GitHub FAQ Chatbot with Fallback Mechanism

BY SHIVANG SHUKLA
12214963

Table of Contents

Introduction

The GitHub FAQ Chatbot is a Telegram-based bot designed to answer questions related to Git and GitHub. It leverages natural language processing (NLP) and a pre-trained FAQ dataset to provide accurate responses. The system is containerized using Docker, ensuring easy deployment and scalability. Key components include:

- **Telegram Bot:** Handles user interactions.
- **Fast API Backend:** Processes queries and retrieves answers.
- **MongoDB:** Stores logs, user data, and analytics.
- **NLP Models:** Sentence-BERT for embeddings and T5 for answer enhancement.

Objectives

1. **User Interaction:** Provide a seamless Telegram interface for users to ask Git/GitHub-related questions.
2. **Efficient Query Handling**: Use NLP to match user questions with the closest FAQ entries.
3. **Answer Enhancement:** Improve answers dynamically using the T5 model.

4. **Logging and Analytics**: Track queries, user interactions, and system performance.
5. **Containerization**: Ensure the system is deployable via Docker for scalability and ease of use.

Implementation

A System Architecture

The system consists of three main services:

- **Bot Service**: Handles Telegram interactions (bot.py).
- **API Service**: Processes queries and retrieves answers (main.py).
- **MongoDB Service**: Stores data and logs.

B Containerization Process

The system is containerized using Docker, with the following steps:

Step 1: Dockerfile Setup

- **Multi-Stage Build**:
    1. **Builder Stage**: Installs Python dependencies, including PyTorch and NLP models.
    2. **Runtime Stage**: Copies only necessary artifacts (e.g., compiled dependencies, models) to the final image, reducing size.

- **Environment Variables**: Configured for Python, Transformers cache, and paths.
- **Model Caching**: Pre-downloaded models are copied to /app/cache to avoid redundant downloads.

Step 2: Docker Compose Configuration

- **Services**:
  1. api: Fast API backend exposed on port 8000.
  2. bot: Telegram bot service.
  3. mongo: MongoDB with persistent storage (mongo_data volume).
- **Dependencies**:
  1. The api service depends on MongoDB being healthy.
  2. The bot service depends on the api service being started.

Step 3: Environment Management

- . env **File**: Stores sensitive configurations like TELEGRAM_BOT_TOKEN and MONGO_URI.
- **Volume for Data Persistence**: MongoDB data is persisted using a Docker volume (mongo_data).

C DOCKER SETUP

a. **Build Images**:

```
docker-compose build
```

c. **Run Containers**:

```
docker-compose up
```

d. **Verify Services**:

i. API: Accessible at http://localhost:8000.
ii. Bot: Logs into Telegram using the provided token.
iii. MongoDB: Available at mongodb://mongo:27017.

D SERVICE CONFIGURATION

- **API Service (**main.py**)**:
  1. Loads the FAQ dataset and pre-trained models on startup.
  2. Provides a /query endpoint to handle user questions.
  3. Enhances answers using T5 and logs queries to MongoDB.
- **Bot Service (**bot.py**)**:
  1. Listens for Telegram messages.
  2. Forwards questions to the API and formats responses for users.
- **MongoDB**:
  1. Stores logs, user interactions, and analytics.

Outcomes

1. **Functional Telegram Bot**: Users can ask Git/GitHub questions and receive accurate, enhanced answers.
2. **Scalable Backend**: The Fast API service efficiently processes queries and scales with Docker.
3. **Persistent Logging**: All interactions are logged in MongoDB for analytics and debugging.
4. **Easy Deployment**: Docker Compose simplifies setup and ensures consistency across environments.
5. **Performance**: NLP models and FAISS indexing enable fast and relevant responses.

The project successfully delivers a robust, containerized chatbot system for GitHub-related FAQs, meeting all outlined objectives.