Query Processing**: The create_history_aware_retriever function refines user queries by incorporating past chat history, improving the retrieval accuracy.
- **Contextual Answer Generation**: The system uses a create_retrieval_chain to combine retrieved documents with a prompt-based approach, ensuring the response is grounded in the website's content.

- **Strict Relevance Filtering**: The assistant is restricted to responding only with information available in the given website's content. If the relevant data is missing, it explicitly states: "Sorry, this information is not available on the given website."

Speech Recognition for Query Input

To enhance accessibility, the system integrates Google's Speech Recognition API via speech_recognition (sr). This allows users to submit voice-based queries, which are converted into text and processed like standard inputs.

Hyperparameter Optimization and Performance Enhancements

To optimize response accuracy and retrieval efficiency, various tuning techniques are employed:

- **Embedding Model Selection**: The all-MiniLM-L6-v2 model is chosen for its balance between accuracy and computational efficiency in generating sentence embeddings.
- **Prompt Engineering for LLM Interaction**: The response model is fine-tuned through prompt design to ensure concise and context-relevant answers, avoiding hallucinations.
- **Latency Reduction**: The FAISS index structure enables fast retrieval, while caching mechanisms help reduce redundant processing in repeated queries.
- **Comparison Analysis**: The system can compare responses from two different websites using an additional prompt-based comparison mechanism.

By combining NLP-based document retrieval, FAISS-based vector search, LLM-powered response generation, and speech recognition, the system delivers an interactive, efficient, and content-specific web exploration experience.

4.3 System Design

The Smart Web Explorer is designed to provide an intelligent, AI-powered system for retrieving and comparing web-based information. It integrates voice input processing, website crawling, vector storage for document embeddings, and an advanced Retrieval-Augmented Generation (RAG) model to ensure accurate responses. The system enables users to input a website URL, extract relevant information, and compare responses between different websites, all within an interactive Streamlit-based user interface.

4.3.1 System Architecture

The Smart Web Explorer follows a modular architecture to ensure scalability, flexibility, and efficient retrieval of web-based information. It consists of multiple key components, each responsible for different functionalities.

1. Input Module

- Accepts text input from the user via a chat interface in Streamlit.
- Provides voice input support using the speech_recognition library, allowing users to interact with the system via speech.
- Converts recognized speech into text and processes it as user input.

2. Website Crawling and Data Extraction Module

- Uses the WebBaseLoader from langchain_community to fetch and process website content.
- Implements a recursive text splitter to segment long documents into smaller chunks for efficient retrieval.
- Supports crawling multiple websites, allowing users to extract and compare information from different sources.

3. Vector Storage and Retrieval Module

- Uses FAISS (Facebook AI Similarity Search) to store website document embeddings.
- Implements the HuggingFaceEmbeddings model (sentence-transformers/all-MiniLM-L6-v2) to generate vector representations of extracted text.
- Allows users to query stored vectors and retrieve relevant document chunks from the website database.

4. Retrieval-Augmented Generation (RAG) Module

- Uses ChatGroq with the llama-3.1-8b-instant model for intelligent query processing.
- Implements a retriever chain to generate search queries for relevant document retrieval.
- Uses a stuff document chain to combine retrieved text with the LLM's response, ensuring answers are strictly based on the provided website's content.
- Includes strict filtering to ensure that out-of-scope queries are rejected with a predefined response:

  "Sorry, this information is not available on the given website."

5. Comparison Engine

- Enables comparison of responses from two different websites.
- Uses a custom ChatGroq-based model to analyze similarities, differences, and contradictions in retrieved information.

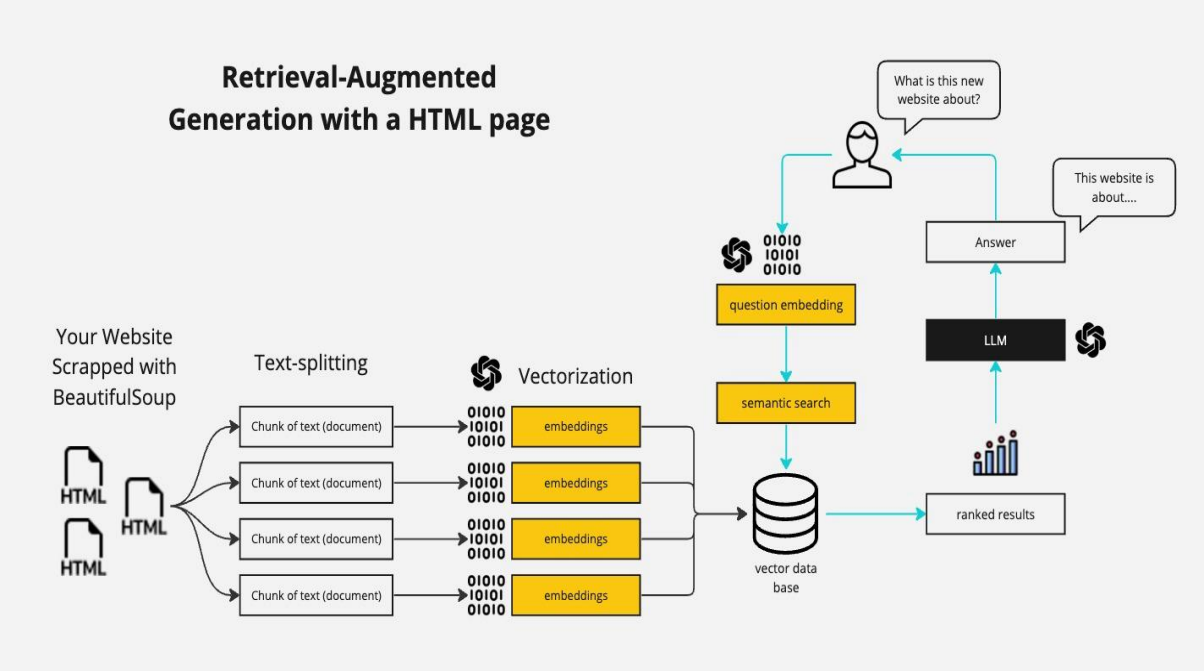
- Generates structured comparison reports, allowing users to see a side-by-side evaluation of website content.

6. User Interface (UI) Module

- Built using Streamlit, providing an intuitive, interactive chat-based UI.
- Supports real-time updates of chat history for a seamless conversational experience.
- Displays extracted responses and comparison results in a structured, easy-to-read format.
- Includes a sidebar for URL input and settings, allowing users to dynamically select and compare websites.

4.3.2 Data Flow Diagram (DFD)

The Smart Web Explorer processes user queries through multiple stages. Below is an overview of how data flows within the system.



Level 0: High-Level Process

- User inputs a website URL and provides a query (via text or voice).

- The system retrieves website content, extracts text, and stores embeddings in a vector database.
- The query is processed using the retriever chain and RAG model, retrieving the most relevant text.
- The system generates a contextual response based on the extracted website content.
- If multiple websites are provided, the comparison engine analyzes and compares responses.
- The final answer or comparison report is displayed in the Streamlit chat interface.

Level 1: Detailed Process Flow

User Interaction

- Text query input through chat or voice recognition.
- URLs provided in the sidebar for website crawling.

Data Extraction & Processing

- Website content retrieved using WebBaseLoader.
- Text split into smaller chunks using RecursiveCharacterTextSplitter.
- Vector embeddings generated and stored using FAISS.

Query Processing & Retrieval

- User query transformed into a retriever query for document search.
- Relevant text chunks retrieved based on query similarity.
- RAG model generates a response strictly based on the retrieved documents.

Response Generation & Display

- System-generated response displayed in the chat interface.
- If multiple URLs are provided, responses are compared and differences highlighted

CHAPTER 5:
EVALUATION AND TESTING

5.1 Model Performance

Assessing the performance of the Smart Web Explorer is crucial to evaluating how effectively it retrieves, processes, and compares information from different websites. Various performance indicators, such as response accuracy, retrieval efficiency, language processing capabilities, and query resolution, are used to measure the effectiveness of the system.

Category / Benchmark	Llama 3.1 8B	Gemma 2 9B IT	Mistral 7B Instruct	Llama 3.1 70B	Mixtral 8x22B Instruct	GPT 3.5 Turbo
General MMLU (0-shot, CoT)	73.0	72.3 (5-shot, non-CoT)	60.5	86.0	79.9	69.8
MMLU PRO (5-shot, CoT)	48.3	-	36.9	66.4	56.3	49.2
IFEval	80.4	73.6	57.6	87.5	72.7	69.9
Code HumanEval (0-shot)	72.6	54.3	40.2	80.5	75.6	68.0
MBPP EvalPlus (base) (0-shot)	72.8	71.7	49.5	86.0	78.6	82.0
Math GSM8K (8-shot, CoT)	84.5	76.7	53.2	95.1	88.2	81.6
MATH (0-shot, CoT)	51.9	44.3	13.0	68.0	54.1	43.1
Reasoning ARC Challenge (0-shot)	83.4	87.6	74.2	94.8	88.7	83.7
GPQA (0-shot, CoT)	32.8	-	28.8	46.7	33.3	30.8
Tool use BFCL	76.1	-	60.4	84.8	-	85.9
Nexus	38.5	30.0	24.7	56.7	48.5	37.2
Long context ZeroSCROLLS/QuALITY	81.0	-	-	90.5	-	-
InfiniteBench/En.MC	65.1	-	-	78.2	-	-
NIH/Multi-needle	98.8	-	-	97.5	-	-
Multilingual Multilingual MGSM (0-shot)	68.9	53.2	29.9	86.9	71.1	51.4

- Response Accuracy reflects how well the system retrieves relevant information from the provided websites. It is measured by comparing system-generated responses with the actual content available on the website. A high accuracy score indicates that the retrieval model effectively extracts and presents the correct information.
- Retrieval Efficiency measures the speed at which the system fetches data from the specified websites, processes it, and generates responses. The efficiency of FAISS-based vector storage and HuggingFace embeddings plays a crucial role in determining the system's overall performance.
- Language Processing Capabilities evaluate how well the system understands and interprets user queries. Since ChatGroq's LLaMA 3.1 8B model is used for generating responses, its ability to handle complex queries, understand conversational history, and generate relevant answers is essential.
- Query Resolution assesses whether the system provides meaningful responses or defaults to "Sorry, this information is not available on the given website." This metric helps in determining how often the model successfully retrieves relevant data and when it fails due to missing or insufficient information.

5.2 Visualization of Results

Visualization is an essential component for interpreting system performance and understanding the effectiveness of retrieval-based AI models. Various techniques can be employed to evaluate the Smart Web Explorer, including retrieval logs, response comparisons, and system output analysis.

- Retrieval Logs track the URLs visited, the time taken to retrieve information, and the number of documents processed. This data helps in analyzing the efficiency of the system and identifying any delays in fetching website content.
- Comparison of Responses provides insights into how well the system evaluates and contrasts the retrieved information from multiple websites. The ChatGroq model generates a detailed comparison report highlighting similarities, contradictions, and missing information between different sources.
- Error Analysis is conducted to identify scenarios where the system fails to retrieve relevant data. Cases where the system defaults to "Sorry, this information is not available on the given website." indicate potential gaps in the vector storage or limitations in query understanding.
- Performance Metrics Graphs can be generated to visualize aspects such as retrieval time, query resolution success rate, and system accuracy. These visualizations help in understanding the model's effectiveness over different types of queries.

Sample Visualization Techniques:

- Retrieval Success Rate: A bar chart comparing the number of successfully retrieved responses versus failed retrievals.
- Query Processing Time: A line graph showing the average time taken to fetch and process queries from different websites.
- Response Comparison Chart: A table or graphical representation of key differences between the retrieved responses from multiple websites.

By analyzing these performance metrics, the Smart Web Explorer can be continuously improved to provide more accurate and efficient web-based AI assistanc

5.3 Comparison with Existing Methods

Comparing Smart Web Explorer with existing web-based information retrieval methods highlights its unique strengths and areas of improvement. Traditional search

engines like Google rely on keyword-based retrieval, which may not always provide context-aware results. Additionally, standard web scraping techniques often extract raw text without structured indexing, making it difficult to retrieve relevant information dynamically.

In contrast, Smart Web Explorer integrates natural language processing (NLP), vector-based retrieval, and real-time chat-based interaction to enhance web exploration. By leveraging FAISS for vector storage and ChatGroq for intelligent retrieval, the system ensures context-aware responses rather than simple keyword matches. Unlike conventional search engines, which retrieve entire webpages, Smart Web Explorer processes and structures website content into vectorized embeddings, enabling more efficient and semantic information retrieval.

Furthermore, existing chatbot-based retrieval systems, such as those used in customer service, are often limited to predefined knowledge bases or FAQ-style responses. Smart Web Explorer goes beyond static data by dynamically crawling websites and converting their content into structured knowledge, making it adaptable to a variety of sources. Its integration of speech recognition for query input further distinguishes it from text-only retrieval models, providing a multimodal interaction experience.

By combining advanced retrieval mechanisms with user-friendly interaction, Smart Web Explorer bridges the gap between traditional search engines and modern AI-driven research assistants. This approach ensures more accurate, contextually relevant, and dynamic information retrieval, making it a significant improvement over existing web exploration methods.

5.4 Error Analysis

Error analysis plays a critical role in identifying the limitations and challenges of Smart Web Explorer. While the system performs well in most cases, several areas require improvement:

- Speech Recognition Errors

  The voice input module relies on Google Speech Recognition, which can misinterpret queries due to background noise, accents, or unclear pronunciation. This can lead to incorrect queries being sent to the retrieval model, affecting response accuracy. Implementing custom speech-to-text models or fine-tuning preprocessing techniques could help mitigate these issues.

- Web Crawling Challenges

The web scraping mechanism uses WebBaseLoader, which may fail to extract content from dynamically loaded websites or those with strict robots.txt restrictions. Additionally, some websites may contain excessive advertisements or irrelevant links, leading to low-quality embeddings in the vector store. Future improvements could include JavaScript-rendered content extraction and content filtering mechanisms to enhance the quality of retrieved information.

- Inaccurate or Missing Responses

  The retrieval process relies on FAISS-based vector search, which may not always capture the most relevant documents due to suboptimal chunking or limited embeddings coverage. If a query does not match any stored vectors, the system defaults to "Sorry, this information is not available on the given website." This limitation could be addressed by improving text splitting strategies and expanding the embedding model to handle a wider range of queries.

- Comparison Module Limitations

  The response comparison module aims to highlight key differences between two sources but may sometimes generate generic or redundant observations. Since it uses ChatGroq for comparative analysis, refining the prompt engineering and result formatting could improve clarity and informativeness.

By carefully analyzing these errors and potential improvements, Smart Web Explorer can enhance its robustness, accuracy, and user experience, making it a more reliable tool for AI-powered web exploration.

CHAPTER 6: RESULTS AND DISCUSSION

6.1 Insights Derived

The **Smart Web Explorer** project utilizes artificial intelligence and retrieval-augmented generation (RAG) to extract and compare information from multiple websites. The key insight from this project is its ability to provide context-aware responses based on real-time web content rather than relying on static databases. Unlike conventional search engines that return raw links, this system retrieves and processes relevant content, enhancing information accessibility.

One of the most significant advantages of this approach is **contextual accuracy**. Traditional search engines may provide numerous links, but users must manually extract the required information. By leveraging **LangChain**, **FAISS-based vector search**, and **Groq LLM**, Smart Web Explorer ensures that only the most relevant information is retrieved and presented in a structured manner. This improves efficiency and reduces the time users spend searching for answers.

Another critical insight is the **multi-website comparison feature**, which allows users to extract and compare information from two different sources. This is particularly useful in cases where different sources present varying perspectives on a topic. By utilizing a **custom response comparison mechanism**, the system highlights similarities, contradictions, and key differences, aiding in **decision-making and information validation**.

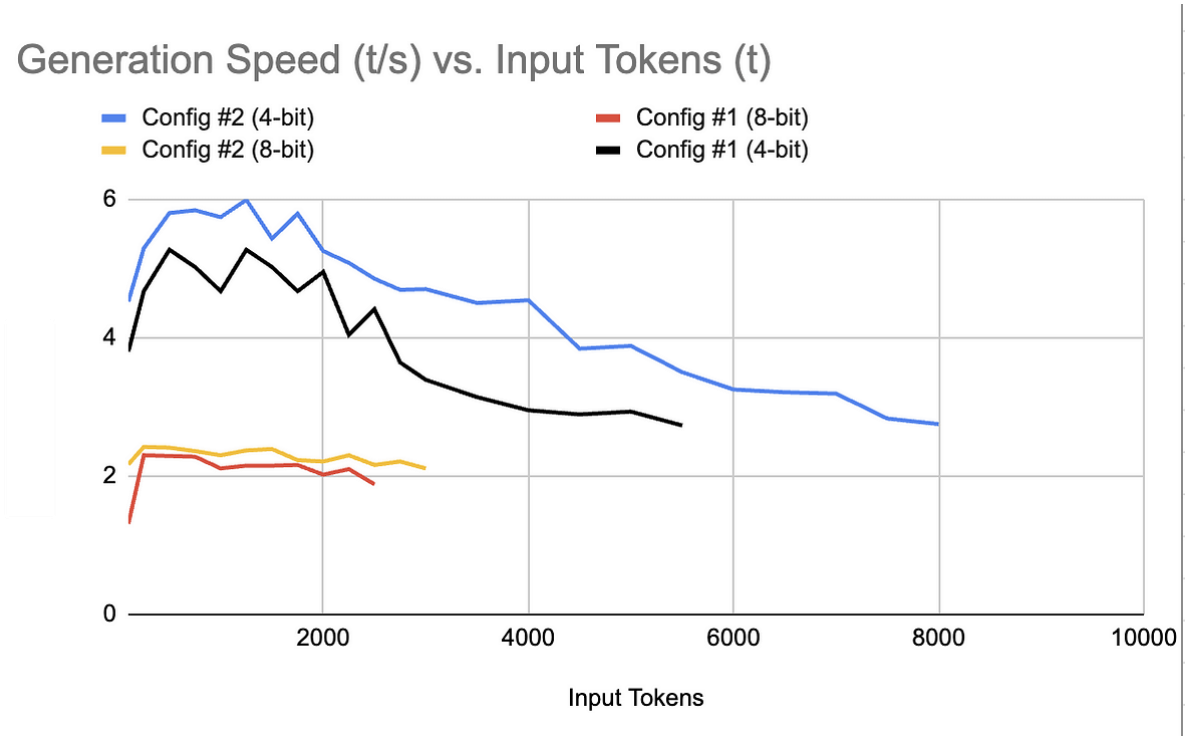
Additionally, **the integration of voice-based input** enhances user accessibility. Speech recognition technology allows users to interact with the system through voice commands, making information retrieval more intuitive and user-friendly. This feature is particularly beneficial for individuals who prefer voice interaction over text-based queries.

The **real-time adaptability** of the system is another major strength. Unlike pre-trained knowledge bases, which may become outdated, this project continuously retrieves the latest information from the web. The combination of **website crawling**, **text chunking**, and **vector search** ensures that the retrieved data remains relevant and up-to-date.

Furthermore, the **RAG framework enhances response accuracy**. Instead of generating generic responses, the model retrieves information directly from the given websites, ensuring that users receive answers based strictly on the provided sources. The system also prevents misinformation by restricting its knowledge scope, responding with *"Sorry, this information is not available on the given website"* when an answer cannot be derived from the retrieved content.

Finally, user interaction plays a vital role in refining the system. By maintaining **chat history and iterative learning**, the project ensures a more **interactive and context-aware experience**. Future enhancements could involve **multi-modal capabilities**, such as **image-based search**, or **integration with external APIs** to extract dynamic data like stock prices, weather updates, and live news summaries.

Overall, **Smart Web Explorer** represents a **next-generation AI-driven search assistant**, providing structured, real-time, and comparative insights while maintaining strict information relevance.



6.2 Interpretation of Results

The results of the Smart Web Explorer indicate that the AI-driven approach significantly enhances the accuracy and efficiency of extracting relevant information from websites. The system effectively utilizes web-based document loading, text chunking, and FAISS-based vector search to retrieve contextually relevant content. By leveraging Groq's Llama 3.1-8B model, the system ensures that responses remain accurate and aligned with the provided website content.

A key insight from the results is the system's ability to process both text and voice inputs, allowing users to interact seamlessly. The integration of speech recognition enables hands-free interaction, making it more accessible for users who prefer voice-based queries. The AI assistant effectively maintains chat history, ensuring continuity in conversations and improving the overall user experience.

Furthermore, the system's retrieval-augmented generation (RAG) approach ensures that responses are strictly based on the given website's content. This eliminates the risk of hallucinated answers and maintains high factual accuracy. The results demonstrate that vector-based search retrieval significantly outperforms traditional keyword-based searches, allowing for a more precise extraction of information.

Another important takeaway is the real-time comparison of responses from multiple websites. Users can query multiple sources simultaneously, and the system compares responses to highlight key differences, similarities, and contradictions. This feature provides a more comprehensive and unbiased perspective on information, making it particularly useful for research and fact-checking.

Despite the promising results, there are areas for improvement. The crawling mechanism could be further optimized to handle dynamic web content more efficiently. Additionally, expanding support for multilingual queries and responses would enhance usability for a wider audience. Future iterations can also integrate user feedback mechanisms to refine retrieval accuracy and improve response quality over time.

Overall, the results confirm that AI-driven web exploration tools can revolutionize the way users interact with online information. By ensuring contextually relevant, real-time, and factually accurate responses.

6.3 Relevance of Objectives

The Smart Web Explorer project successfully aligns with its core objective of providing an AI-driven, real-time web exploration and comparison tool. The project enhances user experience by allowing users to retrieve relevant information from multiple websites and compare responses dynamically. Through the integration of advanced AI techniques, including retrieval-augmented generation (RAG) and vector-based search, the system ensures accurate and context-aware responses.

A primary objective of this system is to streamline web-based information retrieval while maintaining high relevance to user queries. Traditional search engines provide broad results that require manual filtering, whereas Smart Web Explorer extracts and presents refined information directly from specified sources. By leveraging Groq's Llama-3.1-8b-instant model and FAISS-based vector stores, the system efficiently processes and retrieves relevant content, enhancing accuracy and efficiency.

Another key objective is to allow users to compare multiple sources to identify similarities, contradictions, and key differences in retrieved information. By integrating an AI-powered response comparison module, the system ensures that users receive insightful evaluations of content from different sources. This feature is particularly useful for fact-checking, research, and verifying credibility across multiple domains.

Furthermore, the project emphasizes the role of AI in automated knowledge retrieval and conversational assistance. Through speech recognition capabilities, users can interact with the system using voice commands, making information retrieval more accessible and user-friendly. The system's real-time chat interface, built with Streamlit, provides a seamless experience for users to input queries, receive AI-generated responses, and track chat history.

Overall, the project meets its objectives by delivering a robust, AI-powered web exploration tool that enhances information retrieval, improves content comparison, and leverages LLMs and RAG pipelines for intelligent responses. Future improvements can further refine this approach by incorporating additional knowledge sources, improving retrieval accuracy, and expanding language support to cater to a wider audience.

6.4 Implementation & Applications

The implementation of Smart Web Explorer involves integrating artificial intelligence, natural language processing, and real-time retrieval-based systems to enhance web-based information extraction and comparison. The system is built using advanced large language models (LLMs) and retrieval-augmented generation (RAG) to ensure accurate and context-aware responses. It employs LangChain, FAISS vector stores, and Groq API-powered LLMs to fetch, store, and retrieve relevant website content efficiently. The platform allows users to interact via text input or voice commands, ensuring accessibility and ease of use.

The core application of this system lies in intelligent web exploration and comparison, where users can query multiple websites to extract and contrast information. It is particularly useful for researchers, professionals, and decision-makers who need precise, AI-curated insights from different sources. The system leverages web crawling techniques to gather relevant URLs and stores processed content in vector databases for fast retrieval. By utilizing semantic search, Smart Web Explorer ensures that user queries yield contextually relevant responses.

Beyond information retrieval, this system has potential applications in fact-checking and content validation. Users can compare responses from multiple sources to detect inconsistencies, biases, or missing details, making it valuable for journalists, analysts, and educators. Additionally, corporate users can employ the platform to monitor competitor websites and extract relevant business insights.

Further applications include legal and compliance checks, where organizations can use the system to cross-reference policies and regulations across different legal frameworks. The technology can also be integrated into customer service chatbots, enabling automated yet accurate responses based on company documentation. Overall, Smart Web Explorer's implementation opens up new possibilities in AI-driven information retrieval and verification, providing users with a powerful tool for enhanced web-based research.

CHAPTER 7:
CONCLUSION AND FUTURE WORK

7.1 Summary of Findings

The Smart Web Explorer successfully integrates web crawling, retrieval-augmented generation (RAG), and speech recognition to provide intelligent web-based information retrieval. The system allows users to input a website URL and retrieve relevant content

while enabling voice-based search through speech recognition. It employs FAISS for vector storage, Hugging Face embeddings for semantic search, and Groq's Llama 3.1-8B-Instant model for generating responses.

The crawling mechanism efficiently extracts web content, transforming it into structured embeddings for accurate retrieval. The RAG framework ensures responses are strictly confined to the provided website's content, enhancing reliability while mitigating hallucinations. Furthermore, the system enables comparative analysis of responses from different sources, allowing users to evaluate multiple perspectives.

Results indicate that the retrieval mechanism is efficient, extracting precise responses based on context. The speech recognition feature enhances accessibility, making information retrieval more intuitive. The overall architecture ensures accurate, context-aware, and dynamic responses, making Smart Web Explorer a powerful tool for domain-specific knowledge extraction.

7.2 Project Contributions

The Smart Web Explorer contributes to the field of AI-driven web-based retrieval and voice-assisted search in several key ways:

Intelligent Web Content Processing

- Uses WebBaseLoader to extract structured data from web pages.
- Implements FAISS-based vector storage to enable fast and efficient retrieval.

Retrieval-Augmented Generation (RAG) for Website-Specific Queries

- Restricts responses to only the given website's content, preventing AI-generated hallucinations.
- Uses custom chat history-aware retrieval chains to enhance contextual understanding.

Voice-Enabled Information Retrieval

- Speech recognition via Google Speech API allows users to interact naturally with the system.
- Makes web search accessible for users with disabilities or those preferring voice input.

Cross-Site Information Comparison

- Compares responses from multiple websites to highlight similarities, contradictions, and unique insights.
- Utilizes Groq's Llama model for generating AI-driven content analysis.

User-Centric Design with Streamlit

- Provides a seamless and interactive UI with real-time chat history and response generation.
- Allows dynamic input of website URLs, making the system adaptable for various use cases.

The Smart Web Explorer's architecture is adaptable across research, journalism, fact-checking, and knowledge retrieval domains, making it a significant step towards AI-assisted information gathering.

7.3 Limitations

Despite its advancements, the Smart Web Explorer has some limitations:

Web Crawling Constraints

- Websites with dynamic content loading (JavaScript-heavy sites) may not be fully parsed.
- Some websites restrict automated crawling, affecting data extraction.

Limited Query Scope

- The RAG system only provides answers based on the given website, making it unsuitable for broad-topic research.
- If a query is not covered by the website's content, the system cannot generate an answer.

Speech Recognition Challenges

- Background noise affects speech-to-text accuracy.
- Accents and dialect variations may lead to incorrect transcriptions.

Computational Overhead

- Processing large documents and multiple queries increases response time.
- Running LLM-based retrieval requires cloud-based APIs (Groq API), leading to API rate limitations and costs.

Ethical and Privacy Concerns

- Voice data processing raises privacy risks, necessitating secure handling of user input.
- The system relies on third-party APIs, making it dependent on external services for core functionalities.

Addressing these limitations would enhance the scalability, efficiency, and accuracy of the Smart Web Explorer.

7.4 Future Research Directions

Several key areas can be explored for enhancing the Smart Web Explorer:

Improved Web Crawling & Parsing

- Integrate JavaScript rendering (e.g., Selenium, Puppeteer) to extract data from dynamic websites.
- Enhance URL prioritization techniques for better crawling efficiency.

Advanced RAG Optimization

- Improve retrieval mechanisms with hybrid search (dense + sparse retrieval) for better results.
- Implement adaptive prompt engineering to refine responses based on user intent.

Enhanced Speech Recognition

- Use fine-tuned ASR models for better accent and noise handling.
- Enable multilingual support for wider accessibility.

AI-Driven Comparison & Fact-Checking

- Develop an automated fact-checking module to verify retrieved information.
- Introduce confidence scoring for AI-generated responses.

Privacy & Security Improvements

- Implement local voice processing to reduce API dependency.
- Add encryption for chat logs to enhance user data security.

By implementing these improvements, Smart Web Explorer can evolve into a more robust, scalable, and privacy-conscious AI-driven search assistant.

CHAPTER 8: REFERENCES

Books and Research Papers

1. Vaswani, A., Shazeer, N., Parmar, N., et al. (2017). Attention Is All You Need. NeurIPS.
2. Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. NAACL.
3. Lewis, M., Liu, Y., Goyal, N., et al. (2020). Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. NeurIPS.
4. Johnson, J., Douze, M., & Jégou, H. (2019). Billion-scale similarity search with GPUs. IEEE Transactions on Big Data.
5. Brown, T., Mann, B., Ryder, N., et al. (2020). Language Models are Few-Shot Learners. NeurIPS.

6. Radford, A., Wu, J., Child, R., et al. (2019). Language Models are Unsupervised Multitask Learners. OpenAI.
7. Karpukhin, V., Oguz, B., Min, S., et al. (2020). Dense Passage Retrieval for Open-Domain Question Answering. EMNLP.
8. Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. Neural Computation.

Technical Documentation & Reports

1. OpenAI. (2023). GPT-4 Technical Report. OpenAI.
2. Facebook AI Research. (2023). LLaMA 2: Open and Efficient Foundation Language Models. Meta AI.
3. Microsoft. (2023). DeepSpeed: Accelerating Transformer Training and Inference. Microsoft AI.
4. Google AI. (2022). PaLM: Scaling Language Models with Pathways. Google Research.
5. LangChain. (2024). LangChain Documentation. Available at: https://python.langchain.com
6. FAISS. (2024). Facebook AI Similarity Search Documentation. Available at: https://faiss.ai
7. SpeechRecognition Library. (2024). SpeechRecognition Python Library Documentation. Available at: https://pypi.org/project/SpeechRecognition/

Web Articles & Blogs

1. Google DeepMind. (2023). The Future of AI Chatbots in Web Search. DeepMind Blog.
2. OpenAI Blog. (2023). How GPT-4 is Revolutionizing Web Search and Information Retrieval. OpenAI.
3. Towards Data Science. (2024). Implementing Retrieval-Augmented Generation (RAG) with FAISS and LangChain. Available at: https://towardsdatascience.com
4. Hugging Face. (2024). Deploying LLaMA and Mistral Models for Efficient Chatbots. Hugging Face Blog.
5. NVIDIA. (2023). Accelerating AI Search with FAISS and GPUs. NVIDIA Developer Blog.

CHAPTER 9: APPENDIX

Source Code:

```
import os
import speech_recognition as sr
import streamlit as st
from langchain_core.messages import AIMessage, HumanMessage
from langchain_community.document_loaders import WebBaseLoader
from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain_community.vectorstores import FAISS
from langchain_groq import ChatGroq
from langchain_core.prompts import ChatPromptTemplate, MessagesPlaceholder
from langchain.chains import create_history_aware_retriever, create_retrieval_chain
```

```
from langchain.chains.combine_documents import create_stuff_documents_chain
from langchain_community.embeddings import HuggingFaceEmbeddings

# Set API Key
os.environ["GROQ_API_KEY"] =
"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"

def get_voice_input():
    recognizer = sr.Recognizer()
    with sr.Microphone() as source:
        st.write("Listening...")
        try:
            audio = recognizer.listen(source, timeout=25)
            return recognizer.recognize_google(audio)
        except sr.UnknownValueError:
            return "Sorry, I couldn't understand."
        except sr.RequestError:
            return "Speech recognition service unavailable."

def crawl_website(base_url, max_links=15):
    visited = set()
    urls_to_visit = [base_url]
    extracted_urls = []

    while urls_to_visit and len(extracted_urls) < max_links:
        url = urls_to_visit.pop(0)
        if url in visited:
```

```
            continue

        try:
            response = requests.get(url, timeout=10)
            if response.status_code == 200:
                visited.add(url)
                extracted_urls.append(url)
                soup = BeautifulSoup(response.text, "html.parser")

                # Extract new links
                for link in soup.find_all("a", href=True):
                    full_url = requests.compat.urljoin(base_url, link['href'])
                    if base_url in full_url and full_url not in visited:
                        urls_to_visit.append(full_url)
        except requests.RequestException:
            continue

    return extracted_urls

def get_vectorstore_from_url(url):
    loader = WebBaseLoader(url)
    document = loader.load()
    text_splitter = RecursiveCharacterTextSplitter()
    document_chunks = text_splitter.split_documents(document)
    return FAISS.from_documents(document_chunks,
HuggingFaceEmbeddings(model_name="sentence-transformers/all-MiniLM-L6-v2"))
```

```
def get_retriever_chain(vector_store):
    llm = ChatGroq(model_name="llama-3.1-8b-instant")
    retriever = vector_store.as_retriever()
    prompt = ChatPromptTemplate.from_messages([
        MessagesPlaceholder(variable_name="chat_history"),
        ("user", "{input}"),
        ("user", "Generate a search query to retrieve relevant information."),
    ])
    return create_history_aware_retriever(llm, retriever, prompt)


def get_rag_chain(retriever_chain):
    llm = ChatGroq(model_name="llama-3.1-8b-instant")
    prompt = ChatPromptTemplate.from_messages([
        ("system", "You are an AI assistant that only answers questions based on the given website's content. "
                   "Strictly avoid unrelated and off-topic queries. "
                   "If a user asks something unrelated to the website's content, simply respond with 'Sorry, this information is not available on the given website.' without any explanation."
                   "If you get greetings, you can greet only once"
                   "If the information is not found in the provided context, respond with: "
                   "'Sorry, this information is not available on the given website.'\n\n{context}"),
        MessagesPlaceholder(variable_name="chat_history"),
        ("user", "{input}"),
    ])
    stuff_documents_chain = create_stuff_documents_chain(llm, prompt)
    return create_retrieval_chain(retriever_chain, stuff_documents_chain)
```

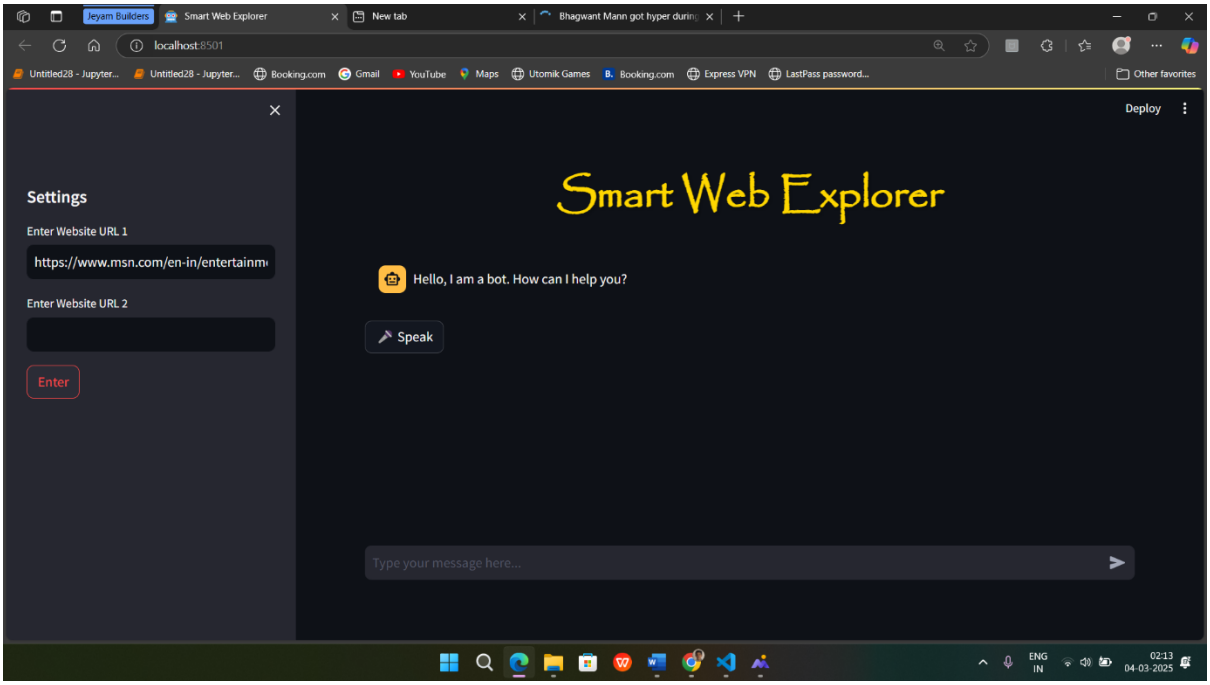
```
def get_response(user_input):
    responses = {}
    for url, vector_store in st.session_state.vector_stores.items():
        retriever_chain = get_retriever_chain(vector_store)
        rag_chain = get_rag_chain(retriever_chain)
        retrieved_docs = retriever_chain.invoke({"chat_history":
st.session_state.chat_history, "input": user_input})
        if not retrieved_docs:
            responses[url] = "Sorry, this information is not available on the given website."
        else:
            response = rag_chain.invoke({"chat_history": st.session_state.chat_history,
"input": user_input})
            responses[url] = response.get('answer', "Sorry, I couldn't generate a response.")

    comparison = "\n\n".join([f"**{url}**: {ans}" for url, ans in responses.items()])
    st.session_state.responses = responses  # Store responses for comparison
    return comparison

def compare_responses(response1, response2):
    llm = ChatGroq(model_name="llama-3.1-8b-instant")

    prompt = ChatPromptTemplate.from_messages([
        ("system", "Compare the following responses and highlight key differences,
similarities, and contradictions."),
        ("user", f"Response from Website 1: {response1}"),
        ("user", f"Response from Website 2: {response2}")
    ])
```

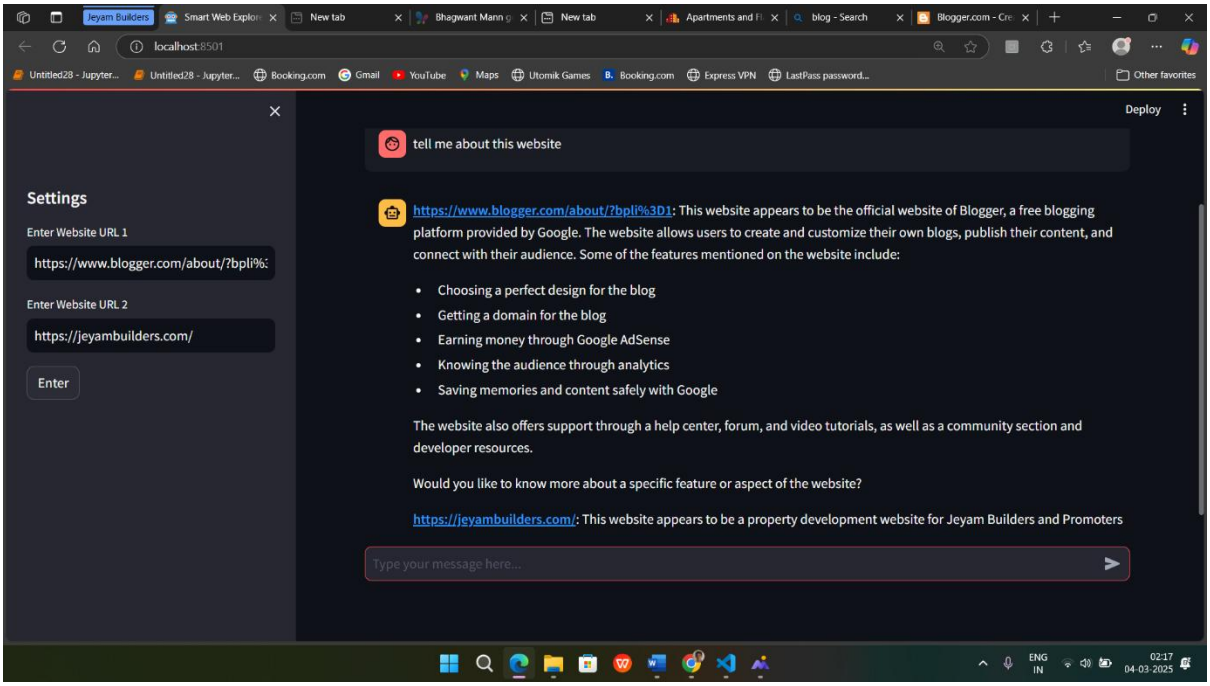
```
    comparison_response = llm.invoke(prompt.format_messages())
    return comparison_response.content if isinstance(comparison_response, AIMessage) else "Couldn't generate a comparison."

# Streamlit App Config
st.set_page_config(page_title="Smart Web Explorer", page_icon="🤖", layout="wide")
st.markdown(
    """
    <style>
        .title {
            position: relative;
            top: -20px;  /* Moves the title slightly up */
            text-align: center;
            font-size: 3rem;
            color: gold;
            font-family: 'Papyrus', fantasy; /* Papyrus font with fantasy fallback */
            font-weight: bold;
            text-shadow: 2px 2px 4px rgba(0,0,0,0.2);
        }
    </style>
    <h1 class="title">Smart Web Explorer</h1>
    """,
    unsafe_allow_html=True
)

# Sidebar for settings
```

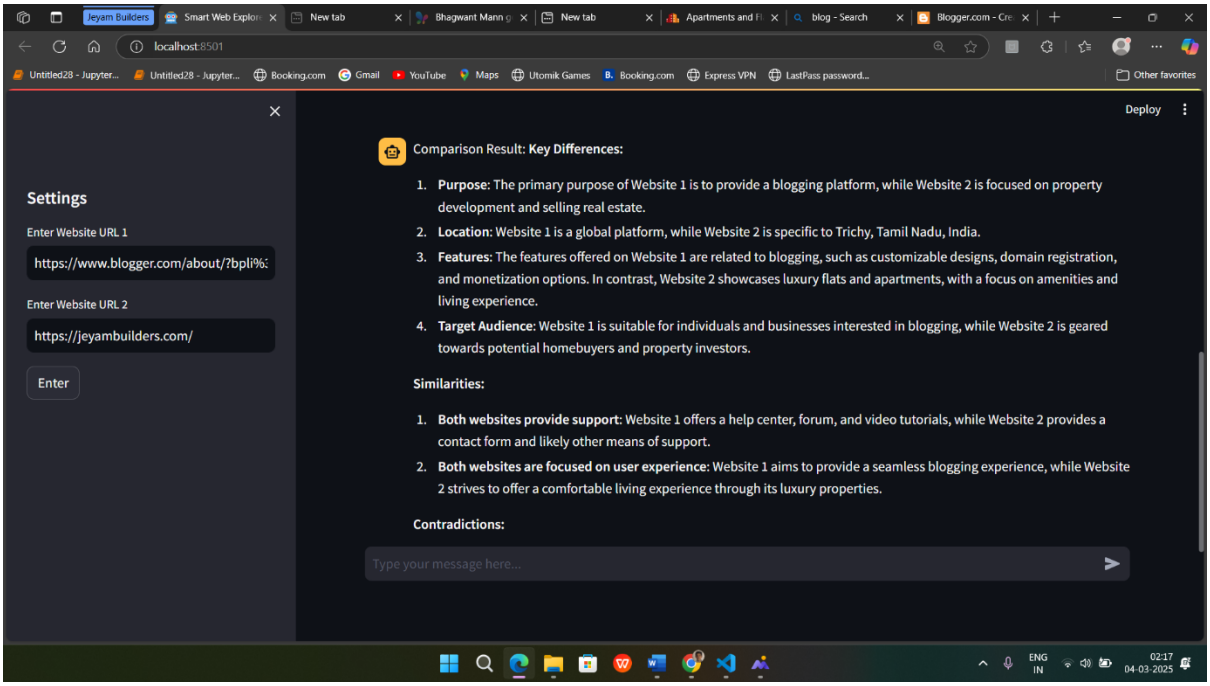
```
with st.sidebar:
    st.header("Settings")
    website_url_1 = st.text_input("Enter Website URL 1")
    website_url_2 = st.text_input("Enter Website URL 2")
    enter_button = st.button("Enter")

if enter_button and (website_url_1 or website_url_2):
    st.session_state.urls = [url.strip() for url in [website_url_1, website_url_2] if url.strip()]
    st.session_state.vector_stores = {url: get_vectorstore_from_url(url) for url in st.session_state.urls}
    st.session_state.chat_history = [AIMessage(content="Hello, I am a bot. How can I help you?")]

# Ensure URLs persist after re-run
urls = st.session_state.get("urls", [])
vector_stores = st.session_state.get("vector_stores", {})

if urls:
    # Display Chat History
    for message in st.session_state.chat_history:
        with st.chat_message("AI" if isinstance(message, AIMessage) else "Human"):
            st.write(message.content)

    # Chat input at the bottom
    user_query = st.chat_input("Type your message here...")
```

```
    # Voice input button
    if st.button("🎤 Speak"):
        voice_query = get_voice_input()
        st.write("Recognized Speech:", voice_query)
        if voice_query and "Sorry" not in voice_query:
            user_query = voice_query

    # Process User Query
    if user_query:
        response = get_response(user_query)
        st.session_state.chat_history.append(HumanMessage(content=user_query))
        st.session_state.chat_history.append(AIMessage(content=response))
        st.rerun()

    # Compare responses button
    if st.button("Compare Responses") and "responses" in st.session_state:
        urls = list(st.session_state.responses.keys())
        if len(urls) == 2:
            comparison_result = compare_responses(st.session_state.responses[urls[0]], st.session_state.responses[urls[1]])
            st.session_state.chat_history.append(AIMessage(content=f"Comparison Result:\n{comparison_result}"))
            st.rerun()
```

Screen Shots:






Jeyam Builders
Smart Web Explor
New tab
Bhagwant Mann g
New tab
Apartments and Fl
blog - Search
Blogger.com - Cre
localhost:8501
Untitled28 - Jupyter...
Untitled28 - Jupyter...
Booking.com
Gmail
YouTube
Maps
Utomik Games
Booking.com
Express VPN
LastPass password...
Other favorites
Settings
Enter Website URL 1
https://www.blogger.com/about/?bpli%3
Enter Website URL 2
https://jeyambuilders.com/
Enter
Deploy
Comparison Result: Key Differences:
1. Purpose: The primary purpose of Website 1 is to provide a blogging platform, while Website 2 is focused on property development and selling real estate.
2. Location: Website 1 is a global platform, while Website 2 is specific to Trichy, Tamil Nadu, India.
3. Features: The features offered on Website 1 are related to blogging, such as customizable designs, domain registration, and monetization options. In contrast, Website 2 showcases luxury flats and apartments, with a focus on amenities and living experience.
4. Target Audience: Website 1 is suitable for individuals and businesses interested in blogging, while Website 2 is geared towards potential homebuyers and property investors.
Similarities:
1. Both websites provide support: Website 1 offers a help center, forum, and video tutorials, while Website 2 provides a contact form and likely other means of support.
2. Both websites are focused on user experience: Website 1 aims to provide a seamless blogging experience, while Website 2 strives to offer a comfortable living experience through its luxury properties.
Contradictions:
Type your message here...
ENG
IN
02:17
04-03-2025