|  |  |  |
| --- | --- | --- |
|  | **MEENAKSHI SUNDARARAJAN ENGINEERING COLLEGE**  **Kodambakkam, Chennai-600024** |  |

# SB3001 - PROJECT-BASED EXPERIENTIAL LEARNING PROGRAM

**DEPARTMENT OF COMPUTER SCIENCE ENGINEERING**

# TOPIC: WIKI CHATBOT

**FACULTY MENTOR : REVATHI P FACULTY EVALUATOR : REVATHI INDUSTRY MENTOR :**

# Project submitted by,

|  |  |  |
| --- | --- | --- |
| **TEAM** | **NAME** | **REGISTER**  **NUMBER** |
| **TEAM LEADER** | *Srirenganathan S* | *311521104057* |

Project report format

# ABSTRACT

1. **INTRODUCTION**
   1. Project Overview
   2. Purpose

# IDEATION AND PROPOSED SOLUTION

* 1. Problem statement definition
  2. Ideation and Brainstorming
  3. Proposed Solution

# REQUIREMENTS ANALYSIS

* 1. Functional Requirements
  2. Non-Functional Requirements

1. **CODE**
2. **OUTPUT**
3. **RESULTS**
4. **ADVANTAGES AND DISADVANTAGES**
5. **CONCLUSION**

# FUTURE SCOPE

# ABSTRACT

The provided Python code defines a class named Wiki Chatbot, which implements a simple chatbot capable of interacting with users, retrieving information from Wikipedia based on user input, and providing responses.

The chatbot utilizes techniques such as web scraping (using BeautifulSoup), text preprocessing (removing punctuation, tokenization, stopword removal, and lemmatization), and cosine similarity scoring to generate responses based on user queries.

Key functionalities include:

* Initializing chat with a greeting message and instructions.
* Receiving user input, with options to end the conversation or request more information.
* Scraping Wikipedia for information on user-specified topics.
* Generating responses based on input queries using TF-IDF vectorization and cosine similarity.
* Preprocessing text data for modeling purposes.

# INTRODUCTION

# The Automated Short Answer (AS) Chatbot project aims to develop a conversational agent capable of engaging users in informative discussions based on topics extracted from Wikipedia. Utilizing natural language processing (NLP) techniques, the chatbot scrapes relevant information from Wikipedia articles, preprocesses the data, and responds to user queries in a coherent and informative manner..

# 2.1 PROJECT OVERVIEW:

# The Wiki Chatbot operates through a Python-based implementation utilizing various libraries such as BeautifulSoup for web scraping, NLTK for text processing, and scikit-learn for TF-IDF modeling. Below are the key functionalities of the chatbot:

# Initialization: Upon initialization, the chatbot greets the user, provides instructions, and awaits input regarding the topic of interest.

# Topic Scraping: Once the user inputs a topic, the chatbot accesses the corresponding Wikipedia page, scrapes relevant paragraphs, and extracts useful information.

# Text Preprocessing: The scraped text undergoes preprocessing, which includes removing punctuation, tokenization, stop-word removal, and lemmatization to enhance the quality of responses.

# Chat Interaction: The chatbot engages in a conversation with the user, responding to queries and providing information based on the scraped Wikipedia content. Users can request more detailed information on a given topic or choose to end the conversation at any time.

# PURPOSE:

The purpose of the Automated Short Answer (AS) Chatbot project is to provide a user-friendly conversational interface for retrieving concise and informative content from Wikipedia articles, facilitating knowledge exploration and natural language interaction.

# IDEATION AND PROPOSED SOLUTION:

# The ideation for the Wiki Chatbot project involved conceptualizing a conversational agent that could access Wikipedia articles, extract relevant information, and engage users in informative discussions. The proposed solution encompasses web scraping to gather data, text preprocessing for improved understanding, and TF-IDF modeling for response generation, ultimately aiming to create an interactive platform for users to explore topics and receive concise answers.

# PROBLEM STATEMENT DEFINITION:

# The problem statement for the Wiki Chatbot project entails developing a conversational agent capable of efficiently retrieving and presenting information from Wikipedia articles based on user-provided topics, integrating web scraping, text preprocessing, and TF-IDF modeling to facilitate meaningful interactions and knowledge exploration.

# IDEATION AND BRAIN STORMING:

# During the ideation and brainstorming phase for the Wiki Chatbot project, several key considerations and ideas were explored to ensure the successful implementation of the chatbot. Here are some of the key points considered:

# User Interaction: Brainstorming focused on creating a user-friendly interface that allows users to easily input topics of interest and engage in conversations with the chatbot. Various options for handling user queries and providing responses were discussed to ensure a seamless and intuitive user experience.

# Data Retrieval: Ideas were generated on how to efficiently retrieve relevant information from Wikipedia articles. This involved brainstorming methods for web scraping, parsing HTML content, and extracting useful data while ensuring accuracy and reliability.

# Text Preprocessing: Brainstorming sessions delved into techniques for preprocessing text data obtained from Wikipedia articles. Strategies for removing noise, such as punctuation and stop words, and for tokenizing and lemmatizing text were explored to improve the quality of responses generated by the chatbot.

# Response Generation: Ideas were generated on how to generate informative responses to user queries based on the scraped data. This involved exploring approaches such as TF-IDF modeling and cosine similarity scoring to identify the most relevant information and formulate coherent responses.

# Error Handling: Brainstorming also included discussions on how to handle errors and edge cases that may arise during the operation of the chatbot. Strategies for detecting and handling issues such as unavailable topics or errors during web scraping were considered to ensure a robust and reliable system.

# PROPOSED SOLUTION:

Web Scraping: Utilizing libraries such as BeautifulSoup and requests, the chatbot scrapes relevant data from Wikipedia articles based on user-specified topics, extracting paragraphs and headings for further processing.

Text Preprocessing: The scraped text undergoes preprocessing steps, including punctuation removal, tokenization, stop-word removal, and lemmatization using NLTK, to enhance the quality and relevance of the extracted information.

TF-IDF Modeling: The chatbot employs TF-IDF vectorization and cosine similarity scoring using scikit-learn to identify the most relevant responses to user queries. This enables the chatbot to generate coherent and informative answers based on the scraped Wikipedia content.

User Interaction: The chatbot provides a user-friendly interface for users to input topics of interest and engage in conversations. It responds to user queries, provides concise information, and offers more detailed responses upon request, creating an interactive and engaging user experience.

# Error Handling: The solution incorporates error handling mechanisms to address issues such as unavailable topics or errors during web scraping. It notifies users of such issues and provides guidance on alternative actions

# to ensure a smooth and reliable operation.

# Error Handling: The solution incorporates error handling mechanisms to address issues such as unavailable topics or errors during web scraping. It notifies users of such issues and provides guidance on alternative actions to ensure a smooth and reliable operation.

# REQUIREMENTS ANALYSIS

# FUNCTIONAL REQUIREMENTS:

# Topic Input:

# Users should be able to input topics of interest for the chatbot to retrieve information from Wikipedia.

# Information Retrieval:

# The chatbot should scrape relevant data from Wikipedia articles based on the user-provided topics.

# It should extract paragraphs and headings from Wikipedia articles for further processing.

# Text Preprocessing:

# The scraped text should undergo preprocessing steps including punctuation removal, tokenization, stop-word removal, and lemmatization to enhance data quality.

# User Interaction:

# The chatbot should provide a user-friendly interface for users to interact with.

# It should respond to user queries in a coherent and informative manner.

# Users should be able to request more detailed information on a given topic.

# Response Generation:

# 4.2 NON FUNCTIONAL REQUIREMENTS:

1. **Performance:**
   * **Response Time:** The chatbot should provide responses within a reasonable time frame, ensuring quick and efficient interactions with users.
   * **Scalability:** It should be capable of handling multiple user interactions simultaneously without significant degradation in performance.
2. **Reliability:**
   * **Availability:** The chatbot should be available and accessible to users at all times, with minimal downtime or service disruptions.
   * **Error Handling:** It should effectively handle errors and exceptions, providing informative error messages to users and logging errors for debugging purposes.
3. **Usability:**
   * **User Interface:** The chatbot interface should be intuitive and easy to navigate, allowing users to input topics and interact with the chatbot effortlessly.
   * **Language Understanding:** It should understand and interpret user queries accurately, providing relevant responses in a natural language format.
4. **Security:**
   * **Data Privacy:** The chatbot should handle user data securely, adhering to data protection regulations and ensuring confidentiality and privacy.
   * **Protection Against Attacks:** It should incorporate security measures to protect against potential attacks such as cross-site scripting (XSS) or SQL injection.

# .

# Code :-

# # Heading: Handwritten Digit Generation using Generative Adversarial Networks (GANs)

# # Importing necessary libraries

# import torch

# import torch.nn as nn

# import torch.optim as optim

# from torchvision import datasets, transforms

# from torch.utils.data import DataLoader

# import numpy as np

# import matplotlib.pyplot as plt

# # Define Generator and Discriminator Networks

# class Generator(nn.Module):

# def \_\_init\_\_(self, latent\_dim=100, output\_dim=784):

# super(Generator, self).\_\_init\_\_()

# self.fc = nn.Sequential(

# nn.Linear(latent\_dim, 256),

# nn.ReLU(),

# nn.Linear(256, output\_dim),

# nn.Tanh()

# )

# def forward(self, x):

# return self.fc(x)

# class Discriminator(nn.Module):

# def \_\_init\_\_(self, input\_dim=784):

# super(Discriminator, self).\_\_init\_\_()

# self.fc = nn.Sequential(

# nn.Linear(input\_dim, 256),

# nn.ReLU(),

# nn.Linear(256, 1),

# nn.Sigmoid()

# )

# def forward(self, x):

# return self.fc(x)

# # Define functions for training

# def train\_discriminator(discriminator, optimizer, real\_data, fake\_data):

# optimizer.zero\_grad()

# real\_prediction = discriminator(real\_data)

# fake\_prediction = discriminator(fake\_data)

# real\_loss = torch.mean(torch.log(real\_prediction))

# fake\_loss = torch.mean(torch.log(1. - fake\_prediction))

# loss = -real\_loss - fake\_loss

# loss.backward()

# optimizer.step()

# return loss.item()

# def train\_generator(generator, optimizer, fake\_data):

# optimizer.zero\_grad()

# prediction = discriminator(fake\_data)

# loss = -torch.mean(torch.log(prediction))

# loss.backward()

# optimizer.step()

# return loss.item()

# # Load MNIST dataset

# transform = transforms.Compose([transforms.ToTensor(), transforms.Normalize((0.5,), (0.5,))])

# mnist\_data = datasets.MNIST(root='./data', train=True, transform=transform, download=True)

# data\_loader = DataLoader(dataset=mnist\_data, batch\_size=64, shuffle=True)

# # Initialize Generator and Discriminator

# generator = Generator()

# discriminator = Discriminator()

# # Define optimizer for Generator and Discriminator

# gen\_optimizer = optim.Adam(generator.parameters(), lr=0.0002)

# dis\_optimizer = optim.Adam(discriminator.parameters(), lr=0.0002)

# # Training the GAN

# num\_epochs = 20

# for epoch in range(num\_epochs):

# for i, (real\_images, \_) in enumerate(data\_loader):

# batch\_size = real\_images.size(0)

# real\_data = real\_images.view(batch\_size, -1)

# fake\_data = generator(torch.randn(batch\_size, 100))

# dis\_loss = train\_discriminator(discriminator, dis\_optimizer, real\_data, fake\_data.detach())

# fake\_data = generator(torch.randn(batch\_size, 100))

# gen\_loss = train\_generator(generator, gen\_optimizer, fake\_data)

# print(f"Epoch [{epoch}/{num\_epochs}], Discriminator Loss: {dis\_loss:.4f}, Generator Loss: {gen\_loss:.4f}")

# # Generating new images

# num\_samples = 16

# z = torch.randn(num\_samples, 100)

# generated\_images = generator(z)

# generated\_images = generated\_images.view(num\_samples, 28, 28)

# # Visualizing generated images

# fig, axes = plt.subplots(4, 4, figsize=(8, 8))

# for i, ax in enumerate(axes.flat):

# ax.imshow(generated\_images[i].detach().numpy(), cmap='gray')

# ax.axis('off')

# plt.show()

# Output:-

# 

# RESULTS:

The user provides the topic "Artificial Intelligence."

The chatbot successfully scrapes data from the corresponding Wikipedia article.

The user asks two questions about artificial intelligence, and the chatbot provides informative responses based on the scraped data.

Finally, the user ends the conversation by typing "bye," and the chatbot bids farewell.

# PERFORMANCE METRICS:

Response Time: Measure average response time to gauge chatbot responsiveness.

Scalability: Assess performance under increasing user loads.

Accuracy: Evaluate correctness and relevance of responses.

Error Rate: Monitor frequency of encountered errors for optimization.

User Satisfaction: Gather feedback to gauge user experience.

Conversational Quality: Evaluate coherence and relevance of interactions.

Resource Utilization: Optimize system resource usage.

Availability: Ensure uptime and accessibility.

Adaptability: Assess ability to handle changes in user behavior and content updates.

# ADVANTAGES AND DISADVANTAGES

Advantages:

Accessible Information: Provides easy access to information from Wikipedia on various topics, enhancing knowledge exploration.

Conversational Interface: Offers a user-friendly conversational interface for interacting with the chatbot, making it intuitive to use.

Efficiency: Automates the process of retrieving and presenting information, saving time and effort for users.

Customizable Responses: Can be tailored to provide short or detailed responses based on user preferences, enhancing user experience.

Scalability: Can handle multiple user queries simultaneously, accommodating a large user base.

Disadvantages:

Reliability on Wikipedia: Relies heavily on the availability and accuracy of information on Wikipedia, which may vary or change over time.

Limited Understanding: May struggle to understand complex queries or nuances in user language, leading to inaccurate or irrelevant responses.

Dependency on Web Scraping: Vulnerable to changes in Wikipedia's HTML structure or content layout, requiring frequent updates to maintain functionality.

Privacy Concerns: May raise privacy concerns if user queries or interactions are logged or stored without consent.

Lack of Personalization: Lacks personalization features, such as user profiles or preferences, limiting its ability to tailor responses to individual users' needs.

# CONCLUSION

In conclusion, the Wiki Chatbot project offers a convenient way to access information from Wikipedia through conversational interactions. While it enhances knowledge exploration and efficiency, challenges include reliance on Wikipedia's data reliability and limitations in understanding complex queries. With ongoing improvements, it has the potential to become a valuable tool for users seeking quick access to curated information.

# FUTURE SCOPE

1. Enhanced Natural Language Understanding: Implement advanced natural language processing techniques to improve the chatbot's ability to understand and respond to complex queries more accurately.
2. Integration of External Sources: Expand the chatbot's capabilities by integrating with additional sources of information beyond Wikipedia, such as academic databases or news articles.
3. Personalization Features: Introduce personalized recommendations and content based on user preferences, browsing history, and feedback to enhance user engagement and satisfaction.
4. Multilingual Support: Incorporate support for multiple languages to cater to a broader user base and facilitate information access for non-English speaking users.
5. Voice Interface: Implement a voice-enabled interface to enable hands-free interaction with the chatbot, enhancing accessibility and usability, particularly in scenarios such as driving or multitasking.