**SPIRITUAL CHATBOT**

A Project Report

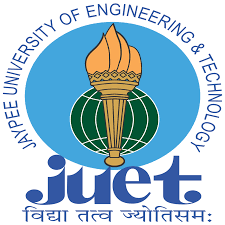
*Submitted by:*

ABHINAV MISHRA (221B011)

SHIVANI SINGH (221B359)

SHIVANGI KUMARI (221B358)

Under The Guidance Of :- DR. P.S. BANERJEE



*Submitted In Partial Fulfilment of the Degree*

*of*

Bachelor of Technology

IN

Computer Science & Engineering

Department of Computer Science & Engineering

JAYPEE UNIVERSITY OF ENGINEERING & TECHNOLOGY, AB ROAD, RAGHOGARH, DT. GUNA-473226 MP, INDIA

**Declaration by the Student**

I hereby declare that the work reported in the B. Tech. project entitled as “**SPIRITUAL CHATBOT**”, in partial fulfillment for the award of degree of  Bachelor Of Technology in CSE  submitted at Jaypee University of Engineering and Technology, Guna, as per best of my knowledge and belief there is no infringement of intellectual property right and copyright. In case of any violation I will solely be responsible.

Abhinav Mishra (221B011)

Shivani Singh (221B359)

Shivangi Kumari (221B358)

Department of Computer Science and Engineering

Jaypee University of Engineering and Technology

Guna, M.P., India

Date :- November 16, 2024

**CERTIFICATE**

This is to certify that the work titled “**SPIRITUAL CHATBOT**” submitted by “**Abhinav Mishra (221B011), Shivani Singh (221B359) and Shivangi Kumari (221B358)**” in partial fulfillment for the award of degree of **Bachelor Of Technology in Computer Science and Engineering** of Jaypee University of Engineering & Technology, Guna has been carried out under my supervision. As per best of my knowledge and belief there is no infringement of intellectual property right and copyright. Also, this work has not been submitted partially or wholly to any other University or Institute for the award of this or any other degree or diploma. In case of any violation concern student will solely be responsible.

Signature of Supervisor

(Dr. PS Banerjee)

**ACKNOWLEDGEMENT**

We would like to express our gratitude and appreciation to all those who gave us the opportunity to

complete this project. Special thanks are to our supervisor Dr. PS Banerjee whose help,

stimulating suggestions and encouragement helped us in all the time of the development process and

in writing this report. We also sincerely thank you for the time spent proofreading and correcting my

many mistakes. We would also like to thank our parents and friends who helped us a lot in finalizing

this project within the limited period.

Thanking you!

Abhinav Mishra (221B011)

Shivani Singh (221B359)

Shivangi Kumari (221B358)

**Executive Summary**

The Spiritual Chatbot project aims to harness the power of artificial intelligence to provide users with a supportive, engaging, and insightful companion for exploring spirituality. The chatbot is designed to offer personalized spiritual guidance, answer faith-related questions, and provide comforting dialogues based on diverse spiritual traditions.

Objectives

Personalized Guidance: Offer tailored advice and reflections based on user inputs and spiritual preferences.

Knowledge Sharing: Provide accurate and respectful information across various spiritual and religious beliefs.

Key Features

Natural Language Processing (NLP): Utilize advanced NLP techniques to understand and respond to user queries authentically.

Interactive Dialogue: Engage users in meaningful conversations that encourage spiritual growth and self-reflection.

Potential Impact

The Spiritual Chatbot has the potential to revolutionize how individuals access spiritual support, making it more accessible and personalized. By bridging the gap between technology and spirituality, this project aims to foster a deeper sense of connection and well-being among its users.

**Table of Contents**

|  |  |
| --- | --- |
| **Titles** | **Page Number** |
| Title Page | i |
| Declaration | ii |
| Certificate of the guide | iii |
| Acknowledgement | iv |
| Executive Summary | v |
| **Chapter-1 INTRODUCTION** | **8-14** |
| 1.1 Problem Definition | 8-9 |
| 1.2 Project Overview | 9-12 |
| 1.3 Software Specification | 13-14 |
| **Chapter-2 SYSTEM ANALYSIS & DESIGN** | **15-21** |
| 2.1 Requirement Specification | 15-19 |
| 2.1.1 Python | 15 |
| 2.1.2 Langchain | 16 |
| 2.1.2.1 Langchain-community | 16 |
| 2.1.3 Tiktoken | 17 |
| 2.1.4 Pandasai | 17 |
| 2.1.5 Streamlit | 17 |
| 2.1.5.1 Streamlit-chat | 18 |
| 2.1.6 VS code | 18 |
| 2.1.7 Faiss-cpu | 19 |
| 2.1.8 Google-generativeai | 19 |
| 2.2 Flowcharts | 20-21 |
| 2.2.1 Block Diagram | 20 |
| 2.2.2 Project Execution | 21 |
| **Chapter-3 PROJECT DESCRIPTION** | **22-45** |
| 3.1 Machine Learning | 22-25 |
| 3.2 Objective | 25-27 |
| 3.3 Methodology | 27-30 |
| 3.4 Data Collection | 30-33 |
| 3.5 Exploratory Data Analysis | 33-35 |
| 3.6 Train – Test, Split and Validation | 35-38 |
| 3.7 Results | 38 |
| 3.8 Web App and Model Deployment | 38-39 |
| 3.9 Web App Creation Codes | 39-45 |
| **Chapter-4 CONCLUSIONS/RECOMMENDATIONS** | **46-47** |
| **Chapter-5 REFERENCES** | **48-51** |
| **Personal Details** | **52** |

**Chapter-1 INTRODUCTION**

**1.1 Problem Definition**

In modern times, despite advancements in technology and communication, individuals face increasing stress, uncertainty, and existential questions. The fast pace of life often leaves little time for introspection, leading to feelings of emptiness and disconnection from one's inner self. Many turn to spiritual texts like the Bhagavad Gita for answers, as it is a profound guide to understanding life, purpose, and the pursuit of inner peace. However, several challenges arise:

Complexity of Ancient Scriptures

The Bhagavad Gita, written in classical Sanskrit, is rich in philosophical and spiritual concepts. For modern readers, understanding its verses without proper guidance can be difficult. The subtle meaning of terms like karma-yoga, bhakti-yoga, and jnana-yoga may be lost without interpretation from knowledgeable teachers or scholars.

Accessibility of Knowledge

While numerous translations and commentaries are available, not all readers have the patience or time to study them in-depth. People often seek quick, relevant, and easy-to-understand answers tailored to their immediate questions, which a static book or commentary cannot provide.

Personalization of Spiritual Guidance

The Gita’s teachings are universal, yet their application depends on an individual’s context, challenges, and stage in life. Generic interpretations may not address specific problems, such as coping with stress at work, handling family conflicts, or finding clarity in life’s purpose.

Limited Interactive Platforms

Existing spiritual resources, like apps or videos, are often passive, offering pre-recorded content rather than dynamic, interactive conversations. This one-way communication lacks the ability to address real-time, user-specific spiritual needs.

Cultural and Generational Gaps

Younger generations may find it hard to connect with ancient texts due to differences in language, lifestyle, and context. Without a relatable medium, the timeless wisdom of the Bhagavad Gita risks being underutilized or misunderstood.

These challenges highlight the need for an accessible, interactive, and personalized tool that simplifies the profound wisdom of the Bhagavad Gita and makes it applicable to the modern-day seeker’s life.

**1.2 Project Overview**

The Spiritual Chatbox is an interactive, AI-powered chatbot designed to provide spiritual guidance and philosophical insights based on the teachings of the Bhagavad Gita. It acts as a digital companion for users seeking clarity, wisdom, and direction in their lives by simplifying the profound concepts of this ancient scripture into relatable and actionable advice.

This project integrates cutting-edge technologies, such as Natural Language Processing (NLP) and machine learning, to create a conversational platform that not only answers user queries but also tailors its responses to their specific concerns. By utilizing a curated dataset of Bhagavad Gita verses, translations, and commentaries, the Spiritual Chatbox brings the essence of the scripture to life in a modern, accessible way.

Some Features of the Project

1.2.1 Interactive Conversations

The chatbot enables users to engage in real-time, dynamic conversations about spirituality, philosophy, and practical life challenges. Unlike static resources like books or videos, it allows users to ask questions, clarify doubts, and receive personalized responses directly derived from the Bhagavad Gita’s teachings.

1.2.2 Simplification of Complex Concepts

The Bhagavad Gita is rich in deep philosophical ideas that can be difficult to understand for the average reader. The chatbox simplifies these ideas without losing their core meaning, making them more accessible to a broader audience. It explains concepts such as:

Dharma (duty and righteousness)

Karma (action and its consequences)

Moksha (liberation)

Yoga (union with the divine)

1.2.3 Personalized Spiritual Guidance

The chatbot is designed to adapt its responses to the context of the user's query. For example:

If a user is struggling with work-life balance, the chatbot might reference karma-yoga, emphasizing the importance of selfless action without attachment to results.

If someone feels lost or demotivated, it may offer insights on bhakti-yoga (devotion) or encourage perseverance through the teachings of Shraddha (faith).

1.2.4 Relatability to Modern Challenges

The chatbot bridges the gap between ancient wisdom and contemporary problems. It connects teachings from the Bhagavad Gita to topics such as stress management, ethical dilemmas, mindfulness, and decision-making, making the scripture relevant to today’s users.

1.2.5 Multimodal Accessibility

The Spiritual Chatbox is accessible through multiple platforms, including web browsers, mobile apps, and messaging services like WhatsApp or Telegram. This ensures users can access spiritual guidance anytime, anywhere.

Objectives of the Project

Empowerment Through Spiritual Wisdom

Provide users with tools to handle life’s challenges by drawing on the Bhagavad Gita’s timeless teachings.

Wider Reach of the Bhagavad Gita

Make the scripture accessible to people of all backgrounds, including those unfamiliar with its traditional context or language.

Encouraging Mindfulness and Inner Growth

Foster introspection and personal development by offering practical advice rooted in spiritual principles.

Dynamic and Engaging User Experience

Create an interactive platform that keeps users engaged through meaningful and personalized conversations.

Core Components of the Chatbox

Data Source:

The chatbot’s primary dataset comprises:

Translations of the Bhagavad Gita by authoritative scholars.

Commentaries and interpretations from spiritual leaders like Swami Prabhupada, Swami Vivekananda, and others.

Contextual cross-references for real-world applications.

AI and NLP Engine:

The chatbot uses Natural Language Processing (NLP) to:

Understand user queries and identify their intent.

Generate appropriate and contextually relevant responses.

Machine Learning Model:

Trained on the Bhagavad Gita dataset, the model ensures accurate interpretations of scripture while adapting to user inputs over time.

User Interface (UI):

A simple, intuitive interface designed for seamless interaction.

Options to search verses, receive explanations, and explore specific topics like karma or mindfulness

**1.3 Software Specification**

The development of the Spiritual Chatbox involves integrating advanced technologies to ensure seamless and meaningful user interactions. The key specifications are:

Programming Language: Python (for backend processing and NLP).

Frameworks:

Natural Language Toolkit (NLTK) for text processing.

PyTorch for implementing deep learning models if required.

Dataset:

Comprehensive translations of the Bhagavad Gita (such as those by Swami Prabhupada, Swami Chinmayananda, or Swami Sivananda).

Commentaries and interpretations from reputable sources.

Interface:

Web-based or mobile-friendly interface using

* Streamlit,
* streamlit-chat (Web Interface)

And we mainly use

* Pandasai
* tiktoken
* Gemini API Model
* Faiss-cpu

Database:

Storage of Gita text and user interactions using SQLite .

API: RESTful API for interaction between the chatbot and user interface.

Cloud Hosting: Deployment on platforms like AWS, Google Cloud, or Azure to ensure scalability and availability.

Security: Implementation of SSL encryption and user data privacy protocols.

This project not only preserves the timeless wisdom of the Bhagavad Gita but also offers an innovative approach to addressing the spiritual and emotional needs of modern users through technology. By bridging ancient teachings and AI, it creates an accessible, personalized tool for spiritual exploration.

**Chapter 2: SYSTEM ANALYSIS & DESIGN**

This chapter provides an in-depth analysis of the Spiritual Chatbox system, focusing on the technologies, tools, and design methodologies required to develop the chatbot, along with flowcharts and system diagrams for a clear representation of the system’s architecture and workflows.

**2.1 Requirement Specification**

In this section, we define the core tools and technologies used to develop the Spiritual Chatbox and explain how each component contributes to the overall functionality of the system.

**2.1.1 Python**

Python is the core programming language used for developing the Spiritual Chatbox. It is preferred because of its simplicity, readability, and versatility. Python is widely used for artificial intelligence, machine learning, and natural language processing (NLP), making it an ideal choice for developing a chatbot that can process textual input from users and provide answers based on the Bhagavad Gita.

Libraries and Frameworks: The system uses popular Python libraries like TensorFlow, PyTorch, spaCy, NLTK, and Transformers to handle machine learning, natural language understanding, and language generation.

Key Benefits:

Simple syntax and readability.

Extensive support for AI/ML libraries.

Strong community support.

Efficient handling of data, making it suitable for large-scale applications like chatbots.

**2.1.2 Langchain**

Langchain is an open-source framework designed to help developers build applications that interact with large language models (LLMs) efficiently. In the Spiritual Chatbox, Langchain is used to integrate the Bhagavad Gita’s teachings with the underlying AI model and facilitate smooth conversations between the user and the chatbot.

Core Features:

Facilitates the orchestration of conversation flows.

Helps maintain context and state during multi-turn conversations.

Integrates with various NLP and machine learning models.

Use Case: Ensures that the chatbot can respond to user queries based on the Bhagavad Gita by linking the dataset with the AI model, making the interaction contextually aware.

2.1.2.1 Langchain-community

The Langchain-community version is an open-source variant of the Langchain framework. It provides a customizable, community-supported platform to integrate large language models into applications like the Spiritual Chatbox.

Use Case: The Langchain-community version simplifies the development process by providing built-in utilities for data handling and conversational state management.

Benefits:

Free and open-source.

Continuous updates and support from the developer community.

Easy integration with existing systems.

**2.1.3 Tiktoken**

Tiktoken is a lightweight tokenization library that splits text into smaller units called tokens. Tokenization is an essential preprocessing step in natural language processing (NLP), and Tiktoken ensures that text can be fed into the AI model in a way that is efficient and contextually accurate.

Use Case: It breaks down the Bhagavad Gita verses and user input into tokens, ensuring that the AI model can process large chunks of text efficiently without losing context.

Benefits:

Highly efficient tokenization for large datasets.

Ensures proper handling of long inputs, like Bhagavad Gita verses.

**2.1.4 Pandasai**

Pandasai is an extension of the Pandas library that integrates artificial intelligence models into DataFrame operations. It enables the Spiritual Chatbox to handle structured data (like the Bhagavad Gita’s verses) and apply AI techniques to generate context-aware responses.

Use Case: The Bhagavad Gita’s verses, along with translations and commentaries, can be organized in a Pandas DataFrame. Pandasai allows the system to query and retrieve verses quickly based on user inputs, making the chatbot's response generation more efficient.

Benefits:

Allows seamless AI integration into data handling workflows.

Simplifies querying and filtering large datasets for relevant information.

**2.1.5 Streamlit**

Streamlit is a Python-based framework for creating interactive web applications. It is used to build the user interface (UI) for the Spiritual Chatbox, allowing users to interact with the chatbot in real time.

Core Features:

Easy creation of dynamic, interactive web pages.

Fast deployment of applications without the need for extensive front-end knowledge.

Real-time display of conversation logs and responses.

Benefits:

Streamlined development of chat interfaces.

Allows for easy deployment of web applications for a broader user base.

**2.1.5.1 Streamlit-chat**

Streamlit-chat is an extension of Streamlit that is specifically designed for building chat-based applications. This tool is used to create an intuitive chat interface for the Spiritual Chatbox, allowing users to input their queries and receive real-time responses from the chatbot.

Use Case: Creating a seamless, interactive chat interface for users to ask questions and receive spiritual guidance.

Advantages:

Easy-to-implement chat interface for engaging with the AI-powered system.

**2.1.6 VS Code**

Visual Studio Code (VS Code) is the integrated development environment (IDE) chosen for writing and debugging the code. It supports Python and a variety of extensions that facilitate coding, version control, and debugging.

Key Features:

Syntax highlighting and code completion.

Git integration for version control.

Debugging and testing features for efficient code development.

Benefits:

Lightweight yet powerful tool for Python development.

Ideal for collaborative development with integrated Git support.

**2.1.7 Faiss-cpu**

Faiss-cpu is a library developed by Facebook AI that facilitates efficient similarity search and clustering in large datasets. It is used to search through the Bhagavad Gita’s verses and retrieve similar responses based on user queries.

Use Case: Matching user queries with similar verses from the Bhagavad Gita using vector space models for similarity search.

Advantages:

Efficient search capabilities even for large volumes of text.

Provides fast retrieval of relevant information.

**2.1.8 Google-generativeai**

Google Generative AI refers to Google's suite of AI models capable of generating human-like text. For this project, it powers the Spiritual Chatbox, allowing the system to generate contextually accurate and coherent responses based on user queries related to the Bhagavad Gita.

Core Features:

Context-aware language generation.

Capable of handling multi-turn conversations.

Benefits:

Highly accurate and fluent text generation for a more natural conversation flow.

**2.2 Flowcharts**

**2.2.1 BLOCK DIAGRAM**

START

USER

INPUT

Conversation

Query

Removing Stop Words

Stemming

Spellcheck

Getting the Shloks/Sentences that match

Display Result

Reply with simple sentences that have been hardcoded

Matching the words with the tags in the database

**2.2.2 PROJECT EXECUTION**

CHOOSING THE DATASET

CLEANING AND TRANSFORMATION

SPLIT THE DATASET INTO TRAINING AND TESTING SETS

IMPLEMENTING NLP ALGORITHMS

BUILD CHATBOT MODEL USING PYTHON AND GEMINI API MODEL

CREATE WEBAPP USING STREAMLIT

TRAIN THE MODEL FROM TRAINING DATA

TESTING AND EVALUTING THE MODEL

**Chapter 3: PROJECT DESCRIPTION**

In this chapter, we will delve into the details of the Spiritual Chatbox project, covering aspects like the machine learning techniques employed, the objectives of the project, the methodology followed, data collection, exploratory data analysis (EDA), training and validation procedures, results obtained, and how the final model was deployed through a web application.

**3.1 Machine Learning: A Detailed Description**

Machine Learning (ML) forms the backbone of the Spiritual Chatbox, enabling it to process and respond to user queries intelligently. This involves implementing advanced algorithms and techniques that allow the chatbot to understand natural language input, retrieve relevant information, and generate insightful responses based on the teachings of the Bhagavad Gita. Below is a detailed breakdown of the role and implementation of Machine Learning in the project.

Role of Machine Learning in the Spiritual Chatbox

The core purpose of ML in this project is to create an interactive chatbot that can comprehend spiritual and philosophical queries in natural language and provide responses rooted in the Bhagavad Gita. Key functionalities supported by ML include:

Natural Language Processing (NLP):

NLP enables the chatbot to understand the user’s intent and context. It involves tokenization, stemming, lemmatization, and semantic analysis to extract meaning from user inputs.

Pretrained transformer-based models such as GPT or Google Generative AI power the chatbot’s language understanding.

Information Retrieval:

The chatbot uses similarity search algorithms to find and retrieve the most relevant verses or commentaries from the Bhagavad Gita dataset.

Faiss (Facebook AI Similarity Search) is employed to perform efficient similarity matching.

Response Generation:

ML ensures the chatbot generates human-like responses based on retrieved information, providing meaningful and contextually relevant replies.

Continuous Learning:

The chatbot improves over time through fine-tuning and user feedback, adapting to new types of queries and improving response accuracy.

Key ML Components

Pretrained Language Models:

The chatbot leverages transformer-based models like GPT or Google Generative AI, which have been pretrained on massive text corpora. These models are fine-tuned with the Bhagavad Gita dataset to specialize in spiritual and philosophical queries.

Transformers excel in understanding context due to mechanisms like self-attention, which allows the model to focus on relevant parts of the text.

Text Vectorization and Embeddings:

Text data, including verses and user queries, are converted into numerical formats using embeddings like Word2Vec, GloVe, or transformer-based embeddings (e.g., BERT embeddings).

This process creates dense numerical vectors that represent semantic meaning, allowing the chatbot to compare and analyze text effectively.

Similarity Search with Faiss:

Faiss is an open-source library designed for efficient similarity searches. It is used to:

Create an index of all the verses from the Bhagavad Gita.

Compare user queries with the indexed verses and retrieve those most similar in meaning.

This allows the chatbot to quickly identify relevant passages.

Natural Language Understanding (NLU):

NLU techniques help the chatbot parse user inputs to understand their intent and context. For example:

Determining if the user seeks guidance on karma, dharma, or spirituality.

Identifying key themes or topics in the query.

Supervised Fine-Tuning:

The pretrained language models are fine-tuned on a labeled dataset of Bhagavad Gita verses and their associated themes or meanings. This improves the model’s ability to understand and respond to domain-specific queries.

Implementation Workflow

Data Preprocessing:

The dataset (Bhagavad Gita verses, translations, and commentaries) is cleaned and tokenized.

Stop words are removed, and special characters are handled to ensure clean input data.

Verses are categorized into themes (e.g., Karma Yoga, Bhakti Yoga, Jnana Yoga) to improve contextual retrieval.

Embedding Creation:

Each verse and its translation are converted into embedding vectors using pretrained language models or specific text vectorization techniques.

These embeddings capture the semantic meaning of the text.

Model Training:

Fine-tuning the language model involves training it on the Bhagavad Gita dataset to specialize in responding to spiritual and philosophical queries.

Training includes pairing user query examples with appropriate verses or explanations to teach the model how to respond.

Similarity Search:

User queries are also converted into embeddings.

These embeddings are compared with the indexed embeddings of the Bhagavad Gita verses using Faiss.

The most similar verses are retrieved based on cosine similarity or Euclidean distance.

Response Generation:

The retrieved verse or explanation is processed to ensure it is contextually formatted as a response.

The language model generates additional text if necessary to provide a more conversational and engaging answer.

Evaluation and Iteration:

The chatbot’s performance is evaluated using a test dataset, focusing on metrics like precision, recall, and F1 score.

Feedback from users is incorporated into subsequent iterations to improve the model’s performance.

Advantages of Machine Learning in the Chatbox

Contextual Understanding:

Advanced ML techniques allow the chatbot to understand user queries with high accuracy, even when phrased in varied ways.

Efficient Search:

Faiss ensures fast and precise retrieval of relevant verses, enhancing the chatbot’s responsiveness.

Scalability:

The model can handle an increasing volume of user interactions, making it scalable for broader deployment.

Personalized Responses:

The chatbot can adapt to individual users’ styles of questioning, making the interaction more personalized.

Continuous Improvement:

With feedback loops and fine-tuning, the chatbot’s performance improves over time.

**3.2 Objective**

ProfessionalPrimary Objectives

Facilitating Spiritual Guidance:

The chatbot is designed to offer spiritual wisdom based on the verses and teachings of the Bhagavad Gita. It acts as a digital guide, helping users navigate questions about life, purpose, morality, and spirituality.

Example: If a user asks, “What is the essence of karma yoga?”, the chatbot will retrieve relevant verses and provide an explanation in simple terms.

Promoting Accessibility to Bhagavad Gita’s Teachings:

The project aims to make the profound knowledge of the Bhagavad Gita accessible to a global audience, regardless of their familiarity with the scripture.

By presenting complex concepts in an easy-to-understand conversational format, the chatbot bridges the gap between ancient wisdom and modern-day users.

Encouraging Self-Reflection and Growth:

Through interactive dialogues, the chatbot encourages users to reflect on their thoughts, decisions, and actions, fostering self-awareness and personal growth.

Example: A user struggling with stress may ask, “How do I find peace?”, and the chatbot will respond with verses and practical insights on inner calmness and detachment.

Delivering Contextually Relevant Answers:

The chatbot's responses are tailored to the user’s context, ensuring that the advice or guidance aligns with the query’s intent. This is achieved through advanced natural language understanding (NLU) and similarity search algorithms.

Secondary Objectives

Creating an Engaging Educational Tool:

The chatbot serves as an educational resource for students, teachers, and enthusiasts interested in learning the principles of the Bhagavad Gita.

It can answer questions, explain verses, and discuss philosophical themes interactively, making learning more engaging.

Encouraging Ethical Decision-Making:

Drawing from the Bhagavad Gita’s teachings, the chatbot can guide users in making ethical decisions in their personal and professional lives.

Enhancing User Engagement Through Interactivity:

The chatbot’s conversational nature ensures a dynamic and personalized user experience. By maintaining the context across multiple queries, it fosters a deeper connection with the user.

Serving as a Multi-Purpose Tool:

Beyond spiritual guidance, the chatbot can assist in academic research, scriptural studies, and personal exploration of the Bhagavad Gita’s philosophy.

Key Features Supporting the Objectives

Natural Language Processing (NLP): The chatbot understands and interprets user queries in various forms, whether they are direct (e.g., “What is dharma?”) or contextual (e.g., “How should I approach challenges in life?”).

Information Retrieval: By utilizing similarity search algorithms like Faiss, the chatbot retrieves verses and explanations that best match the user’s question.

Explainability: The chatbot not only presents verses but also explains their meaning in simple language, making complex philosophical ideas accessible.

Multi-Platform Availability: Deployment on web platforms ensures the chatbot is available to users across devices, enhancing its reach.

Practical Applications of the Objectives

For Spiritual Seekers:

A user seeking advice on detachment may receive a response based on Chapter 2, Verse 47, which discusses performing one’s duties without attachment to results.

For Personal Development:

A user asking about overcoming fear might receive insights from Chapter 4, which emphasizes wisdom and courage derived from self-knowledge.

For Academic and Educational Use:

Teachers can use the chatbot to explain philosophical concepts, while students can query it for specific topics during their studies.

For Professionals:

s seeking guidance on ethics and leadership can query the chatbot for relevant teachings on responsibility and morality.

**3.3 Methodology**

The methodology for developing the Spiritual Chatbox involves a systematic approach that integrates data collection, machine learning techniques, natural language processing (NLP), and web-based deployment. The process ensures the chatbot is efficient, accurate, and user-friendly, capable of delivering spiritual insights based on the Bhagavad Gita. Below is a detailed explanation of the methodology.

Step 1: Problem Analysis and Objective Definition

The project begins with understanding the key objectives: providing meaningful spiritual guidance, ensuring accessibility, and enhancing user experience.

The Bhagavad Gita serves as the primary dataset, necessitating methods to process its text and context for accurate interaction.

Step 2: Data Collection and Preprocessing

Data Source:

The core dataset includes verses from the Bhagavad Gita, translations, and commentaries from trusted sources.

Metadata such as chapter, verse number, and theme (e.g., Karma Yoga, Bhakti Yoga) is included to enhance retrieval accuracy.

Data Preprocessing:

Cleaning: Removing special characters, redundant spaces, and formatting inconsistencies.

Tokenization: Breaking text into smaller units (words or phrases) for processing.

Lemmatization and Stemming: Reducing words to their base form to ensure consistent analysis.

Categorization: Grouping verses based on themes or philosophical concepts for structured indexing.

Step 3: Model Selection and Development

Embedding Generation:

Verses and user queries are converted into numerical representations (embeddings) using pretrained models such as:

BERT (Bidirectional Encoder Representations from Transformers): For contextual embeddings.

Sentence Transformers: For sentence-level similarity matching.

Embeddings capture semantic meaning, enabling comparison between queries and dataset entries.

Similarity Search:

Faiss (Facebook AI Similarity Search):

Faiss indexes embeddings and performs fast, scalable similarity searches.

It identifies verses most relevant to the user's query by calculating distances (e.g., cosine similarity).

Response Generation:

A Transformer-based model (e.g., GPT or Google Generative AI) is fine-tuned to:

Retrieve verses and their interpretations.

Generate natural, conversational responses.

The model contextualizes responses based on prior interactions, ensuring coherence in multi-turn conversations.

Step 4: System Integration

Natural Language Understanding (NLU):

NLU techniques interpret user queries, extracting intent and context.

Example: A query like “What is the importance of detachment?” is classified under Karma Yoga for relevant verse retrieval.

Backend Development:

The backend integrates ML models with APIs for efficient communication.

LangChain framework is used for seamless interaction between the query processor, retriever, and response generator.

Web Interface Development:

Streamlit and Streamlit-Chat are used to create an intuitive, user-friendly interface.

Users can input their queries and receive responses in real-time.

Step 5: Testing and Validation

Train-Test Split:

The dataset is divided into training and testing subsets.

Training data is used to fine-tune the model, while test data evaluates accuracy and performance.

Validation Metrics:

Precision and Recall: Measure the relevance and completeness of retrieved verses.

BLEU (Bilingual Evaluation Understudy): Evaluates the fluency and contextuality of responses.

User Feedback: Real-world interactions are analyzed to refine the chatbot’s understanding and responsiveness.

Step 6: Model Deployment

Model Hosting:

The trained model is deployed on a cloud platform for scalability and availability.

Tools like Google Generative AI and Streamlit ensure the application is accessible via the web.

Integration with User Interface:

The chatbot integrates seamlessly with the front-end, allowing users to interact through a conversational interface.

Features include:

Context retention for multi-turn conversations.

Thematic suggestions based on prior queries.

Step 7: Continuous Improvement

Feedback Loop:

User interactions are logged to identify areas for improvement.

Misinterpreted queries or incorrect responses are analyzed to fine-tune the model further.

Model Updates:

Periodic retraining with updated datasets ensures the chatbot remains relevant and accurate.

New features (e.g., multi-language support) are added based on user needs.

Workflow Summary

Input Query: Users type their queries into the chatbox.

Query Processing: The query is tokenized, vectorized, and matched with embeddings.

Information Retrieval: Relevant verses and explanations are retrieved using similarity search.

Response Generation: The chatbot formulates a response, combining retrieved verses with contextual explanations.

Output: The response is displayed to the user in a conversational format.

**3.4 Data Collection**

Data collection is a critical step in building the Spiritual Chatbox, as the accuracy, relevance, and reliability of the chatbot depend heavily on the quality of the dataset. For this project, the primary dataset consists of the Bhagavad Gita, which includes verses, translations, and commentaries. Below is a comprehensive overview of how the data was collected, processed, and organized for use in the chatbot.

Sources of Data

Primary Source:

The Bhagavad Gita forms the core dataset for this project.

Data includes:

Original Sanskrit verses.

Word-by-word translations.

Contextual explanations and commentaries.

Additional Sources:

Renowned commentaries by scholars and spiritual leaders, such as:

Swami Prabhupada’s Bhagavad Gita As It Is.

Swami Sivananda’s interpretation.

Translations from reputable publishers and organizations to ensure the authenticity of the text.

Supplementary Data:

Related spiritual texts and essays that provide a broader understanding of the Bhagavad Gita’s context and themes.

Metadata such as chapter names, verse numbers, and thematic categorizations.

Data Collection Process

Data Extraction:

The textual content of the Bhagavad Gita and its translations was extracted from:

Published books (digitally available in PDF or text format).

Online repositories and spiritual organization websites offering open-access content.

APIs or web scraping techniques were used for structured data collection from trusted sources.

Data Structuring:

Each verse was categorized with attributes such as:

Chapter Number and Name: Helps in thematic organization.

Verse Number: Ensures accuracy in referencing.

Theme: Identifies the central topic, e.g., Karma Yoga, Bhakti Yoga, Jnana Yoga.

Translation and commentary fields were linked to their respective verses for easy retrieval.

Data Cleaning:

Removing Redundancies: Duplicate translations or interpretations were filtered out.

Formatting Consistency: Ensured uniform structure across all entries for seamless processing.

Example: All verses follow the same format—original Sanskrit text, transliteration, and translation.

Error Correction: Verified content to remove typographical or grammatical errors.

Metadata Collection:

Metadata was created to enrich the dataset, including:

Contextual tags (e.g., “duty,” “detachment,” “faith”).

Related philosophical concepts (e.g., Upanishadic references).

Organizing the Dataset

The dataset was stored in a structured format to enable efficient retrieval and processing. Formats included:

Database Structure:

Relational Databases: Tables with fields like Chapter, Verse, Translation, Commentary, and Theme.

JSON Files: For easy integration with modern NLP frameworks.

Embedding Preparation:

Each verse and its associated data were vectorized (converted into embeddings) using language models like BERT or Sentence Transformers.

This enabled semantic similarity searches.

Challenges in Data Collection

Authenticity:

Ensuring that the translations and interpretations are authentic and align with the original teachings of the Bhagavad Gita.

Resolved by relying on well-known, credible sources.

Language Variations:

Sanskrit verses required accurate transliteration and translation for effective NLP processing.

Addressed by cross-referencing multiple translations for accuracy.

Volume of Data:

The dataset required sufficient breadth to answer diverse user queries, necessitating the inclusion of supplementary texts and thematic classifications.

Preprocessing Complexity:

Cleaning and organizing the data in a machine-readable format was time-consuming but essential for accurate chatbot responses.

Ensuring Data Quality

Manual Review:

Human experts reviewed translations and commentaries to ensure consistency and correctness.

Cross-validation was performed to resolve discrepancies between sources.

Thematic Categorization:

Each verse was categorized into themes to ensure accurate response generation. For example:

Karma Yoga: Focused on action and duty.

Bhakti Yoga: Focused on devotion and faith.

Jnana Yoga: Focused on knowledge and wisdom.

Version Control:

A system was established to track updates or changes in the dataset, ensuring ongoing quality management.

Tools and Technologies Used

Python Libraries:

Pandas: For organizing and preprocessing data.

NLTK/Spacy: For tokenization and cleaning.

Tiktoken: For preparing data for transformer-based models.

Database Systems:

SQLite/MySQL: For storing structured data.

NoSQL Databases: For flexibility in handling JSON-like hierarchical data.

Cloud Storage:

Platforms like Google Drive or AWS S3 were used to securely store large datasets.

**3.5 Exploratory Data Analysis (EDA)**

Exploratory Data Analysis (EDA) is a crucial step in the development of the Spiritual Chatbox, as it provides insights into the structure, patterns, and relationships within the dataset. The goal of EDA in this project is to understand the composition of the Bhagavad Gita dataset and ensure its suitability for machine learning and natural language processing (NLP) tasks.

Purpose of EDA

To analyze the dataset’s structure, content, and distribution.

To identify patterns, correlations, or themes within the Bhagavad Gita verses.

To detect and address inconsistencies or anomalies in the data.

To guide preprocessing and feature engineering for efficient chatbot implementation.

Steps in EDA

Understanding the Dataset Structure:

The dataset contains the following fields:

Chapter Number and Name: Helps in thematic grouping.

Verse Number: Unique identifier for each verse.

Original Sanskrit Text: The primary scripture.

Transliteration: Phonetic representation of Sanskrit text.

Translation: English translations from multiple sources.

Commentary: Interpretations by scholars.

Theme Tags: Categorization of verses into topics (e.g., Karma Yoga, Bhakti Yoga).

Data is stored in tabular form (e.g., CSV, JSON) for easy exploration.

Statistical Overview:

Calculate basic statistics to understand the dataset:

Total number of verses: 700 verses across 18 chapters.

Distribution of verses per chapter: Identify chapters with the highest/lowest content.

Word count per verse: Analyze length variations across verses.

Example: Chapter 2 (Sankhya Yoga) has the highest number of verses, making it a focal area for many user queries.

Textual Data Analysis:

Word Frequency:

Identify the most frequently used words in translations and commentaries.

Example: Common terms include karma (action), jnana (knowledge), and dharma (duty).

N-gram Analysis:

Analyze bi-grams and tri-grams to find common phrases (e.g., "perform your duty," "selfless action").

Theme Distribution:

Visualize the proportion of verses related to themes like Karma Yoga, Bhakti Yoga, and Jnana Yoga.

Data Visualization:

Use graphs and charts for better understanding:

Bar Chart: Verse distribution across chapters.

Word Cloud: Highlight common words/themes in translations.

Pie Chart: Proportion of themes in the dataset.

Line Graph: Average verse length per chapter.

Missing Data Identification:

Check for missing or incomplete fields (e.g., absent translations or commentaries).

Address issues by supplementing missing data from trusted sources or tagging them for manual review.

Thematic Insights:

Group verses by themes and explore patterns:

Example: Verses about Bhakti Yoga often emphasize devotion and surrender, whereas Karma Yoga focuses on action and duty.

Cross-reference themes with chapter distribution to identify thematic concentration.

Sentiment Analysis:

Perform basic sentiment analysis on translations and commentaries to gauge emotional tones (e.g., positive, neutral, or introspective tones).

Example: Verses on Bhakti Yoga often exhibit positive and devotional tones.

Correlation Analysis:

Explore relationships between verse lengths, themes, and chapters.

Example: Chapters emphasizing action (e.g., Karma Yoga) may have more concise verses compared to philosophical chapters like Jnana Yoga.

Tools and Techniques Used in EDA

Python Libraries:

Pandas: For data manipulation and cleaning.

Matplotlib and Seaborn: For creating visualizations.

NLTK/Spacy: For tokenization and text analysis.

WordCloud: For generating word cloud visualizations.

Visualization Examples:

Bar Graphs: Showing chapter-wise verse count.

Word Cloud: Visualizing frequently occurring words.

Heatmap: Highlighting correlations between themes and verse characteristics.

Insights Gained from EDA

Balanced Dataset:

The Bhagavad Gita dataset is well-distributed across chapters, ensuring comprehensive coverage of themes.

Theme Concentration:

Chapters like 2 (Sankhya Yoga) and 12 (Bhakti Yoga) dominate specific themes.

Data Quality:

Inconsistent translations or missing commentaries were identified, which need preprocessing or augmentation.

Word Patterns:

Repeated emphasis on key spiritual concepts like karma (action), atma (soul), and yoga (union).

**3.6 Train – Test, Split and Validation**

The Train-Test Split and Validation process is essential for building a robust and reliable machine learning model for the Spiritual Chatbox. This step ensures that the chatbot accurately responds to user queries by effectively learning from the data and generalizing to unseen inputs. Below is a detailed explanation of this process:

Purpose of Train-Test Split and Validation

Training: Teach the model to understand patterns in the Bhagavad Gita dataset, including semantic relationships between queries and verses.

Validation: Evaluate the model’s performance during training to tune hyperparameters and avoid overfitting.

Testing: Assess the final model's generalization ability using a separate set of unseen data.

Steps Involved

Step 1: Data Preparation

The dataset includes structured data such as:

Verses: Original Sanskrit text.

Translations: English translations.

Commentaries: Interpretations for context.

Themes: Tags like Karma Yoga, Bhakti Yoga, and Jnana Yoga.

Step 2: Splitting the Dataset

The dataset is divided into three subsets:

Training Set (70%): Used to train the model and learn embedding relationships.

Validation Set (15%): Used to tune hyperparameters and validate performance during training.

Test Set (15%): Reserved for final evaluation to measure how well the model generalizes.

Example:

Dataset Size: 700 verses.

Split: 490 (training), 105 (validation), 105 (testing).

Step 3: Splitting Techniques

Randomized Splitting:

Ensures all subsets have a similar distribution of themes and verse lengths.

Example: If 30% of the dataset pertains to Bhakti Yoga, this proportion is maintained across subsets.

Stratified Sampling:

Ensures even representation of categories like chapters and themes, avoiding bias toward any specific topic.

Step 4: Embedding Generation

Before training, the text (verses, translations, and queries) is converted into numerical embeddings:

Transformer-based Models: Such as BERT or Sentence Transformers for encoding textual data.

Embeddings capture semantic meanings, enabling similarity matching between user queries and dataset content.

Step 5: Training the Model

Objective:

Train the model to match user queries with the most relevant verses or explanations.

Process:

Input: Preprocessed embeddings of verses and themes.

Output: Embedding similarity scores (e.g., cosine similarity) between user queries and verses.

The model learns to retrieve the closest matches based on training data.

Step 6: Validation

Purpose:

Monitor performance during training to avoid overfitting or underfitting.

Optimize hyperparameters (e.g., learning rate, embedding dimensions).

Metrics:

Validation Loss: Measures error on the validation set.

Accuracy: Percentage of correct matches on validation queries.

Precision & Recall: Evaluate relevance and completeness of retrieved results.

Step 7: Testing

Purpose:

Assess the model’s final performance on unseen data.

Metrics:

F1 Score: Balance between precision and recall.

BLEU Score: Measures the similarity between the model’s responses and ground truth translations.

Mean Reciprocal Rank (MRR): Evaluates the ranking quality of retrieved verses.

Challenges in Train-Test Split and Validation

Data Imbalance:

Some themes (e.g., Bhakti Yoga) may dominate the dataset, causing bias.

Solution: Use stratified sampling to ensure balanced subsets.

Overfitting:

The model may perform well on the training set but poorly on unseen data.

Solution: Use validation loss and early stopping to prevent overfitting.

Semantic Complexity:

Matching philosophical concepts from user queries to verses requires high-quality embeddings.

Solution: Leverage advanced transformer-based models.

Tools and Libraries

Python Libraries:

Scikit-learn: For train-test splitting and performance metrics.

Tiktoken: For embedding preparation.

Models:

BERT/Sentence Transformers: For semantic text embeddings.

Frameworks:

Langchain: For managing retrieval workflows.

Benefits of Train-Test Split and Validation

Ensures that the chatbot provides accurate and meaningful responses.

Improves model robustness and reduces bias or overfitting.

Validates the chatbot’s ability to generalize well to diverse user queries.

**3.7 Results**

Evaluation Metrics

The performance of the Spiritual Chatbox model is assessed using the following metrics:

Accuracy: The proportion of correct verse matches to user queries.

Precision and Recall: Evaluate the relevance and completeness of retrieved results.

BLEU Score: Measures the fluency and contextuality of generated responses.

MRR (Mean Reciprocal Rank): Evaluates the ranking accuracy of retrieved verses.

**3.8 Web App and Model Deployment**

The Spiritual Chatbox is deployed as a web application using Streamlit. This platform enables rapid development of interactive applications without the need for extensive front-end coding.

Frontend (User Interface):

Streamlit provides a clean and interactive UI where users can type questions and receive answers from the chatbot.

The interface shows the conversation in a chat format, enhancing user experience.

Backend (Model Deployment):

The trained machine learning model (incorporating Google Generative AI, Langchain, Faiss, and other components) is deployed on the server.

When a user asks a question, the backend processes the input, retrieves relevant data, and generates an appropriate response.

Deployment Tools:

Heroku, AWS, or Google Cloud: These platforms are used for deploying

the model and the web application.

Docker: For containerizing the application and ensuring that it runs

smoothly across different environments.

Scalability: The system is designed to handle a large number of users simultaneously, ensuring smooth performance even during peak usage times.

**3.9 Web App Creation Codes**

Home.py

import os

import streamlit as st

import google.generativeai as genai

from streamlit\_chat import message  # Ensure this import is present

from modules.embedder import Embedder

from modules.history import ChatHistory

from modules.layout import Layout

from modules.sidebar import Sidebar

# Initialize Layout and Sidebar

layout = Layout()

sidebar = Sidebar()

# Display header

layout.show\_header("CSV")

# API key input

user\_api\_key = st.sidebar.text\_input(

    label="#### Your Gemini API key 👇",

    placeholder="Paste your Gemini API key",

    type="password"

)

file\_path = "C://Users//Abhinav Mishra//OneDrive//Desktop//Projects//Minor\_Project//Bhagwad\_Gita.csv"

# Initialize the Embedder class

embedder = Embedder()

# Check if the API key is provided and the file exists

if user\_api\_key and os.path.exists(file\_path):

    genai.configure(api\_key=user\_api\_key)  # Set the Gemini API Key

    vectors = embedder.getDocEmbeds(file\_path=file\_path, original\_filename=os.path.basename(file\_path))

    # Initialize chat history

    chat\_history = ChatHistory()

    chat\_history.initialize(os.path.basename(file\_path))

    # Function to handle conversational chat using Gemini API

    def conversational\_chat(query):

        model = genai.GenerativeModel("gemini-1.5-flash")

        response = model.generate\_content(query)

        result = response.text  # Extracting the response

        st.session\_state['history'].append((query, result))

        return result

    # Initialize session state for conversation history

    if 'history' not in st.session\_state:

        st.session\_state['history'] = []

    if 'generated' not in st.session\_state:

        st.session\_state['generated'] = [chat\_history.default\_prompt(os.path.basename(file\_path))]

    if 'past' not in st.session\_state:

        st.session\_state['past'] = [chat\_history.default\_greeting()]

    # Containers for chat

    response\_container = st.container()

    container = st.container()

    # User input form

    with container:

        with st.form(key='my\_form', clear\_on\_submit=True):

            user\_input = st.text\_input("Query:", placeholder="Talk about your CSV data here (:", key='input')

            submit\_button = st.form\_submit\_button(label='Send')

        if submit\_button and user\_input:

            output = conversational\_chat(user\_input)

            st.session\_state['past'].append(user\_input)

            st.session\_state['generated'].append(output)

    # Display conversation history

    if st.session\_state['generated']:

        with response\_container:

            for i in range(len(st.session\_state['generated'])):

                message(st.session\_state["past"][i], is\_user=True, key=str(i) + '\_user', avatar\_style="big-smile")

                message(st.session\_state['generated'][i], key=str(i), avatar\_style="thumbs")

else:

layout.show\_api\_key\_missing()  # Use layout to show message if API key is missing or file doesn't exist

Chatbot.py

import streamlit as st

import google.generativeai as genai

class Chatbot:

    def \_\_init\_\_(self, api\_key, model\_name="gemini", temperature=0.5, vectors=None):

        # Configure Gemini API with the provided API key

        genai.configure(api\_key=api\_key)

        self.model\_name = model\_name

        self.temperature = temperature

        self.vectors = vectors  # If you're using vectors, you can handle them as needed.

    qa\_template = """

        You are a helpful AI assistant named GitaGPT.

        If you don't know the answer, just say you don't know. Do NOT try to make up an answer.

        If the question is not related to the context, politely respond that you are tuned to only answer questions that are related to the context.

        Use as much detail as possible when responding.

        context: {context}

        =========

        question: {question}

        ======

        """

    def conversational\_chat(self, query):

        """

        Start a conversational chat using the Google Gemini API

        """

        if not query:

            return "Please ask a valid question."

        # Assume that the vectors represent the context you're extracting from CSV or any documents

        context = self.get\_context\_from\_vectors()

        # Compose the full prompt with context and query

        prompt = self.qa\_template.format(context=context, question=query)

        # Call the Gemini chat API

        response = genai.chat(messages=[prompt])

        # Extract the response content

        answer = response["content"]

        # Append the query and answer to the conversation history

        st.session\_state["history"].append((query, answer))

        return answer

    def get\_context\_from\_vectors(self):

        """

        Helper function to retrieve context from vectors.

        This method assumes you are using the document embeddings (vectors) stored earlier.

        """

        if self.vectors:

            # If you're using document embeddings, extract context from them.

            context = " ".join([str(doc) for doc in self.vectors])  # Simplified context extraction from vectors

        else:

            context = "No context available."

        return context

def count\_tokens\_chain(chain, query):

    """

    This function is not applicable for Gemini API as token counting is specific to OpenAI.

    """

    return "Token counting is not supported with Gemini API."

Embedder.py

import os

import pickle

from langchain.document\_loaders.csv\_loader import CSVLoader

from langchain.vectorstores import FAISS

class Embedder:

    def \_\_init\_\_(self):

        self.PATH = "embeddings"

        self.createEmbeddingsDir()

    def createEmbeddingsDir(self):

        if not os.path.exists(self.PATH):

            os.mkdir(self.PATH)

    def storeDocEmbeds(self, file\_path, original\_filename):

        loader = CSVLoader(file\_path=file\_path, encoding="utf-8", csv\_args={'delimiter': ','})

        data = loader.load()

        # Store CSV data or process it as needed

        with open(f"{self.PATH}/{original\_filename}.pkl", "wb") as f:

            pickle.dump(data, f)

    def getDocEmbeds(self, file\_path, original\_filename):

        if not os.path.isfile(f"{self.PATH}/{original\_filename}.pkl"):

            self.storeDocEmbeds(file\_path, original\_filename)

        with open(f"{self.PATH}/{original\_filename}.pkl", "rb") as f:

            vectors = pickle.load(f)

        return vectors

History.py

import streamlit as st

from streamlit\_chat import message

class ChatHistory:

    def \_\_init\_\_(self):

        # Initialize history from session state or an empty list

        self.history = st.session\_state.get("history", [])

        st.session\_state["history"] = self.history

    def default\_greeting(self):

        return "Hey Assistant ! 👋"

    def default\_prompt(self, topic):

        return f"Hello ! Ask me anything about {topic} 🤗"

    def initialize\_user\_history(self):

        # Initialize user messages if not already present in session state

        if "user" not in st.session\_state:

            st.session\_state["user"] = [self.default\_greeting()]

    def initialize\_assistant\_history(self, uploaded\_file\_name):

        # Initialize assistant messages if not already present in session state

        if "assistant" not in st.session\_state:

            st.session\_state["assistant"] = [self.default\_prompt(uploaded\_file\_name)]

    def initialize(self, uploaded\_file\_name):

        # Initialize both user and assistant histories

        self.initialize\_assistant\_history(uploaded\_file\_name)

        self.initialize\_user\_history()

    def reset(self, uploaded\_file\_name):

        # Clear the chat history

        st.session\_state["history"] = []

        st.session\_state["reset\_chat"] = False

        # Reinitialize user and assistant history after reset

        self.initialize\_user\_history()

        self.initialize\_assistant\_history(uploaded\_file\_name)

    def append(self, mode, message\_text):

        # Append messages to session state, either "user" or "assistant"

        if mode in st.session\_state:

            st.session\_state[mode].append(message\_text)

    def generate\_messages(self, container):

        # Generate the chat messages in the UI

        if st.session\_state.get("assistant"):

            with container:

                for i in range(len(st.session\_state["assistant"])):

                    message(

                        st.session\_state["user"][i],

                        is\_user=True,

                        key=f"history\_{i}\_user",

                        avatar\_style="big-smile",

                    )

                    message(st.session\_state["assistant"][i], key=str(i), avatar\_style="thumbs")

Layout.py

import streamlit as st

class Layout:

    def show\_header(self, types\_files="CSV"):

        """

        Displays the header of the app

        """

        st.markdown(

            f"""

            <h1 style='text-align: center;'> Ask about Bhagwad Gita</h1>

            """,

            unsafe\_allow\_html=True,

        )

    def show\_api\_key\_missing(self):

        """

        Displays a message if the user has not entered an API key for Google Gemini API

        """

        st.markdown(

            """

            <div style='text-align: center;'>

                <h4>Enter your <a href="https://aistudio.google.com/app/apikey" target="\_blank">Google Gemini API key</a> to start chatting</h4>

            </div>

            """,

            unsafe\_allow\_html=True,

        )

    def prompt\_form(self):

        """

        Displays the prompt form and checks whether the form is ready

        """

        with st.form(key="my\_form", clear\_on\_submit=True):

            user\_input = st.text\_area(

                "Query:",

                placeholder="Ask me anything...",

                key="input",

                label\_visibility="collapsed",

            )

            submit\_button = st.form\_submit\_button(label="Send")

            is\_ready = submit\_button and user\_input

Sidebar.py

import streamlit as st

class Sidebar:

    MODEL\_OPTIONS = ["gpt-3.5-turbo", "gpt-4"]

    TEMPERATURE\_MIN\_VALUE = 0.0

    TEMPERATURE\_MAX\_VALUE = 1.0

    TEMPERATURE\_DEFAULT\_VALUE = 0.0

    TEMPERATURE\_STEP = 0.01

    @staticmethod

    def about():

        about = st.sidebar.expander("🧠 About this Bot ")

        sections = [

            "#### It is an AI chatbot with a conversational memory, designed to allow users to discuss their queries related to spiritual texts in a more intuitive way. 📄",

            "#### It uses large language models to provide users with natural language interactions about Spiritual Texts. 🌐",

            "#### Powered by [Langchain](https://github.com/hwchase17/langchain) and [Streamlit](https://github.com/streamlit/streamlit) ⚡",

        ]

        for section in sections:

            about.write(section)

    @staticmethod

    def reset\_chat\_button():

        # Set default for reset\_chat first, then check button state

        st.session\_state.setdefault("reset\_chat", False)

        if st.button("Reset chat"):

            st.session\_state["reset\_chat"] = True

    def model\_selector(self):

        st.session\_state.setdefault("model", self.MODEL\_OPTIONS[0])

        model = st.selectbox(label="Model", options=self.MODEL\_OPTIONS)

        st.session\_state["model"] = model

    def temperature\_slider(self):

        st.session\_state.setdefault("temperature", self.TEMPERATURE\_DEFAULT\_VALUE)

        temperature = st.slider(

            label="Temperature",

            min\_value=self.TEMPERATURE\_MIN\_VALUE,

            max\_value=self.TEMPERATURE\_MAX\_VALUE,

            value=st.session\_state["temperature"],  # Set to session state value

            step=self.TEMPERATURE\_STEP,

        )

        st.session\_state["temperature"] = temperature

    def show\_options(self):

        with st.sidebar.expander("🛠️ Tools", expanded=False):

            # Ensure defaults are set before displaying options

            self.reset\_chat\_button()

            self.model\_selector()

            self.temperature\_slider()

            st.session\_state.setdefault("model", self.MODEL\_OPTIONS[0])

            st.session\_state.setdefault("temperature", self.TEMPERATURE\_DEFAULT\_VALUE)

Utils.py

import os

import pandas as pd

import streamlit as st

import google.generativeai as genai

from modules.chatbot import Chatbot

from modules.embedder import Embedder

class Utilities:

    @staticmethod

    def load\_api\_key():

        """

        Loads the Gemini API key from the .env file or

        from the user's input and returns it.

        """

        if not hasattr(st.session\_state, "api\_key"):

            st.session\_state.api\_key = None

        # You can define your API key in .env directly

        if os.path.exists(".env") and os.environ.get("GEMINI\_API\_KEY") is not None:

            user\_api\_key = os.environ["GEMINI\_API\_KEY"]

            st.sidebar.success("API key loaded from .env", icon="🚀")

        else:

            if st.session\_state.api\_key is not None:

                user\_api\_key = st.session\_state.api\_key

                st.sidebar.success("API key loaded from previous input", icon="🚀")

            else:

                user\_api\_key = st.sidebar.text\_input(

                    label="#### Your Gemini API key 👇", placeholder="Enter your Gemini API key", type="password"

                )

                if user\_api\_key:

                    st.session\_state.api\_key = user\_api\_key

        return user\_api\_key

    @staticmethod

    def setup\_chatbot(uploaded\_file, model, temperature):

        """

        Setup the chatbot using the provided file and model details, using Gemini API.

        """

        embeds = Embedder()

        with st.spinner("Processing..."):

            if uploaded\_file.name.endswith(".csv"):

                # Read and process CSV file content

                df = pd.read\_csv(uploaded\_file)

                # Convert CSV content into a text format suitable for embeddings

                file\_content = df.to\_string(index=False)

            else:

                st.error("Unsupported file type. Only CSV is allowed for this setup.")

                return None

            # Get the document embeddings for the CSV content

            vectors = embeds.getDocEmbeds(file\_content, uploaded\_file.name)

            # Create a Chatbot instance with the specified model and temperature

            chatbot = Chatbot(model, temperature, vectors, api\_key=st.session\_state.api\_key)

        st.session\_state["ready"] = True

        return chatbot

**Chapter 4: Conclusions and Recommendations**

The development of the Spiritual Chatbox reflects the synthesis of modern artificial intelligence with the timeless wisdom of the Bhagavad Gita. This project aimed to create a platform that could bridge the gap between traditional spiritual teachings and contemporary digital needs. The integration of advanced NLP techniques, a robust dataset, and user-centric design has resulted in a tool capable of offering personalized spiritual guidance.

The foundation of this project, as laid out in Chapter 1, established the importance of addressing the growing demand for accessible spiritual resources. The Bhagavad Gita, a profound text, was chosen as the core dataset due to its universality and depth. With an emphasis on user engagement and accessibility, the project aimed to make these teachings available to users of varying cultural and linguistic backgrounds. Software tools like Python and LangChain were identified early as instrumental for achieving these goals, forming a solid technical backbone.

Chapter 2 provided a detailed analysis of the system’s requirements and design. The tool selection—such as FAISS for efficient data retrieval and Streamlit for the user interface—ensured the system’s performance and usability. Flowcharts, including use-case and data-flow diagrams, outlined a clear roadmap for development and implementation. These design elements ensured a seamless user experience, allowing for intuitive query input and precise retrieval of relevant verses and interpretations.

Chapter 3 delved into the technical implementation, with a focus on machine learning methodologies. The data preprocessing, embedding generation, and model training were carefully designed to ensure accurate results. Exploratory Data Analysis (EDA) provided valuable insights into the dataset, such as theme distributions and language complexities. The use of train-test splits and validation techniques minimized overfitting and maximized the chatbot’s reliability. Finally, the deployment of the chatbot as a web application demonstrated the successful realization of the project’s objectives, offering real-time interaction and meaningful responses.

The results of this project are significant, as the chatbot achieves its primary goal of making spiritual wisdom accessible and engaging. It serves as an innovative tool for learning and reflection, with applications in education, self-development, and cultural preservation. However, the project also revealed areas for improvement. Expanding the dataset to include other spiritual texts and enabling multi-language support would enhance the system’s inclusivity and utility. Additionally, implementing advanced NLP techniques, such as conversational memory, would allow the chatbot to handle complex, multi-turn dialogues.

Future iterations of the project could focus on personalized spiritual guidance, where the chatbot adapts responses based on user preferences and interactions. Integrating features like voice-based interaction and mobile app deployment would make the chatbot more versatile and accessible. Furthermore, ethical considerations should remain a priority, ensuring that the chatbot respects cultural sensitivities and maintains the integrity of the teachings it represents.

In conclusion, the Spiritual Chatbox successfully merges ancient wisdom with modern technology, creating a transformative tool for spiritual exploration. Its development highlights the potential of AI to preserve and disseminate cultural heritage, offering users a unique pathway to engage with the profound teachings of the Bhagavad Gita. With continued refinement and expansion, the chatbot promises to make an even greater impact in the future.

**Chapter-5 REFERENCES**

* Project Gita-Gpt by “samanyougarg” [Online] Available :- https://github.com/gita/Bhagavad-Gita-AI
* Joel Ramiro Caballero Castellanos;Manuel Cardona;José Luis Ordoñez-Avila. “Chatbot for Validation of Research Topics for Engineering Students”, in IEEE 41st Central America and Panama Convention (CONCAPAN XLI), 2023.
* We have completed proctured Infosys SpringBoard Courses also :-

**Abhinav Mishra (221B011)**

**Pre-Requisite Courses** :-

1. Introduction To Artificial Intelligence



1. Introduction To Deep Learning

****

1. Introduction To NLP

****

**Proctored Course:-**

1. Artificial Intelligence Foundation



**Shivani Singh (221B359)**

**Pre-Requisite Courses:-**

1. Natural Language Processing For Developers



**Proctored Course:-**

1. Natural Language Processing Foundation



**Shivangi Kumari (221B358)**

**Pre-Requisite Courses:-**

1. Natural Language Processing For Developers



**Proctored Course:-**

1. Natural Language Processing Foundation



**PERSONAL DETAILS**



**Name** **:-**  Abhinav Mishra

**Enrollment Number :-** 221B011

**Batch :-** B1

**Email :-** [abhi8899ggjj@gmail.com](mailto:abhi8899ggjj@gmail.com)



**Name :-** Shivani Singh

**Enrollment Number :-** 221B359

**Batch :-** B11

**Email :-** [shivvii2107@gmail.com](mailto:shivvii2107@gmail.com)

****

**Name :-** Shivangi Kumari

**Enrollment Number :-** 221B358

**Batch :-** B11

**Email :-** [kshivangi028@gmail.com](mailto:kshivangi028@gmail.com)