
Generative AI Professional
AI and Data Science Track

Final Project Report

**Track Instructor:** Mohamed Agoor

**Group Code:** RS-2433 ALX1_AIS2_S1e

---

# Personalized Conversational Commerce System **PCCS**

---


| | |
|---|---|
| Mariam Ahmed Fathy | MariamAhmed2024@alexu.edu.eg |
| Omar Hani | amora.elshanawany@gmail.com |
| Ali Amr | aliamreducation@gmail.com |
| khaled Nabawi | cds.Khalednabawi60078@alexu.edu.eg |
| Maha Ossama | Salemmaha021@gmail.com |

# 1 Introduction

The Personalized Conversational Commerce System (PCCS) aims to transform e-commerce by utilies artificial intelligence (AI) to create a highly engaging and tailored shopping experience. By integrating conversational AI, PCCS facilitates real-time, interactive communication, allowing for personalized product recommendations based on individual preferences. Additionally, the system incorporates dynamic pricing strategies, enabling businesses to adjust prices in response to market trends and customer demand.
This innovative approach enhances user satisfaction and optimizes overall business performance by fostering deeper connections between consumers and brands. This report explores the key components and methodologies of PCCS, highlighting its potential to reshape the future of online shopping.

# 2 Conversational AI Using Chatbot

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, categories, and flags. The goal is to build a chatbot to understand customer queries and provide accurate, context-aware responses in real-time.

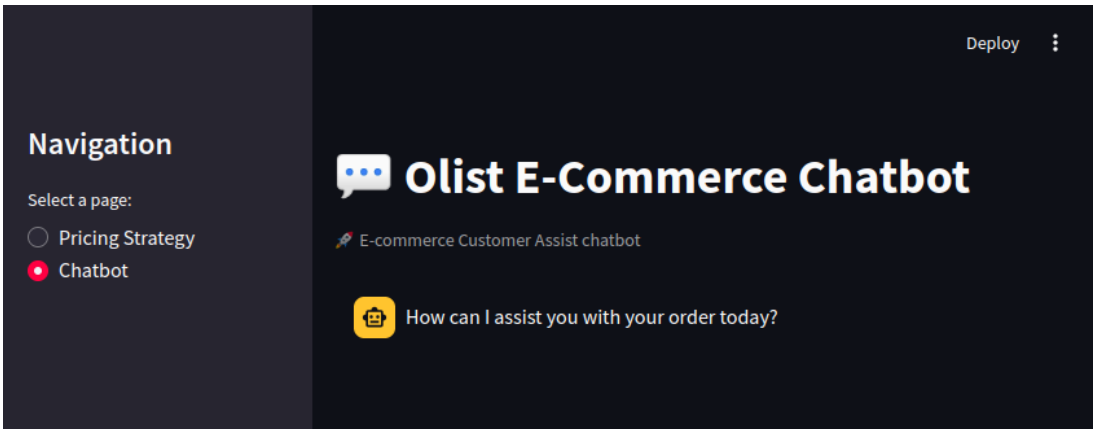


Figure 1: Streamlit User Interface for chatbot

## 2.1 Data Preprocessing

Effective data preprocessing is crucial to ensure 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:

### 2.1.1 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.

#### 2.1.2 Replacing the Special Tokens

The prompts column had queries and questions with special tokens representing items like `{{Order Number}}`. A mapping dictionary was created to map those special tokens to corresponding randomly sampled values from the *olist* dataset.

#### 2.1.3 Adding Special Queries

Variants of queries like "What is the store website?" were added to teach the model store-specific details, such as the URL of the store.

#### 2.1.4 Encoding Categorical Data

To prepare the categorical features (intents, categories, and flags) for model training:

- **Label Encoding:** This was applied to convert each category, intent, and flag into numerical labels, as these features represent distinct classes rather than continuous values.
- **Tokenization:** The 'instruction' and 'response' columns were tokenized to obtain encodings for model training.

#### 2.1.5 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 to train the model to understand user queries and generate responses.
- **Evaluation Set:** A portion of the data was reserved for validation testing to evaluate the chatbot's generalization to unseen prompts and responses.

### 2.2 Model Training

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

### 2.2.1 Hyperparameters Used

Several key hyperparameters were selected to optimize the training process:

- **Learning Rate:** A small learning rate was chosen to ensure the model adapted slowly and retained pre-trained knowledge. The `Trainer` class 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 given the size of the DialoGPT model.
- **Epochs:** The model was trained for 3 epochs, which was sufficient for fine-tuning on the dataset without overfitting.
- **Warmup Steps:** 500 warmup steps were used to stabilize the model during the initial stages of training.
- **Weight Decay:** A weight decay of 0.01 was applied to prevent the model from becoming overly complex and mitigate overfitting.

### 2.2.2 Training Techniques

The fine-tuning process leveraged several advanced techniques:

- **Transfer Learning:** The model was initialized with weights from the pre-trained GPT-2 model, allowing it to build on existing knowledge, reducing training time, and improving response generation.
- **Optimization Algorithms:** The AdamW optimizer was used to update model weights during training, ensuring a balance between speed and accuracy.
- **Evaluation Strategy:** Intermediate evaluations were performed every 500 steps to track performance and prevent overfitting. The best-performing model was saved based on these evaluations.

### 2.2.3 Challenges Faced

Several challenges were encountered during training:

- **Model Size:** GPT-2 is a large model, making memory management critical during fine-tuning with a batch size of 8. The model and dataset were optimized to fit within available resources.
- **Overfitting:** Since the dataset was specialized, overfitting was a risk. Techniques like weight decay, warmup steps, and early stopping based on evaluation metrics were employed to mitigate it.
- **Response Diversity:** The chatbot initially generated repetitive responses, a behavior gradually improved through fine-tuning on the e-commerce dataset.

## 2.3 Integration with API

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

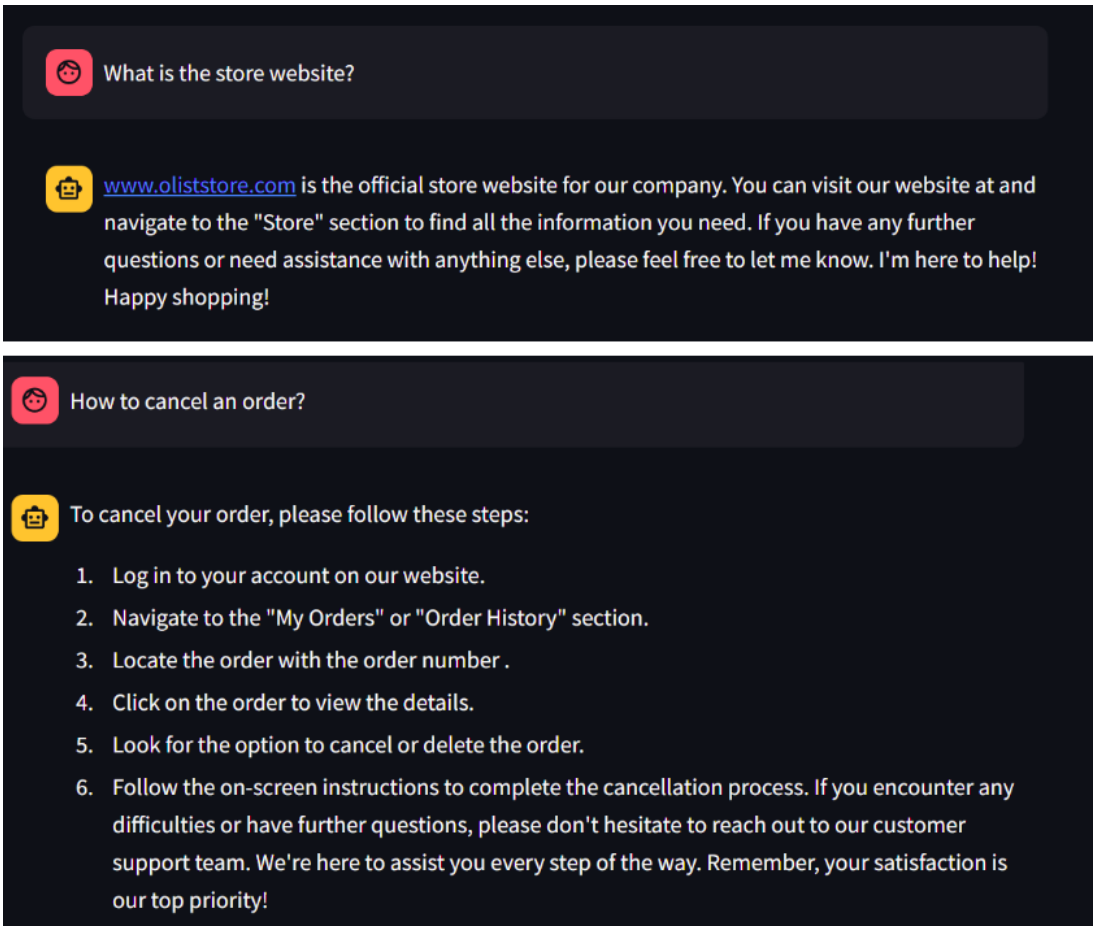


Figure 2: Chatbot Outbot Using Streamlit

### 2.3.1 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. It is deployed on a GPU (if available) for faster inference; otherwise, it runs on a CPU.

**API Endpoints:** A single API endpoint (`/generate-response`) handles 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 removes placeholder tokens such as `{{Order Number}}`, `{{Person Name}}`, and `{{Delivery City}}` before passing the prompts to the model to ensure accurate and context-appropriate responses.


Figure 3: FastAPI for Chatbot

**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.

# 3 Pricing Strategy Prediction

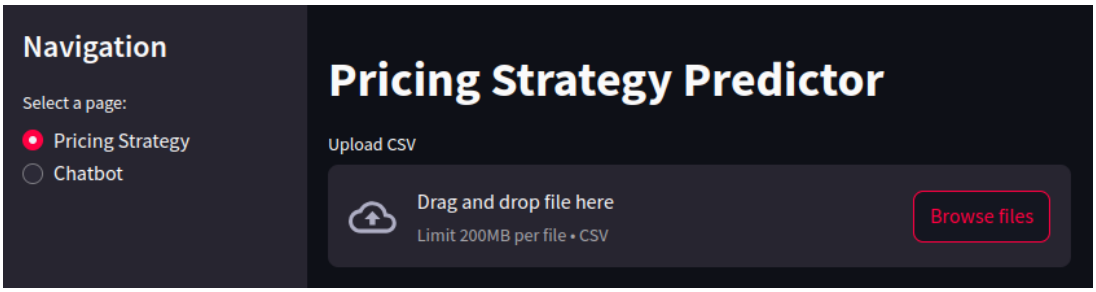


Figure 4: Streamlit User Interface for Pricing Ptrdiction

## 3.1 Dataset Preparation

The "Olist" dataset used in this analysis is a comprehensive e-commerce dataset that integrates various tables to provide insights into customer behavior, product details, and sales performance. The following subsections provide detailed descriptions of the steps involved in processing the dataset.

### 3.1.1 Merging Tables and Data Structure

The dataset was created by merging multiple tables from an SQLite database, resulting in 17,798,047 rows and 46 columns. It contains information about customer orders, product details, seller information, and pricing. This rich dataset provides a broad view of the e-commerce process, from order placement to product delivery.

### 3.1.2 Product Categorization

To simplify the analysis, we created a classification function to map product categories into broader groups such as "Beauty & Health," "Fashion," "Electronics," and others. Each product in the dataset

was assigned to one of these broader categories based on its detailed description. This grouping allowed for more manageable and insightful analysis across different product types.

#### 3.1.3 Outlier Detection and Price Cleaning

We performed outlier detection on the price data using the Interquartile Range (IQR) method. Prices that were higher than $Q3 + 1.5 \times IQR$ were considered outliers and removed from the dataset. This cleaning process ensured that extreme price values did not skew the analysis and provided more realistic insights into pricing strategies.

#### 3.1.4 Price Simulation

In addition to cleaning the prices, we generated a comparator price for each product-seller combination within the same purchase month. A random price variation between -20 and 20 was applied to simulate different pricing strategies. This comparator price allows for the analysis of price competition and variability among sellers.

#### 3.1.5 Sampling the Dataset

Given the size of the dataset, we extracted a random sample of 20,000 rows for detailed analysis. This sample preserves the structure and diversity of the full dataset, allowing us to explore trends and insights while maintaining computational efficiency.

### 3.2 Model: `GenAIPricingTransformer`

The `GenAIPricingTransformer` is a deep learning model designed to predict optimal pricing strategies in a dynamic market environment. It employs a Transformer-based architecture, adapted from the GPT-2 framework, to generate pricing recommendations based on various features. The architecture is composed of several key components:

- **Input Embedding Layer**: The input features (e.g., product characteristics, market demand) are first embedded into a 256-dimensional space using a fully connected layer. This transformation prepares the features for further processing by the Transformer model.
- **Transformer Encoder**: The core of the model consists of a Transformer encoder with multiple self-attention heads and layers (specifically, 8 heads and 6 layers). This allows the model to capture complex relationships between different features and their temporal dependencies. The model uses GELU as the activation function, and dropout is applied to prevent overfitting.
- **Output Heads**: Three distinct output heads provide different types of predictions:
  - **Price Prediction Head**: A sequence of fully connected layers that produces a single price prediction.
  - **Market Condition Prediction Head**: A separate fully connected network predicts market conditions, such as supply-demand ratios.
  - **Elasticity Prediction Head**: Another fully connected head estimates price elasticity, indicating how sensitive customer demand is to changes in price.

## 3.3 Model Training

The training process is conducted using Mean Squared Error (MSE) loss to minimize prediction errors for price and market conditions. The optimizer used is AdamW with a learning rate scheduler, and gradient clipping is applied to stabilize the training process. The training includes a validation loop to ensure generalization, and early stopping is triggered if the validation loss plateaus.

### 3.3.1 Training Summary

The training of the model was conducted over a total of 30 epochs. The following metrics summarize the training performance:

- Number of epochs trained: 30
- Initial training loss: 0.6789
- Final training loss: 0.0540
- Best validation loss: 0.0218
- Initial learning rate: 0.001000
- Final learning rate: 0.000500
- Best model achieved at epoch: 30

### 3.3.2 Training Stability Metrics

The training stability was assessed using the following metrics:

- Training loss standard deviation: 0.1151
- Validation loss standard deviation: 0.0508

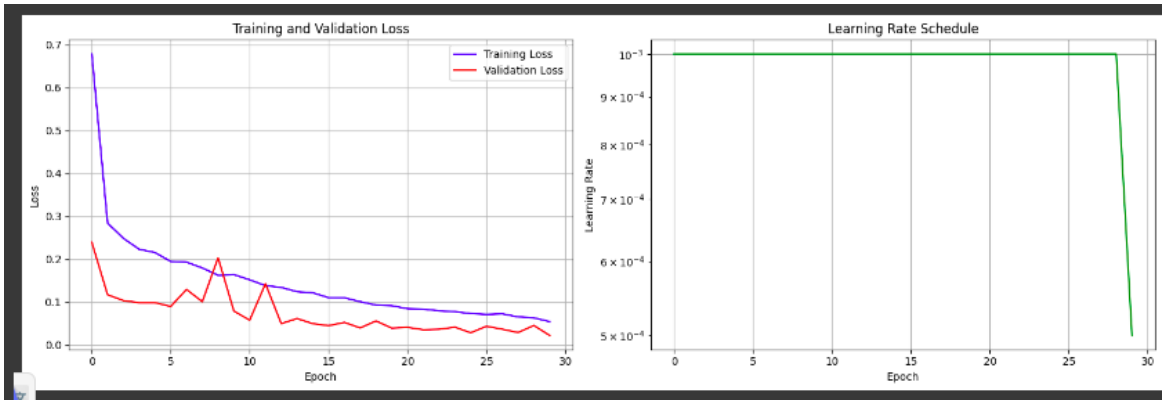


Figure 5: Training Losses and Learning Rate

## 3.4 Preprocessing Pipeline

The preprocessing phase ensures the model receives normalized and well-structured data. Key preprocessing steps include:

- **Price Normalization**: Prices are normalized using a predefined minimum value for numerical stability.
- **Datetime Feature Extraction**: Datetime columns are expanded into hour, day, and month components to provide temporal insights.
- **Feature Engineering**: Several new features are computed, including product volume (derived from product dimensions), market density, and category demand.
- **Log Transformation**: Certain price-related features are log-transformed to mitigate the effects of outliers.
- **Categorical Encoding**: Categorical variables (e.g., product categories, seller and customer states) are label-encoded.

## 3.5 Predictive Output

When the model is used for prediction, the output includes three values:

1. **Price Prediction**: The recommended price based on market conditions and product features.
2. **Market Condition Prediction**: A classification of the current market conditions.
3. **Price Elasticity**: A score reflecting how sensitive the demand is to price changes.

## 3.6 Example Pricing Strategy

Once the model generates predictions, the results can be analyzed for optimal pricing strategies. For example:

- **Market Condition**: 0.85 (indicating strong demand).
- **Price Elasticity**: 0.75 (indicating moderate sensitivity to price changes).
- **Recommended Prices**: \$45.20 (for a given strategy).

This architecture enables the model to adapt pricing strategies in real-time, enhancing business decisions by accounting for both market dynamics and consumer behavior.

## 3.7 Pricing Strategy Streamlit Integration

The core of the pricing strategy lies in the `GenAIPricingStrategy` class, which encapsulates the necessary methods for preprocessing input data, preparing features for model training, and generating predictions. Users can upload a CSV file containing relevant product data through the Streamlit

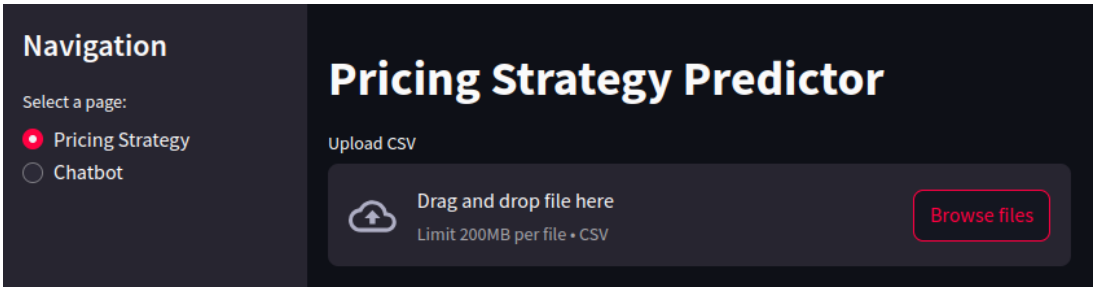


Figure 6: Streamlit User Interface for Pricing

interface, which serves as the frontend for the application. Upon uploading the file, the application displays a preview of the data, allowing users to verify its contents before processing.

To prepare the data for modeling, the application applies a series of preprocessing steps, including feature extraction and transformation. These steps ensure that the data is structured appropriately for input into the pricing model. The model is then loaded, and its architecture is dynamically configured based on the dimensions of the prepared feature set.

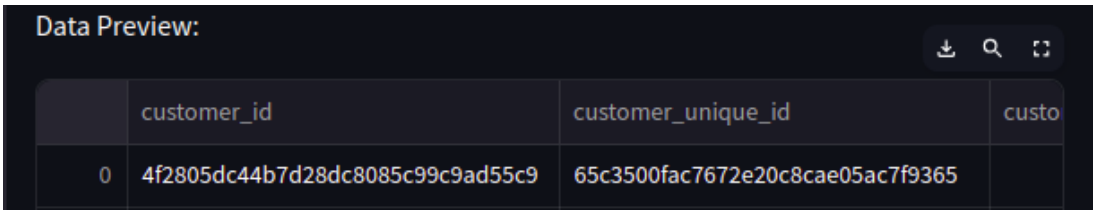


Figure 7: Sample Data for Testing Pricing

Once the model is ready, it processes the input data to generate pricing recommendations. The predictions comprise three key outputs: market condition, price elasticity, and recommended prices for the products. The market condition reflects the current state of the market, while price elasticity indicates how responsive the demand for a product is to changes in price.

After obtaining the predictions, the application displays the pricing strategy recommendations to the user, providing insights that inform decision-making regarding pricing adjustments. By leveraging the predictions from the pricing strategy model, businesses can optimize their pricing approaches, enhance competitiveness, and maximize revenue opportunities in a dynamic market environment.

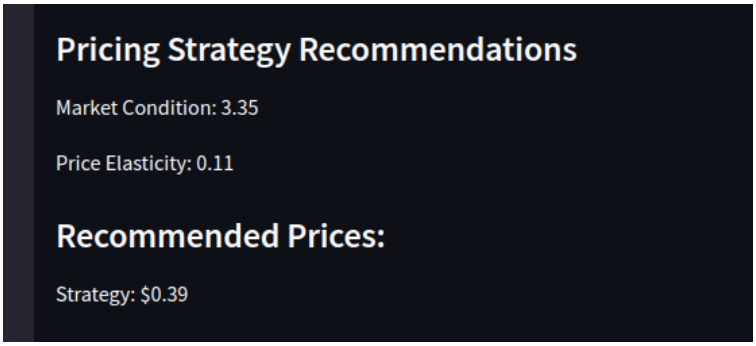


Figure 8: Pricing Model Output

# 4 Conclusion

The PCCS represents a significant advancement in the e-commerce landscape. By harnessing the power of AI and machine learning, this project will create a more personalized, efficient, and engaging shopping experience for users while simultaneously driving business growth and optimizing operational efficiency. We seek support to bring this vision to life and transform the future of online shopping.