

AI-Powered Customer Care

AI-Powered Customer Care Chatbot - Overview

The **AI-Powered Customer Care Chatbot** is designed to provide **instant responses** to customer inquiries regarding **orders, shipping, returns, and support**. It uses **predefined responses** for common queries and can **fall back to an AI model (DeepSeek LLM via Ollama)** when no predefined answer is found. This ensures that the chatbot delivers **fast and accurate responses** while handling **unrecognized queries intelligently**.

Project Files & Their Roles

File	Purpose
<code>main.py</code>	Gradio UI - Handles user input and displays responses.
<code>customer_chatbot.py</code>	Query Processing - Matches predefined responses or asks for rephrasing.
<code>predefined_responses.py</code>	Static Answers - Contains hardcoded responses for common queries.
<code>utils.py</code>	Text Normalization & Similarity Matching - Helps in query comparison.
<code>llm_fallback.py</code>	Fallback AI Processing - Calls DeepSeek AI if no predefined response matches.
<code>config.py</code>	Settings - Defines similarity thresholds and default AI model.

Explanation of Each File & Key Functions

1 `main.py` - The Gradio UI

This file sets up the **Gradio-based chatbot interface**, allowing users to:

- **Enter customer service questions** (e.g., “Where is my order?”).
- **Receive instant responses** from predefined replies.
- **Select an AI model** (if fallback AI is enabled).

◆ Key Functions:

- `chat_interface(query, model)` → Calls `process_query()` from `customer_chatbot.py`.
- `get_available_models()` → Fetches available Ollama AI models.

Code Snippet (UI Integration with Gradio)

```
submit_button.click(
    fn=chat_interface,
    inputs=[query_input, model_selector],
    outputs=response_output
)
```

2 `customer_chatbot.py` - Handling Queries

This file processes customer questions by:

- **Checking for predefined responses** using a **similarity threshold**.
- **Returning an appropriate response** if a match is found.
- **Asking the user to rephrase** if no match is detected.
- ♦ **Key Function:**
 - `process_query(query, model)` → Matches a predefined response or requests rephrasing.

📌 Code Snippet (Handling Predefined Responses)

```
if predefined_response and score >= FALLBACK_THRESHOLD:
    return predefined_response
else:
    return "I'm sorry, but I can only assist with questions about orders, shipping, or returns."
```

3 `predefined_responses.py` - Static Answers

This file contains **pre-set responses** to common customer queries like **returns, tracking, and support contacts**.

- ♦ **Example Queries & Responses:**
 - “Where is my order?” → `"You can track your order using website.com/tracking"`
 - “What is your return policy?” → `"Our return policy allows returns within 30 days."`
 - “How can I contact support?” → `"You can reach support at support@example.com"`

📌 Code Snippet (Predefined Responses)

```
PREDEFINED_RESPONSES = {
    "where is my order": "You can track your order using website.com/tracki
ng",
    "what is your return policy": "Our return policy allows returns within 30 d
ays..."
}
```

4 `utils.py` - Query Normalization & Matching

This file helps **normalize text** and **find the best predefined response**.

◆ Key Functions:

- `normalize_text(text)` → Converts text to lowercase and removes punctuation.
- `calculate_similarity(query, predefined_query)` → Compares the user's query to predefined ones.
- `get_best_predefined_response(query)` → Finds the closest predefined answer.

📌 Code Snippet (Similarity Matching)

```
def calculate_similarity(query, predefined_query):
    query_words = set(normalize_text(query).split())
    predefined_words = set(normalize_text(predefined_query).split())
    common_words = query_words.intersection(predefined_words)
    return len(common_words) / len(predefined_words)
```

5 `llm_fallback.py` - AI Response Handling

This file **calls DeepSeek AI (Ollama) as a backup** when no predefined response is found.

◆ Key Function:

- `run_llm_query(query, model)` → Uses `subprocess.run()` to query the AI.

📌 Code Snippet (Fallback AI Processing)

```
command = ["ollama", "run", model, query]
result = subprocess.run(command, capture_output=True, text=True, check
```

```
=True, encoding="utf-8")  
return result.stdout.strip()
```

6 `config.py` - Settings & Thresholds

This file defines **system-wide settings**, including:

- **Minimum similarity score** required for predefined responses.
- **Default AI model for fallback queries.**

Code Snippet (Threshold & Model Settings)

```
FALLBACK_THRESHOLD = 0.5  
DEFAULT_LLM_MODEL = "deepseek-r1:1.5b"
```

Summary of How Everything Works Together

- 1 User enters a customer service question.
- 2 System checks predefined responses for a close match.
- 3 If a match is found, it provides an **instant answer**.
- 4 If no match is found, it asks the user to **rephrase the query**.
- 5 (Optional) If **AI fallback is enabled**, it queries DeepSeek AI for a response.

How to Run the Project

- 1 Run the chatbot:

```
python main.py
```