Project Report for Customer Support Chatbot

Introduction

In today's digital world, customer support demands have increased as companies manage a high volume of customer inquiries on social media. Traditional customer service solutions can struggle with managing diverse queries efficiently and providing personalized responses. This project aims to build a customer support chatbot that can generate concise and contextually accurate responses using the Llama 3.2 instruct model.

The dataset used for this project, Customer Support on Twitter, offers modern English conversations between consumers and customer support agents, providing a focused, natural, and succinct basis for model training and evaluation.

Data and Process Overview

The dataset is structured as a CSV file, where each row represents a tweet with the following attributes:

- **tweet_id**: A unique ID for the tweet.
- **author_id**: An anonymized user ID.
- **inbound**: Indicates if the tweet is from the customer.
- **created_at**: The timestamp of the tweet.
- **text**: The content of the tweet.
- **response_tweet_id**: IDs of tweets that responded to this tweet.
- **in_response_to_tweet_id**: ID of the tweet this tweet is responding to.

Work Completed

Data Cleaning and Preparation

- Handled missing values by either filling them with placeholder values or removing irrelevant rows.

- Segmented the dataset into conversation threads using `inbound`, `response_tweet_id`, and `in_response_to_tweet_id` fields.
- Linked each customer inquiry with its corresponding responses to capture the full context of each conversation.
- Structured the input conversations in JSON format for consistency and compatibility with downstream processing. The structure is as follows:

JSON Input Structure

```
[
  {
    "Company_name": "company_name_here"
  },
  {
    "conversation": [
      {
        "role": "Customer",
        "message": "text_here"
      },
      {
        "role": "Company",
        "message": "text_here"
      }
    ]
  }
]
```

Entity and Intent Classification

- **Entities Extracted**:
  1. **Product**: Name or type of product mentioned in the conversation.
  2. **Service**: Name or type of service discussed.
  3. **Issue Type**: Nature of the problem or complaint.
  4. **Time Frame**: Any time period or duration mentioned.
  5. **Other**: Any other relevant information.
- **Intents Classified**:
  1. **Complaint**: Expressions of dissatisfaction or frustration with a product, service, or experience.
  2. **Request for Assistance**: Requests for help, support, or clarification.

3. **Inquiry**: Questions about products, services, or policies.
4. **Feedback**: Opinions, suggestions, or gratitude (positive, negative, or neutral).
5. **Escalation Request**: Requests to speak to a manager or higher authority for faster resolution.
6. **Account or Billing Issue**: Problems or queries about account management, billing, refunds, or payment.
7. **Technical Support**: Requests for troubleshooting or technical assistance.
8. **Delivery or Fulfillment Issue**: Issues with order delivery, delays, or couriers.
9. **Promotion or Offer Inquiry**: Questions or complaints about promotions or offers.
10. **Uncategorized**: Messages that don't fit into the above categories.

- Used Llama 3.2 3B instruct model for both entity extraction and intent classification.
- Applied structured prompts to accurately identify entities and classify intents, storing the results in a structured JSON format for further use.

Example Output for Entity and Intent Extraction

Example Output

```
{
  "Company_name": "ChipotleTweets",
  "entities": [
    {
      "role": "Customer",
      "entities": {
        "product": "food",
        "service": "Chipotle",
        "issue_type": "Contamination",
        "time_frame": null,
        "other": null
      }
    },
    {
      "role": "Company",
      "entities": {
        "product": null,
        "service": "food service",
        "issue_type": null,
        "time_frame": null,
        "other": null
      }
    }
  ],
  "classified_intents": ["Complaint", "Request for Assistance"]
}
```

Ontology Roadmap

This section details the steps for building and integrating an ontology-based chatbot system. The process includes extracting relationships using Llama, creating and merging ontologies using RDFlib, and integrating the ontology into a chatbot system.

1. Relationship Extraction using Llama

- **Objective**: Extract relationships between entities and intents to establish meaningful connections for the ontology.
- **Steps**:
  1. **Define Relationship Types**: Identify common relationships:

    - `hasIssue`: Links a product/service to an issue.
    - `appliesTo`: Links an issue to an intent.
    - `resolvesWith`: Links an issue to a resolution.
    - `providedBy`: Links a product/service to a provider.
  2. **Use Llama for Relationship Extraction**: Pass entities and intents into Llama using structured prompts to infer relationships.
  3. **Automate the Extraction**: Loop through the dataset to extract relationships for all conversations.

- **Deliverables**: - Relationships extracted as RDF triples (subject, predicate, object).

2. Ontology Creation using RDFlib

- **Objective**: Represent extracted entities, intents, and relationships in a structured ontology.
- **Steps**:
  1. **Define Ontology Classes and Properties**:
     - Classes: `Product`, `IssueType`, `Intent`, `Resolution`.
     - Properties:
       * `hasIssue`: Links products/services to issues.
       * `appliesTo`: Links issues to intents.
       * `resolvesWith`: Links issues to resolutions.
  2. **Create RDF Triples**: Represent the extracted data as RDF triples.
  3. **Serialize the Ontology**: Save the ontology as an RDF/XML file for querying and integration.
- **Deliverables**: - An RDF/OWL file representing the ontology structure.

3. Ontology Merging

- **Objective**: Combine relationships extracted from different batches or data sources into a unified ontology.
- **Steps**:
  1. **Load Existing Ontology**: Parse the current ontology file using RDFlib.
  2. **Check for Duplicates**: Ensure no duplicate triples are added by comparing new triples with existing ones.
  3. **Resolve Conflicts**: Handle conflicting triples by:
     - Retaining the most frequently occurring relationship.
     - Resolving based on confidence or context from the dataset.

4. **Merge New Relationships**: Add validated triples to the ontology graph.
5. **Save the Updated Ontology**: Serialize the merged ontology back to an RDF/XML file.

- **Deliverables**: - A unified RDF/OWL file with all relationships integrated.

4. Querying the Ontology

- **Objective**: Retrieve relevant knowledge from the ontology to enrich chatbot responses.
- **Steps**:

  1. Define SPARQL queries for retrieving context.
  2. Query the ontology dynamically based on user input.
  3. Integrate the retrieved data into chatbot prompts.

- **Example Use Case**: - **SPARQL Query**: Find all issues linked to a product like `Food`.
- **Deliverables**: - Query results to enrich chatbot conversations.

5. Embedding and Similarity Retrieval

- **Objective**: Enhance chatbot capabilities by retrieving similar questions and answers from a vector database.
- **Steps**:

  1. Embed conversations using a transformer model like MiniLM-L12-v2.
  2. Store embeddings with metadata in a vector database such as Pinecone or FAISS.
  3. Query the database with new inputs to find the most similar stored conversations.

- **Deliverables**: - Retrieved similar conversations to provide context-aware chatbot responses.

6. Integrating the Ontology with the Chatbot

- **Objective**: Use the ontology and vector database to improve chatbot responses.
- **Steps**:

  1. Query the ontology to retrieve relationships, intents, and resolutions.
  2. Use embeddings to find similar historical conversations.
  3. Combine ontology data and retrieved conversations to enrich chatbot prompts.

- **Deliverables**: - A chatbot that reasons with the ontology and retrieves context-aware responses.

7. Evaluate and Refine the Ontology

- **Objective**: Test and improve the ontology for accuracy and relevance.
- **Steps**:
  1. Measure chatbot accuracy and response quality before and after ontology integration.
  2. Identify and resolve inconsistencies in the ontology.
  3. Expand the ontology with new entities, intents, and relationships based on feedback.
- **Deliverables**: - A refined ontology and improved chatbot system.

Conclusion

This roadmap provides a comprehensive plan for building an ontology-based chatbot system. By leveraging Llama for relationship extraction, RDFlib for ontology creation and merging, and embedding-based retrieval for similarity matching, the chatbot can deliver accurate, context-aware responses.

Agentic AI System Roadmap with Guardrail Agent

This roadmap outlines the development of an Agentic AI system for a chatbot that integrates autonomous agents to handle specific tasks, such as relationship extraction, ontology management, semantic pairing, and safe response generation. A dedicated Guardrail Agent ensures system reliability and ethical considerations.

1. Define the Agentic System

- **Objective**: Design a system of autonomous agents to manage specific tasks in the chatbot pipeline.
- **Agents and Responsibilities**:
  1. **Data Preprocessing Agent**: Cleans and structures input data.
  2. **Relationship Extraction Agent**: Extracts relationships between entities and intents.
  3. **Ontology Management Agent**: Creates and updates the ontology.
  4. **Query and Retrieval Agent**: Retrieves relevant context from the ontology and vector database.
  5. **Response Generation Agent**: Generates chatbot responses based on context.
  6. **Guardrail Agent**: Ensures responses meet safety, ethical, and quality standards.
- **Deliverables**: - A modular system architecture with clearly defined roles for each agent.

2. Guardrail Agent

- **Objective**: Monitor and ensure the safety, ethical standards, and accuracy of chatbot outputs.
- **Responsibilities**:
  1. Enforce content policies:
     - Prevent harmful, biased, or inappropriate responses.
  2. Validate factual accuracy:
     - Cross-check outputs against trusted knowledge sources.
  3. Manage response fallback:
     - Redirect to safe fallback messages if the system cannot confidently generate a response.
  4. Monitor system performance:
     - Flag patterns of failure for further investigation and refinement.
- **Steps**:
  1. Intercept responses from the Response Generation Agent.
  2. Check outputs against predefined safety and ethical guidelines.
  3. Use external APIs or databases for real-time fact-checking.
  4. Return approved responses or trigger fallback mechanisms.
- **Deliverables**: - Safe, ethical, and accurate chatbot responses. - Logs of flagged responses for system improvement.

3. Data Preprocessing Agent

- **Objective**: Prepare raw input data for downstream tasks.
- **Steps**:
  1. Remove irrelevant or noisy data.
  2. Structure conversations into JSON format.
  3. Extract entities and intents for further processing.
- **Deliverables**: - Cleaned and structured input data.

4. Relationship Extraction Agent

- **Objective**: Identify relationships between entities and intents for ontology creation.
- **Steps**:
  1. Pass structured data to Llama for relationship extraction.
  2. Validate relationships based on domain-specific rules.
  3. Forward extracted relationships to the Ontology Management Agent.
- **Deliverables**: - Relationships in RDF triple format.

5. Ontology Management Agent

- **Objective**: Maintain and update the ontology with new relationships.
- **Steps**:
  1. Add new relationships to the ontology.
  2. Check for duplicate or conflicting triples.
  3. Resolve conflicts using predefined rules.
  4. Save updated ontology and notify other agents.
- **Deliverables**: - An up-to-date RDF/OWL ontology.

6. Query and Retrieval Agent

- **Objective**: Retrieve relevant knowledge from the ontology and vector database.
- **Steps**:
  1. Query the ontology for relationships and resolutions.
  2. Retrieve similar queries and answers from the vector database.
  3. Combine results into a unified context for response generation.
- **Deliverables**: - Unified context combining ontology and vector database results.

7. Response Generation Agent

- **Objective**: Generate accurate and context-aware chatbot responses.
- **Steps**:
  1. Format retrieved context into a system prompt.
  2. Use a language model (e.g., Llama or GPT) to generate a response.
  3. Send the response to the Guardrail Agent for validation.
- **Deliverables**: - Generated chatbot response ready for validation.

8. Semantic Pairing Integration

- **Objective**: Enhance response quality by leveraging semantically paired data.
- **Steps**:
  1. Link semantically similar queries and answers using a graph-based approach.
  2. Integrate paired results into the context provided to the Response Generation Agent.
- **Deliverables**: - Enhanced context for response generation.

9. Evaluate and Refine the System

- **Objective**: Ensure continuous improvement of the chatbot system.
- **Steps**:
  1. Measure response quality, safety, and accuracy.
  2. Analyze flagged issues from the Guardrail Agent.
  3. Refine agent logic, ontology structure, and semantic pairing mechanisms.
- **Deliverables**: - Improved chatbot performance and reliability.

Conclusion

This roadmap integrates a Guardrail Agent into an Agentic AI system to ensure safety, ethical compliance, and high-quality responses. The modular approach enables scalability, while semantic pairing enhances context for generating accurate and reliable chatbot interactions.

Semantic Pairing

What is Semantic Pairing?

Semantic pairing links semantically similar questions in a graph structure to provide richer context for chatbot responses. When a similar question is retrieved, its associated answer is fetched from the dataset.

Steps for Semantic Pairing

1. **Add Questions to the Graph**:
   - Represent each question as a node in the graph.
   - Compute embeddings for each question using a transformer model (e.g., MiniLM-L12-v2).

- Connect nodes with edges based on cosine similarity scores above a threshold (e.g., 0.8).

2. **Retrieve Question-Answer Pairs**:

   - When a user inputs a query, compute its embedding and find the most similar questions in the graph.
   - Retrieve the answers associated with these questions from the dataset.
   - Combine the retrieved question-answer pairs into a unified context.

Benefits of Question-Only Graphs

Advantages of Question-Only Graphs

- **Simplified Graph Structure**: Reduces complexity by focusing only on semantic relationships between questions.
- **Dynamic Pairing**: Retrieves answers dynamically, ensuring responses remain accurate and contextually relevant.
- **Scalability**: Easily accommodates new questions without restructuring the graph.

Example Workflow

Example

- **User Query**: "What happens if my package is delayed?"
- **Most Similar Questions Retrieved**:
  - "Why is my delivery late?"
  - "How do I handle late packages?"
- **Retrieved Question-Answer Pairs**:
  - Question: "Why is my delivery late?"
    Answer: "Delays may occur due to weather or courier issues."
  - Question: "How do I handle late packages?"
    Answer: "Contact customer support or check the tracking details."
- **Context for Response Generation**:
  - Combine retrieved answers to generate a cohesive response:

    "Delays may occur due to weather or courier issues. We recommend contacting customer support or checking the tracking details for more information."

References

Ontology Development

- **Ontology Development 101: A Guide to Creating Your First Ontology**
  Author: Natalya F. Noy and Deborah L. McGuinness
  Summary: A foundational guide offering step-by-step methodologies for ontology creation.
  Source: Ontology Development 101
- **Best Practices of Ontology Development**
  Organization: CUBRC, Inc.
  Summary: Insights into developing consistent and integrative ontologies for semantic web strategies.
  Source: NIST Best Practices

Agentic AI Frameworks

- **Microsoft's Agentic AI Frameworks: AutoGen and Semantic Kernel**
  Organization: Microsoft

Summary: Open-source frameworks for building AI agent systems with a focus on orchestration and autonomy.
Source: Microsoft's Agentic AI Frameworks

- **Agentic Frameworks for Generative AI Applications**
  Platform: Analytics Vidhya
  Summary: Overview of frameworks for developing autonomous AI agents with decision-making capabilities.
  Source: Analytics Vidhya

- **The Dawn of the AI Agent Economy: Finding Your Ontological Core**
  Platform: WordLift
  Summary: Insights into leveraging ontological structures to enhance AI systems in service-based applications.
  Source: WordLift Blog