Project Summary – EpiRacing CHATBOT PROJECT

Submitted by: ALAGO CHIEMELA VICTOR

Projects Dates: September 2024.

Completed in partial fulfillment of the requirements for the course:

Semester 4 Final Project

At:

EPITA, SCHOOL OF ENGINEERING AND COMPUTER SCIENCE

Academic Year: 2023/2024

Paris, 30th September 2024.

Introduction

Welcome to the EpiRacing Chatbot, an innovative project designed to enhance the experience of EPITA students and enthusiasts in the exhilarating world of motorsports. This project was crafted not just to serve information but to engage users in meaningful interactions, facilitating access to detailed event information, club activities, and comprehensive user support through a seamlessly integrated digital platform.

Developed with advanced machine learning techniques and robust web technologies, the EpiRacing Chatbot is tailored to meet the specific needs and interests of the racing community at EPITA. It adheres rigorously to a set of detailed project requirements, ensuring a conversational agent that not only delivers information but also understands and interacts with users in a context-aware manner. This initiative bridges the gap between digital interaction and real-world racing enthusiasm, fostering a vibrant community of informed and engaged members.

Project Goals

The main objectives of the EpiRacing Chatbot included:

- Developing a robust chatbot using PyTorch for natural language understanding.
- Implementing a user-friendly web interface using Flask and Jinja.
- Ensuring user authentication for personalized experiences.
- Providing real-time responses about club events, registration, and general inquiries.

Implementation Details

Backend Development

- **Model Training:** Utilized PyTorch to train a neural network on a predefined dataset (intents.json) which includes patterns and responses for different conversational intents such as greetings, event information, and user assistance. I talked more about this in detail further down in this documentation.

Files: model.py, train.py, nltk_utils.py, intents.json.

- **Chatbot Logic:** The chat logic in chat.py processes user inputs using natural language processing techniques implemented in nltk_utils.py to understand and generate appropriate responses. This also includes the integration of Openai responses to support the model for better overall performance.

Files: chat.py

- Dual-**Response System in Chat Logic:** The chat logic within chat.py is engineered to handle user queries with a dual-response mechanism. It first attempts to generate responses using a locally trained PyTorch model. If the confidence level of the model's response is below a predetermined threshold (e.g., 0.75), it seamlessly switches to querying the OpenAI's ChatGPT API to fetch a response. This hybrid approach ensures that the chatbot maintains high-quality interactions, providing reliable and contextually appropriate responses even when the initial model output is uncertain.

Files: chat.py

- **Application and Routing:** The Flask application defined in app.py handles all server-side routing and user session management. It serves the front-end pages, processes login and registration requests, and directs API calls to the chatbot for processing user queries. This file is crucial for integrating the user interface with the backend logic, ensuring that requests are authenticated, and responses are correctly returned to the user.

Files: app.py

- **Database Integration:** To manage user data and session information securely, the project integrates a database system via Flask's session management capabilities. User credentials and session data are stored in a structured format, allowing efficient retrieval and updating of user information. This setup not only supports the authentication process through Flask-Login but also ensures that user interactions with the chatbot are personalized and secure. The database plays a crucial role in maintaining the integrity and security of user data while providing a robust backend framework for the application.

Files: app.py

Frontend Development

- **User Interface:** Developed using HTML, CSS, and JavaScript, with Flask serving the HTML pages. The interface includes home page, login, registration, and profile management features.

Files: base.html, login.html, register.html, profile.html

- **Interactivity:** JavaScript and AJAX were used for handling real-time chat interactions without page reloads, enhancing the user experience. This also ensures that the chat is saved until the session is finished.

Files: app.js

Security Measures

- Implemented user authentication using Flask-Login to manage user sessions securely. Passwords are hashed for secure storage, and CSRF protection is enabled for all forms.

Files: app.py

- Ensured that there are no major loopholes in my code architecture and writing user SonarLint extension, hence improving the app security.

System Architecture

The system architecture includes:

- **Frontend:** HTML/CSS/JS with Bootstrap for responsive design.
- **Backend:** Flask application serving dynamic web pages and handling RESTful requests.
- **Databases:** Python sqlite3 to store and retrieve user information and privacy concerns.
- **AI Model:** PyTorch model loaded in the backend to process and respond to user queries.

Model Training in Detail

- **Intents JSON (intents. json):** The intents.json file acts as the core dataset for training your chatbot. It is structured to include various “intents” that the chatbot can recognize, each with associated “patterns” (user inputs that the model should learn to recognize) and “responses” (how the bot should respond to those inputs). Each intent represents a different type of conversation or question a user might have, such as greetings, asking about event details, or seeking user assistance. Here’s an example structure:

```
{
  "intents": [
    {
      "tag": "greeting",
      "patterns": ["Hello", "Hi", "Hey"],
```

```
    "responses": ["Hi there!", "Hello! How can I help you?"]
   },
   {
    "tag": "event_information",
    "patterns": ["When is the next race?", "List upcoming events"],
    "responses": ["The next race is on Sunday, 10th October."]
   }
  ]
}
```

• **Neural Network Model (model.py):** The model.py file defines the neural network architecture used for understanding user intents. This file typically includes a class defining the neural network with layers suitable for natural language processing tasks. A common setup might involve:

- **Input Layer**: Accepts a vector of size input_size, which corresponds to the vocabulary size. This layer is responsible for receiving the input feature set.
- **Hidden Layers**: One or more layers for processing features.
- **Output Layer**: The final layer, l3, maps the features from the hidden space to num_classes, which is the total number of intent tags. This layer is crucial for classifying the input into one of several categories.

• **Training Script (train.py):** train.py manages the training process using the PyTorch library. It reads the intents.json, processes the data into a suitable format (tokenization, stemming, creating a bag of words), splits it into features (X) and labels (Y), and then feeds it into the neural network for training. The script will typically involve:

- Loading and preprocessing data using functions from nltk_utils.py.
- Defining a training loop that repeatedly feeds data through the model, calculates the loss, and adjusts the model weights using backpropagation.
- Saving the trained model to a file like data.pth for later use in predictions.

• **Natural Language Utilities (nltk_utils.py):** This file includes helper functions for text processing needed before the training or during the inference, such as:

- **Tokenization**: Breaking sentences into words or tokens.
- **Stemming**: Reducing words to their base or root form.
- **Bag of Words**: Creating a fixed-size vector from a sentence based on the vocabulary, which is used as input for the neural network.

Together, these files and functionalities form the backbone of your chatbot's ability to understand and process user queries effectively. By training the model with diverse and comprehensive data (intents.json), and utilizing robust natural language processing techniques (nltk_utils.py), your chatbot is equipped to provide accurate and helpful responses, all managed through the training script (train.py) that orchestrates the learning process and saves the trained model for future use.

Some Challenges I Faced and Resolved

- Ensuring security and privacy of the user data during authentication and chat interactions.
- Handling varied user intents, since a user relatively ask very unrelated questions.
- Maintaining Chatbot Accuracy and Relevance

Future Enhancements I Can Make

- Expand the dataset for training the chatbot to cover more diverse intents and scenarios.
- Implement advanced NLP features like sentiment analysis to enhance interaction quality.
- Enhance security features and add support for multi-language interfaces.

Conclusion

The EpiRacing Chatbot stands as a testament to the potential of artificial intelligence to transform user engagement in niche communities. Through its dynamic interactions and tailored responses, the chatbot has significantly enhanced how club members at EPITA access and interact with information about motorsports.

This project is not only a demonstration of cutting-edge technology application but also a crucial step towards enriching the motorsport culture at EPITA. It has established a new standard for how clubs can interact with their members digitally, making the thrilling world of racing more accessible to a broader audience.

As the chatbot continues to evolve, it promises to play a pivotal role in the digital strategy of the motorsport club, driving forward both engagement and technological innovation.