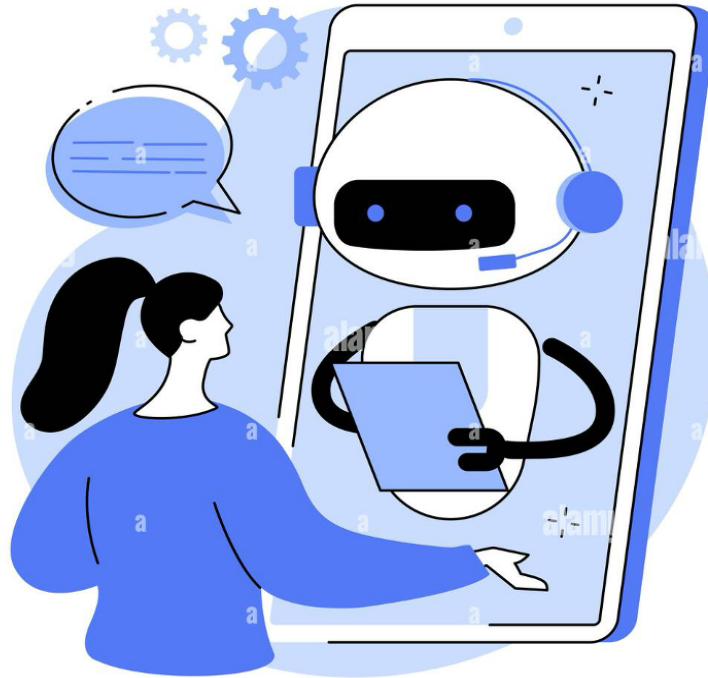


CSYE 7380 - Final Project Documentation: Simplified Digital Twin for a Customer Support System

Table of Contents

- [Introduction](#)
- [Project Goals](#)
- [Business Process Definition](#)
- [Data Representation](#)
- [System Architecture](#)
- [LLM Integration](#)
- [Digital Twin Simulation](#)
- [Using the Digital Twin](#)
- [Frontend and Backend Interaction](#)
- [Potential Enhancements](#)
- [Technologies Used](#)
- [Setup and Installation](#)
- [Demonstration Steps](#)
- [Challenges and Solutions](#)
- [Conclusion](#)

1. Introduction



The Simplified Digital Twin for a Customer Support System is a project designed to automate and simulate the process of handling customer support tickets. The system uses a **Large Language Model (LLM)** to:

- Categorize tickets based on user descriptions.
- Generate automated responses to support queries.
- Incorporate a feedback loop to improve the quality of responses over time.
- Enhance responses with Retrieval-Augmented Generation (RAG) for additional context.

This digital twin acts as a virtual representation of a real-world support system, reducing manual intervention and improving response efficiency.

2. Project Goals

1. **Automate Ticket Categorization:** Classify support tickets using an LLM.
2. **Generate Initial Responses:** Provide relevant, automated responses to user queries.
3. **Simulate Support Workflow:** Develop a digital twin that mirrors real-world support processes.
4. **Incorporate RAG:** Enhance response accuracy using external knowledge sources.
5. **Implement a Feedback Loop:** Collect user feedback to iteratively improve responses.

3. Business Process Definition

The customer support ticket system involves the following steps:

1. **Receiving a Ticket:** The user submits a support ticket with a description of their issue via the frontend interface.
2. **Categorizing the Ticket:** The LLM categorizes the ticket based on the description (e.g., Authentication Issues, Payment Issues).
3. **Generating an Initial Response:** The LLM provides an initial response tailored to the category of the ticket.
4. **Updating Ticket Status:** The system updates the status of the ticket (e.g., In Progress, Resolved) after processing.



4. Data Representation

Ticket Data Structure

Tickets are represented as Python dictionaries:

```
{
  "ticket_id": 1,
  "description": "I cannot log into my account.",
  "category": "Authentication/Login Issues",
  "response": "Please check your credentials and try resetting your password.",
  "status": "In Progress"
}
```

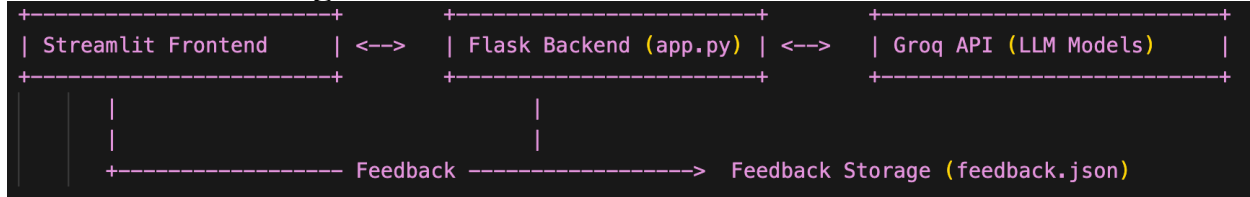
Feedback Data Structure

Feedback data collected from users and stored in: feedback.json.

```
{ } feedback.json > ...
[
  {
    "description": "login issue ",
    "model": "gemma-7b-it",
    "rating": 5,
    "feedback": "thanks "
  }
]
```

5. System Architecture

Architecture Diagram



Components

- **Frontend (Streamlit):** User interface for submitting tickets, selecting models, and providing feedback.
- **Backend (Flask):** Handles processing requests, categorization, response generation, and storing feedback.
- **Groq API (LLMs):** Provides LLMs like gemma-7b-it and llama3-8b-8192 for processing.
- **Feedback Storage:** Collects user feedback in feedback.json for future improvements.

6. LLM Integration

Categorization Function

```
# Function to categorize a ticket using the selected model
def categorize_ticket(description, model, use_rag=False):
    prompt = f"Categorize this support ticket: {description}"
    if use_rag:
        prompt = f"Use RAG to categorize this support ticket: {description}"

    response = client.chat.completions.create(
        model=model,
        messages=[{"role": "user", "content": prompt}],
        max_tokens=50
    )
    category = response.choices[0].message.content.strip()
    return category
```

Response Generation Function

```
# Function to generate a response using the selected model
def generate_response(category, description, model, use_rag=False):
    prompt = f"Generate a support response for a {category} issue: {description}"
    if use_rag:
        prompt = f"Use RAG to generate a support response for a {category} issue: {description}"

    response = client.chat.completions.create(
        model=model,
        messages=[{"role": "user", "content": prompt}],
        max_tokens=200
    )
    response_text = response.choices[0].message.content.strip()
    return response_text
```

7. Digital Twin Simulation

The backend functions (categorize_ticket and generate_response) act as the core components of the digital twin.

These functions simulate:

1. Ticket Categorization
2. Response Generation
3. Feedback Collection

8. Using the Digital Twin

Workflow Simulation

1. **Submit a Ticket:** Users describe their issue in the Streamlit frontend.
2. **Categorize & Respond:** The backend categorizes the ticket and generates a response.
3. **Review & Feedback:** Users provide feedback on the response, which is stored for improvement.

9. Frontend and Backend Interaction

The **Customer Support Chatbot** leverages a **Streamlit frontend** and a **Flask backend** to handle user interactions and process support tickets.

The interaction flow is as follows:

1. Frontend (Streamlit):

- The user inputs a ticket description, selects a model, and optionally enables RAG mode.
- Sends a POST request to the Flask backend (/process_ticket endpoint) with the ticket description and model details.
- Displays the category and generated response received from the backend.
- Allows users to submit feedback, which is sent to the backend (/submit_feedback endpoint).

2. Backend (Flask):

- Receives the ticket data, categorizes the ticket, and generates a response using the selected LLM model.
- Saves feedback into feedback.json when submitted.
- Handles communication with the Groq API for LLM processing.

Frontend Interaction (Streamlit)

```
# Button to send the query
if st.button("Submit Ticket"):
    if user_query:
        with st.spinner("Processing your ticket..."):
            try:
                response = requests.post(
                    "http://127.0.0.1:5000/process_ticket",
                    json={"description": user_query, "model": model_name, "use_rag": use_rag}
                )
                if response.status_code == 200:
                    data = response.json()
                    category = data["category"]
                    reply = data["response"]

                    # Display the response
                    st.subheader("Category:")
                    st.write(category)

                    st.subheader("Response:")
                    st.write(reply)

                    # Store the conversation
                    st.session_state.conversation_history.append(f"**You:** {user_query}")
                    st.session_state.conversation_history.append(f"**Bot:** {reply}")

                    # Store the current query and response for feedback submission
                    st.session_state.latest_query = user_query
                    st.session_state.latest_model = model_name
                    st.session_state.latest_reply = reply

            else:
                st.error(f"Error: {response.json().get('error', 'Unknown error occurred')}")

        except Exception as e:
            st.error(f"An error occurred: {e}")
```


Feedback Submission Interaction

```
if st.button("Submit Feedback"):
    try:
        feedback_response = requests.post(
            "http://127.0.0.1:5000/submit_feedback",
            json={
                "description": st.session_state.latest_query,
                "model": st.session_state.latest_model,
                "rating": rating,
                "feedback": feedback_text
            }
        )

        if feedback_response.status_code == 200:
            st.success("Thank you for your feedback!")
            # Optionally clear the feedback form after submission
            del st.session_state["latest_query"]
            del st.session_state["latest_model"]
            del st.session_state["latest_reply"]
        else:
            st.error("Failed to submit feedback.")

    except Exception as e:
        st.error(f"An error occurred: {e}")
```

Backend Endpoints (Flask)

```
@app.route("/process_ticket", methods=["POST"])
def process_ticket():
    data = request.get_json()
    description = data.get("description")
    model = data.get("model")
    use_rag = data.get("use_rag", False)

    if not description:
        return jsonify({"error": "Please provide a ticket description."}), 400

    try:
        # Categorize the ticket and generate a response
        category = categorize_ticket(description, model, use_rag)
        response = generate_response(category, description, model, use_rag)

        return jsonify({"category": category, "response": response})
    except Exception as e:
        return jsonify({"error": str(e)}), 500

@app.route("/submit_feedback", methods=["POST"])
def submit_feedback():
    feedback_data = request.get_json()
    feedback_file = "feedback.json"

    try:
        # Check if the file exists and contains valid JSON
        if os.path.exists(feedback_file):
            with open(feedback_file, "r") as file:
                try:
                    feedback_list = json.load(file)
                except json.JSONDecodeError:
                    feedback_list = [] # Initialize with an empty list if the file is corrupted
        else:
            feedback_list = []

        # Add the new feedback
        feedback_list.append(feedback_data)

        # Save the updated feedback list
        with open(feedback_file, "w") as file:
            json.dump(feedback_list, file, indent=4)

        return jsonify({"message": "Feedback submitted successfully."})
```

10. Potential Enhancements

- **Complex Ticket States:** Add states like Escalated, Pending Review.
- **Priority Levels:** Implement priority handling and SLA tracking.
- **Real-Time API Integration:** Connect to external APIs like Zendesk.
- **Advanced Feedback Loop:** Automatically adjust LLM prompts based on feedback.

11. Technologies Used

- Backend: Flask
- Frontend: Streamlit
- Models: Gemma-7b-it, LLaMA3-8b-8192
- API: Groq API
- Environment: Python 3.9, dotenv for environment variables

12. Setup and Installation

Follow these steps to set up and run the project on your local machine:

Prerequisites

1. Python 3.9 or higher
2. Dependencies listed in requirements.txt
3. Groq API Key (set in the .env file)

```
project-root/
|-- app.py           # Flask backend
|-- frontend.py      # Streamlit frontend
|-- run_chatbot.py   # Script to run both frontend and backend
|-- requirements.txt # List of dependencies
|-- sample_tickets.json # Sample ticket data
|-- feedback.json    # Stores user feedback
└── .env             # Environment variables (GROQ API key)
```

Installation Instructions

```
git clone <[your-repo-url](https://github.com/Nehagopinath15neu/GenAIFinal-Project.git)>  
cd <[your-repo-directory](https://github.com/Nehagopinath15neu/GenAIFinal-Project.git)>
```

Create and Activate a Virtual Environment

- python3 -m venv myenv
- `source` myenv/bin/activate

Install Dependencies

- pip install -r requirements.txt

Set Up Environment Variables

- GROQ_API_KEY=<your-groq-api-key>

Run the Project

- python run_chatbot.py

Access the Frontend

- http://localhost:8501

13. Demonstration Steps

1. Submit a Support Ticket:
 - o Enter a brief description of your issue.
 - o Select the "gemma-7b-it" or "llama3-8b-8192" model.
 - o Optionally, enable RAG Mode.
 - o Click "Submit Ticket."

Welcome to CSYE 7380 - Final Project

Customer Support Chatbot

Enter your support ticket description:

login issue

Choose Model:

gemma-7b-it

☒ Enable RAG Mode

Submit Ticket

Conversation History

With disabled RAG Mode

Welcome to CSYE 7380 - Final Project

Customer Support Chatbot

Enter your support ticket description:

login issue

Choose Model:

gemma-7b-it

☐ Enable RAG Mode

Submit Ticket

2. View the Response:

- The system will display the category and response generated by the LLM.

Category:

RAG Category for Login Issue Ticket

Level: Problem **Area:** Authentication **Component:** Login Process **Sub-area:** Session Management

Rationale:

- **Problem:** Indicates an issue affecting functionality.
- ****Authentication**

Response: ⇄

RAG Support Response for Login Issue Ticket

Issue: Login issue

Level: Problem

Affected Area: Authentication

Component: Login Process

Sub-area: Session Management

Possible Causes:

- Incorrect login credentials
- Account blocked due to multiple failed login attempts
- Technical issues with the login server
- Browser compatibility issues
- Cookies disabled in browser

Suggested Actions:

1. Verify Login Credentials:

- Ensure you're using the correct username and password.
- Check for any typos.
- Try logging in from a different browser or device.

2. Check Account Status:

- Verify if your account is suspended or blocked due to excessive login attempts.
- If so, contact our support team for assistance.

3. Technical Troubleshooting:

- Ensure your internet connection is stable.
- Clear your browser's cache and cookies.
- Try using a different browser or incognito mode.

Disabled RAG Mode Response

Customer Support Chatbot

Enter your support ticket description:

login issue

Choose Model:

gemma-7b-it

☐ Enable RAG Mode

Submit Ticket

Category:

Technical Support Category: User Account

Specific Issue: Login Difficulties

Response:

Support Response for User Account - Login Difficulties

Subject: Login Difficulties - Account [Account Name]

Dear [Customer Name],

Thank you for contacting us regarding login difficulties with your account. We understand that this can be frustrating, and we're happy to assist you.

Possible causes for login issues:

- Incorrect login credentials (username or password)
- Account disabled or suspended
- Browser compatibility issues
- Cookies disabled

Please verify the following:

- **Login credentials:** Are you using the correct username and password? Have you recently changed your password?
- **Account status:** Check if your account is disabled or suspended (check your email for any notifications).
- **Browser compatibility:** Ensure you're using a supported browser (latest versions of Chrome, Firefox, Safari, or Edge).
- **Cookies enabled:** Cookies allow websites to remember your login information. Make sure they're enabled in your browser settings.

3. Provide Feedback:

- Rate the response and add additional comments.
- Click "Submit Feedback" to save the feedback.

Rate the Response

Rate the quality of this response:

- ☐ 1
- ☐ 2
- ☐ 3
- ☐ 4
- ☒ 5

Additional feedback (optional):

very nice, great work

Submit Feedback

Thank you for your feedback!

Backend Feedback saving

```
{ } feedback.json > ...
[
  {
    "description": "login issue ",
    "model": "gemma-7b-it",
    "rating": 5,
    "feedback": "thanks "
  },
  {
    "description": "login issue",
    "model": "gemma-7b-it",
    "rating": 5,
    "feedback": "very nice, great work"
  }
]
```


4. View Conversation History:

- Previous conversations are displayed below the input form.

Conversation History

You: login issue

Bot: ## Support Response for User Account - Login Difficulties

Subject: Login Difficulties - Account [Account Name]

Dear [Customer Name],

Thank you for contacting us regarding login difficulties with your account. We understand that this can be frustrating, and we're happy to assist you.

Possible causes for login issues:

- Incorrect login credentials (username or password)
- Account disabled or suspended
- Browser compatibility issues
- Cookies disabled

Please verify the following:

- **Login credentials:** Are you using the correct username and password? Have you recently changed your password?
- **Account status:** Check if your account is disabled or suspended (check your email for any notifications).
- **Browser compatibility:** Ensure you're using a supported browser (latest versions of Chrome, Firefox, Safari, or Edge).
- **Cookies enabled:** Cookies allow websites to remember your login information. Make sure they're enabled in your browser settings.

14. Challenges and Solutions

Challenges Faced

- **Model Compatibility:**
 - Some models like llama2-13b-4096 and claire-3b-256 were not accessible due to API restrictions.
- **Feedback Storage:**
 - Implementing a reliable method to store feedback without overwriting existing data.
- **SSL Compatibility:**
 - The warning related to LibreSSL required suppressing with `NotOpenSSLWarning`.

Solutions Implemented

- **Working Models:**
 - Selected models like "**gemma-7b-it**" and "**llama3-8b-8192**" that were verified to work with the Groq API.
- **Feedback Loop:**
 - Feedback is appended to `feedback.json` to ensure data is not lost.
- **Suppressing Warnings:**
 - Added code to suppress `NotOpenSSLWarning` to avoid unnecessary terminal warnings.



15. Conclusion

The **Simplified Digital Twin for a Customer Support System** effectively demonstrates the integration of LLMs for automating ticket categorization and response generation.

The system includes:

- **Automated Categorization and Responses.**
- **Retrieval-Augmented Generation (RAG)** for enhanced responses.
- **User Feedback Loop** for continuous improvement.
- **Streamlit Frontend** and **Flask Backend** for seamless user interaction.

This project showcases how AI can streamline customer support workflows, reducing the need for human intervention while maintaining efficiency.

Useful Links

- Groq API Documentation - <https://console.groq.com/docs/api-reference#chat>
- Streamlit Documentation - <https://docs.streamlit.io/>
- Flask Documentation - <https://flask.palletsprojects.com/en/stable/>

GitHub Repository -

<https://github.com/Nehagopinath15neu/GenAIFinal-Project>