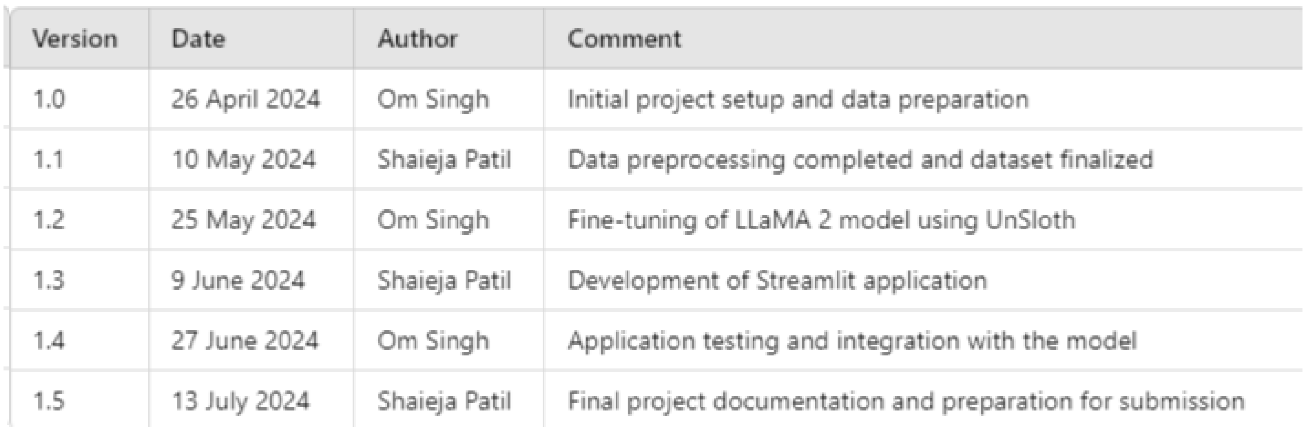
Low Level Design

Product Chat Assistant

Written By	Om Singh, Shaileja Patil
Document Version	1.5
Last Revised Date	13 – June -2024

Document Control

Change Record:



Version	Date	Author	Comment
1.0	26 April 2024	Om Singh	Initial project setup and data preparation
1.1	10 May 2024	Shaieja Patil	Data preprocessing completed and dataset finalized
1.2	25 May 2024	Om Singh	Fine-tuning of LLaMA 2 model using UnSloth
1.3	9 June 2024	Shaieja Patil	Development of Streamlit application
1.4	27 June 2024	Om Singh	Application testing and integration with the model
1.5	13 July 2024	Shaieja Patil	Final project documentation and preparation for submission

Contents

1. Introduction

The Product Chat Assistant is an AI-driven application designed to enhance the online shopping experience. It interacts with users naturally, providing detailed product information, answering queries, and negotiating prices to improve customer satisfaction and drive sales.

Key Features:

1. Product Information Retrieval:
   - Retrieves detailed product specifications, features, pricing, and availability for informed decision-making.

2. Interactive Conversations:
   - Engages users in natural, context-aware conversations for seamless interactions.

3. Price Negotiation:
   - Allows users to negotiate prices, simulating real-world shopping experiences.

4. Personalized Recommendations:
   - Analyzes user preferences and browsing history to offer tailored product suggestions.

5. Multi-Channel Support:
   - Integrates with websites, mobile apps, and social media for a consistent user experience across platforms.

2. Scope

The Product Chat Assistant aims to develop an AI assistant for personalized product support.

Functional Requirements

- Chat Interface: Interface for product-related questions.
- Model Integration: Fine-tuned LLaMA 2 for responses.
- Bargaining Feature: Price negotiation.
- Product Information: Descriptions, prices, and reviews.

Data Requirements

- Dataset: 370 rows, 12 columns from OpenAI and Gemini.
- Feature Enhancement: Append conversation scripts and prompts.
- Storage: Push dataset to Hugging Face.

Technical Requirements

- Fine-Tuning: Use UnSloth's FastLanguageModel.
- Training: Implement SFTTrainer.
- Deployment: Use Streamlit for frontend.

Deliverables

- Functional Assistant: User-friendly interface.
- Trained Model: Fine-tuned LLaMA 2.
- Dataset: Available on Hugging Face.

Timeline

- Data Collection: 1 week
- Model Training: 3 weeks
- Frontend Development: 1 week

Resources

- Project Lead: Mandalor09
- Technical: UnSloth, Hugging Face, Streamlit
- Data: OpenAI, Gemini

3. Architecture

Data Generation: Creates training data for the AI model using OpenAI and Gemini.

➢ **Data Preprocessing:** Cleans and prepares the data for training.

➢ **Model Training:** Fine-tunes a pre-trained LLAMA model (UnSloth) for conversation.

➢ **Conversation Design:** Creates conversation scripts and prompts for interaction.

➢ **Data & Model Deployment:** Uploads data to Hugging Face Hub and deploys the model for access.

➢ **User Interface:** Builds a Streamlit interface for user interaction with the model.

This highlights the key steps: data preparation, model training, deployment, and user interface creation.

3.1) Data Preparation Using OpenAI and Gemini

Data Acquisition:

- ✓ OpenAI and Gemini: Data is gathered from two main sources: OpenAI and Gemini.
- ✓ OpenAI API: Leveraging OpenAI's text generation capabilities, product descriptions, reviews, and ratings are extracted through API calls.
- ✓ Gemini API: Similarly, Gemini's API provides detailed product information, including company names, categories, specifications like size and color, and pricing details (original, discounted, and additional discounts.

Data Integration:

Combined Dataset: Data from both sources is merged to create a comprehensive dataset.

Weighted Contribution: OpenAI provides a smaller yet richer set (70 rows) focused on descriptive content, while Gemini contributes a larger set of product details (300 rows). The total dataset reaches 370 rows.

Column Alignment: To ensure data consistency, specific columns are carefully matched during the merging process:

i. Product_Company_Name
ii. Product_Category
iii. Product_Name
iv. Product_Description (from OpenAI)
v. Product_Color
vi. Product_Size
vii. Product_Original_Price
viii. Product_Discounted_Price
ix. Product_Additional_Discount
x. Average_Rating (from OpenAI)
xi. Reviews (from OpenAI)

4.

3.2) Data Preprocessing:

This stage ensures your data is clean, consistent, and ready for effective modeling. Here's what's involved:

Data Cleaning:

- Missing Values: The process identifies missing values in your data (e.g., empty cells). Techniques like filling numerical data with the average and categorical data with the most frequent value (mode) address these missing entries.
- Duplicate Removal: Duplicate rows are identified and eliminated to avoid skewed results and maintain data integrity.

Data Normalization:

- Standardization: Techniques like Min-Max scaling or Z-score normalization are applied to transform data ranges. This ensures all features have a similar scale, preventing specific features from dominating the model's learning process and contributing to better convergence and accuracy.

- Data Transformation:
  Tailored Transformations: Data might undergo specific transformations depending on its type. For example, applying a log or square root transformation to numerical data can improve model performance. Categorical data, like color options, might be converted using one-hot encoding, where each category has its own dedicated feature.

- Feature Engineering:
  Contextual Features: To help the model understand the context of product inquiries, additional features are created. This might involve adding a "Conversation Script" feature capturing the dialogue or narrative related to the product, and an "Input Prompt" feature detailing the initial user query or command.

Output: A Refined Dataset Ready for Action

Following these steps, you'll have a preprocessed dataset with a well-defined structure, containing 370 rows and 14 columns. This includes:

Product Information: Company name, category, name, description, color, size, pricing details (original, discounted, additional discounts).

Review & Rating: Average rating and reviews (from OpenAI).

Conversational Context: Conversation Script and Input Prompt (new features).

This comprehensive and refined dataset is now suitable for various tasks like machine learning, statistical analysis, or business intelligence applications. The combined data from OpenAI and Gemini provides a broader perspective for analysis and potentially leads to more accurate predictive models in downstream tasks.

3.3)Pushing the Dataset to Hugging Face:

The next step involves uploading your preprocessed dataset to Hugging Face. This online platform allows you to share your data with others and access data shared by the community. Here's a simplified overview of the process:

Import Libraries: Libraries like pandas and the datasets library from Hugging Face are used to load and prepare the dataset.

Push to Hub: The push_to_hub method is used with your Hugging Face API token to upload the dataset. This makes your data readily available for future projects or for anyone interested in exploring product-related conversations with potential price negotiations.

3.4)Fine-Tuning the Model:

While large language models like LLaMA 2 offer impressive capabilities, they are often trained on generic datasets. Fine-tuning allows you to specialize the model for your specific task – in this case, handling product inquiries with potential price negotiation. However, traditional fine-tuning approaches can be slow and resource-intensive for large models.

Introducing Unsloth:

Unsloth is a library designed to address the challenges of fine-tuning large language models. Here's how it simplifies the process:

Efficient Memory Management: Unsloth optimizes memory usage, making it more efficient to work with large models like LLaMA 2.

Simplified API: The library provides a user-friendly API for fine-tuning, reducing the complexity involved in the process.

Flexibility: Unsloth works with various transformer models, offering broader applicability beyond just LLaMA 2.

Fine-Tuning Steps with Unsloth: Putting it into Practice

The text provides a code example outlining the steps for fine-tuning LLaMA 2 with Unsloth. Here's a high-level overview:

Install Unsloth: Specific commands are provided to install Unsloth and its dependencies.

Import Libraries: Necessary libraries for working with Unsloth, the pre-trained model, and the dataset are imported.

Load Pre-trained Model: The code snippet demonstrates loading the pre-trained LLaMA 2 model and its tokenizer from Unsloth.

Prepare Dataset: The code snippet shows how to load your preprocessed dataset using the datasets library.

Fine-Tuning with Unsloth: Here, the code demonstrates two approaches for fine

4) Downloading the Model and Adapter

This section focuses on acquiring the necessary components to run your fine-tuned LLaMA 2 model. Here's what you'll do:

- **Download the Pre-trained Model:** You'll use a tool called snapshot_download from the huggingface_hub library to download a pre-trained LLaMA 2 model from the Hugging Face Hub. This pre-trained model has already been trained on a massive dataset of text and code, giving it a strong foundation for understanding language.

- **Download the Adapter:** In addition to the base model, you'll also download a specific adapter file. An adapter is like a specialization for the model, allowing it to focus on a particular task. In this case, the adapter you download will be trained to handle product inquiries and potentially negotiate prices.

- **Loading the Model and Adapter:** Once you have both the pre-trained model and the adapter downloaded, you'll use libraries like transformers to load them into your application. This makes the model's capabilities available for use.

5) Creating a Streamlit App for Product Chat Assistant

This section guides you through building an application where users can interact with the fine-tuned LLaMA 2 model. Here's the breakdown:

- **Setting Up the Application:** You'll use a framework called Streamlit to create a web application. Streamlit simplifies the process of building user interfaces without requiring extensive web development knowledge.
- **Initializing the Model:** Within the Streamlit application, you'll load the previously downloaded and prepared LLaMA 2 model and its tokenizer. The tokenizer helps the model understand the individual words and phrases used in a conversation.
- **Building the User Interface:** The Streamlit app will have a chat interface where users can type in their questions about products. You'll define functionalities to:
    - Allow users to select a product they're interested in.
    - Capture user input as questions about the product.
    - Display the model's generated responses to the user's questions.

Key Functionalities:

- When a user asks a question, the application will send the question through the loaded tokenizer.
- The tokenizer breaks the question down into a format the model can understand.
- The model then processes the question and generates a response based on its training data and the adapter's specialization.
- The generated response is passed back through the tokenizer to convert it into human-readable language, and finally displayed to the user in the chat interface.
- The adapter helps the model focus on product information and potentially engage in basic price negotiations within its responses.

By following these steps, you'll create a Streamlit application that allows users to interact with the fine-tuned LLaMA 2 model in a conversational manner to explore product details.

6) Deployment Considerations

This section explores the deployment aspects of the Streamlit application utilizing the fine-tuned LLaMA 2 model. Here's why deployment considerations are crucial:

High Computational Requirements: The LLaMA 2 model is a large language model, and running it efficiently demands significant computational resources. This translates to needing a powerful graphics processing unit (GPU) like an A100 or T4 to handle the model's processing needs.

Deployment Options: Streamlit applications can be deployed in various environments. However, due to the high computational needs of LLaMA 2, free deployment options might not be suitable. Cloud-based solutions equipped with powerful GPUs become the preferable choice.

Lightning.ai Integration (Optional): While not strictly necessary, Lightning.ai can be considered for deployment. It offers benefits like:

- **Simplified Deployment:** Streamlines the process of moving your LLaMA 2 model from development to a production setting, making it readily accessible by the Streamlit application.
- **Scalability:** If your application experiences a surge in user traffic, Lightning.ai can help scale the model to accommodate the increased demand.
- **Management and Monitoring:** Provides tools to manage and monitor the deployed model's performance, ensuring smooth operation.

Choosing the Right Approach:

The ideal deployment approach depends on your specific needs and resources. Here are some factors to consider:

- **Expected User Traffic:** If you anticipate a large user base, a cloud-based solution with sufficient processing power is essential.
- **Budget Constraints:** Free deployment options might be suitable for initial testing or small-scale applications. However, for production environments with high traffic, cloud-based solutions with appropriate GPU resources will likely be necessary.

7) Conclusion: A Look Back and a Look Forward

This section summarizes the project's achievements and highlights future considerations.

Recap of Key Achievements:

- **Data Preparation:** A comprehensive dataset encompassing product descriptions, customer reviews, and simulated conversations was created using Gemini and OpenAI. This dataset provided the foundation for effectively fine-tuning the model.
- **Data Preprocessing:** The dataset was meticulously cleaned by handling missing values, removing duplicates, and incorporating features like "Conversation Script" to enhance the model's understanding of product inquiries.
- **Model Fine-Tuning:** The LLaMA 2 model was fine-tuned with Unsloth to specialize in comprehending and responding to user inquiries related to products and potentially negotiate prices.
- **Streamlit Application Development:** A user-friendly Streamlit application was built to facilitate interaction with the model. Users can choose products, ask questions, and engage in basic price negotiations through a chat interface.

Challenges and Future Directions:

- **Deployment Considerations:** As discussed earlier, the high computational demands of the LLaMA 2 model pose a challenge for deployment. Cloud-based solutions are likely necessary for production environments.
- **Model Explainability:** While the model can generate responses, further exploration into explaining its reasoning behind specific responses could be valuable.

Overall Significance:

This project showcases the successful integration of advanced machine learning techniques with practical application development. The resulting interactive "bargaining bot" demonstrates the potential of AI to enhance user experiences in e-commerce environments. While deployment constraints require attention, the provided resources - a demo video and a GitHub repository - ensure the project's functionality and innovation can be effectively shared.

This project not only underlines the power of AI in interactive applications but also offers valuable insights into the complexities of model deployment and resource management. As AI technology continues to evolve, we can expect even more sophisticated and impactful applications in the future.