Chatbot report

The chatbot is powered by a fine-tuned language model based on GPT-2, leveraging large-scale conversational data to generate human-like responses. The model has been trained on a dataset that includes (prompts, intents of the prompts, categoris, flags). The goal is to build a chatbot that can understand customer queries and provide accurate, context-aware responses in real-time.

**Data Preprocessing**

Effective data preprocessing is crucial to ensure that the dataset is clean and ready for training the chatbot model. In this project, the dataset used for training the GPT-2-based chatbot is referred to as prompt\_data, which consists of four key features: **prompts**, **intents of the prompts**, **categories**, and **flags**. The following steps were taken to prepare the data for fine-tuning the model:

* **Feature Selection**

The prompt dataset includes relevant features essential for training a chatbot to understand user input:

* **Prompts**: These are the core input data, consisting of user queries or statements. The model was trained to generate appropriate responses based on these inputs.
* **Intents**: These represent the underlying purpose or goal behind each prompt, helping the model understand what action or response is expected.
* **Categories**: Each prompt was classified into a specific category, such as order-related, product-related, or general inquiry. This aided the model in narrowing down the context when generating responses.
* **Flags**: These were additional indicators used to capture specific conditions or alert statuses associated with a query, enhancing the chatbot's ability to address special cases or exceptions.
* **Replacing the special tokens:**
  + The prompts column had queries and questions with special tokens representing some things like {{Order Number}} so we created a mapping dictionary to map those special tokens at their corresponding randomly sampled values from the olist dataset.
* Adding some special queries like variants of “"what is the store website?” to get the model to know the URL of the store.
* **Encoding Categorical Data**
  + To prepare the categorical features (intents, categories, and flags) for model training:
  + Label Encoding was applied to convert each category, intent, and flag into numerical labels, as these features represent distinct classes rather than continuous values. This encoding allowed the model to learn from categorical variables without introducing bias.
  + Tokenizing the ‘instruction’ and the ‘response’ columns to get their encodings and that will be fed to the model.
* Data Splitting
  + To ensure a well-rounded training process, the dataset was split into three subsets:
  + Training Set: The majority of the data was used for training the model to understand user queries and responses.
  + Evaluation Set: A separate portion of the data was reserved for validation testing, allowing the chatbot to be evaluated on unseen prompts and responses to ensure generalization to new queries.

**Model Training**

The chatbot is powered by a fine-tuned GPT-based language model, utilizing the pre-trained GPT-2 medium model from openai-community as the starting point. Fine-tuning this model on the prompt dataset allowed it to better understand and respond to customer queries in the e-commerce context. Below is an outline of the fine-tuning process:

* **Hyperparameters Used**

Several key hyperparameters were selected to optimize the training process:

* + Learning Rate: A small learning rate was chosen to ensure that the model adapted slowly and did not forget the pre-trained knowledge. This was managed by the Trainer class, which automatically adjusted the learning rate during training.
  + Batch Size: Both the training and evaluation batch sizes were set to 8, ensuring a balance between computation efficiency and memory usage, especially considering the size of the DialoGPT model.
  + Epochs: The model was trained for 3 epochs, which was sufficient to fine-tune the model on the specific dataset without overfitting.
  + Warmup Steps: A total of 500 warmup steps were introduced to help the model stabilize during the initial stages of training.
  + Weight Decay: A weight decay of 0.01 was applied to prevent the model from becoming too complex, helping to mitigate overfitting.
* **Training Techniques**
  + The fine-tuning process leveraged several advanced training techniques:
  + Transfer Learning: The model was initialized with weights from the pre-trained GPT-2 model, allowing it to build upon existing knowledge. This approach significantly reduced training time and improved the chatbot’s ability to generate coherent responses.
  + Optimization Algorithms: The AdamW optimizer was used to update model weights during training, balancing the need for speed and accuracy while minimizing loss.
  + Evaluation Strategy: The training process included intermediate evaluations every 500 steps to track model performance and prevent overfitting. The best-performing model was saved based on these evaluations.

**Challenges Faced**

During the training process, several challenges were encountered:

* Model Size: Given that GPT-2 is still a relatively large model, memory management was a key challenge, especially when fine-tuning with a batch size of 8. To overcome this, the model and dataset were carefully optimized to fit within the available resources.
* Overfitting: Since the dataset was relatively specialized, there was a risk of overfitting. This was addressed by using techniques such as weight decay, warmup steps, and early stopping based on evaluation metrics.
* Response Diversity: Ensuring that the chatbot generated diverse and relevant responses was another challenge. The pre-trained model sometimes exhibited repetitive behavior, which was gradually improved through fine-tuning on the e-commerce-specific dataset.

This process ensured that the fine-tuned chatbot model could generate context-aware, relevant responses tailored to customer queries in real-time.

**Integration with API**

The chatbot is integrated into a web application using a FastAPI backend to handle chat interactions and a Streamlit app for the frontend. This architecture allows seamless communication between the user interface and the chatbot model, providing real-time responses to user queries. Below is an outline of the integration process.

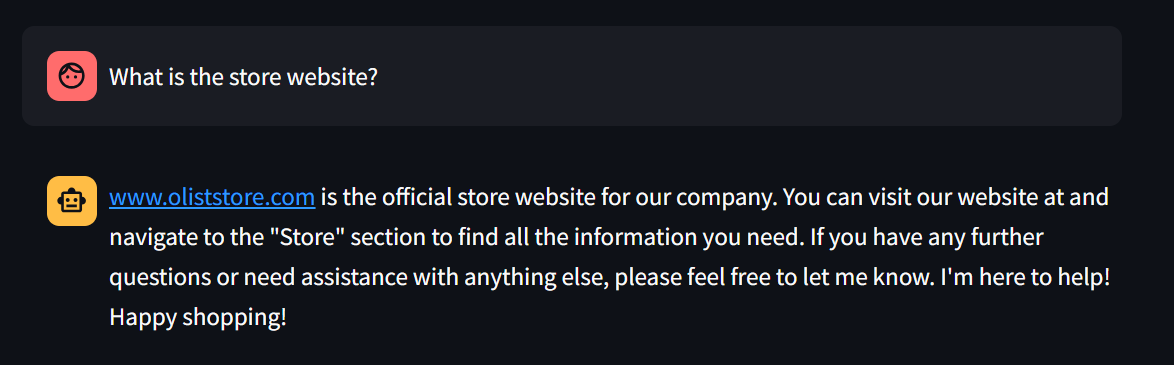
* **Backend Architecture**

The backend is built using FastAPI, a modern, fast web framework for building APIs. The API interacts with the fine-tuned GPT-2 model, processing incoming chat prompts and generating responses.

* + Model Loading: The chatbot model is loaded into memory when the FastAPI app starts, making it available for subsequent API requests. The model is deployed on GPU (if available) for faster inference; otherwise, it runs on the CPU.
  + API Endpoints: A single API endpoint (/generate-response) is created to handle chat interactions. When a user sends a prompt, it is processed through this endpoint, and the chatbot generates a response based on the fine-tuned model.
  + Data Cleaning: The API cleans prompts by removing placeholder tokens such as {{Order Number}}, {{Person Name}}, and {{Delivery City}} before passing them to the model. This ensures that the chatbot generates accurate, context-appropriate responses.
  + Response Generation: The chatbot uses various decoding strategies such as beam search, top-p sampling, and repetition penalties to produce coherent, human-like responses. The API also post-processes the output by removing placeholders and cleaning up punctuation to enhance readability.

A screenshot of a computer

Description automatically generated

 A screenshot of a computer

Description automatically generated