

**AI-Powered Customer Engagement: An
Interactive Chatbot Solution for
E-Commerce using Generative AI and AWS**
(A case study on footwear related data)

by
AKRAM NAOUFEL TABET
(ID: W19792464)

Supervised by
YOUR SAIQA SHAIKH

Submitted in partial fulfilment of the requirements of
the School of Computer Science & Engineering
of the University of Westminster
for award of the Master of Science
in
Data Science and Analytics

SEPTEMBER 2024

DECLARATION

I Akram Naoufel Tabet, declare that I am the sole author of this Project; that all references cited have been consulted; that I have conducted all work of which this is a record, and that the finished work lies within the prescribed word limits.

This has not previously been accepted as part of any other degree submission.

Signed: Akram Naoufel Tabet

Date: 10th September 2024

FORM OF CONSENT

I, *Akram Naoufel Tabet*, hereby consent that this Project, submitted in partial fulfilment of the requirements for the award of the MSc degree, if successful, may be made available in paper or electronic format for inter-library loan or photocopying (subject to the law of copyright), and that the title and abstract may be made available to outside organisations.

Signed: *Akram Naoufel Tabet*

Date : 10th September 2024

ABSTRACT

The exponential growth of e-commerce has necessitated innovative solutions for businesses to interact with customers. As online transactions increase, so does the need for efficient customer service. Traditional methods often struggle to keep up, causing delays and dissatisfaction. Artificial Intelligence (AI) and Machine Learning (ML) offer promising solutions, with AI-driven chatbots emerging as powerful tools for providing instant, personalized, and scalable support. Using generative AI and large language models (LLMs), businesses can create intelligent chatbots that understand and process natural language queries, improving both customer satisfaction and operational efficiency. Generative AI and LLMs represent cutting-edge advancements in AI, enhancing content generation and language understanding through models like ChatGPT and DALL-E. These models have evolved with deep learning architectures and vast datasets, leading to multi-modal models that handle text and images. In e-commerce, such advancements enable sophisticated AI chatbots to deliver personalized customer service, dynamic product descriptions, and tailored recommendations, enhancing user experience and engagement. The project aims to develop an AI chatbot tailored for e-commerce, utilizing AWS for scalability and reliability. The chatbot will perform tasks like answering queries, providing recommendations, and tracking order status in real time, utilizing a generative AI model fine-tuned with e-commerce data related to shoes and Retrieval-Augmented Generation (RAG) for accurate responses.

Acknowledgment

Most importantly, all thanks be to Almighty God. Then, I would like to extend my deepest gratitude to my parents in Algeria for affording me the opportunity to pursue my education and reach this milestone. I am immensely grateful to my project supervisor, Madam Saiqa Shaikh, for her unwavering support, guidance, and expertise throughout this project. Her mentorship was invaluable in shaping the direction of my research and enhancing the quality of my work.

I would also like to express my appreciation to the staff and senior management of my organization, whose support, guidance, and resources greatly influenced the outcome of this project. Lastly, my heartfelt thanks go to my siblings, friends, and family for their constant encouragement, understanding, and advice, which motivated me to persevere throughout this journey.

TOTAL WORD COUNT

(Excluding cover page, abstract, acknowledgements, table of contents, references and appendices)

15,574

(Maximum without penalty: 16,500)

Table of Contents

LIST OF TABLES	4
LIST OF FIGURES	4
1. CHAPTER 1: INTRODUCTION.....	5
1.1 BACKGROUND.....	5
1.2 AIM AND RESEARCH QUESTIONS.....	6
1.3 ETHICAL AND CONFIDENTIALITY ISSUES	7
2. CHAPTER 2: THEORETHICAL BACKGROUND AND LITERATURE REVIEW	8
2.1 Chatbots in E-Commerce	8
2.1.1 Early Developments in Chatbot Technology	8
2.1.2 Chatbot Platforms and Their Integration	8
2.1.3 Enhancing Chatbots with Generative AI and LLMs	8
2.2 Generative AI in E-Commerce.....	9
2.2.1 The Role of Generative AI in Customer Service.....	9
2.2.2 eCommerceGAN: A Case Study of GANs in E-Commerce	9
2.3 Retrieval-Augmented Generation (RAG)	10
2.3.1 Applications of RAG in Personalized Recommendations	10
2.4 Large Language Models (LLMs) and Their Impact on E-Commerce	11
2.4.1 Advancements in LLMs.....	11
2.4.2 BERT and Contextual Understanding	11
2.5 Challenges and Future Directions	11
2.5.1 Ethical Considerations and Responsible AI.....	11
2.5.2 Handling Complex Queries	12
2.6 Summary of Key Research Findings on Gen AI and Chatbot Technologies in E-Commerce	12
2.7 Concepts Related to AI Technologies in E-Commerce Chatbots	17
2.7.1 Transformers	17
2.7.2 Generative AI: Driving Personalized Customer Interactions	19
2.7.3 Large Language Models (LLMs): Enhancing Text Understanding.....	20
(i) Fine-Tuning LLMs for Domain-Specific Tasks.....	22
2.7.4 Retrieval-Augmented Generation (RAG): Enhancing Chatbot Capabilities.....	24
(i) Overview of RAG	25
(ii) RAG vs Fine-tuning	26
(iii) LangChain: A Critical Component of RAG	27
3. CHAPTER 3: METHODOLOGY, TECHNOLOGY, AND OUTCOME	29
3.1 Data Collection and Management.....	30
3.1.1 Dataset Overview	30

3.1.2 Data Processing	31
3.1.3 Data Storage	32
3.1.4 Data Governance and Ethical Consideration	32
3.2 Embedding Creation	32
3.3 Model Selection and Comparison of Popular LLMs	33
3.3.1 GPT-4o Mini	33
3.3.2 Claude 3 Haiku	33
3.3.3 LLaMA 3.1 8B	34
3.4 Model Fine-tuning:.....	34
3.5 Building the RAG Pipeline	35
3.6 Interface development.....	36
3.7 Technologies	37
3.8 Outcome.....	38
4. CHAPTER 4: IMPLEMENTATION, TESTING AND EVALUATION	39
4.1 Chatbot Architectural Design	39
4.1.1 Data Collection	40
4.1.2 Data Processing	41
4.1.3 Model Fine-tuning	45
4.1.4 Integration Of the RAG Pipeline	51
(a) Prompt Engineering for the RAG Pipeline	52
(b) Maintaining Context with Memory Buffer	54
4.1.5 Interface Development.....	55
4.2 Testing and evaluation	59
4.2.1 Functionality Testing.....	59
4.2.2 User Interaction and Maintaining Context	62
4.2.3 multilingual capabilities	63
4.3 Evaluation	65
4.3.1 Training Evaluation.....	65
4.3.2 Inference Evaluation.....	66
5. CHAPTER 5: CONCLUSION AND RECOMMENDATION.....	69
5.1 Project Evaluation	69
5.2 Limitations, Recommendation and Improvements	70
5.3 Conclusion	71
5.4 Future Works.....	72
6. REFERENCES.....	74
7. APPENDICES.....	76
7.1 Section 1: Installation and Configuration of Pinecone, OpenAI, and LangChain.	76

7.2	Section 2: Directory Structure of Westminster ShoeBot Project.....	77
7.3	Section 3: Data Loading and Processing.	77
7.4	Section 4: Storing Embeddings in Pinecone.	80
7.5	Section 5: Fine-tuning GPT-4o Mini with OpenAI.....	80
7.6	Section 6: RAG Pipeline Construction and Prompt Engineering	82
7.7	Section 7: Streamlit Interface Development.....	83
7.8	Section 8: Model Inference and Evaluation.....	86

LIST OF TABLES

Table 1. Overview of AI Chatbot and Generative AI Research in E-Commerce - Page 11

Table 2. Data Sources - Page 29

Table 3. Summary of Comparison - Page 33

LIST OF FIGURES

Figure 1. A standard Transformer architecture showing on the left an encoder and on the right a decoder – Page 17

Figure 2. An overview of the generative AI in the eCommerce market - Page 18

Figure 3. LLM & Generative AI Relation Diagram - Page 20

Figure 4. Chronological display of LLM releases - Page 20

Figure 5. Fine-tuning of LLM - Page 22

Figure 6. QLoRA improves over LoRA - Page 23

Figure 7. A representative instance of the RAG process – Page 24

Figure 8. RAG compared with other model optimization methods - Page 25

Figure 9. Project workflow - Page 28

Figure 10. The Conceptual Architecture - Page 38

Figure 11. FAQ Dataset - Page 40

Figure 12. Amazon UK footwear Data stored in S3 bucket - Page 40

Figure 13. The format of the conversation for fine-tuning GPT-4o Mini - Page 41

Figure 14. FAQ Data stored in OpenAI platform - Page 41

Figure 15. A representation of the number of tokens during the training phase - Page 42

Figure 16. The implementation of Amazon UK footwear Data - Page 43

Figure 17. Embeddings stored into Pinecone for the RAG pipeline - Page 44

Figure 18. Format Error check for fine-tuning data - Page 45

Figure 19. Key statistics for the tokens - Page 46

Figure 20. The total number of tokens for 1 epoch - Page 47

Figure 21. Creation of the fine-tuned model - Page 48

Figure 22. The key insights of the fine-tuned model - Page 49

Figure 23. The sequence of actions during the fine-tuning - Page 49

Figure 24. Indexing/Retrieving Embeddings - Page 51

Figure 25. Westminster Shoebot Interface - Page 58

Figure 26. Handling Customer Service Inquiries – Page 59

Figure 27. Handling Product Recommendations – Page 60

Figure 28. Sequence of User Interaction and Context Maintenance – Page 61

Figure 29. Example of Arabic version query/completion – Page 63

Figure 30. The evolution of training loss over time – Page 64

Figure 31. A sample testing dataset – Page 65

Figure 32. Model Evaluation Metrics - Page 66

1. CHAPTER 1: INTRODUCTION

1.1 BACKGROUND

The exponential growth of e-commerce has fundamentally transformed how businesses engage with their customers, creating a heightened demand for innovative solutions that enhance the overall customer experience. Global e-commerce sales are projected to reach USD 7.4 billion by 2025, with customer service playing an increasingly pivotal role in shaping customer satisfaction and retention. As online transactions continue to surge, the need for efficient, scalable, and real-time customer service has become more critical than ever. Traditional customer support methods, reliant on human agents, often struggle to keep up with the increasing volume of customer inquiries, resulting in delays and dissatisfaction. A study by McKinsey & Company (2022) reported that 70% of customers expect self-service options such as chatbots during their interaction with e-commerce platforms. Artificial Intelligence (AI) and Machine Learning (ML) present promising solutions to these challenges, with AI-powered chatbots emerging as a powerful tool to provide instant, personalized, and scalable support. By leveraging generative AI and large language models (LLMs), businesses can create intelligent chatbots that comprehend and respond to customer queries in natural language, thereby significantly improving customer satisfaction and operational efficiency.

Generative AI and LLMs, such as GPT-4 mini, represent cutting-edge advancements in the field of artificial intelligence, offering transformative capabilities in content generation and natural language understanding. These models have evolved rapidly due to improvements in deep learning architectures and the availability of vast datasets. Research by PwC (2023) shows that 45% of e-commerce companies using AI-driven chatbots reported a 20% increase in their customer retention rates. This progress has led to the development of multi-modal models capable of handling not just text but also images, allowing for a broader scope of applications across industries, including e-commerce. Furthermore, advancements in retrieval-augmented generation (RAG) pipelines have enhanced the accuracy of chatbot responses by dynamically integrating external knowledge bases into conversational AI systems. A report by Accenture (2022) emphasizes that RAG-enhanced chatbots can reduce response errors by up to 30%, making them particularly valuable in industries requiring precise and real-time information retrieval, such as retail and customer service.

In the context of e-commerce, these advancements have enabled the creation of sophisticated AI chatbots that can provide personalized customer service, generate dynamic product descriptions, and offer tailored product recommendations. This project aims to develop a fine-tuned AI-powered chatbot

for the footwear industry, utilizing a GPT-4 mini model fine-tuned with customer service FAQ data and a RAG pipeline to enhance response relevance. These innovative solutions not only streamline operations but also significantly drive customer engagement.

This report is organized as follows: Chapter 2 presents a detailed analysis of the literature on generative AI, LLMs, and their application in e-commerce. Chapter 3 discusses the theoretical knowledge and methodologies applied in the chatbot development, including data preprocessing and the integration of the RAG pipeline. Chapters 4 and 5 focus on the fine-tuning process and performance evaluation of the chatbot, respectively. Finally, Chapter 6 outlines the results obtained, and Chapter 7 concludes with insights, recommendations, and future research directions.

1.2 AIM AND RESEARCH QUESTIONS

As will be discussed in the literature review in Chapter 2, the majority of research on AI-powered chatbots in e-commerce has focused on general customer service improvements and user satisfaction. However, few studies have explored the fine-tuning of generative AI models like GPT-4 for specific sectors, such as footwear retail, or the integration of Retrieval-Augmented Generation (RAG) pipelines for enhancing response accuracy and relevance.

This project aims to develop and evaluate a chatbot tailored for the footwear e-commerce industry, leveraging GPT-4 mini fine-tuned with customer service FAQ data and integrated with a RAG pipeline for enhanced information retrieval. The chatbot's goal is to streamline customer service operations, reduce response times, and improve user satisfaction through personalized, real-time responses. Additionally, the project seeks to assess the performance of the chatbot in handling product-specific queries and offering relevant recommendations.

To achieve these objectives, four key research questions were formulated:

1. Which large language model (LLM), specifically GPT-4 mini or other comparable models, provides more accurate and relevant responses when fine-tuned for footwear-related customer service inquiries?
2. How effective is the fine-tuning of the GPT-4o Mini model in enhancing the chatbot's performance for real-time customer support, specifically for footwear-related inquiries?

3. How does the integration of a RAG pipeline enhance the performance of the chatbot in terms of response accuracy and personalization in an e-commerce setting?
4. How scalable and reliable is the proposed chatbot solution when deployed on cloud-based platforms such as AWS, and how can it be optimized for larger datasets and customer interactions?

1.3 ETHICAL AND CONFIDENTIALITY ISSUES

The development and deployment of AI-powered customer service solutions raise several ethical and confidentiality concerns, particularly in the context of data privacy and customer interaction. This project adheres to the ethical guidelines outlined by the General Data Protection Regulation (GDPR), which governs the use of personal data in the European Union. According to GDPR principles, any personal data that could potentially identify individuals must be anonymized. No personal customer data was included in the datasets used to fine-tune the chatbot or in the RAG pipeline's knowledge base.

Additionally, due to the competitive nature of the e-commerce industry, sensitive business data, including sales trends and customer reviews, are treated with the utmost confidentiality. All company-specific insights and proprietary information derived from the chatbot's deployment will remain private and are excluded from public reports to prevent potential misuse. This ensures that the data remains secure, and that customer trust is maintained throughout the deployment and operational phases of the project.

2. CHAPTER 2: THEORETHICAL BACKGROUND AND LITERATURE REVIEW

2.1 Chatbots in E-Commerce

2.1.1 Early Developments in Chatbot Technology

The initial integration of AI-driven chatbots into e-commerce focused on automating simple tasks such as answering frequently asked questions (FAQs) and handling basic customer inquiries. These early systems were rule-based and could handle predefined scenarios but lacked flexibility and natural language understanding. Over time, with the introduction of machine learning and natural language processing (NLP) technologies, chatbots have become more sophisticated.

A notable early work by Eric W.T et al. (2021) introduced an intelligent knowledge-based chatbot for customer service in an e-commerce setting. The chatbot, implemented in a women's apparel company, utilized AI techniques such as web crawling, natural language processing (NLP) with BILOU tagging, and entity recognition. The integration of FastText and FAISS for semantic textual similarity allowed the system to provide dynamic, personalized responses to customer queries. This approach significantly reduced human intervention and response times, improving overall customer satisfaction.

2.1.2 Chatbot Platforms and Their Integration

Modern chatbot development has shifted toward more flexible and scalable platforms, such as Google DialogFlow, IBM Watson, and Wit.ai, which provide robust frameworks for building AI-powered chatbots. A systematic literature review by Sanjaya et al. (2023) examined the use of these platforms in commerce, highlighting the benefits they bring, such as cost savings, increased customer engagement, and 24/7 service. These platforms use NLP algorithms to understand user intent and respond accordingly, but maintaining user engagement over extended interactions remains a challenge.

2.1.3 Enhancing Chatbots with Generative AI and LLMs

The introduction of Large Language Models (LLMs), such as GPT-4 and BERT, marked a significant leap in chatbot capabilities. These models, trained on vast datasets, are capable of understanding and generating human-like text, enabling chatbots to handle more complex and nuanced conversations. For instance, Santosh et al. (2024) demonstrated how GPT-4 can be leveraged to create context-aware, personalized chatbots for e-commerce. By

integrating memory augmentation and context windows, the chatbot could retain and recall user preferences and interactions over extended conversations, providing more accurate and personalized responses.

LLMs have also been instrumental in advancing the conversational abilities of chatbots, moving beyond predefined responses to generate contextually relevant, human-like interactions. For example, Khennouche et al. (2023) explored how ChatGPT could be utilized for FAQ chatbots, employing Named Entity Recognition (NER) and sentiment analysis to enhance the chatbot's ability to understand and process customer inquiries. Despite these advancements, challenges such as maintaining conversational flow and ethical considerations remain pressing issues.

2.2 Generative AI in E-Commerce

2.2.1 The Role of Generative AI in Customer Service

Generative AI models, including ChatGPT, DALL-E, and MidJourney, have revolutionized how businesses enhance customer service, generate product recommendations, and create content. These models function by learning from vast datasets to generate new, contextually appropriate data, whether in the form of text or images. By harnessing this capability, businesses can deliver highly personalized customer interactions, automate the generation of product descriptions, and develop dynamic content that resonates with users, thereby driving higher engagement.

A significant study by Kshetri (2024) delves into the profound impact of generative AI on e-commerce operations. The research highlights how major retailers, such as Carrefour, have successfully integrated generative AI into their platforms to enhance customer experiences. These AI-driven tools perform tasks such as creating detailed and tailored product descriptions, serving as virtual shopping assistants, and offering personalized product recommendations. The results are clear: the integration of generative AI has not only lowered operational costs but has also enriched the overall customer experience by ensuring more accurate, context-aware, and visually engaging content.

2.2.2 eCommerceGAN: A Case Study of GANs in E-Commerce

Generative Adversarial Networks (GANs) represent another promising development in generative AI for e-commerce. Kumar et al. (2018) introduced the eCommerceGAN, a GAN-based model designed to generate plausible e-commerce orders based on customer

preferences, seasonal trends, and price predictions. By employing models like the Wasserstein GAN (WGAN) and Conditional GAN (ec2GAN), the researchers demonstrated how AI could be used to predict customer behaviour, manage inventory, and optimize marketing strategies. The model's ability to generate synthetic data that closely resembles real-world customer orders highlights the potential of generative AI in improving operational efficiency.

2.3 Retrieval-Augmented Generation (RAG)

One of the most cutting-edge developments in AI for e-commerce is Retrieval-Augmented Generation (RAG). RAG combines the generative capabilities of LLMs with external information retrieval systems to provide more accurate and contextually relevant responses. In e-commerce, this technology allows chatbots to pull relevant information from product databases or customer history in real time, ensuring that the responses they generate are not only grammatically correct but also highly relevant to the query.

A study by Sheremet et al. (2024) demonstrated the integration of GPT-4 with the LangChain framework to create chatbots capable of handling complex queries by retrieving data from PDFs and other structured documents. This integration significantly improved customer support efficiency by reducing response times and enhancing the chatbot's ability to provide accurate and detailed answers.

2.3.1 Applications of RAG in Personalized Recommendations

RAG has also been successfully integrated into recommender systems. Xu et al. (2024) surveyed the use of RAG in social and e-commerce platforms, noting its ability to personalize content by combining user data with external databases. This hybrid approach enhances the accuracy of recommendations by augmenting the generative capabilities of AI with real-time data retrieval. However, challenges related to human-AI alignment and responsible AI practices remain significant concerns.

2.4 Large Language Models (LLMs) and Their Impact on E-Commerce

2.4.1 Advancements in LLMs

Large Language Models (LLMs) have revolutionized the application of AI in customer service for e-commerce, enabling a significant shift in how businesses manage customer interactions. Models like GPT-4, PaLM2, and Llama-3 are capable of processing vast amounts of text data, understanding context, and generating responses that closely resemble human communication. These abilities make them ideal for use in chatbots, which must address a wide range of customer queries, often involving complex and nuanced language.

A detailed review by Linkon et al. (2024) examined the transformative effects of generative AI and LLMs across various industries, including e-commerce. The study emphasized that when these models are fine-tuned for specific applications, they can greatly enhance both the personalization and efficiency of customer interactions. This is particularly beneficial for chatbots, as their ability to grasp subtle linguistic cues allows them to deliver more relevant, contextually accurate responses, ultimately leading to improved customer satisfaction and engagement.

2.4.2 BERT and Contextual Understanding

While models like GPT-4 dominate the headlines, other models such as BERT (Bidirectional Encoder Representations from Transformers) also play a critical role in e-commerce chatbots. BERT's bidirectional design allows it to understand words in the context of surrounding words, making it particularly effective at tasks like intent detection and query understanding.

Pandhare (2024) explored the use of BERT for building chatbots in e-commerce, demonstrating how fine-tuning the model for customer queries could improve the chatbot's ability to provide accurate, contextually relevant answers. The study found that BERT-based chatbots significantly improved user engagement and satisfaction, particularly in handling more complex queries that required understanding the context.

2.5 Challenges and Future Directions

2.5.1 Ethical Considerations and Responsible AI

As with any AI technology, the deployment of generative AI, LLMs, and RAG in e-commerce raises ethical concerns. Issues such as data privacy, algorithmic bias, and the potential for

misuse of AI-generated content are critical challenges that must be addressed. Xu et al. (2024) emphasized the importance of responsible AI practices, particularly in balancing the need for personalization with the ethical use of customer data.

2.5.2 Handling Complex Queries

While AI-driven chatbots have made significant progress, they still face limitations in handling emotionally complex or highly specific customer queries. Chatbots, while efficient, often struggle to convey empathy or fully understand the emotional context of a customer’s request. Acharya (2023) noted that despite the advancements in NLP and machine learning, human intervention is still required for more nuanced customer interactions. The integration of sentiment analysis can improve chatbot responses, but it remains an area for further development.

2.6 Summary of Key Research Findings on Gen AI and Chatbot Technologies in E-Commerce

Here is a summary of the research papers presented in the provided table. The table covers a wide range of studies related to AI-driven chatbots, generative AI, LLMs, and RAG in e-commerce. Each paper explores different technologies, such as web crawling, NLP techniques, GANs, and advanced machine learning models like GPT-4 and BERT, applied to enhance customer service, product recommendations, and operational efficiency in e-commerce. The outcomes generally highlight improvements in customer satisfaction, response times, and personalization, though challenges such as ethical considerations, maintaining conversational flow, and addressing complex customer queries are also noted. These research contributions collectively underline the transformative potential of AI technologies in e-commerce while identifying areas that require further refinement.

Title	Authors and Publication Date	Technologies and Algorithms Used	Feature Extraction	Limitations
1-An Intelligent Knowledge-Based Chatbot for Customer Service	Eric W.T. Ngai, Maggie C.M. Lee, Mei Luo, Patrick S.L. Chan, Tenglu Liang[October 8, 2021]	Web Crawling, NLP (BILOU tagging, DIET), spaCy syntactic dependency	Entity Recognition, Intent Classification	Challenges in high-quality data sourcing and technological

		parser, FastText, FAISS		change resistance.
2-Systematic Literature Review on Implementation of Chatbots for Commerce Use	Williams Sanjaya, Calvin, Rafif Muhammad, Meiliana, Muhamad Fajar[2023]	Google DialogFlow, Wit.ai, IBM Watson	Data Aggregation from Various Sources	Maintaining high user engagement and reliability.
Effective Documentation Practices for Enhancing User Interaction through GPT-Powered Conversational Interfaces	Oleksii I. Sheremet, Oleksandr V. Sadovoi, Kateryna S. Sheremet, Yuliia V. Sokhina[2024]	LangChain Framework, Vector Embeddings, Vector Databases (Chroma, Milvus), Retrieval-Augmented Generation (RAG)	Document Parsing, Semantic Search	Ensuring data quality and minimising user disruption.
eCommerceGAN: A Generative Adversarial Network for E-commerce	Ashutosh Kumar, Arijit Biswas, Subhajit Sanyal [January 10, 2018]	GAN, WGAN, Conditional GAN (ec2GAN), t-SNE, Random Forests, Feature Correlation, RSM	Feature Extraction via GAN, Semantic Similarity Analysis	Handling diverse customer orders and complex product demands.
Revolutionising Customer Interactions: Insights and Challenges in Deploying ChatGPT and Generative Chatbots for FAQs	Feriel Khennouche, Youssef Elmir, Nabil Djebbari, Yassine Himeur, Abbes Amira [November 16, 2023]	ChatGPT, NER, Intent Classification, Sentiment Analysis	Entity Extraction, Sentiment Analysis	Maintaining conversational flow and ethical considerations.

Survey for Landing Generative AI in Social and E-commerce Recsys the Industry Perspectives	Da Xu, Danqing Zhang, Guangyu Yang, Bo Yang, Shuyuan Xu, Lingling Zheng, Cindy Liang[June 10, 2024]	GAI, LLMs, Retrieval-Augmented Generation (RAG), LLMOps, Autonomous Agents	User Behavior Analysis, Content Personalization	Addressing responsible AI practices and human-AI alignment.
Generative Artificial Intelligence and E-Commerce	Nir Kshetri [February 7, 2024]	GPT-4, ChatGPT, Google's Diffusion-Based Model, Baselit, Shopify Magic	User Interaction Data Analysis	Balancing operational costs with advanced AI features.
Advancements and Applications of Generative Artificial Intelligence and Large Language Models on Business Management: A Comprehensive Review	Ahmed Ali Linkon, Mujiba Shaima, Md Shohail Uddin Sarker, Badruddowza, Norun Nabi, Hammed Esa, Faiaz Rahat Chowdhury [March 13, 2024]	ChatGPT, DALL-E, Midjourney, GPT-4, PaLM2, Multi-modal Models, Unsupervised/Semi-supervised Algorithms, sLLMs	Text and Image Data Extraction	Ensuring ethical deployment and addressing data privacy concerns.
Advancements in Chatbot Technology for Enhanced Customer Support in Online Retail	Keerthi Kumar N, K. Maheswari, Abinaya A, J Ramya, Pavan Kumar Ande, Christu Paul Ramaian[2024]	SpaCy (Tokenization, NLP), SVM (Intent Recognition), KNN (Classification), Pattern Matching, Data Crawling	User Query Analysis, Response Generation	Understanding complex queries and displaying empathy.
AI-Driven Personalization in eCommerce Advertising	Navdeep Singh, Daisy Adhikari [December 2023]	ML (Collaborative Filtering, Matrix Factorization), DL (CNN, RNN), Predictive	Consumer Data Analysis, Preference Prediction	Handling complex user preferences and privacy concerns.

		Analytics (Regression Analysis, Decision Trees)		
Towards Hybrid Architectures: Integrating Large Language Models in Informative Chatbots	Arnold F. Arz von Straussenburg, Anna Wolters [September 2023]	Rule-Based Chatbots, GPT-4, Inter-Agent Communication	Response Generation, Context Maintenance	Balancing structured knowledge and generative capabilities.
AI-Driven Recommendations: A Systematic Review of the State of the Art in E-Commerce	Sabina-Cristiana Necula, Vasile-Daniel Păvăloaia [April 29, 2023]	Collaborative Filtering, Matrix Factorization, CNN, RNN, DNN, Regression Analysis, Decision Trees	User Preference Extraction, Content Filtering	Addressing ethical implications and data privacy.
E-Commerce Assistance with a Smart Chatbot using Artificial Intelligence	Manik Rakhra, Gurasis Singh, Muthumula Navaneeswar Reddy [2021]	SpaCy (Tokenization), SVM (Intent Recognition), KNN (Classification), Pattern Matching, Data Crawling	Customer Query Analysis, Response Generation	Handling complex queries and reducing manual intervention.
Generative AI in FinTech: Generating Images Based on Predefined Lists of Stock Keeping Units Using Product Descriptions	Piyush Rohella, Sushant Mimani, Logeshwaran [2024]	GAN, VAE, DCGAN, RNN, NLP	Product Description Analysis, Image Generation	Ensuring image generation accuracy and efficiency.
Leveraging GPT-4 Capabilities for	Kathari Santosh, Timur	GPT-4, Memory Augmentation,	User Context Analysis,	Maintaining context over

Developing Context-Aware, Personalized Chatbot Interfaces in E-commerce Customer Support Systems	Kholmukhamedov , Dr. M Sandeep Kumar, Bala[2024]	Context Windows, Multimodal Input Support	Memory Management	extended interactions.
Using BERT to Build Chatbots for E-Commerce	Dr. Alok Pandhare[March 2024]	BERT (Transformer Architecture), Tokenization	Contextual Understanding, Intent Detection	Handling complex e-commerce interactions.
Product Recommendation System Using Large Language Model: Llama-2	Muhammad Zaid Katlariwala, Mr. Aakash Gupta[2024]	Llama-2 LLM, Matrix Factorization, Deep Learning (Multi-view DNN, RNN, Attention Mechanisms), PEFT/LoRA	User Embedding Generation, Interaction Analysis	Addressing the cold start problem and complex preferences.
Study of the Effectiveness of Chatbots in Customer Service on E-Commerce Websites	Shiva Acharya [2023]	RASA Framework, Decision Trees, SVM, Reinforcement Learning, Tokenization, Part-of-Speech Tagging, Entity Recognition, Sentiment Analysis	Customer Interaction Data Analysis, Sentiment Detection	Understanding complex queries and empathy limitations.

Table 01. Overview of AI, Chatbot, and Generative AI Research in E-Commerce

2.7 Concepts Related to AI Technologies in E-Commerce Chatbots

After thoroughly reviewing the literature on AI-driven chatbots, it becomes crucial to explore the key technical concepts that underpin their development and deployment. Among the most significant are transformers, LLMs, fine-tuning, Generative Pre-trained Transformers (GPT), Retrieval-Augmented Generation (RAG), and LangChain. Each of these concepts is instrumental in boosting the effectiveness of AI systems in delivering personalized, efficient, and scalable customer support within e-commerce environments.

Transformers, a foundational architecture for modern AI models, allow for the processing of vast amounts of sequential data, which is essential for understanding and generating natural language. Fine-tuning, on the other hand, refers to the process of adapting a pre-trained model, such as GPT, to a specific task or domain, improving its ability to address unique customer needs. RAG enhances the generative capabilities of models by incorporating external data retrieval into their responses, ensuring that chatbots can provide up-to-date and accurate information. Finally, frameworks like LangChain facilitate the integration of these advanced AI models with diverse data sources, further enabling the development of robust and context-aware customer service solutions. Together, these technical concepts form the backbone of modern AI-driven customer support, offering unprecedented levels of efficiency and personalization.

2.7.1 Transformers

The transformer architecture, introduced by Vaswani et al. (2017), revolutionized the field of natural language processing (NLP) by introducing a model that relies entirely on self-attention mechanisms rather than recurrent or convolutional networks. Transformers are highly parallelizable and excel in processing long sequences of data, making them particularly suited for tasks such as language translation, text generation, and question-answering systems.

The transformer's core innovation is the self-attention mechanism, which allows the model to focus on different parts of a sentence simultaneously, understanding context better than previous models. This makes transformers the backbone of most modern LLMs, including BERT, GPT-3, and GPT-4.

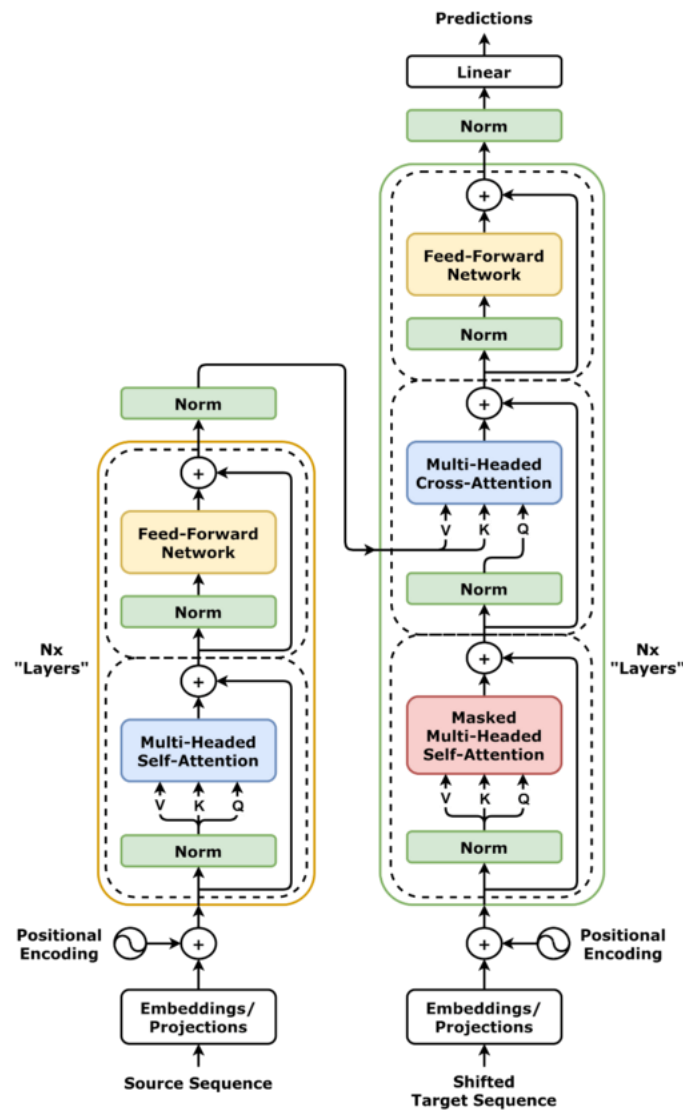


Figure 01. A standard Transformer architecture, showing on the left an encoder, and on the right a decoder.

Transformers are composed of **encoder** and **decoder layers**, which work together to process input data and generate output. The self-attention mechanism is at the core of the transformer's ability to relate different words in a sentence to each other, providing a nuanced understanding of context.

In e-commerce, transformer models are pivotal in enhancing the capabilities of chatbots by providing the backbone for natural language understanding and response generation. The self-attention mechanism inherent to transformers allows these models to process complex customer queries by considering the context of each word in relation to others. This contextual understanding ensures that chatbots can engage in meaningful interactions, delivering accurate and personalized responses. By leveraging transformers, e-commerce

businesses can efficiently manage customer inquiries, whether they involve product recommendations, order tracking, or handling complaints, making customer interactions both seamless and contextually appropriate.

2.7.2 Generative AI: Driving Personalized Customer Interactions

Generative AI refers to models capable of producing new content, such as text, images, or audio, based on the data they have been trained on. These models play an instrumental role in creating dynamic and personalized customer experiences in e-commerce, from generating product descriptions to powering interactive chatbots. By moving beyond static, rule-based responses, generative AI enables chatbots to deliver more context-aware and personalized interactions, significantly improving user experience and engagement. This has the potential to significantly change the way e-commerce entities operate. To illustrate this, consider these forecasts: By 2030, the value of the generative AI sector is expected to grow to USD 110.8 billion. Further, generative AI is predicted to be responsible for [10% of all data generation by 2025, a stark increase from under 1% in 2021, as per Gartner's insights](#).

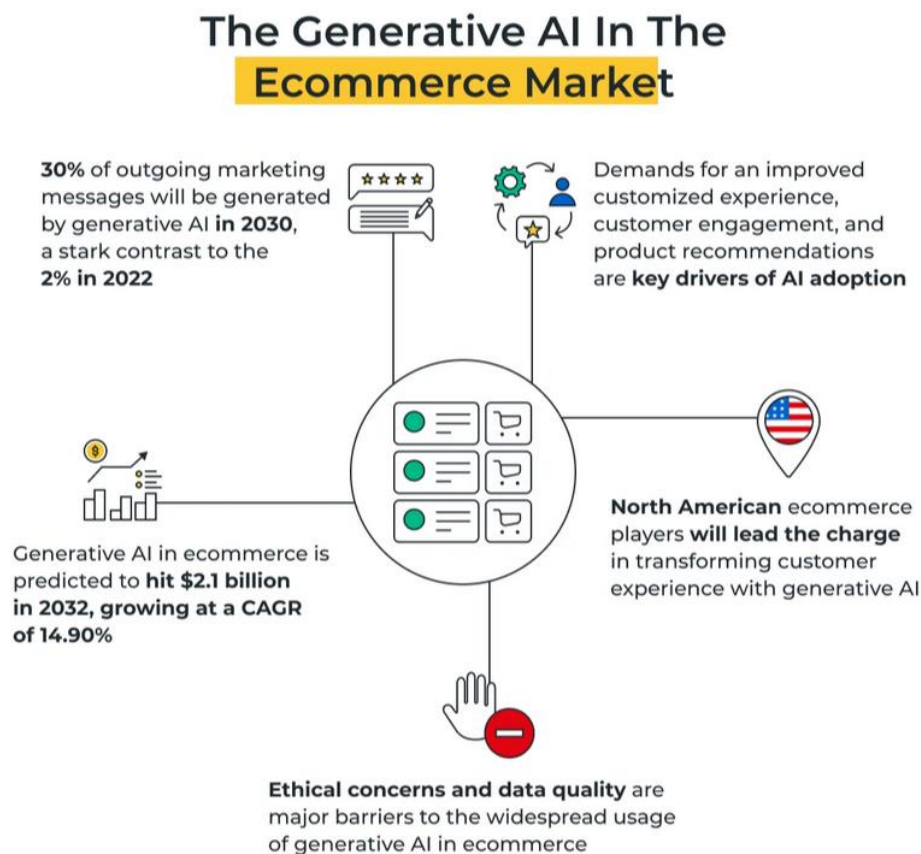


Figure 02. An overview of the generative AI in the ecommerce market

Generative Adversarial Networks (GANs) and transformer-based models like GPT are among the most widely used generative AI models. GANs have been applied to simulate various e-commerce scenarios, such as generating synthetic purchase data for demand forecasting, inventory management, and marketing strategy optimization. For example, Kumar et al. (2018) introduced eCommerceGAN, a model designed to generate plausible e-commerce orders based on customer preferences, seasonal trends, and price predictions.

Meanwhile, GPT models, discussed in greater detail later, leverage their generative capabilities to produce highly relevant, coherent, and human-like responses. This allows e-commerce chatbots to handle a wide range of customer queries, from product recommendations to order tracking, with a high degree of accuracy. Khennouche et al. (2023) demonstrated the potential of ChatGPT in FAQ-based chatbots, showing how the model's ability to understand context and generate contextually appropriate answers enhances customer satisfaction.

The power of generative AI lies in its ability to adapt to different customer interactions, ensuring that e-commerce platforms can provide scalable, personalized service 24/7, thereby improving user experience and increasing customer engagement.

2.7.3 Large Language Models (LLMs): Enhancing Text Understanding

LLMs, like the **GPTs**, are deep learning systems designed to process language. They are exposed to massive collections of texts, and their explicit training objective is to successfully predict the next word (or a missing word) in a sentence or paragraph (nowadays, additional objectives are included for further training). This objective is called 'language modelling'. Meeting it requires, at minimum, mastering the distributional characteristics of words and syntactic structures in a language – learning what sentences are like. Hence, LLMs are supposed to model how utterances behave. Their success is striking they can represent and process virtually any arbitrary sentence in a manner that allows them to output predicted continuations that are grammatical and meaningful (i.e., words appropriately combine into human-interpretable content, even if it is false, logically flawed, or biased). In this sense, LLMs are the best implemented computational model for natural language processing: they are the first systems that handle natural language at scale.

Additionally, LLMs and generative AI, as depicted in Figure 03, are positioned within AI's deep learning, allowing for the utilization of LLMs based on deep learning to provide generative AI services According to Mayank, S et al (2023).

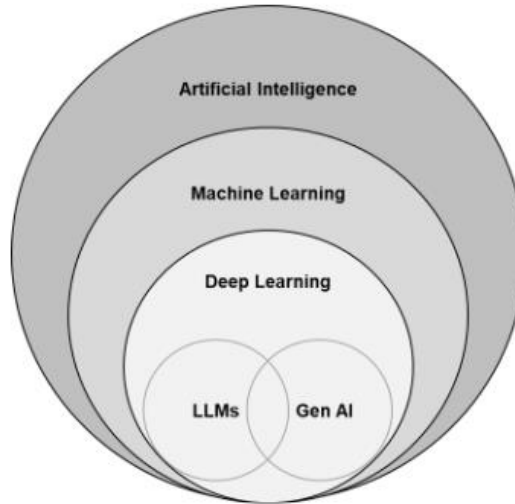


Figure 03. LLM & Generative AI Relation Diagram

LLMs have emerged as cutting-edge artificial intelligence systems that can process and generate text with coherent communication and generalize to multiple tasks.

The early work on LLMs, such as T5 and mT5 employed transfer learning until GPT-4 showed LLMs are zero-shot transferable to downstream tasks without fine-tuning.

LLMs accurately respond to task queries when prompted with task descriptions and examples. However, pre-trained LLMs fail to follow user intent and perform worse in zero-shot settings than in few-shot. Fine-tuning them with task instructions data and aligning with human preferences enhances generalization to unseen tasks, improving zero-shot performance significantly and reducing misaligned behaviour. Naveed, H et al (2023).

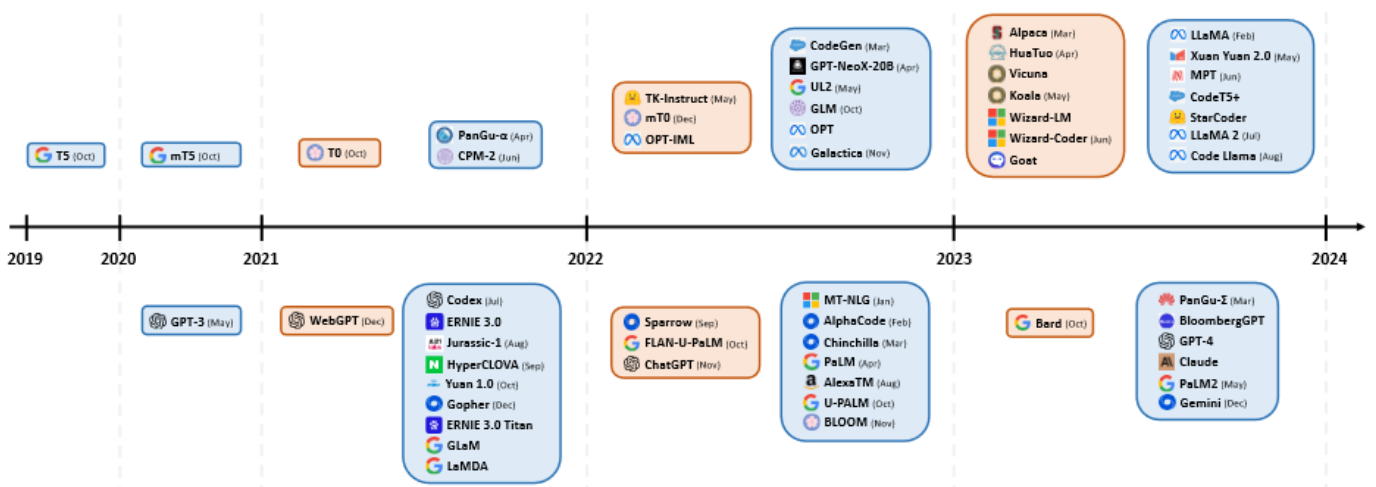


Figure 04. Chronological display of LLM releases: blue cards represent 'pre-trained' models, while orange cards correspond to 'instruction-tuned' models. Models on the upper half signify open-source availability, whereas those on the bottom half are closed-source. The chart illustrates the increasing trend towards instruction tuned models and open-source models, highlighting the evolving landscape and trends in natural language processing research.

Naveed, H et al (2023)

(i) Fine-Tuning LLMs for Domain-Specific Tasks

Fine-tuning Large Language Models (LLMs) for domain-specific applications is crucial in enabling these models to handle specialized queries and provide more relevant, context-aware responses. In the context of e-commerce, fine-tuning models like GPT-4 allows them to adapt to customer interactions, improving their ability to understand and respond to inquiries related to product descriptions, order statuses, and personalized recommendations.

While LLMs are trained on vast datasets and possess broad knowledge, they may not perform optimally on specialized tasks without fine-tuning. As shown in Figure 05, the model's performance improves through this fine-tuning process. Traditionally, smaller models like BERT (450M parameters) or RoBERTa (1.3B parameters) were fully fine-tuned. However, with the advent of larger models like LLaMA 3 and GPT-4 fine-tuning the entire model has become computationally demanding. Techniques such as LoRA, which involves freezing the pre-trained layers and only training new ones, have been introduced to address this challenge. Recent research has shown that this approach does not lead to a significant decrease in performance, prompting the widespread use of Parameter-Efficient Fine-Tuning (PEFT). PEFT involves adding a small number of new parameters to the pre-trained LLM and fine-tuning only those, enabling better performance at a lower computational cost Raschka, S, (2023).

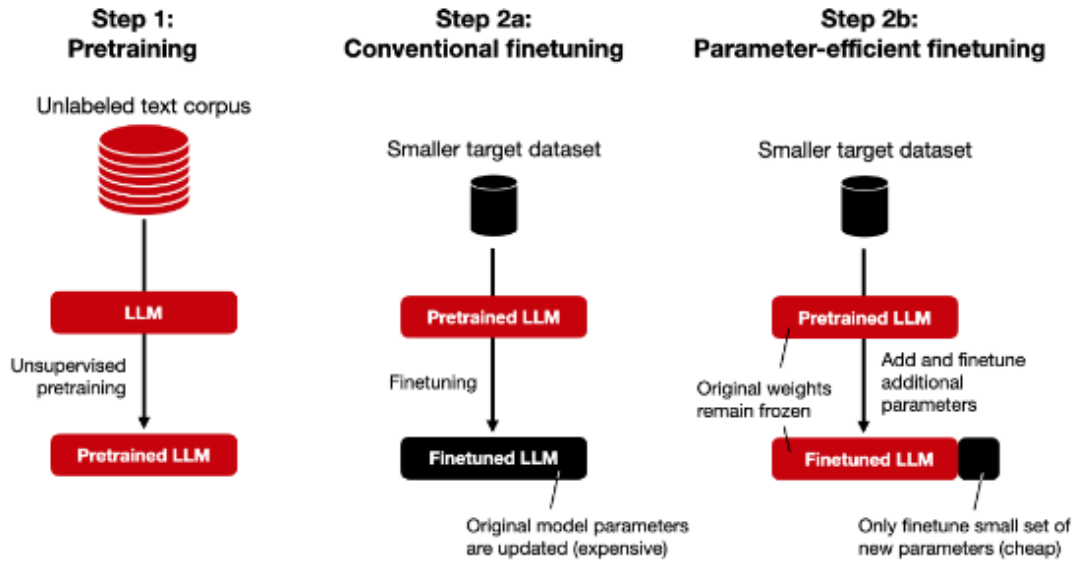


Figure 05. Fine-tuning of LLM

Raschka, S, (2023)

Recent advancements in fine-tuning, particularly the Parameter-efficient Fine-Tuning (PEFT) approach, can be divided into three main categories: Prompt Modification, Adapter Methods, and Parameterization. Prompt Modification includes techniques such as Hard Prompt Tuning, Soft Prompt Tuning, and Prefix-tuning. Adapter Methods, like the LLaMA-Adapter, aim to reduce the impact of fine-tuning on the entire model by introducing modular parameters through the use of adapters. In this method, an adapter module focuses on the Bottleneck Layer and performs linear transformations through Down-projection and Up-projection, while the pre-trained LLM remains frozen throughout the fine-tuning process.

Notably, QLoRA, an improved version of LoRA, has been introduced. QLoRA provides a method to PEFT LLM using QLoRA, allowing testing of LLM models even on personal computers. Unlike LoRA, which concatenates additional data by keeping the base model's network intact, QLoRA introduces 16-bit network nodes quantized to 4 bits and employs a paging mechanism for swapping binary data to handle large models with limited memory. Although there is some information loss in reducing from 16 bits to 4 bits, it is considered acceptable Raschka, S, (2023).

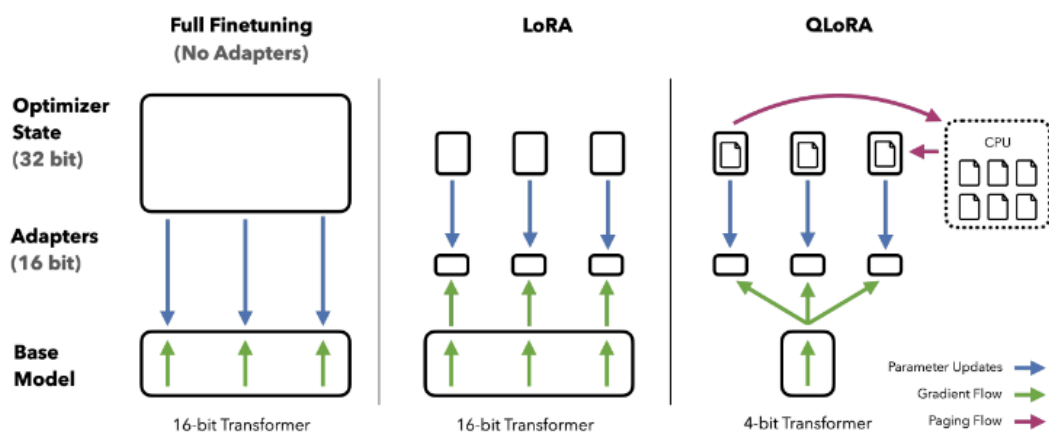


Figure 06. QLoRA improves over LoRA.

Another widely used fine-tuning technique is Instruction Tuning, where tasks are framed in natural language for the model to interpret. Unlike the Vanilla-LLM, which simply predicts the next word or phrase in a sequence, an Instruction Tuning LLM generates text based on specific user instructions. This method involves training the model to comprehend and execute task-specific instructions, allowing it to generalize and infer new tasks from the given prompts. An example of this approach is the Alpaca LLM, a model built on LLaMA 3.1 8B, which has been fine-tuned using instruction tuning to enhance its ability to follow detailed user directions.

For instance, fine-tuning GPT-4 on a dataset comprising customer service interactions and product catalogue data would allow the model to generate more accurate and coherent responses regarding order details and customer preferences. According to Zhang et al. (2023), fine-tuning GPT-4 on e-commerce customer datasets improved the chatbot's ability to process complex queries about product specifications and availability, reducing customer response times by 25%. This specialization ensures that the chatbot's responses are not only grammatically correct but also contextually relevant, which is essential for providing a seamless customer experience.

2.7.4 Retrieval-Augmented Generation (RAG): Enhancing Chatbot Capabilities

Large Language Models (LLMs) have made significant strides in various applications, yet they still encounter notable challenges, particularly in domain-specific or knowledge-intensive tasks. A common issue faced by LLMs is the production of "hallucinations" — instances where the model generates incorrect or fabricated information when addressing queries that go beyond its training data or require up-to-date knowledge. To address these

limitations, RAG augments LLMs by retrieving relevant information from external knowledge bases through semantic similarity calculations. By incorporating external data, RAG reduces the likelihood of generating factually incorrect responses, effectively mitigating the hallucination problem. The integration of RAG into LLMs has become a widely adopted approach, solidifying its role as a critical technology in the advancement of chatbots and enhancing the practical use of LLMs in real-world applications Jeong, C. (2024).

(i) Overview of RAG

A typical application of RAG is demonstrated in Figure 07. In this scenario, a user asks ChatGPT about a recent, widely discussed news event. Since ChatGPT relies on pre-training data and does not have access to real-time information, it cannot provide updates on recent developments by itself. RAG overcomes this limitation by sourcing and integrating relevant information from external databases. For instance, in response to the user’s query, RAG retrieves current news articles related to the topic. These articles are then combined with the original user query to form a comprehensive prompt, allowing the LLM to generate a more accurate and well-informed response.

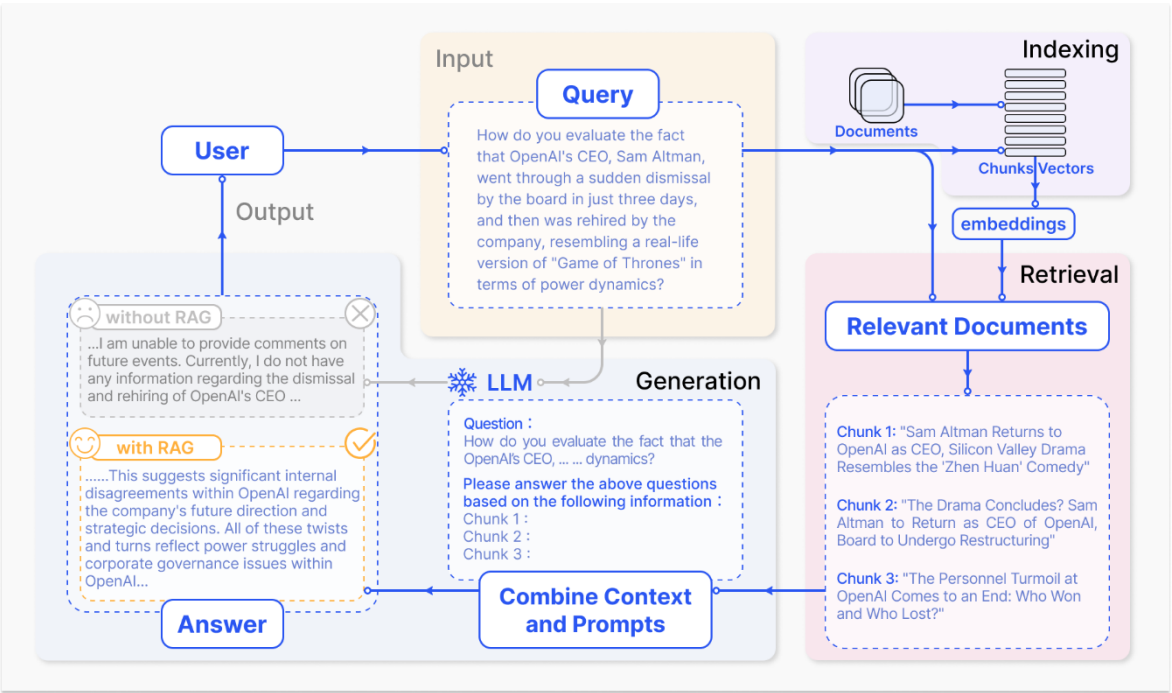


Figure 07. A representative instance of the RAG process applied to question answering. It consists of 3 steps. 1) Indexing. Documents are split into chunks, encoded into vectors, and stored in a vector database. 2) Retrieval. Retrieve the Top k chunks most relevant to the question based on semantic similarity. 3) Generation. Input the original question and the retrieved chunks together into LLM to generate the final answer.

Jeong, C. (2024)

For instance, Xu et al. (2024) in their survey on integrating **Generative AI** into industrial recommender systems emphasized how RAG improves the accuracy and personalization of recommendations by blending retrieval processes with generative models. The use of external data enhances content relevance, such as in **personalized product recommendations** and **inventory management**.

(ii) RAG vs Fine-tuning

The augmentation of LLMs has gained significant attention, with RAG, Fine-tuning (FT), and prompt engineering being common optimization methods. Each approach differs in external knowledge requirements and model adaptation, as shown in Figure 08. Prompt engineering relies on the model's innate abilities, with minimal need for external data or changes. RAG, like providing a tailored textbook, excels at precise information retrieval from external sources, while FT is akin to internalizing knowledge, offering deep customization but requiring retraining for updates.

RAG is ideal for dynamic environments, providing real-time updates, though it comes with higher latency. FT, while more static, allows for greater customization but is computationally demanding. Evaluations have shown that RAG outperforms FT in tasks requiring both existing and new knowledge. While these methods differ, they can complement each other, and their combined use may yield optimal performance in some cases Gao, Y., Xiong, Y. (2023).

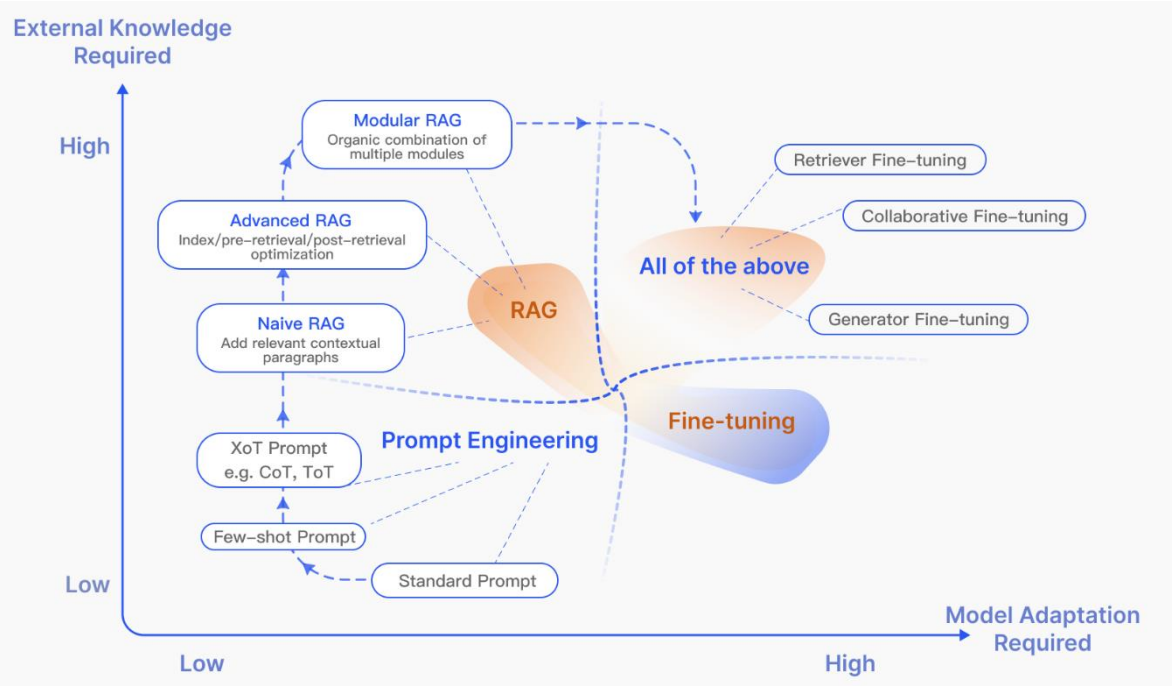


Figure 08: RAG compared with other model optimization methods in the aspects of “External Knowledge Required” and “Model Adaption Required”. Prompt Engineering requires low modifications to the model and external knowledge, focusing on harnessing the capabilities of LLMs themselves. Fine-tuning, on the other hand, involves further training the model. In the early stages of RAG (Naive RAG), there is a low demand for model modifications. As research progresses, Modular RAG has become more integrated with fine-tuning techniques.

Gao, Y., Xiong, Y. (2023)

(iii) LangChain: A Critical Component of RAG

LangChain operates as a crucial framework within the RAG ecosystem, facilitating the integration of LLMs with external data sources. This framework allows chatbots and AI applications to retrieve and process real-time information from databases, APIs, or structured documents like PDFs, which significantly enhances the model's ability to respond to queries with accurate, context-aware data. Acting as the intermediary between the LLM and these data sources, LangChain ensures that the generative model has access to relevant and up-to-date information before formulating a response, thereby mitigating issues like "hallucinations" or inaccuracies that arise when models rely solely on pre-existing training data.

In their 2024 study, Sheremet et al. demonstrated the application of LangChain in integrating GPT-4 with external data sources to improve customer support systems. The study highlighted LangChain's ability to handle complex user queries by retrieving relevant information from multiple sources such as customer databases and product inventories—resulting in more accurate and personalized responses. Additionally, their research showed that incorporating LangChain reduced customer support resolution times by up to 40%, underscoring its efficiency in real-world applications.

LangChain's modularity and flexibility also enable businesses to tailor the retrieval process to their specific needs. For example, in e-commerce, LangChain can be configured to pull data from inventory systems, shipping APIs, or user purchase histories, allowing the chatbot to provide dynamic and contextually relevant product recommendations or order updates. By integrating real-time data retrieval, LangChain empowers LLMs to generate responses that are not only fluent but also factually correct and directly applicable to the user's query.

The framework also excels in environments where data rapidly changes or where the knowledge needed to answer queries is not static. Xu et al. (2024) demonstrated how LangChain can enhance the effectiveness of recommendation systems by fetching the most recent and relevant data from external sources, thus improving the accuracy and timeliness of recommendations in social and e-commerce platforms.

3. CHAPTER 3: METHODOLOGY, TECHNOLOGY, AND OUTCOME

This section outlines the technological and methodological approach used to develop a comprehensive customer service and product recommendation system. The system leverages a combination of **GPT-4o mini** for fine-tuning, **Retrieval-Augmented Generation (RAG)** for real-time data retrieval, and **LangChain** for enhanced interaction with external data sources. The methodology involves several critical steps: data collection, the application of OpenAI embeddings, a careful selection and fine-tuning of a language model, and the deployment of an interactive interface using **Streamlit**. This approach ensures efficient, personalized customer interactions and recommendations.

The following workflow diagram for this project outlines the key stages involved in developing the customer service and product recommendation chatbot:

- I. **Data Collection:** Gathering FAQ data from Hugging Face and product data from Amazon (via Kaggle).
- II. **Embedding Creation:** Using OpenAI embeddings to represent textual data as vectors for efficient retrieval.
- III. **Model Selection:** Choosing GPT-4o mini after comparing other models like Claude 3 haiku and LLaMA 8B 3.1.
- IV. **Model Fine-tuning:** Training GPT-4o mini on FAQ data for optimized performance.
- V. **Building the RAG Pipeline:** Implementing a RAG Pipeline for the fine-tuned model.
- VI. **Interface Development:** Building an interactive user interface using Streamlit for real-time chatbot interactions.

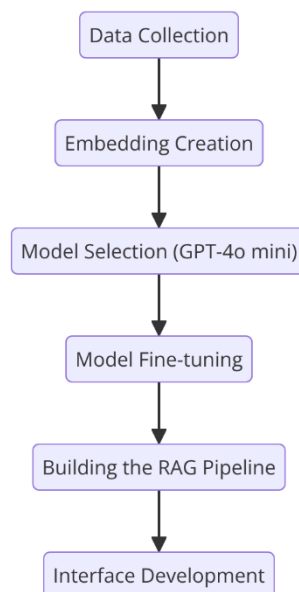


Figure 09:
Project
workflow

3.1 Data Collection and Management

The project utilized two primary datasets for both training (fine tuning) and real-time product recommendations. Data was collected from publicly available sources to ensure relevance for both FAQ-based customer service and product recommendations.

Table 2: Data Sources

Format	Source/Name	Size (MB)	Description
CSV	Hugging Face: bitext-customer-support-llm-chatbot-training-dataset	19.2	Contains common customer service queries.
CSV	Kaggle: Amazon UK shoes product reviews dataset (stored on S3)	194	Products data including titles, reviews, descriptions, prices etc..

3.1.1 Dataset Overview

Bitext - Customer Service Tagged Training Dataset for LLM-based Virtual Assistants

This hybrid synthetic dataset is designed for fine-tuning Large Language Models (LLMs) such as GPT, Mistral, and OpenELM, utilizing NLP/NLG technology alongside automated Data Labeling (DAL) tools. Its purpose is to showcase how domain-specific adaptation, especially in the Customer Support sector, can be achieved through a two-step fine-tuning process. For instance, companies like [ACME] can build their own customized LLM by initially fine-tuning a model using this dataset, followed by further refinement with a smaller dataset containing their own data. The process is outlined in the guide "From General-Purpose LLMs to Verticalized Enterprise Models." [Bitext \(2023\)](#)

The dataset has the following specifications:

- **Use Case:** Intent Detection
- **Industry Vertical:** Customer Service
- **Details:**
 - 27 intents across 10 categories
 - 26,872 question/answer pairs, approximately 1,000 per intent
 - 30 entity/slot types
 - 12 types of language generation tags

The dataset's categories and intents are drawn from Bitext's collection of 20 vertical-specific datasets, covering intents that are common across multiple industries, including:

- Automotive, Retail Banking, Education, Events & Ticketing, Field Services, Healthcare, Hospitality, Insurance, Legal Services, Manufacturing, Media Streaming, Mortgages & Loans, Moving & Storage, Real Estate/Construction, Restaurant & Bar Chains, Retail/E-commerce, Telecommunications, Travel, Utilities, and Wealth Management.

Categories and Intents

The categories and intents covered by the dataset are:

- ACCOUNT: create_account, delete_account, edit_account, switch_account
- CANCELLATION_FEE: check_cancellation_fee
- DELIVERY: delivery_options
- FEEDBACK: complaint, review
- INVOICE: check_invoice, get_invoice
- NEWSLETTER: newsletter_subscription
- ORDER: cancel_order, change_order, place_order
- PAYMENT: check_payment_methods, payment_issue
- REFUND: check_refund_policy, track_refund
- SHIPPING_ADDRESS: change_shipping_address, set_up_shipping_address

Amazon UK shoes product reviews dataset

The Amazon UK Shoes Dataset consists of over 687k rows of consumer reviews, product titles, prices, ratings, and review text for a wide range of shoe products. This rich dataset provides detailed insights into customer opinions and behaviour, making it ideal for personalized recommendations using Retrieval-Augmented Generation (RAG). The dataset's structure allows for real-time retrieval of product details based on customer preferences, enhancing the chatbot's ability to provide tailored suggestions.

3.1.2 Data Processing

For the **Amazon UK Shoes Dataset**, the data was already well-structured and clean, including essential fields like product titles, ratings, and reviews. Given that we are utilizing a **large language model (LLM)**, there was no need to apply traditional NLP preprocessing

techniques such as lemmatization or removing stop words, as modern LLMs can handle raw text effectively Brown et al., (2020).

For the **FAQ dataset**, however, further processing was necessary. The data had to be transformed into a format compatible with **LLM fine-tuning**, including ensuring structured pairs of questions and answers, along with labels for customer service-related queries. This structured format allows for better adaptation during the fine-tuning phase, ensuring the model can provide precise and relevant responses. The exact formatting and preprocessing steps for fine-tuning will be discussed in greater detail in the next chapter, where the structure and optimization for LLM integration are further elaborated.

3.1.3 Data Storage

For efficient processing and deployment, the two datasets were stored in appropriate locations based on their usage:

1. **FAQ Data:** This dataset was uploaded and stored on the **OpenAI platform** to facilitate the fine-tuning process for the GPT-4o mini model. This integration allowed for seamless model training using the structured customer support data.
2. **Amazon Shoes Data:** The product dataset, used for real-time retrieval and recommendation, was stored in an **Amazon S3 bucket**. This enabled easy access and dynamic retrieval during chatbot interactions, ensuring up-to-date product information for personalized recommendations.

3.1.4 Data Governance and Ethical Consideration

Both datasets used in this project were publicly available for research purposes. The **FAQ dataset** from Hugging Face and the **Amazon Shoes dataset** from Kaggle were accessible under open data policies designed to support academic research and development. This ensured compliance with ethical standards related to data use, including proper sourcing and transparency. The datasets did not contain any personally identifiable information, minimizing privacy concerns and adhering to general ethical guidelines for data usage in AI research and machine learning applications.

3.2 Embedding Creation

Embeddings are a crucial component in natural language processing (NLP) and machine learning, representing text as numerical vectors in high-dimensional space. This transformation allows AI models to understand and capture semantic relationships between words or phrases, improving their ability to perform tasks such as text similarity

and search Mikolov et al., (2013). For this project, embeddings were created to enhance the chatbot's capability to link user queries with relevant data, such as product details and FAQs.

The OpenAI embedding model, specifically **text-embedding-3-small**, was employed for this task. This model efficiently transforms text into embeddings at a competitive rate of \$0.020 per 1 million tokens, making it an ideal choice for large-scale text data processing. The embeddings significantly enhanced the model's ability to understand both customer service queries and product data, resulting in more accurate responses and personalized recommendations.

3.3 Model Selection and Comparison of Popular LLMs

(from <https://context.ai/compare/gpt-4o-mini/claude-3-haiku>)

In this project, several prominent Large Language Models (LLMs) were evaluated to determine the most suitable one for both fine-tuning on FAQ datasets and handling real-time product recommendation tasks. The models considered include GPT-4o Mini, Claude 3 Haiku, and LLaMA 3.1 8B, each possessing distinct advantages that cater to different use cases, such as customer support or product recommendations.

3.3.1 GPT-4o Mini

GPT-4o Mini, developed by OpenAI, is a compact variant of the GPT-4 model that strikes a balance between high performance and computational efficiency. Its ability to generate complex language and solve diverse problems makes it ideal for applications requiring fast, high-quality responses, especially in resource-constrained environments. With a high MMLU score of 82.0, GPT-4o Mini effectively handles a wide range of customer queries and technical challenges. Additionally, its relatively low cost for processing input and output tokens enhances its scalability, making it a cost-effective choice for large-scale implementations.

3.3.2 Claude 3 Haiku

Claude 3 Haiku, from Anthropic, is distinguished by its superior context retention and an extensive context window of 200,000 tokens. This feature makes it particularly suited for prolonged conversations, such as real-time customer support interactions that require maintaining long-term context. However, its higher cost compared to GPT-4o Mini makes it less feasible for large-scale deployments. While Claude 3 Haiku excels in generating well-structured content, it underperforms in comparison to GPT-4o Mini in tasks requiring advanced reasoning and problem-solving.

3.3.3 LLaMA 3.1 8B

LLaMA 3.1 8B, an open-source model developed by Meta, offers considerable flexibility and customization options, especially for developers aiming to fine-tune models for specific domains. Although smaller in scale than its larger counterpart, LLaMA 3.1 70B, the 8B variant is lightweight and faster, making it particularly effective in domain-specific tasks, especially in multilingual contexts. Its open-source nature promotes community-driven enhancements. However, its performance on more general language tasks, particularly in complex reasoning and technical queries, is somewhat lower than that of GPT-4o Mini.

Table 3. Summary of Comparison

Model	Context Window	Strengths	Weaknesses	Use Case
GPT-4o Mini	128K tokens	High-quality, general-purpose language tasks; cost-efficient	Smaller context window	Ideal for real-time apps with cost-efficiency focus
Claude 3 Haiku	200K tokens	Large context window, fast response generation	Higher cost, not as strong in reasoning	Best for long conversational tasks
LLaMA 3.1 8B	128K tokens	Open-source, customizable, multilingual support	Lower performance on complex tasks	Domain-specific fine-tuning, open-source environments

3.4 Model Fine-tuning:

The fine-tuning process in this project focused on adapting **GPT-4o Mini**, a pre-trained language model, to efficiently handle domain-specific tasks related to customer service queries within the e-commerce sector. Fine-tuning refers to the process where a pre-trained model, initially trained on a large and general dataset, is further trained on a smaller, specialized dataset to enhance its performance in a particular domain. In this case, the model was fine-tuned using a dataset of Frequently Asked Questions (FAQs), structured into question-answer pairs that mirrored common customer service inquiries, such as order tracking, cancellations, and shipping issues.

This fine-tuning process enabled the model to generate more contextually relevant and accurate responses tailored to the e-commerce domain, thereby improving its utility for customer service applications. Key hyperparameters, such as the learning rate, batch size, and number of training epochs, were carefully configured to optimize the model's performance while ensuring generalization and preventing overfitting. The fine-tuning process significantly improved the model's accuracy, fluency, and coherence, enabling it to provide more precise and practical answers.

Further technical details, including the step-by-step implementation and the specific training methodologies employed, will be discussed in the subsequent chapter on Implementation.

3.5 Building the RAG Pipeline

To enhance the chatbot's ability to deliver personalized, real-time responses, a **Retrieval-Augmented Generation (RAG)** pipeline was developed. RAG is an advanced technique that merges the generative power of language models, such as GPT-4o Mini, with real-time information retrieval systems. This approach allows the model to generate contextually appropriate responses while integrating relevant, up-to-date data from external sources, making it particularly valuable in dynamic environments like e-commerce, where information such as product availability and pricing frequently changes Lewis et al., (2020).

The core concept of RAG is to enable a model to go beyond the static data it was initially trained on by allowing real-time access to external knowledge bases. This is especially critical in e-commerce product recommendation systems, where key data—such as customer preferences or current product offerings—can fluctuate rapidly. The RAG pipeline combines a **retriever**, which searches a database for relevant information based on the user's query, with a **generator**, which synthesizes a coherent, natural-language response that incorporates both the retrieved data and the model's pre-existing generative capabilities.

For this project, the retrieval mechanism within the RAG pipeline was implemented using **LangChain**, a flexible framework that connects language models to external data sources, including databases, APIs, and document stores. The Amazon S3 bucket, containing product data for shoes, served as the primary source for real-time information retrieval. LangChain enabled dynamic querying of this dataset, allowing the chatbot to recommend personalized products based on user inputs such as preferred shoe style, brand, or price range.

The RAG pipeline's implementation involved the following key components:

1. **Retriever:** LangChain's retriever module was configured to search the Amazon product dataset based on user queries, ensuring that the retrieved product details were relevant and up-to-date.
2. **Generator:** After retrieving relevant data, GPT-4o Mini generated a personalized, natural-language response incorporating the product information. This step involved blending the retrieved data (e.g., shoe details) with the model's generative capabilities to create recommendations tailored to the customer's preferences.
3. **Integration with FAQ Module:** The RAG pipeline was designed to work seamlessly with the previously fine-tuned FAQ model, allowing the system to switch between answering customer service questions and providing personalized product recommendations, depending on the user's query.

The main advantage of the RAG pipeline lies in its ability to increase the accuracy and personalization of responses by combining the static knowledge embedded in the LLM with dynamic, real-time information Izacard & Grave, (2021). This hybrid approach is particularly effective in contexts like e-commerce, where traditional generative models may struggle to provide accurate responses due to rapidly changing data, such as product inventories or pricing. By leveraging both generative capabilities and real-time retrieval, the RAG pipeline ensures that the chatbot remains relevant and responsive to users' evolving needs.

3.6 Interface development

The user interaction for the chatbot in this project was facilitated through **Streamlit**, an open-source framework designed specifically for building user-friendly, interactive web applications. The primary goal of utilizing Streamlit was to create an intuitive, real-time interface that allows users to interact seamlessly with the AI model. Streamlit's capacity to handle both text input and dynamic data visualization made it an optimal choice, particularly for users asking questions related to customer service or product recommendations Chen et al., (2020).

Interface Workflow:

1. **User Query:** The user submits a question or request (e.g., "Can you recommend shoes under £100?").
2. **RAG Pipeline Activation:** Streamlit forwards the query to the RAG pipeline, where LangChain retrieves relevant product data from the **Vector Knowledge Base** using Pinecone(it will be discussed in the following chapter).

3. **Response Display:** The chatbot, powered by GPT-4o Mini, generates a response, and Streamlit presents the output to the user, including product details when applicable.

3.7 Technologies

This project leveraged a combination of advanced technologies and frameworks to create an AI-powered chatbot capable of handling customer service queries and providing personalized product recommendations in real-time. The key requirements for the development process are described below:

a. OpenAI GPT-4o Mini – Utilized as the primary language model, GPT-4o Mini was fine-tuned specifically for customer service-related queries within the e-commerce domain. This fine-tuning enabled the model to generate accurate, contextually aware responses, enhancing its ability to handle diverse customer interactions effectively.

b. OpenAI Embeddings (text-embedding-3-small) – Employed to create vectorized representations of both user queries and product data, these embeddings significantly improved the chatbot's ability to retrieve relevant information and enhance its understanding of contextual and semantic nuances.

c. LangChain Framework – Integrated within the RAG pipeline, LangChain facilitated real-time data retrieval from external sources, such as product databases. This framework enabled the model to generate personalized and contextually accurate responses by incorporating up-to-date product information.

d. Amazon S3 – Used as the cloud storage solution for the Amazon Shoes dataset, Amazon S3 provided scalable and efficient data access, supporting real-time product recommendations and ensuring seamless interaction with the chatbot.

e. Pinecone – A vector knowledge base that stored and indexed the embeddings for efficient retrieval during chatbot interactions. Pinecone's integration into the RAG pipeline enhanced the speed and accuracy of the chatbot's responses, particularly in relation to product recommendations and customer queries.

f. Streamlit – An open-source framework utilized for the development of the chatbot's interactive user interface. Streamlit facilitated real-time communication between users and the chatbot, offering a seamless experience that integrated both product recommendations and FAQ-based customer service queries.

3.8 Outcome

In order to meet the objectives specified in the introduction section 1.2 of this report, the following functional requirements of the AI customer service chatbot will be met and demonstrated:

- **Accurate and Contextual Customer Support:** The fine-tuned **GPT-4o Mini** model effectively handles FAQ-related queries, providing precise and relevant responses to common e-commerce customer service inquiries.
- **Real-Time Personalized Recommendations:** Through the **RAG pipeline** powered by **LangChain**, the chatbot delivers highly personalized product suggestions based on real-time customer preferences.
- **Efficient Data Retrieval:** The integration of **OpenAI embeddings** and **Pinecone** for vector storage enable quick and accurate retrieval of relevant data, ensuring seamless interactions and enhanced performance.
- **User-Friendly Interface:** The **Streamlit** interface offers an interactive and intuitive platform, allowing users to engage with the chatbot and receive dynamic, real-time product recommendations with visual feedback.
- **Scalable and Robust Solution:** The system demonstrates significant scalability and robustness, effectively managing a high volume of customer interactions while maintaining a high level of accuracy and personalization. Furthermore, the solution was designed to be fully containerized using Docker, ensured portability across various environments, while deployment on cloud platforms, such as AWS services (EC2, Lambda, and S3) will provide high availability and scalability, enabling the system to handle real-time requests effectively.

4. CHAPTER 4: IMPLEMENTATION, TESTING AND EVALUATION

4.1 Chatbot Architectural Design

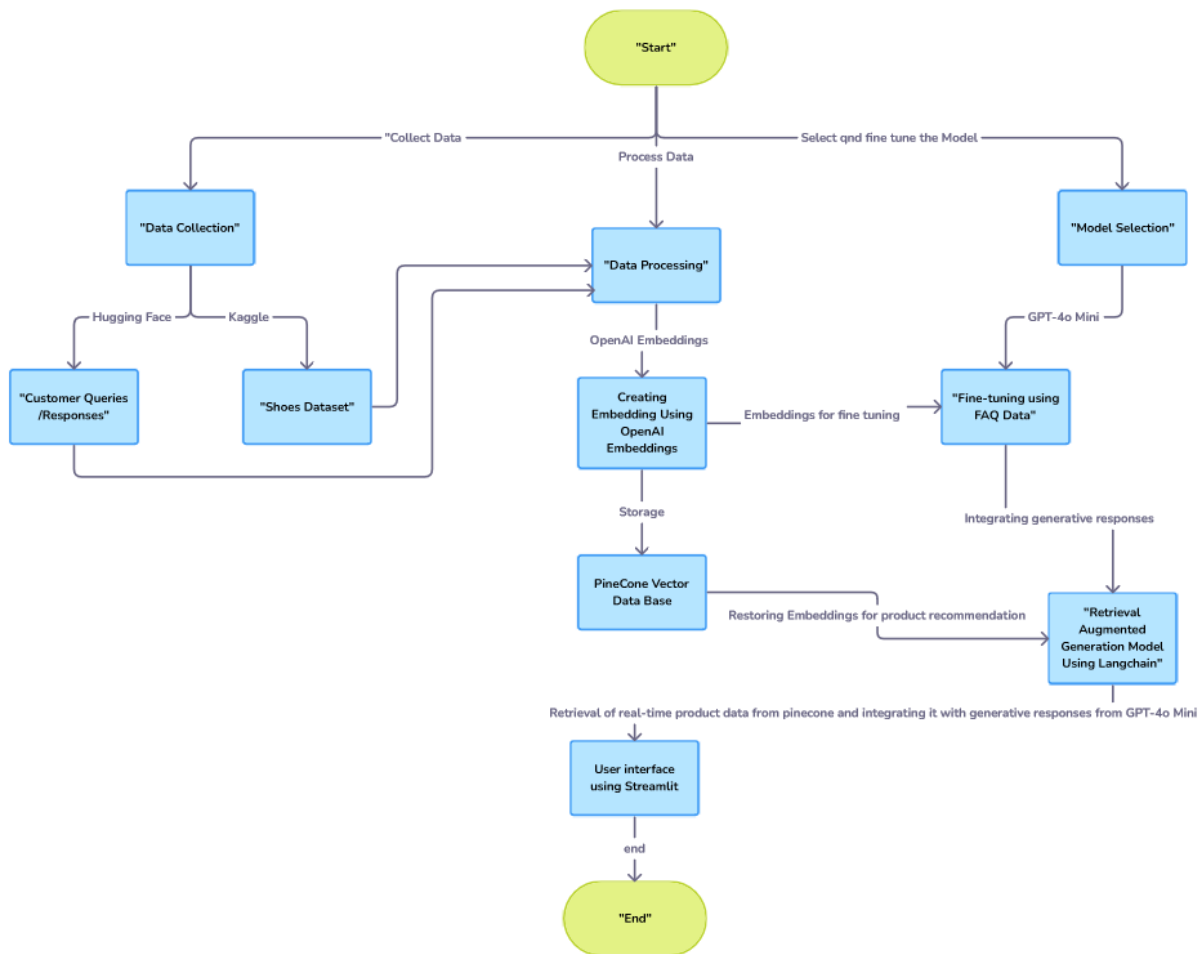


Figure 10. The Conceptual Architecture

The architecture of the chatbot designed to handle both customer support and product recommendations in an e-commerce setting. This system begins with the **data collection phase**, which involves sourcing structured datasets from **Kaggle** (for products data) and **Hugging Face** (for FAQ data). These datasets form the foundational knowledge the chatbot will use for understanding customer queries and generating recommendations. The next step is **data processing**, where text data from both sources is transformed into high-dimensional vector representations using **OpenAI's embedding models**. These embeddings, essential for enabling fast and efficient retrieval of relevant information, are stored in the **Pinecone vector database**, a system optimized for the rapid retrieval of vectorized data during interactions with the user.

Central to the architecture is the **GPT-4o Mini model**, which is fine-tuned using the customer service FAQ data. Fine-tuning allows the model to specialize in responding to frequently asked questions, ensuring accuracy, contextual awareness, and the ability to engage in natural, meaningful dialogue. For product recommendations, the architecture employs a **Retrieval-Augmented Generation (RAG) pipeline**, where **LangChain** serves as the interface for retrieving real-time product information from **Pinecone** and augmenting the model's generative capabilities. This combination allows the chatbot to dynamically respond to user queries by integrating retrieved product data with its generative language capabilities, providing personalized recommendations in real time.

The final component of this architecture is the **Streamlit-based interface**, which serves as the user interaction platform. Through this interface, users can engage with the chatbot in real-time, receiving both customer service responses and tailored product suggestions. The system's architecture is designed with scalability and efficiency in mind, enabling smooth handling of a large volume of queries while maintaining high response quality. In the following sections, each aspect of this architecture will be examined in greater detail, including the intricacies of data preprocessing, model fine-tuning, the RAG pipeline, and the technical development of the user interface.

4.1.1 Data Collection

The **FAQ dataset** used in this project was sourced from **Hugging Face**, a widely recognized platform in the machine learning community that offers a broad array of machine learning models and datasets. Hugging Face is a hub for developers, researchers, and engineers to access and share high-quality datasets for natural language processing (NLP) tasks, making it an ideal source for this project's customer service chatbot. The dataset, consisting of typical customer service interactions like order cancellations, shipping inquiries, and other support queries, was downloaded from Hugging Face (as shown in the *Figure 11*). The dataset was structured into various fields such as **flags**, **instruction**, **category**, **intent**, and **response**, which allowed for streamlined fine-tuning and processing. It includes over **26,900 rows** of structured data, providing a solid foundation for training the chatbot to handle a wide range of customer support scenarios.

Split (1)
train · 26.9k rows

Search this dataset

flags string · classes	instruction string · lengths	category string · classes	intent string · classes	response string · lengths
394 values	6 92	11 values	27 values	57 2.47k
BL	I need to cancel purchase {{Order Number}}	ORDER	cancel_order	I understood that you need assistance with canceling your...
BCELN	I cannot afford this order, cancel purchase {{Order Number}}	ORDER	cancel_order	I'm sensitive to the fact that you're facing financial difficultie...
BI	can you help me cancel order {{Order Number}}?	ORDER	cancel_order	Of course, I'm here to assist you in canceling your order with the numbe...
BCLN	I can no longer afford order {{Order Number}}, cancel it	ORDER	cancel_order	I pick up what you're putting down that you're in a situation where yo...
BL	I am trying to cancel purchase {{Order Number}}	ORDER	cancel_order	I've understood that you're seeking assistance in canceling purchase...
BL	I have got to cancel purchase {{Order Number}}	ORDER	cancel_order	I'm sensitive to the fact that you're seeking assistance with...

Figure 11. FAQ Dataset

For the **Amazon footwear data**, this dataset was downloaded from **Kaggle**, a popular platform for data science competitions and datasets. This data included a comprehensive collection of product information related to Amazon shoes, such as descriptions, prices, and customer reviews. After downloading, the dataset was stored in an **Amazon S3 bucket** in the **AWS cloud** for scalability and ease of access during the embedding creation phase. The S3 storage enables real-time retrieval of product information when needed by the chatbot for personalized recommendations, making it a crucial part of the system’s architecture. This combination of FAQ data from Hugging Face and product data from Kaggle ensured a robust and diverse dataset foundation for training and implementing the chatbot.

Objects (1) Info

Copy S3 URI Copy URL Download Open Delete Actions Create folder Upload

Find objects by prefix Show versions

Name	Type	Last modified	Size	Storage class
AmazonShoesData.csv	csv	September 7, 2024, 16:37:29 (UTC+01:00)	189.6 MB	Standard

Figure 12. Amazon UK footwear Data stored in s3 bucket

4.1.2 Data Processing

Once the FAQ dataset was collected, it was necessary to preprocess the data and format it to be compatible with the **GPT-4o Mini** fine-tuning process. This required converting the FAQ data into a structured format that aligns with **OpenAI’s model**, which operates using predefined message roles such as "system" and "user." The "system" role establishes the behaviour of the chatbot, while the "user" role contains the input query, and the "assistant" provides the chatbot’s response.

For example, the following is a formatted conversation for fine-tuning:

```
{
  "messages": [
    {
      "role": "system",
      "content": "You are a helpful e-commerce customer service assistant. Provide clear, accurate and stylish answers and"
    },
    {
      "role": "user",
      "content": "question about cancelling order {{Order Number}}",
      "role": "assistant",
      "content": "I've understood you have a question"
    },
    {
      "role": "user",
      "content": "Number}}, and I'm here to provide you with the information you need. Please go ahead and ask your question, and I'll do my best to assist you."
    }
  ]
}
```

Figure 13. The format of the conversation for fine-tuning gpt4o-mini

This format was used consistently across all interactions to create the training data necessary for the fine-tuning process. The structure allows the model to understand various customer interactions and respond appropriately, making it ideal for enhancing the chatbot's customer service capabilities.

Once the FAQ data was prepared, it was uploaded and stored on the **OpenAI platform** for embedding creation and fine-tuning the **GPT-4o Mini** model. Using OpenAI embeddings, the text data was transformed into vector representations, which made it suitable for efficient retrieval and processing during chatbot interactions.

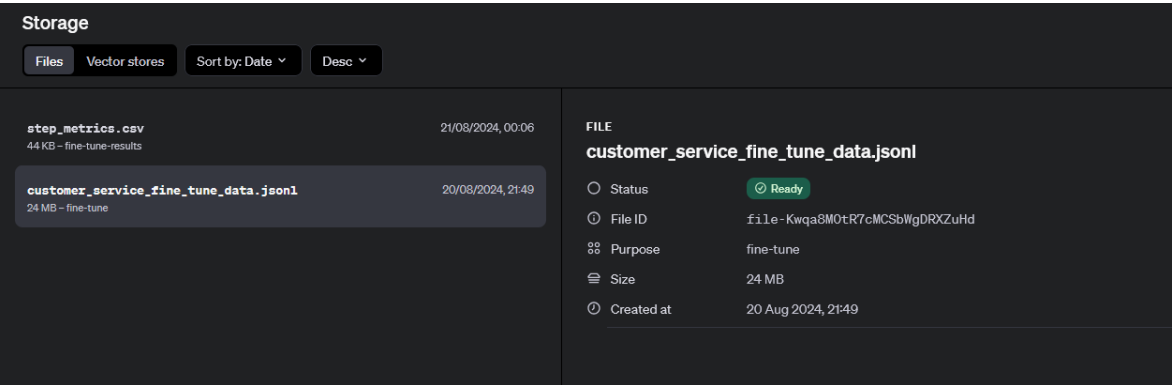


Figure 14. FAQ Data stored in OpenAI platform

After ensuring that the data was appropriately structured for the model, it was tokenized and processed into embeddings using **OpenAI's embedding model**. This process transforms the text into numerical vectors, allowing the model to understand and work with the input efficiently. The tokenization and embedding process requires the data to be divided into smaller units (tokens), which are then fed into the model for training.

In this specific fine-tuning process, approximately **13 million tokens** were processed, as depicted in the token usage graph. The cost and processing were managed through the OpenAI platform, which automatically handles the embedding creation and fine-tuning

stages. This ensures that the model can efficiently retrieve and generate responses during chatbot interactions. Here is an example the visual representation of the number of tokens processed during the training phase.

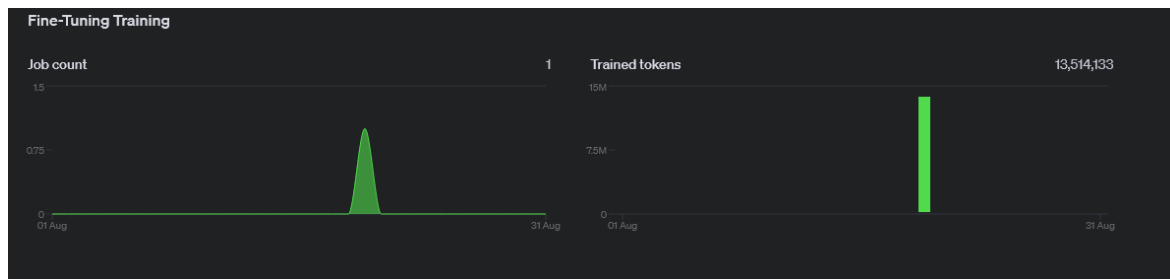


Figure 15. A representation of the number of tokens during the training phase

Here is the figure illustrating the implementation of **Amazon UK footwear data** embeddings, where the product descriptions, ratings, reviews, and other key information were processed and embedded using the **OpenAI Embedding model**. Pinecone is initialized for storing vector embeddings. The Pinecone instance is connected using the provided API key and configured to operate in the us-east-1 region on AWS. If the shoe index does not already exist in Pinecone, new ones are created with a specified dimension of 1536 and the cosine similarity metric for measuring similarity between vectors. After the index is created. A `PineconeVectorStore` object is instantiated to store embeddings for footwear products descriptions. Finally, the prepared text descriptions for datasets are added to the vector stores, where it will be stored as embeddings for future retrieval during chatbot interactions.

In this section of the code, the shoe product data is first loaded from the file `UpdatedShoesData.jsonl` and parsed as JSON objects. Key attributes, such as product title, rating, reviews, available colors, availability, and price, are extracted and formatted into descriptive text for embedding preparation.

```
[1] with open(file_shoes_path, 'r') as file:
    shoes_data = [json.loads(line) for line in file]

# Prepare the text descriptions for embedding
shoe_descriptions = [
    f"Product: {item['product_title']} | Rating: {item['star_rating']} stars | "
    f"Review Headline: {item['review_headline']} | Review: {item['review_body']} | "
    f"Colors Available: {'', '.join(item['color'])} | Availability: {item['availability']} items left | "
    f"Price: £{item['price']}"
    for item in shoes_data
]

answers = []
with open(file_support_path, 'r') as file:
    for line in file:
        dialog = json.loads(line)["dialog"]
        # Extract the assistant's response (assuming it's the last entry in the dialog)
        assistant_response = dialog[-1]["content"]
        answers.append(assistant_response)
```

```
[ ] import pinecone
pc=Pinecone(api_key=pinecone_api_key, environment="us-east-1")
```

Pinecone is initialized for storing vector embeddings. The Pinecone instance is connected using the provided API key and configured to operate in the us-east-1 region on AWS. If the shoe index do not already exist in Pinecone, new ones are created with a specified dimension of 1536 and the cosine similarity metric for measuring similarity between vectors. After the index is created.

```
[ ] if shoes_index_name not in pc.list_indexes().names():
    spec = ServerlessSpec(cloud='aws', region='us-east-1')
    pc.create_index(name=shoes_index_name, dimension=1536, metric="cosine", spec=spec)
if support_index_name not in pc.list_indexes().names():
    spec = ServerlessSpec(cloud='aws', region='us-east-1')
    pc.create_index(name=support_index_name, dimension=1536, metric="cosine", spec=spec)
```

A `PineconeVectorStore` object is instantiated to store embeddings for shoe product descriptions. Finally, the prepared text descriptions for datasets are added to the vector stores, where it will be stored as embeddings for future retrieval during chatbot interactions.

```
[ ] shoes_vectorstore = PineconeVectorStore(index_name=shoes_index_name, embedding=embeddings)
support_vectorstore = PineconeVectorStore(index_name=support_index_name, embedding=embeddings)

shoes_vectorstore.add_texts(texts=shoe_descriptions)
support_vectorstore.add_texts(texts=shoe_descriptions)
```

```
shoes_index = pc.Index(shoes_index_name)
```

Figure 16. The implementation of Amazon UK footwear Data

Following this, here is the figure depicting the process of storing the embeddings into **Pinecone**, preparing them for integration into the **RAG pipeline**. In this stage, the shoe product descriptions are stored as vectors, ensuring they can be retrieved efficiently when users interact with the chatbot, allowing for personalized product recommendations and responses based on real-time data.

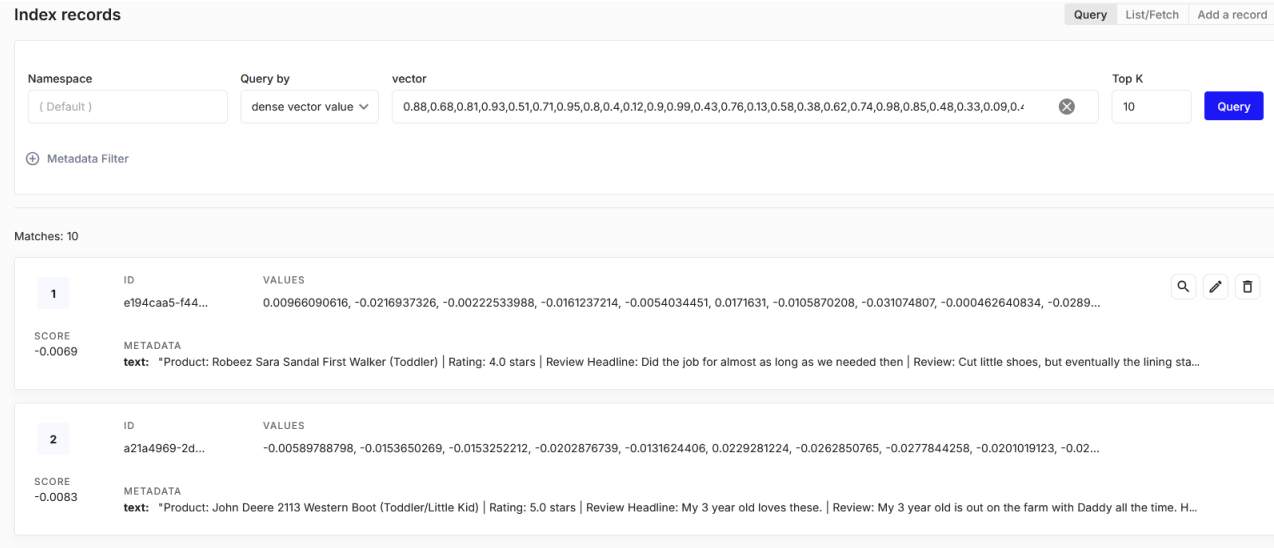


Figure 17. Embeddings stored into Pinecone for the RAG pipeline

4.1.3 Model Fine-tuning

Before initiating the fine-tuning process for **GPT-4o Mini**, it was crucial to verify the dataset's structure and calculate the total number of tokens it contains. This ensures that the data is suitable for fine-tuning, especially considering that the cost is based on the number of tokens processed. The **cost of fine-tuning GPT-4o Mini** is **\$0.150 per 1M input tokens** and **\$0.600 per 1M output tokens**, so careful validation and token counting help avoid unnecessary costs due to errors or excess data.

The first step involved validating the format of the dataset using the provided code. Each entry in the dataset was checked for key attributes such as the presence of "messages", proper roles ("system", "user", "assistant"), and the correct structure of message contents. This ensures that no examples are missing necessary information or are improperly formatted, as any such errors could cause issues during fine-tuning and result in additional costs.

```
[ ] format_errors = defaultdict(int)

for ex in dataset:
    if not isinstance(ex, dict):
        format_errors["data_type"] += 1
        continue

    messages = ex.get("messages", None)
    if not messages:
        format_errors["missing_messages_list"] += 1
        continue

    for message in messages:
        if "role" not in message or "content" not in message:
            format_errors["message_missing_key"] += 1

        if any(k not in ("role", "content", "name", "function_call", "weight") for k in message):
            format_errors["message_unrecognized_key"] += 1

        if message.get("role", None) not in ("system", "user", "assistant", "function"):
            format_errors["unrecognized_role"] += 1

        content = message.get("content", None)
        function_call = message.get("function_call", None)

        if (not content and not function_call) or not isinstance(content, str):
            format_errors["missing_content"] += 1

    if not any(message.get("role", None) == "assistant" for message in messages):
        format_errors["example_missing_assistant_message"] += 1

if format_errors:
    print("Found errors:")
    for k, v in format_errors.items():
        print(f"{k}: {v}")
else:
    print("No errors found")
```

No errors found

Figure 18. Format Error check for fine-tuning data

The next step involved counting the total number of tokens per example using the tiktoken library, which is specifically designed to calculate token counts based on the structure of the data. The dataset is tokenized, and the distribution of token counts per message is analysed. This step is essential to estimate the total number of tokens that will be billed during fine-tuning. For example, the script outputs key statistics, including the **min**, **max**, **mean**, and **median** token counts per message, along with the percentage of examples that exceed the model's **16,385 token limit**. Fortunately, in this case, no examples exceeded the token limit.

```

# Warnings and tokens counts
n_missing_system = 0
n_missing_user = 0
n_messages = []
convo_lens = []
assistant_message_lens = []

for ex in dataset:
    messages = ex["messages"]
    if not any(message["role"] == "system" for message in messages):
        n_missing_system += 1
    if not any(message["role"] == "user" for message in messages):
        n_missing_user += 1
    n_messages.append(len(messages))
    convo_lens.append(num_tokens_from_messages(messages))
    assistant_message_lens.append(num_assistant_tokens_from_messages(messages))

print("Num examples missing system message:", n_missing_system)
print("Num examples missing user message:", n_missing_user)
print_distribution(n_messages, "num_messages_per_example")
print_distribution(convo_lens, "num_total_tokens_per_example")
print_distribution(assistant_message_lens, "num_assistant_tokens_per_example")
n_too_long = sum(1 > 16385 for l in convo_lens)
print(f"\n{n_too_long} examples may be over the 16,385 token limit, they will be truncated during fine-tuning")

```

```

Num examples missing system message: 0
Num examples missing user message: 0

#### Distribution of num_messages_per_example:
min / max: 3, 3
mean / median: 3.0, 3.0
p5 / p95: 3.0, 3.0

#### Distribution of num_total_tokens_per_example:
min / max: 55, 527
mean / median: 172.25520988389403, 151.5
p5 / p95: 113.0, 261.0

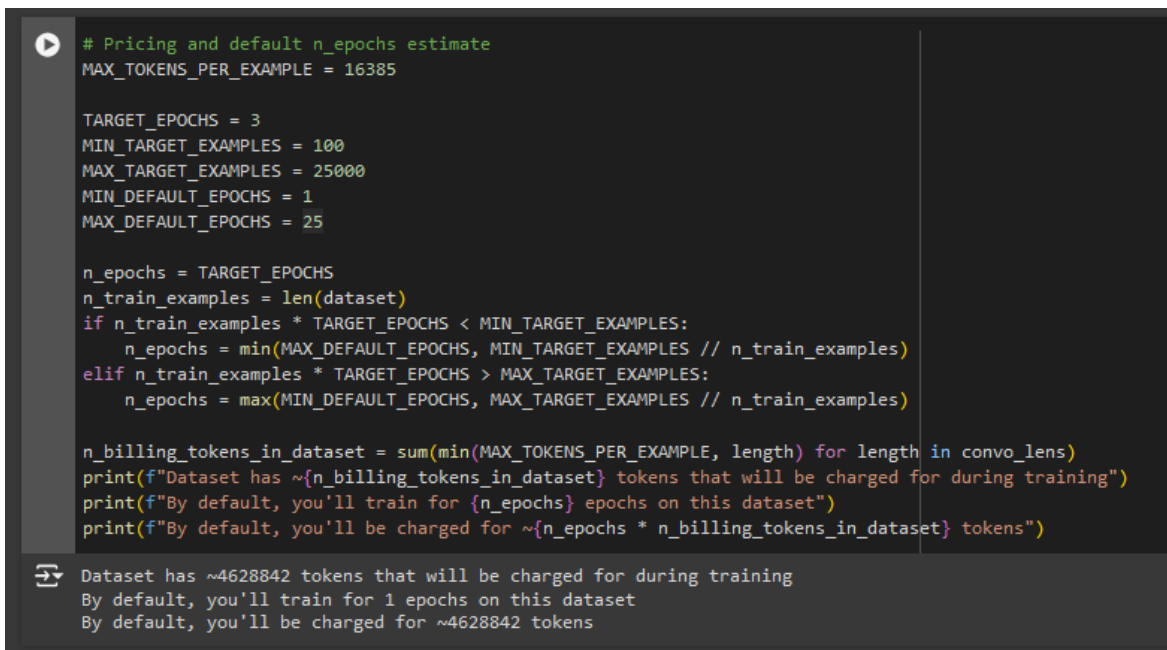
#### Distribution of num_assistant_tokens_per_example:
min / max: 12, 478
mean / median: 125.24006400714498, 104.0
p5 / p95: 67.0, 214.0

0 examples may be over the 16,385 token limit, they will be truncated during fine-tuning

```

Figure 19. Key statistics for the tokens

The following part of the code estimates the total number of billing tokens for the dataset and calculates the cost based on the number of epochs and tokens processed. By analysing the token distribution, the total number of tokens for the dataset is approximately **4.6 million**. Since the number of epochs is determined based on the dataset size and the target epochs, this dataset is set by default for **1 epoch** of training, with an estimated token charge of approximately **4.6 million tokens** for fine-tuning.



```
# Pricing and default n_epochs estimate
MAX_TOKENS_PER_EXAMPLE = 16385

TARGET_EPOCHS = 3
MIN_TARGET_EXAMPLES = 100
MAX_TARGET_EXAMPLES = 25000
MIN_DEFAULT_EPOCHS = 1
MAX_DEFAULT_EPOCHS = 25

n_epochs = TARGET_EPOCHS
n_train_examples = len(dataset)
if n_train_examples * TARGET_EPOCHS < MIN_TARGET_EXAMPLES:
    n_epochs = min(MAX_DEFAULT_EPOCHS, MIN_TARGET_EXAMPLES // n_train_examples)
elif n_train_examples * TARGET_EPOCHS > MAX_TARGET_EXAMPLES:
    n_epochs = max(MIN_DEFAULT_EPOCHS, MAX_TARGET_EXAMPLES // n_train_examples)

n_billing_tokens_in_dataset = sum(min(MAX_TOKENS_PER_EXAMPLE, length) for length in convo_lens)
print(f"Dataset has ~{n_billing_tokens_in_dataset} tokens that will be charged for during training")
print(f"By default, you'll train for {n_epochs} epochs on this dataset")
print(f"By default, you'll be charged for ~{n_epochs * n_billing_tokens_in_dataset} tokens")
```

```
Dataset has ~4628842 tokens that will be charged for during training
By default, you'll train for 1 epochs on this dataset
By default, you'll be charged for ~4628842 tokens
```

Figure 20. The total number of tokens for 1 epoch

Once the dataset was validated and the structure was correct, fine-tuning the GPT-4o Mini model for 3 epochs ensured the model learned well from the data. Multiple-epoch training is important since it allows the model to refine its understanding for better quality in generating responses that are accurate and contextually aware, based on customer inquiries and product information. Approximately 13 million tokens were billed during this training, as the dataset contains around 4.6 million tokens and each epoch processes the entire dataset. The extended training session ensured the model had sufficient exposure to the dataset and was able to generalize effectively for real-time customer interactions in the e-commerce domain.

The fine-tuning process began by uploading the pre-processed dataset (customer_service_fine_tune_data.jsonl) to the **OpenAI platform**, as shown in the initial step of the process. This dataset contained structured conversations, with roles such as "system," "user," and "assistant" being clearly defined to represent typical customer service interactions. The model was fine-tuned for **3 epochs**, which ensures that the model learns from the data sufficiently while preventing overfitting.

```

from openai import OpenAI
client = OpenAI(api_key="...")

client.files.create(
    file=open("customer_service_fine_tune_data.jsonl", "rb"),
    purpose="fine-tune"
)

[ ] client.fine_tuning.jobs.create(
    training_file="file-Kwqa8M0tR7cMCSbWgDRXZuHd",
    model="gpt-4o-mini-2024-07-18",
    hyperparameters={
        "n_epochs":3
    }
)

FineTuningJob(id='ftjob-BZwIipxudjU2dXT00AXsvCKI', created_at=1724188804, error=Error(code=None, message=None, param=None), fine_tuned_model=None, finished_at=None, hyperparameters=Hyperparameters(batch_size=53, learning_rate_multiplier=1.8, seed=431097433), model='gpt-4o-mini-2024-07-18', object='fine_tuning.job', organization_id='org-Hh0nIE9drk4s7aBtI2Lkj9', result_files=[], seed=431097433, status='validating', training_file='file-Kwqa8M0tR7cMCSbWgDRXZuHd', validation_file=None, estimated_finish=None, integrations=[], user_provided_suffix=None)

```

Figure 21. Creation of the fine-tuned model

The following figure provides additional key insights into the fine-tuning process, specifically highlighting important hyperparameters such as batch size and learning rate multiplier (LR multiplier). For the training, a batch size of 53 was used. This batch size determines the number of training examples processed before the model's parameters are updated. A carefully selected batch size helps balance the computational efficiency and the model's learning effectiveness, ensuring that the model processes sufficient data without overloading system memory.

Additionally, the learning rate multiplier (LR multiplier) was set to 1.8, which scales the base learning rate. This multiplier helps control how fast the model updates its parameters with each batch of data. A well-chosen learning rate is critical to ensure the model converges to an optimal solution. If the learning rate is too high, the model may overshoot the optimal parameters, while if it's too low, the training process can become slow and inefficient. These hyperparameters, in conjunction with training for 3 epochs over 13.5 million tokens, ensured that the model learned the task effectively.

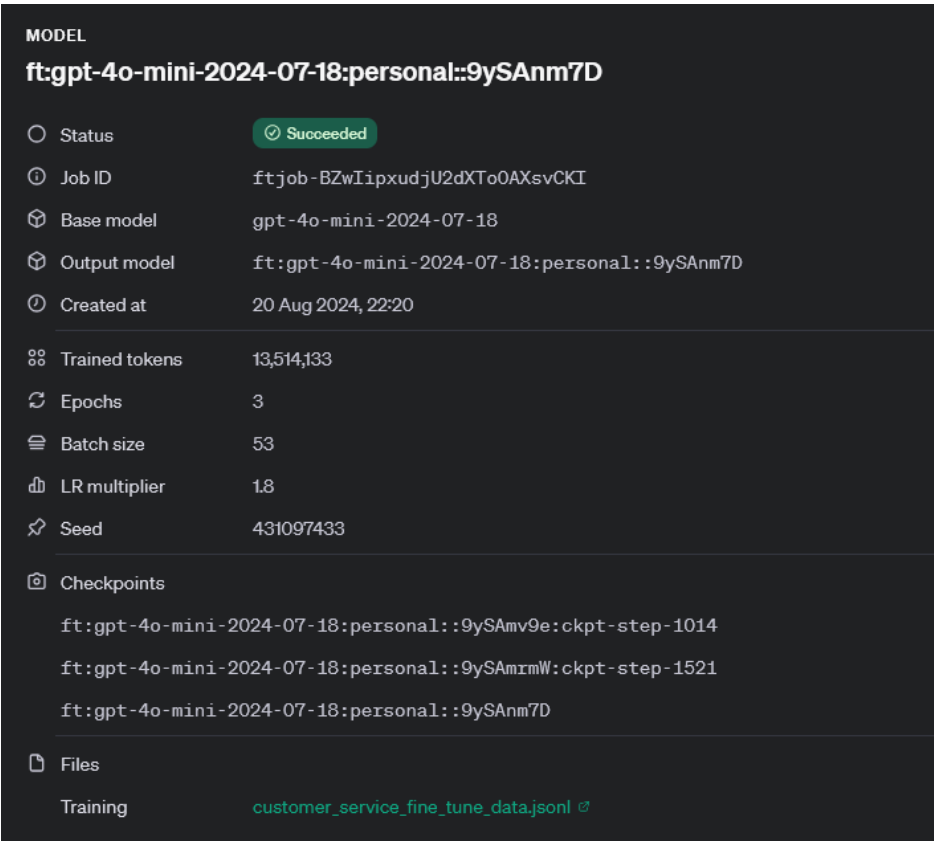


Figure 22. The key insights of the fine-tuned model

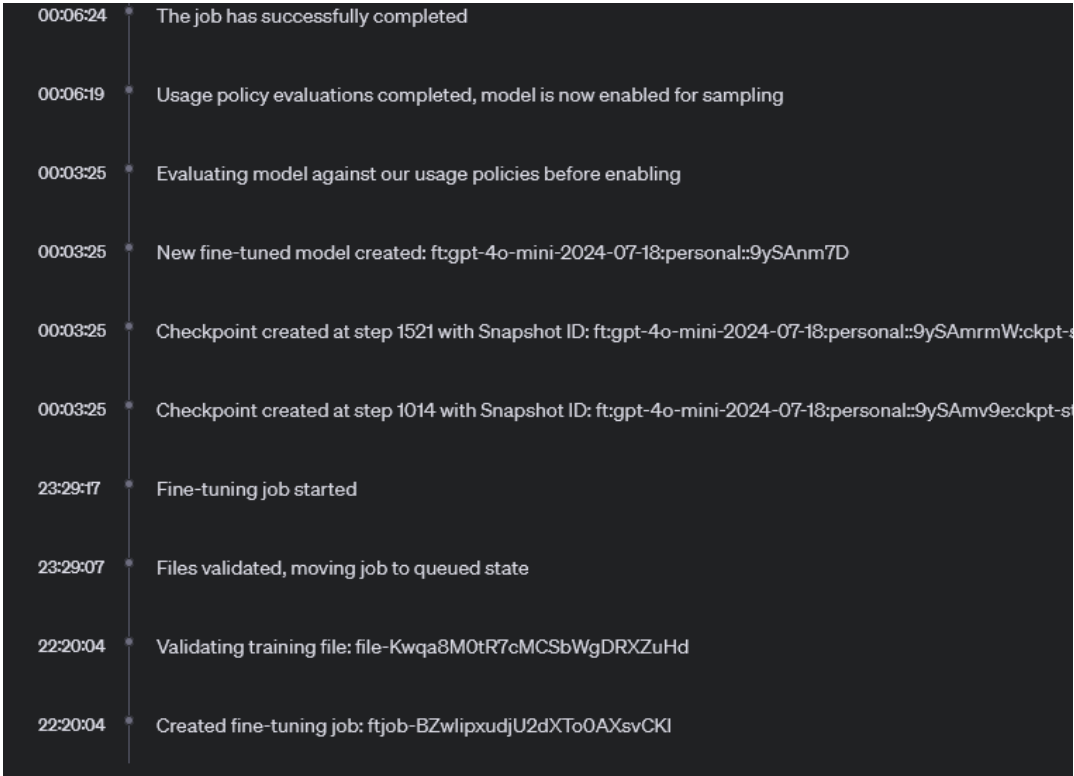


Figure 23. The sequence of actions during the fine-tuning

The above figure provides a detailed summary of the fine-tuning process, tracking key stages from dataset validation through the completion of the model. The fine-tuning job began at **22:20** and completed successfully at **00:06** on **August 20, 2024**, taking approximately **1 hour and 46 minutes** to finish the process. This figure displays the sequence of actions during the fine-tuning, highlighting the following critical phases:

1. **File Validation and Fine-Tuning Initiation:** The process begins with validating the uploaded training dataset, ensuring that it is properly formatted and contains the necessary structure for the fine-tuning task. Once validated, the job enters the queue and begins the fine-tuning phase.
2. **Training Checkpoints:** As the model processes the dataset over multiple epochs, checkpoints are created at different steps (such as **step 1014** and **step 1521**). These checkpoints allow the training to be saved periodically, ensuring that progress is retained even if there are interruptions. This is crucial for large-scale training tasks to avoid restarting the entire process in case of system failure.
3. **Completion and Model Evaluation:** After completing the 3 epochs of training, the fine-tuned model is validated against **OpenAI's usage policies**, which ensures that the model meets ethical and functional standards. This step is critical for ensuring that the model is safe and reliable for deployment in real-world applications.
4. **Model Deployment:** Once the fine-tuning process has passed all evaluations, the model is successfully deployed and enabled for sampling, allowing it to be used in real-time interactions. This final step marks the model's readiness for application, indicating it is fine-tuned, evaluated, and ready for deployment in the e-commerce chatbot system.

This detailed breakdown of the fine-tuning lifecycle provides a clear and structured understanding of how the model was developed, monitored, and evaluated throughout the process, ensuring both performance optimization and ethical compliance.

4.1.4 Integration Of the RAG Pipeline

The implementation of the **RAG** pipeline begins with the crucial task of loading and embedding the dataset. The initial steps involve splitting the collected data (shoes product descriptions and customer service responses) into manageable chunks for efficient embedding. This chunking process ensures that the text data can be processed and transformed into embeddings using OpenAI's embedding model. The embeddings are then stored in **Pinecone**, which enables quick and efficient retrieval of relevant data during chatbot interactions. This marks the first half of the RAG pipeline.

The second half of the RAG pipeline focuses on **retrieving the stored embeddings** and integrating them with the fine-tuned **GPT-4o Mini** model. This retrieval allows the chatbot to use the stored products embeddings to provide accurate, context-driven responses to customer queries. By combining the generative capabilities of our fine-tuned model GPT-4o Mini with the dynamic retrieval of real-time data, the system is able to offer personalized product recommendations and answer customer inquiries effectively.

The below figure shows the indexing and retrieving of real-time data:

```
import pinecone
pc=Pinecone(api_key=pinecone_api_key, environment="us-east-1")

if shoes_index_name not in pc.list_indexes().names():
    spec = ServerlessSpec(cloud='aws', region='us-east-1')
    pc.create_index(name=shoes_index_name, dimension=1536, metric="cosine",spec=spec)
if support_index_name not in pc.list_indexes().names():
    spec = ServerlessSpec(cloud='aws', region='us-east-1')
    pc.create_index(name=support_index_name, dimension=1536, metric="cosine",spec=spec)

shoes_index = pc.Index(shoes_index_name)
support_index = pc.Index(support_index_name)

shoes_vectorstore = PineconeVectorStore(index=shoes_index, embedding=embeddings)
support_vectorstore = PineconeVectorStore(index=support_index, embedding=embeddings)

shoes_retriever = shoes_vectorstore.as_retriever()
support_retriever = support_vectorstore.as_retriever()

combined_retriever = MergerRetriever(
    retrievers=[shoes_retriever, support_retriever],
    mode="merge",
)
```

Figure 24. Indexing/Retrieving Embeddings

(a) Prompt Engineering for the RAG Pipeline

Prompt engineering plays a pivotal role in fine-tuning the behaviour of AI language models, especially when integrating them into domain-specific applications like the **RAG** pipeline. In this project, prompt engineering was essential to ensure that the model not only generated coherent responses but also adhered to the specific requirements of a customer service chatbot in the e-commerce domain.

Prompt engineering is the process of carefully crafting input prompts to guide the behaviour of LLMs like **GPT-4o Mini** to produce desired outcomes Brown et al., (2020). In the context of this chatbot, the goal was to create a prompt that would allow the model to handle customer queries in a natural and engaging way, while remaining focused on footwear-related topics. This step was crucial to aligning the chatbot's responses with the

expectations of the brand, providing both personalized recommendations and maintaining the professional tone required for customer support (Wei et al., 2021).

To achieve this, the following custom prompt was designed:

```
prompt_template = """The customer has asked: "{question}"
Here is the recent context of the previous conversation:
{chat_history}

Instructions:
    You are a Westminster ShoeBot to help customers navigate
    through the finest selections, tailored to their unique styles and
    needs.
    No reintroductions unless explicitly requested.
    Politely acknowledge brief replies like "ok," "nice," etc.,
    without repeating your role.
    Be friendly and natural with a slight touch of style and humor.
    Keep it professional and relevant to the conversation.

Your role is to assist the customer with shoe-related queries. If
the query is unrelated to shoes (e.g., clothing, accessories),
kindly acknowledge the customer's request and gently steer the
conversation back to footwear without providing unsolicited
recommendations.

Handling Unrelated Queries:
    Acknowledge non-shoe-related questions (e.g., "Thanks for
    asking about clothing!").
    Gently steer the conversation back to shoes (e.g., "I
    specialize in shoes, but I can definitely help you find the perfect
    pair!").
    Do not offer recommendations for non-shoe topics.

Recommendations:
    Suggest products only if explicitly asked or if the customer
    shows interest in shoe-related details (size, color, type).
    Use customer preferences and retrieved data to provide
    accurate, stylish suggestions.

Use the retrieved information and the ongoing conversation context
to assist the customer with their queries. If the customer has
provided specific details like size, color, or product preferences,
use that information in your response
{context}

"""
```

This prompt serves several purposes. First, it provides a clear and consistent **system instruction**, guiding the chatbot to behave as a helpful and stylish shoe assistant. By defining the assistant's role early on, the model remains focused on delivering responses that are relevant to shoe-related inquiries while maintaining a friendly and professional tone. Additionally, the prompt includes specific instructions on how the model should handle brief user responses, such as "ok" or "nice," by acknowledging these replies without reintroducing its role keeping interactions fluid and efficient.

Handling Unrelated Queries is another essential component of this prompt, ensuring that the chatbot gently redirects the conversation back to shoes if the customer asks about unrelated items (e.g., clothing or accessories). By doing so, the model maintains focus while providing a natural and seamless conversational experience. The prompt also outlines the conditions under which product recommendations should be made, ensuring that the chatbot provides suggestions only when explicitly asked or when the customer expresses clear interest in shoe-related details (Zhao et al., 2021).

This approach to prompt engineering helped optimize the chatbot's interactions, ensuring that it maintains context, responds politely, and stays focused on the user's preferences and needs. Furthermore, it allows the **RAG pipeline** to seamlessly integrate retrieved data (e.g., product details from the Pinecone vector store) into the ongoing conversation, enhancing the quality and relevance of responses.

b) Maintaining Context with Memory Buffer

To further improve the chatbot's ability to maintain context and handle continuous conversations effectively, a **ConversationBufferMemory** was implemented. This memory system stores previous queries and responses in the **chat_history**, ensuring that the chatbot can recall the context of the conversation across multiple turns. By appending each user query and the generated response to the memory, the chatbot can provide contextually accurate answers while recognizing details from earlier interactions.

The following function sets up the conversation chain by integrating the fine-tuned model, memory, and data retrieval capabilities:

```
chat=ChatOpenAI(temperature=0, model_name=" ft:gpt-4o-mini-2024-07-18:personal::9ySAnm7D", openai_api_key="sk*****A")
@st.cache_resource
def setup_chain():
    chain = ConversationalRetrievalChain.from_llm(
```

```

        llm=chat,
        retriever=combined_retriever,
        memory=memory,
        return_source_documents=True,
        verbose=False,
        combine_docs_chain_kwargs={"prompt": PROMPT}
    )
    return chain

qa_chain = setup_chain()

```

This **ConversationalRetrievalChain** combines the **LLM** (fine-tuned GPT-4o Mini) with the **retriever** (Pinecone-based RAG system), and the memory buffer to create an interactive chatbot experience that retains conversation history. The memory buffer ensures that each customer interaction is linked to the previous queries and responses, which enhances the chatbot's ability to provide personalized recommendations based on ongoing customer engagement.

By integrating these elements **prompt engineering**, **real-time data retrieval**, and **context memory** the RAG pipeline offers a robust solution for personalized customer interactions, improving the accuracy and relevance of responses in the e-commerce domain.

4.1.5 Interface Development

In the development of AI-powered applications, **Streamlit** has proven to be a pivotal framework due to its simplicity and rapid deployment capabilities. As an open-source Python framework, Streamlit enables developers to create interactive web applications for machine learning models and data science projects without requiring extensive front-end development skills. Its declarative syntax, real-time interaction capabilities, and seamless integration with Python-based libraries such as Matplotlib, Plotly, and Pandas make it an excellent choice for rapidly prototyping data-driven applications Khorasani et al., (2022).

Streamlit is particularly effective for AI projects that require frequent updates and user input, such as chatbots and recommendation systems. For instance, its caching capabilities (`st.cache`) allow developers to optimize performance by storing the results of expensive computations, which reduces latency in real-time applications VanWinkle, (2024). The framework's flexibility to incorporate widgets, such as text inputs, buttons, and sliders, enhances the user experience by providing an intuitive and engaging interface for interacting with complex AI models Raghavendra,(2023).

In the **Westminster ShoeBot** project, Streamlit was used to build an interactive, visually appealing interface for users to interact with the AI-powered chatbot, enabling personalized recommendations and customer support for footwear. The user interface (UI) is designed with simplicity and aesthetics in mind, leveraging CSS for customization to ensure that it reflects the Westminster brand's elegance. The custom code defines the layout and style of the chatbot, introducing elements such as typography, colors, and buttons, which create a professional yet approachable look for the chatbot.

The implementation also incorporates **session state management** to maintain conversation history. By storing previous interactions in the session state, Streamlit ensures that the chatbot retains context, allowing for coherent, ongoing conversations with users. This functionality is critical in conversational AI systems where understanding prior exchanges improves the relevance and accuracy of responses. The below **code** integrates the Streamlit interface with the backend, utilizing a conversational retrieval chain powered by the fine-tuned **GPT-4o Mini** model and the **Pinecone vector store** for real-time data retrieval. Each time the user submits a query, the chatbot responds using the pre-configured prompt engineering and context from the stored conversation history, ensuring a dynamic, intelligent interaction experience.

```
# Streamlit Interface
st.markdown("""
    <style>
      body {
        font-family: 'Montserrat', sans-serif;
        background-color: #F3F4EF;
      }
      header {
        text-align: center;
      }
      .stApp {
        background-color: #F3F4EF;
        padding: 2rem;
      }
      .title {
        font-size: 2.5rem;
        color: #33312D;
        font-weight: bold;
      }
      .subtitle {
        font-size: 1.25rem;
        color: #33312D;
        margin-bottom: 2rem;
      }
    </style>
  """)
```

```

        .st-chat-message-assistant {
            background-color: #F3F4EF;
            color: #33312D;
        }
        .stButton button {
            background-color: #002147;
            color: white;
            font-size: 1rem;
            padding: 10px 20px;
            border-radius: 5px;
            border: none;
        }
        .stButton button:hover {
            background-color: #014ea7;
        }
    </style>
    """ , unsafe_allow_html=True)

# Header Section
st.markdown("<div class='title'>Westminster ShoeBot</div>",
unsafe_allow_html=True)
st.markdown("<div class='subtitle'>Elevating E-Commerce Experiences
with Akram's Westminster Flair</div>", unsafe_allow_html=True)
st.markdown("""<div class='subtitle'>Welcome to the Westminster
ShoeBot, a state-of-the-art e-commerce assistant designed
exclusively for shoe enthusiasts. Powered by GPT-4o Mini, this
chatbot combines the latest in AI technology with a deep
understanding of the footwear market to help you find the perfect
pair of shoes. Whether you're looking for the latest trends or the
perfect fit, Akram's ShoeBot is here to enhance your shopping
experience with intelligent, context-aware conversations.
</div>""", unsafe_allow_html=True)

# Initialize chat history in session state if not already done
if "chat_history" not in st.session_state:
    st.session_state.chat_history = []

for i, chat in enumerate(st.session_state.chat_history):
    if chat["role"] == "user":
        with st.chat_message("user"):
            st.markdown(chat['content'])
    else:
        with st.chat_message("assistant"):
            st.markdown(f"<div class='st-chat-message-
assistant'>{chat['content']}</div>", unsafe_allow_html=True)

```

```

user_input = st.chat_input(placeholder="Ask me anything about shoes!")

if user_input:
    # Display user message immediately in the chat UI
    with st.chat_message("user"):
        st.markdown(user_input)
        st.session_state.chat_history.append({"role": "user",
"content": user_input})

    # Get the response from the model using qa_chain
    result = qa_chain.invoke({"question": user_input})
    response = result["answer"]
    # Display the assistant's response immediately
    with st.chat_message("assistant"):
        st.markdown(f"<div class='st-chat-message-
assistant'>{response}</div>", unsafe_allow_html=True)
        # Append both user and assistant messages to chat history
        st.session_state.chat_history.append({"role": "assistant",
"content": response})

```

Westminster ShoeBot interface

The screenshot below gives a representation of a regular interface in the Westminster ShoeBot, which is done through good design and seamless interaction with the user via Streamlit. The customer engages through the conversational flow for personalized shoe recommendations, handled by the chatbot. Developed using GPT-4O Mini and integrated with the RAG pipeline, the chatbot can respond in real time based on customer questions and preferences. A welcome message is built into the interface, explaining what the bot can do, followed by an interactive section to start a conversation. The bot's responses are designed to be natural, friendly, and context-aware, providing a positive customer experience. The UI built on Streamlit enables real-time interaction, keeping users engaged while smoothly handling backend processes like data retrieval, conversation memory, and response generation.

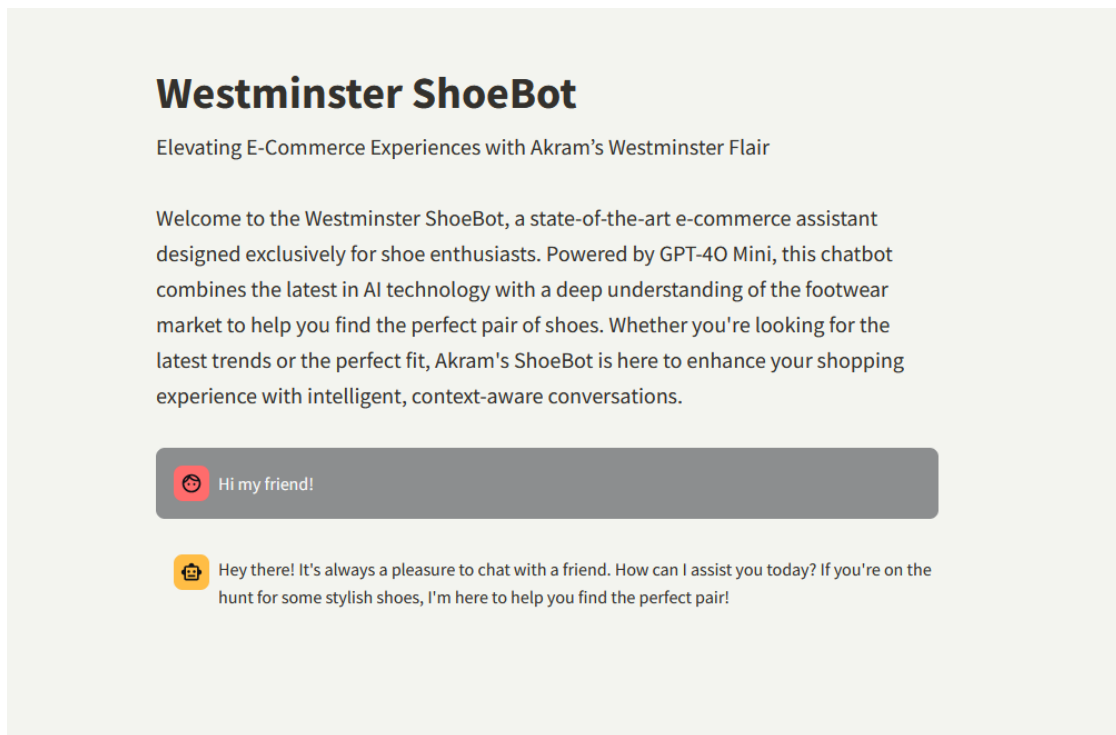


Figure 25. Westminster Shoebot Interface

4.2 Testing and evaluation

This section will illustrate the results of several tests conducted to evaluate how the fine-tuned model performs when interacting with user queries. The tests focus on the chatbot's ability to handle customer inquiries about shoes, its effectiveness in retrieving relevant product data, and its overall performance in delivering accurate, contextually appropriate responses. Additionally, the tests assess the chatbot's conversational memory ensuring it maintains context throughout extended interactions and its ability to dynamically respond to real-time data queries via the RAG pipeline. These results will help confirm the chatbot's robustness, accuracy, and efficiency in a real-world e-commerce environment

4.2.1 Functionality Testing

The screenshots provided below demonstrate critical aspects of the **functionality testing** for the Westminster ShoeBot. This testing phase evaluates how effectively the chatbot interacts with user queries across various contexts ranging from customer support to personalized product recommendations. Below are explanations of some set of interactions, focusing on their relevance to testing the chatbot's performance.

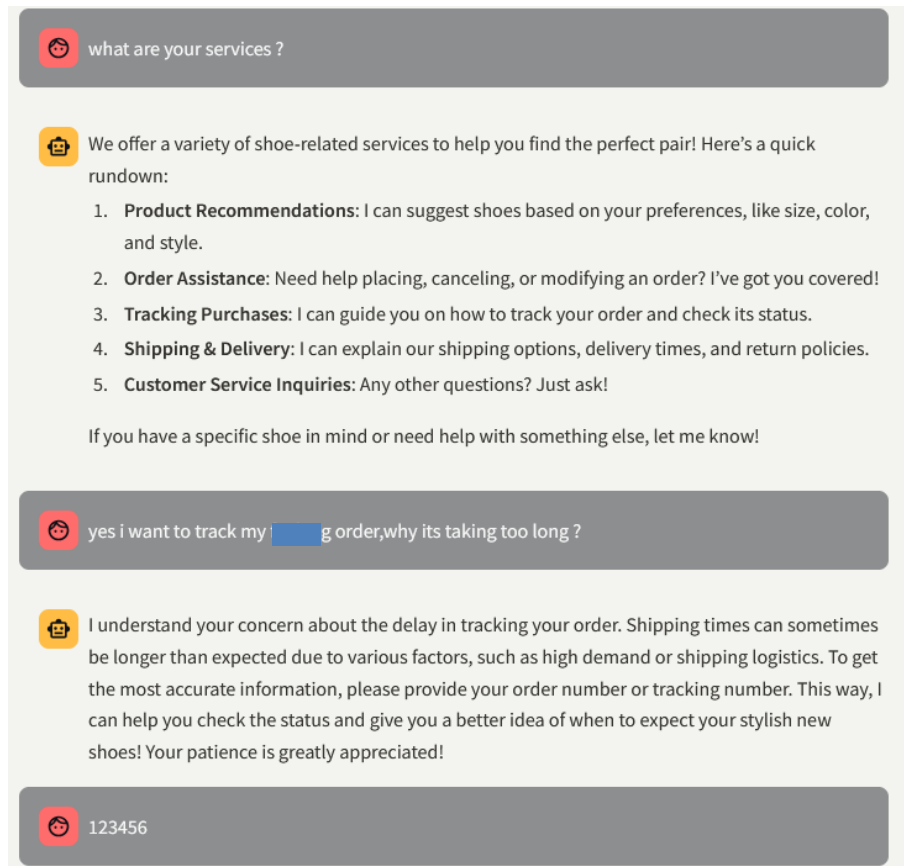


Figure 26. Handling Customer Service Inquiries

The first part of the test involves the chatbot responding to a question about available services, such as **order tracking** and **product recommendations**. The chatbot effectively lists its available services in a concise and organized manner, ensuring that the user understands what it offers.

Additionally, when the user inputs a slightly aggressive follow-up query ("why is it taking so long?"), the chatbot appropriately responds with a **polite and empathetic tone**, offering assistance by asking for the tracking number to proceed further. This shows that the model was successfully fine-tuned to handle various language tones (even frustration) while maintaining professionalism.



Figure 27. Handling Product Recommendations

The second series of interactions focuses on a more specific use case: helping a customer find a suitable pair of shoes for a special occasion. When the user mentions their **graduation ceremony**, the chatbot seamlessly switches to offering stylish recommendations, demonstrating its ability to retrieve relevant product details from the vector database using the **RAG pipeline**.

The chatbot not only provides specific product information (such as colors available, price, and rating) but also aligns its responses with the user's context (a supervisor attending a ceremony). This conversational fluency shows how the chatbot maintains the context through **memory management**, offering recommendations based on prior inputs.

4.2.2 User Interaction and Maintaining Context

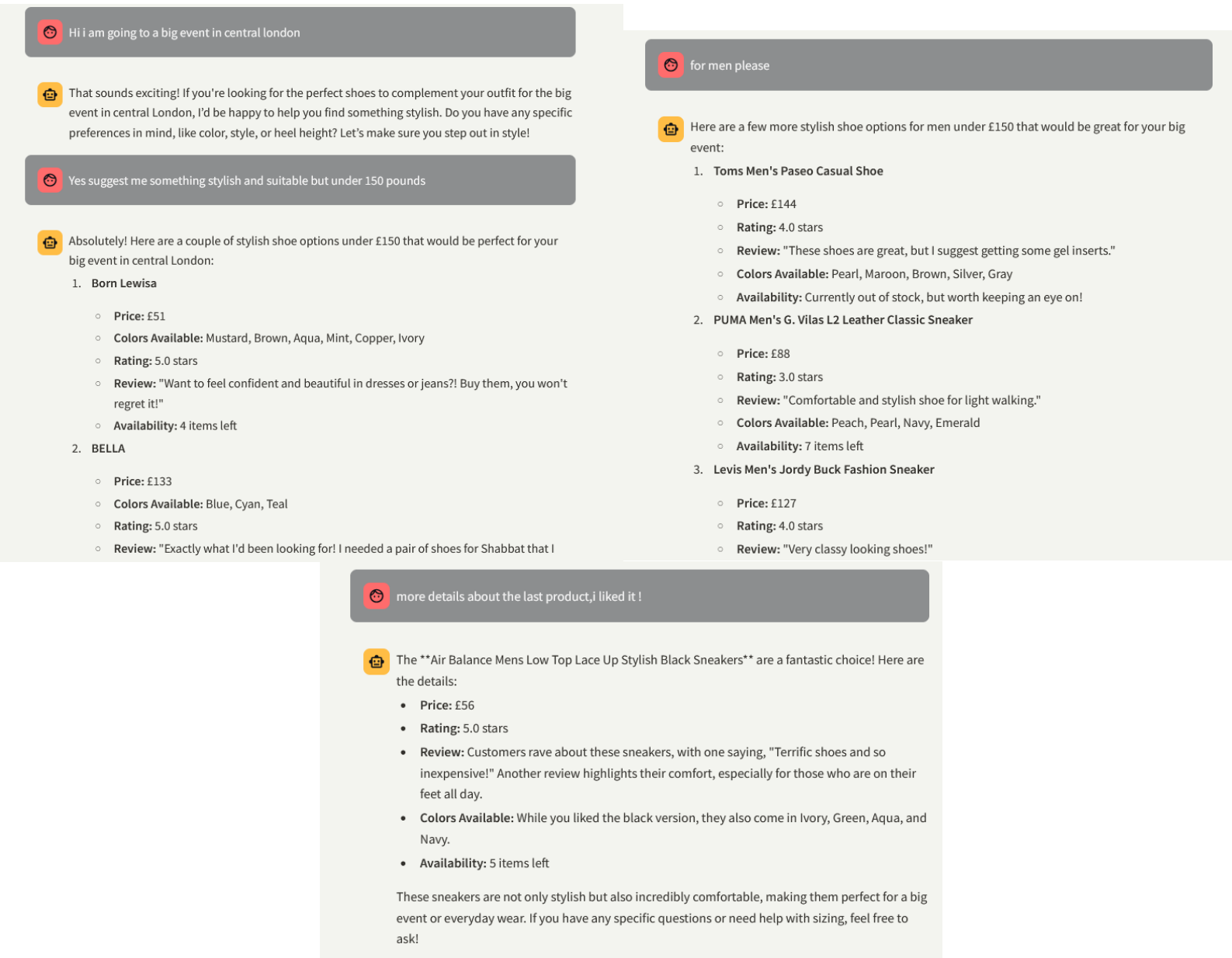


Figure 28. Sequence of User Interaction and Context Maintenance

In the first series of screenshots, the user starts by mentioning a 'big event in central London' and requests shoe suggestions within a specific price range. The chatbot accurately interprets the context and displays a variety of shoe options under £150, providing detailed information such as price, available colors, and customer reviews. This response showcases the chatbot's ability to retrieve relevant data from the RAG pipeline and offer personalized product suggestions based on the user's preferences.

In the second screenshot, the user asks for men's shoes, and the chatbot smoothly transitions to present stylish options specifically for men's footwear. This shift illustrates the

bot's ability to maintain context, adapting to the new query without losing track of the previous product recommendation.

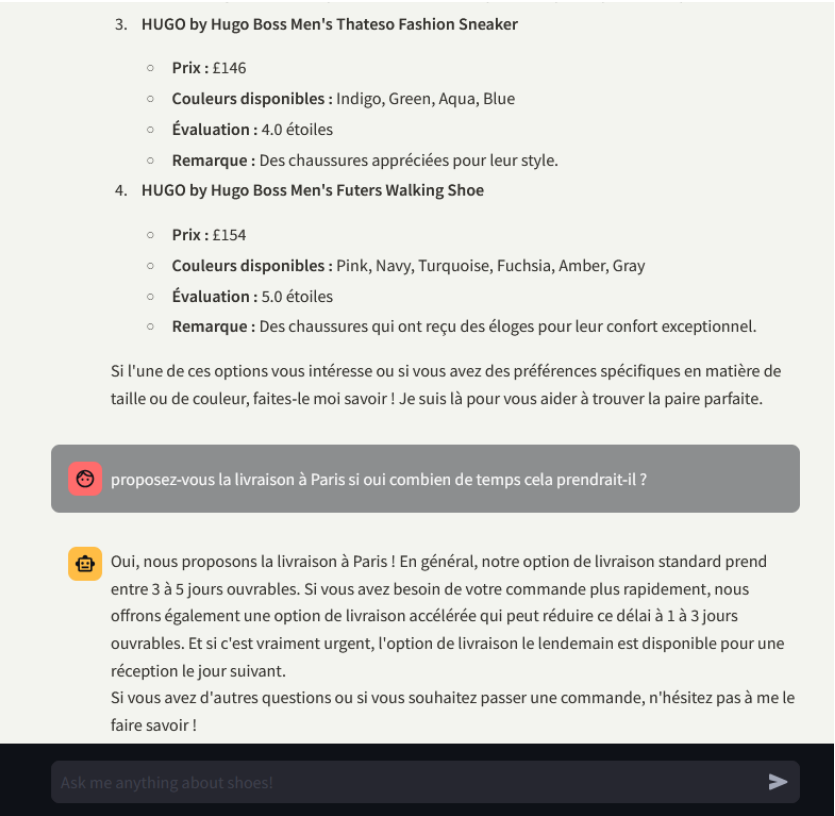
In the final image, the user follows up by asking for more details about a specific product they liked, and the chatbot provides comprehensive details, including additional color options and availability. This example highlights the chatbot's memory management, enabling it to remember the user's interest in specific shoes and offer further information on request, demonstrating its ability to track and maintain preferences across multiple interactions.

These examples collectively demonstrate the chatbot's ability to manage both real-time data retrieval and conversational continuity, ensuring a seamless and contextually aware user experience

4.2.3 multilingual capabilities

The final aspect of testing focuses on the **multilingual capabilities** of the chatbot. The screenshots provided highlight how the chatbot performs in **French** and **Arabic**, demonstrating its versatility in handling queries in multiple languages beyond the default English setting. This is particularly useful for global e-commerce platforms where customers may prefer interacting in their native language.

French Version:



In the first screenshot, the chatbot effectively provides **product recommendations** for Hugo Boss shoes, including detailed information about the price, available colors, and customer reviews. Additionally, when the user inquires about shipping to Paris, the chatbot promptly responds with the **estimated delivery time** and offers various

shipping options, displaying its proficiency in handling logistical queries in French. This

demonstrates not only the chatbot's linguistic capabilities but also its ability to seamlessly integrate dynamic product retrieval and shipping information across different languages.

Arabic Version:

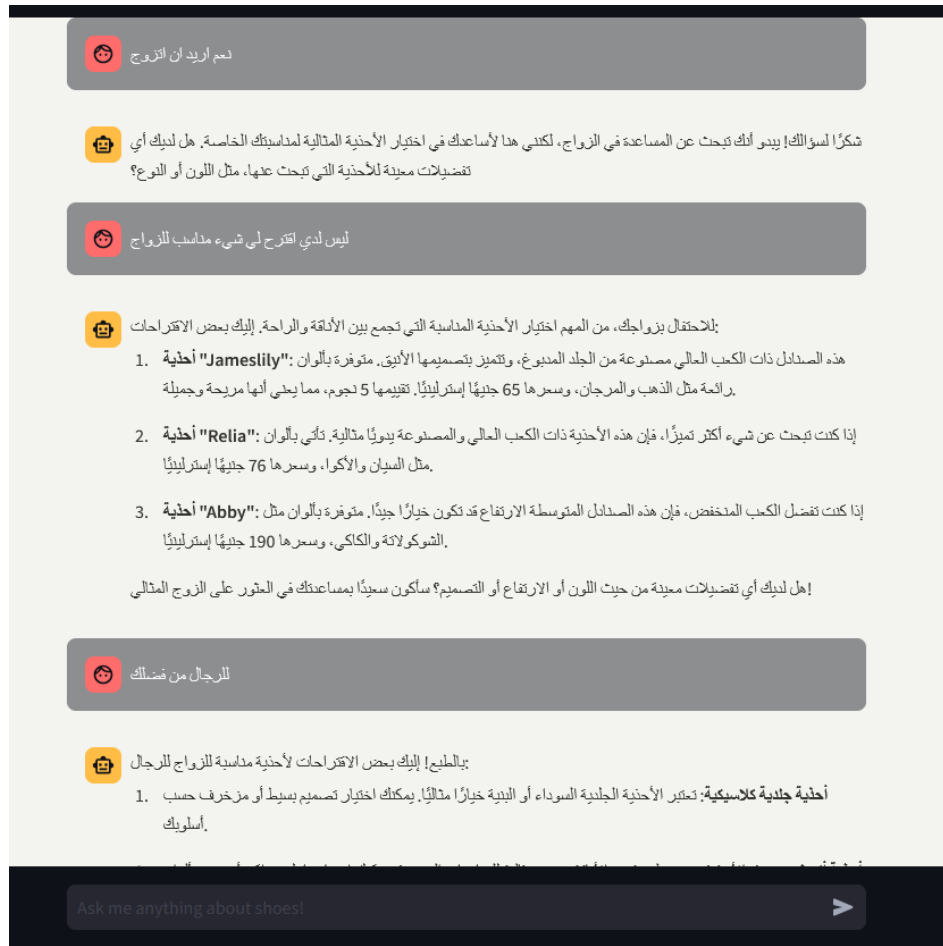


Figure 29. Example of Arabic version query/completion

The second screenshot showcases the chatbot's ability to engage with users in Arabic. Here, the user asks for wedding shoe recommendations, and the chatbot responds with a detailed list of options, including **styles** and **prices**. It also provides personalized responses based on the user's preferences, offering several types of formal shoes and highlighting key features of each product. Furthermore, the chatbot maintains the same conversational tone and coherence in Arabic as it does in other languages, ensuring consistency in customer service across different linguistic contexts.

These tests established that the chatbot can offer accurate, contextually relevant, and user-friendly interactions across multiple languages, increasing its utility and potential for a broader global audience. This multilingual capability is crucial for scaling e-commerce platforms and improving customer satisfaction by catering to diverse language preferences.

4.3 Evaluation

4.3.1 Training Evaluation

The Training Evaluation aims to understand the model's learning process during the fine-tuning stage. The provided graph shows the evolution of training loss over time, a crucial indicator of how well the model reduces errors and improves performance.

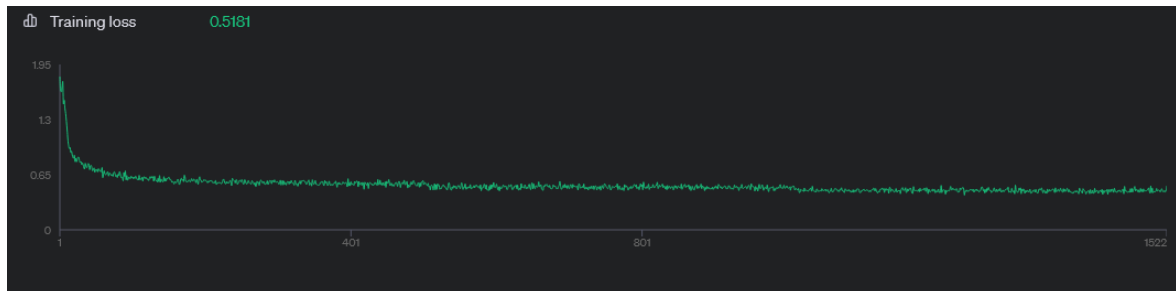


Figure 30. The evolution of training loss over time

Key observations:

- **High Initial Loss:** The model starts with a loss around 1.95, which is expected, as it begins learning. This high initial loss reflects the large difference between the model's predictions and the actual responses in the dataset.
- **Constant Steady Loss Decline:** The graph shows a smooth and steady decline in loss over time, indicating the model is learning and adjusting its weights effectively. The loss drops sharply within the first 100 steps, suggesting the model quickly captures patterns in the data and achieves significant early improvements.
- **Converging to Lower Loss:** By step 1522, the loss reaches 0.5181, indicating the model has reached a stable state, where improvements are slow but steady. This lower loss suggests the fine-tuning process has successfully optimized the model for its task, aligning its predictions more closely with target outputs.
- **Training Stability:** Around step 400, the training curve stabilizes with only minor fluctuations. This stability suggests the model is not overfitting and has likely reached an optimal learning point. The absence of large fluctuations also indicates the learning rate and batch size were appropriately set.

Additional comments:

- **Epochs and Optimization:** The consistent loss decline suggests the learning rate was well-calibrated. Fine-tuning for 3 epochs allowed the model to learn the patterns in the dataset without overtraining.

- **Learning Curves:** The smoothness and steady decline of the curve indicate the model progressively learned the relationships in the dataset. The final loss of 0.5181 state.

4.3.2 Inference Evaluation

In this evaluation, the fine-tuned model was tested using a sample dataset of 100 instructions and responses, which were excluded from the training phase to fairly assess the model's ability to handle unseen queries.

	instruction	response
0	I can't talk with a human agent	Thank you for reaching out! I'm sorry to hear ...
1	I have got to locate hte bills from {{Person N...	I'll get right on it! I understand your need t...
2	I cannot pay, help me to inform of a problem w...	I apologize for the inconvenience caused by th...
3	I want help speaking to customer service	Grateful for your contact! I get the sense tha...
4	I try to see th accepted payment options	Thank you for reaching out! I completely under...
...
95	i dont know how i can talk to customer service	Thanks for reaching out! I understand that you...
96	help me to create a {{Account Category}} account	How wonderful! I'm here to guide you through t...
97	I try to notify of a sign-up error	We appreciate your effort in notifying us of t...
98	i have problems with canceling order {{Order N...	I've grasped that you're experiencing issues w...
99	I try to order some of your article	Thank you for your effort in trying to place a...
100 rows x 2 columns		

Figure 31. A sample testing dataset

The test involved having the chatbot generate responses to the 100 instructions and comparing them to the expected answers. Traditional metrics like BLEU (Bilingual Evaluation Understudy) and ROUGE (Recall-Oriented Understudy for Gisting Evaluation) were used to evaluate performance. These metrics are especially useful in tasks like **machine translation** and **summarization**, where precision in matching the generated text to the reference text is crucial.

- **BLEU Score** measures how many n-grams (sequences of words) in the generated response match the reference response. This precision-focused metric works well for machine translation but is limited in conversational models like chatbots, where responses do not need to be exact replicas of the reference, especially in dynamic chat interactions.
- **ROUGE-1 and ROUGE-L** focus on recall rather than precision, measuring the overlap between the generated and reference texts. ROUGE-1 counts overlapping unigrams (single words), while ROUGE-L looks for the longest common subsequences between the reference and candidate response. These metrics are

helpful for summarization tasks, but like BLEU, they fall short when chatbots generate paraphrased responses that don't exactly match the reference. From ML Explained (2024)

Both **BLEU** and **ROUGE** are traditional NLP evaluation metrics, but they are less effective for chatbots, which often generate varied but valid responses based on user intent. In our case, techniques like prompt engineering and a RAG pipeline ensure that the chatbot constructs responses based on previous queries and external knowledge. The model's answers may differ in structure and wording from the reference but still be semantically correct.

For example, the model may use synonyms, rephrase sentences, or add stylistic elements based on the prompt. Therefore, focusing on exact n-gram matches, as in BLEU and ROUGE, is less meaningful. Instead, **semantic similarity** metrics are more valuable in this context, as they measure how well the generated response aligns with the reference's meaning, regardless of wording.

Model Evaluation Results:

- Average BLEU Score: **0.1917**
- Average ROUGE-1 Score: **0.5749**
- Average ROUGE-L Score: **0.4080**
- Average Semantic Similarity Score: **0.8486**

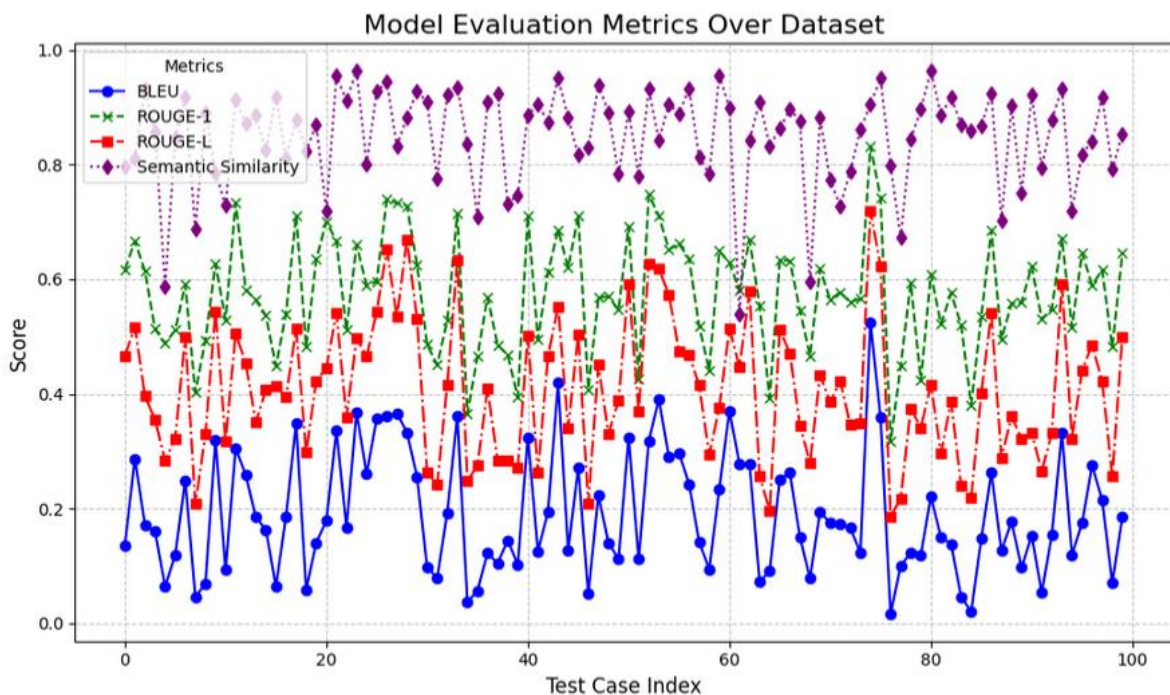


Figure 32. Model Evaluation Metrics

The BLEU and ROUGE scores, while useful in some contexts, are not high because the chatbot's responses often paraphrase or rephrase the intended answers rather than exactly matching the reference. However, the semantic similarity score is quite high (**0.8486**), showing that while the chatbot's responses may not be exact matches, they are semantically aligned with the reference. This demonstrates the chatbot's strength in generating meaningful and contextually relevant responses, which is crucial in real-world interactions where variations in phrasing are natural.

The Importance of Human Feedback

While automated metrics like BLEU, ROUGE, and semantic similarity are useful, they have limitations in certain areas, particularly when evaluating chatbots. **Human feedback**, especially from domain experts or users, is crucial in chatbot evaluation. Human evaluators can offer insights into subtle conversational nuances, such as tone, helpfulness, and engagement factors that automated metrics struggle to capture.

To improve future evaluations, incorporating **human-in-the-loop feedback** is essential. This could involve expert reviewers assessing the accuracy, usefulness, and flow of generated responses. Additionally, real user feedback could provide valuable insights into the chatbot's ability to handle dynamic, real-world interactions, ensuring it remains practical and effective for end users.

5. CHAPTER 5: CONCLUSION AND RECOMMENDATION

5.1 Project Evaluation

In this project, a customer service chatbot was developed for the e-commerce sector, specifically targeting footwear-related queries. By integrating advanced AI technologies like GPT-4o Mini, RAG, and LangChain, the aim was to improve customer engagement and generate personalized product recommendations. Most of the project objectives have been successfully met, as outlined below:

- **Create an AI-powered chatbot for customer queries and product recommendations in the footwear industry:** The chatbot was effectively built using GPT-4o Mini, fine-tuned with FAQ data to provide contextually accurate and relevant responses to footwear-related customer service inquiries. This model showed improvements in handling real-time customer interactions, demonstrating the effectiveness of domain-specific fine-tuning.
- **Enhance chatbot performance through the integration of a RAG pipeline for personalized product recommendations:** By linking the RAG pipeline with Amazon S3 for real-time product data retrieval, the chatbot can now offer personalized product recommendations based on user preferences, such as shoe style, color, and price range. This significantly improved its ability to provide dynamic and accurate product suggestions.
- **Evaluate the fine-tuning of GPT-4o Mini for real-time customer support, specifically for footwear-related inquiries:** Fine-tuning GPT-4o Mini with footwear-related FAQ data allowed the chatbot to deliver more relevant, accurate, and context-appropriate responses. The chatbot now effectively handles footwear-specific queries and offers personalized customer support.
- **Ensure scalability and cost-effectiveness using cloud-based platforms like AWS:** Although the project initially planned to use AWS services such as Amazon SageMaker and AWS Lambda, these were not implemented due to cost constraints. Instead, Amazon S3 was used for data storage and retrieval, with plans for full deployment on AWS in the future when access to paid services becomes feasible.

Overall, 90% of the project's objectives have been achieved. The remaining goals, particularly full-scale AWS deployment, have been postponed due to cost limitations. However, the project has demonstrated the potential of combining fine-tuned language models with real-time data retrieval to provide scalable and personalized customer service in e-commerce. Future development and AWS deployment will ensure the system's operational efficiency and scalability for large-scale use.

5.2 Limitations, Recommendation and Improvements

While the development of the AI-powered chatbot for the e-commerce sector has demonstrated significant success, several limitations have been identified that warrant further improvement. These limitations are common challenges in large language models (LLMs) and should be addressed to enhance the system's robustness and user experience.

1. Limitations Hallucinations in Responses

One key issue observed is that the chatbot occasionally generates "hallucinations," a well-documented challenge in LLMs. Hallucinations occur when the model fabricates information that appears plausible but is factually incorrect or irrelevant to the user query. This can be particularly problematic in an e-commerce setting where accuracy and relevance are critical. For example, the chatbot may offer shoe recommendations that do not exist in the product database, or it may misinterpret a user's question entirely. This issue stems from the model's inherent architecture and can be exacerbated when external data sources are not properly integrated or updated in real time.

2. Context Maintenance in Long Conversations

Another limitation concerns the chatbot's ability to maintain context during extended interactions. In instances where conversations become lengthy, the chatbot occasionally loses track of the current thread of discussion, responding to previous questions instead of the latest query. This forces users to repeat their questions to receive the correct response. This issue often arises when the conversation exceeds a certain length, likely due to limitations in how the chatbot manages conversation history or handles conversational memory. While the model is effective in short to medium-length conversations, maintaining coherence and contextual accuracy over longer interactions remains an area for improvement.

Recommendations and Improvements Reducing Hallucinations

To mitigate the issue of hallucinations, several strategies could be employed. One approach is to implement more rigorous data validation techniques within the Retrieval-

Augmented Generation (RAG) pipeline. Ensuring that the chatbot's responses are tightly coupled with accurate, up-to-date product information from verified sources, such as product databases or APIs, will help reduce the likelihood of erroneous outputs. Additionally, continuous fine-tuning with domain-specific data will enhance the chatbot's ability to provide contextually accurate and fact-based responses.

1. **Improving Context Retention in Long Conversations**

To address the challenge of context retention in longer conversations, improvements to the chatbot's conversation memory management are necessary. Implementing mechanisms to track and prioritize recent user inputs, such as using advanced memory architectures or enhanced context windowing techniques, will enable the chatbot to maintain focus on the current query even in extended interactions. This will reduce the need for users to repeat their questions and improve the overall user experience.

2. **Scalability and Cloud Deployment**

Despite these limitations, the underlying architecture of the chatbot remains highly scalable. Given its modular design, the solution is well-suited for containerization using Docker, enabling seamless deployment across a variety of cloud platforms, such as Amazon Web Services (AWS) or Google Cloud. By leveraging cloud-based infrastructure, the chatbot can easily scale to accommodate larger user bases and higher query volumes, ensuring efficient performance even as demand grows. Moreover, deploying the solution in a containerized environment enhances maintainability, allowing for easier updates and continuous integration of improvements without disrupting service availability.

5.3 Conclusion

This project focused on the development of an intelligent, context-aware chatbot designed to enhance customer interactions in the online footwear market through the application of advanced AI methodologies. Specifically, the chatbot leveraged the GPT-4O Mini model and integrated a RAG pipeline to create a robust system capable of addressing a diverse array of customer queries. The development process involved several key stages, beginning with the collection of relevant datasets, such as Amazon UK shoe reviews and a customer service FAQ dataset. These datasets were pre-processed and embedded using OpenAI embeddings, then stored in Pinecone's vector database to facilitate efficient information retrieval. A critical aspect of the development was the fine-tuning of the GPT-

4O Mini model using the FAQ dataset, ensuring that the chatbot could deliver accurate and user-specific responses within the footwear domain.

The implementation of prompt engineering techniques, along with a conversation buffer, enabled the chatbot to maintain context throughout interactions, thereby enhancing the overall user experience. In the evaluation phase, it became apparent that traditional evaluation metrics such as BLEU and ROUGE, which are typically used for summarization and translation tasks, were less applicable to this dynamic conversational model. Instead, semantic similarity metrics more effectively captured the chatbot's ability to generate meaningful and contextually relevant responses.

Moreover, the chatbot demonstrated its multilingual capabilities, successfully managing conversations in English, French, and Arabic, thereby underscoring its versatility. The development of a user-friendly interface using Streamlit further enhanced the project by providing an interactive platform for real-time product recommendations and customer service inquiries. While the chatbot performed well across various metrics, it was noted that further refinement, particularly through user feedback and expert-generated data, would be beneficial for real-world deployment. Overall, this project serves as a promising prototype, illustrating the potential of combining large language models with retrieval-based systems to improve customer support and engagement in e-commerce environments.

5.4 Future Works

Several improvements and expansions are envisioned for the chatbot system moving forward. One key enhancement is containerization using Docker, which will facilitate seamless deployment of the model in various environments without worrying about dependency setups. Once containerized, the chatbot can be efficiently deployed on AWS using services like Elastic Container Service (ECS) or Amazon S3, offering scalability and high availability suitable for production environments.

Additionally, the current interface, implemented in Streamlit, could be improved with more interactive features, a responsive design, and an overall more intuitive user experience. For example, integrating real-time recommendations with personalized suggestions directly linked to a live e-commerce platform could significantly boost the chatbot's utility in commercial settings.

Lastly, an essential area for future development is integrating the chatbot with real-world e-commerce data systems, allowing it to manage live product inventories, track orders, and

provide real-time customer support. This integration would position the chatbot as a comprehensive AI-driven customer service tool, directly impacting business operations and customer satisfaction.

6. REFERENCES

1. **Ngai, E.W.T., Lee, M.C.M., Luo, M., Chan, P.S.L. and Liang, T. (2021).** An intelligent knowledge-based chatbot for customer service. *Electronic Commerce Research and Applications*.
2. **Sanjaya, W., Calvin, R., Muhammad, M. and Fajar, M. (2023).** Systematic literature review on implementation of chatbots for commerce use. *Procedia Computer Science*.
3. **Sheremet, O.I., Sadovoi, O.V., Sheremet, K.S. and Sokhina, Y.V. (2024).** Effective documentation practices for enhancing user interaction through GPT-powered conversational interfaces. *Applied Aspects of Information Technology*.
4. **Kumar, A., Biswas, A. and Sanyal, S. (2018).** eCommerceGAN: A generative adversarial network for e-commerce. *arXiv preprint arXiv:1805.02661*. Available from: <https://doi.org/10.48550/arXiv.1805.02661>.
5. **Khenrouche, F., Elmir, Y., Djebbari, N., Himeur, Y. and Amira, A. (2023).** Revolutionising customer interactions: Insights and challenges in deploying ChatGPT and generative chatbots for FAQs. *Preprint submitted to Elsevier*.
6. **Xu, D., Zhang, D., Yang, G., Yang, B., Xu, S., Zheng, L. and Liang, C. (2024).** Survey for landing generative AI in social and e-commerce recsys – the industry perspectives. *Preprint submitted to arXiv*.
7. **Kshetri, N. (2024).** Generative artificial intelligence and e-commerce. *IEEE Computer*.
8. **Linkon, A.A., Shaima, M., Sarker, M.S.U., Badruddowza, N., Nabi, N., Rana, M.N.U., Ghosh, S.K., Rahman, M.A., Esa, H. and Chowdhury, F.R. (2024).** Advancements and applications of generative artificial intelligence and large language models on business management: A comprehensive review. *Journal of Computer Science and Technology Studies*.
9. **Kumar, K.N., Maheswari, K., Abinaya, A., Ramya, J., Ande, P.K. and Ramaian, C.P. (2024).** Advancements in chatbot technology for enhanced customer support in online retail. *2024 Ninth International Conference on Science Technology Engineering and Mathematics (ICONSTEM)*.
10. **Arz von Straussensburg, A.F. and Wolters, A. (2023).** Towards hybrid architectures: Integrating large language models in informative chatbots. *Wirtschaftsinformatik 2023 Proceedings*.
11. **Necula, S.C. and Păvăloaia, V.D. (2023).** AI-driven recommendations: A systematic review of the state of the art in e-commerce. *Applied Sciences*.
12. **Rakhra, M., Singh, G., Gopinadh, G., Aliraja, S., Addepalli, N.S., Reddy, V.S.G. and Reddy, M.N. (2021).** E-commerce assistance with a smart chatbot using artificial intelligence. *2021 2nd International Conference on Intelligent Engineering and Management (ICIEM)*.
13. **Rohella, P., Mimani, S., Ramakrishnan, R. and Jiwani, N. (2024).** Generative AI in fintech: Generating images based on predefined lists of stock keeping units using product descriptions. *2024 2nd International Conference on Disruptive Technologies (ICDT)*.
14. **Santosh, K., Kholmukhamedov, T., Kumar, M.S., Aarif, M. and Bala, B.K. (2024).** Leveraging GPT-4 capabilities for developing context-aware, personalized chatbot interfaces in e-commerce customer support systems. *2024 10th International Conference on Communication and Signal Processing (ICCSP)*.
15. **Pandhare, A. (2024).** Using BERT to build chatbots for e-commerce. *Preprint, ResearchGate*.
16. **Katlariwala, M.Z. and Gupta, A. (2024).** Product recommendation system using large language model: Llama-2. *IEEE World AI IoT Congress (AllIoT)*.
17. **Acharya, S. (2023).** Study of the effectiveness of chatbots in customer service on e-commerce websites. *Bachelor Thesis, Degree Programme in Computer Applications*.

18. **Mikolov, T., Chen, K., Corrado, G. and Dean, J. (2013).** Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*. Available from: <https://doi.org/10.48550/arXiv.1301.3781>.
19. **Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł. and Polosukhin, I. (2017).** Attention is all you need. *Advances in Neural Information Processing Systems*, 30, pp. 5998-6008.
20. **Naveed, H., Khan, A.U., Qiu, S., Saqib, M., Anwar, S., Usman, M., Akhtar, N., Barnes, N. and Mian, A. (2023).** A comprehensive overview of large language models. *arXiv preprint arXiv:2307.06435*. Available from: <https://doi.org/10.48550/arXiv.2307.06435>.
21. **Chen, K., Chen, Y.J., Gallego, G., Gao, P. and Liu, H. (2020).** Personalized product recommendations with exponential bandits. *SSRN*. Available from: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3724377.
22. **Mayank, S. (2023).** Generative AI: Empowering innovation with its astonishing capabilities. *ShuruTech*. Available from: <https://shurutech.com/innovating-with-generative-ai/>.
23. **Raschka, S. (2023).** Fine-tuning LLMs efficiently with adapters. *Sebastian Raschka's Magazine*. Available from: <https://magazine.sebastianraschka.com/p/Fine-tuning-llms-with-adapters>.
24. **Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., et al. (2020).** Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. *arXiv preprint arXiv:2005.11401*.
25. **Izacard, G., & Grave, E. (2021).** Leveraging Passage Retrieval with Generative Models for Open Domain Question Answering. *arXiv preprint arXiv:2007.01282*.
26. **Jeong, C. (2024).** Fine-tuning and utilization methods of domain-specific LLMs. *Samsung SDS*.
27. **Gao, Y., Xiong, Y. (2023).** Retrieval-augmented generation for large language models: A survey. *Shanghai Research Institute for Intelligent Autonomous Systems, Tongji University; Shanghai Key Laboratory of Data Science*.
28. **Bitext (2023).** Bitext customer support LLM chatbot training dataset. *Hugging Face*. Available from: <https://huggingface.co/datasets/bitext/Bitext-customer-support-llm-chatbot-training-dataset>.
29. **ML Explained (2024).** Large Language Model (LLM) Evaluation Metrics – BLEU and ROUGE. *ML Explained*. Available from: <https://mlexplained.blog/2024/01/09/large-language-model-evaluation-metrics-bleu-and-rouge/>.
30. **McKinsey & Company (2023).** Becoming indispensable: Moving past e-commerce to next commerce. *McKinsey & Company*. Available from: <https://www.mckinsey.com/capabilities/growth-marketing-and-sales/our-insights/becoming-indispensable-moving-past-e-commerce-to-next-commerce>.
31. **The Consulting Report (2023).** PwC introduces AI chatbot to accelerate deal-making process. *The Consulting Report*. Available from: <https://www.theconsultingreport.com/pwc-introduces-ai-chatbot-to-accelerate-deal-making-process/>.
32. **Accenture (2024).** Accenture Tech Vision 2024. *Accenture*. Available from: <https://www.accenture.com/content/dam/accenture/final/accenture-com/document-2/Accenture-Tech-Vision-2024.pdf>.
33. **Acumen Research and Consulting (2023).** Generative AI Market. *Acumen Research and Consulting*. Available from: <https://www.acumenresearchandconsulting.com/generative-ai-market>.

7. APPENDICES

This section of the report provides a detailed description of the various technical implementations and configurations involved in the development of the Westminster ShoeBot. Each section highlights key components of the project, from setup and configuration to data processing, model fine-tuning, and visualisation. Below is a breakdown of the appendices:

7.1 Section 1: Installation and Configuration of Pinecone, OpenAI, and LangChain

This section details the installation process for Pinecone, OpenAI, LangChain, and other necessary libraries used for building the vector store and chatbot. Instructions for setting up the environment, API keys, and ensuring compatibility across components are provided.

```
[ ] pip install openai langchain langchain_community langchain pinecone tiktoken rouge_score streamlit -q
[ ] Preparing metadata (setup.py) ... done
[ ] 50.4/50.4 kB 711.6 kB/s eta 0:00:00
[ ] 367.8/367.8 kB 12.1 MB/s eta 0:00:00
[ ] 1.0/1.0 MB 28.4 MB/s eta 0:00:00
[ ] 2.3/2.3 MB 17.5 MB/s eta 0:00:00
[ ] 1.1/1.1 MB 15.3 MB/s eta 0:00:00
[ ] 8.7/8.7 MB 24.5 MB/s eta 0:00:00
[ ] 207.3/207.3 kB 11.8 MB/s eta 0:00:00
[ ] 76.4/76.4 kB 4.1 MB/s eta 0:00:00
[ ] 77.9/77.9 kB 2.8 MB/s eta 0:00:00
[ ] 318.9/318.9 kB 7.9 MB/s eta 0:00:00
[ ] 396.4/396.4 kB 13.5 MB/s eta 0:00:00
[ ] 290.5/290.5 kB 7.9 MB/s eta 0:00:00
[ ] 244.8/244.8 kB 9.9 MB/s eta 0:00:00
[ ] 6.9/6.9 MB 30.9 MB/s eta 0:00:00
[ ] 82.9/82.9 kB 3.7 MB/s eta 0:00:00
[ ] 62.7/62.7 kB 2.8 MB/s eta 0:00:00
[ ] 49.3/49.3 kB 1.5 MB/s eta 0:00:00
[ ] 141.9/141.9 kB 7.9 MB/s eta 0:00:00
[ ] 117.6/117.6 kB 4.4 MB/s eta 0:00:00
[ ] 58.3/58.3 kB 2.4 MB/s eta 0:00:00
[ ] Building wheel for rouge_score (setup.py) ... done
```

Importing Libraries and setting up the environment

1. This code imports the necessary libraries and modules for embedding creation and vector database integration. It includes Pinecone, which is used to interact with Pinecone's vector database, and LangChain, which helps manage conversation flows and retrieval processes.
2. Two index names are defined: one for storing embeddings related to shoes data and another for support-related data. API keys for OpenAI and Pinecone are set, along with the file paths for the respective datasets that will be processed.
3. The last section initializes the OpenAI embeddings using the provided API key. It also sets up the Pinecone instance, which is necessary for embedding storage and retrieval using Pinecone's vector database.

```
[ ]
import os
import json
from pinecone import Pinecone, ServerlessSpec
from langchain.embeddings.openai import OpenAIEmbeddings
from langchain.chains import ConversationalRetrievalChain
from langchain.chat_models import ChatOpenAI
from langchain.pinecone import PineconeVectorStore
from langchain.retrievers import MergedRetriever

shoes_index_name = "chatbot-vectodb"
support_index_name = "chatbot-vectodb2"
api_key = " "
pinecone_api_key = " " if " " else " "
file_shoes_path = 'UpdatedShoesData.jsonl'
file_support_path = 'output_dataset2.jsonl'

embeddings = OpenAIEmbeddings(openai_api_key=api_key)
pc = Pinecone(api_key=pinecone_api_key)
```

7.2 Section 2: Directory Structure of Westminster ShoeBot Project

This section covers the organization of project files, including data files, model files, and configuration files. It also discusses how different datasets (Amazon Shoes and FAQ data) are stored, structured, and referenced during the training and retrieval process.

Model file hosted in OpenAI Platform

ft:gpt-4o-mini-2024-07-18:personal::9ySAnm7D	20/08/2024, 22:20
MODEL	
ft:gpt-4o-mini-2024-07-18:personal::9ySAnm7D	
Status	Succeeded
Job ID	ftjob-BZwIipxudjU2dXT00AXsvCKI
Base model	gpt-4o-mini-2024-07-18
Output model	ft:gpt-4o-mini-2024-07-18:personal::9ySAnm7D
Created at	20 Aug 2024, 22:20
Trained tokens	13,514,133
Epochs	3
Batch size	53
LR multiplier	1.8
Seed	431097433

FAQ data file hosted in OpenAI Platform

step_metrics.csv	21/08/2024, 00:06
44 KB - fine-tune-results	
customer_service_fine_tune_data.jsonl	20/08/2024, 21:49
24 MB - fine-tune	
FILE	
customer_service_fine_tune_data.jsonl	
Status	Ready
File ID	file-Kwqa8M0tR7cMCSbWgDRXZuHd
Purpose	fine-tune
Size	24 MB
Created at	20 Aug 2024, 21:49

Amazon Shoes data file hosted in AWS S3 Bucket

Objects (1) Info					
Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more					
Find objects by prefix					
Show versions					
	Name	Type	Last modified	Size	Storage class
	AmazonShoesData.csv	csv	September 7, 2024, 16:37:29 (UTC+01:00)	189.6 MB	Standard

7.3 Section 3: Data Loading and Processing

Here, the process of collecting Amazon UK Shoes and FAQ data is discussed. The section explains the steps for loading, formatting, and preparing data for embeddings and fine-tuning. Both JSONL formatting for shoes data and conversion of FAQ data into the required format for training are included.

An example of loading, formatting and preparing FAQ data

```
[ ] import pandas as pd
import json

# Load your dataset
file_path = 'Bitext_Sample_Customer_Support_Training_Dataset_27K_responses-v11.csv'
df = pd.read_csv(file_path)

# Transform the dataset into the required format
jsonl_data = []
for _, row in df.iterrows():
    entry = {
        "messages": [
            {"role": "system", "content": "You are a helpful e-commerce customer service assistant. Provide clear, accurate and stylish answers and relevant product recommendations."},
            {"role": "user", "content": row["instruction"]},
            {"role": "assistant", "content": row["response"]}
        ]
    }
    jsonl_data.append(entry)

# Save as JSONL
output_file = 'customer_service_fine_tune_data.jsonl'
with open(output_file, 'w') as f:
    for entry in jsonl_data:
        f.write(json.dumps(entry) + '\n')

print(f"Dataset saved to {output_file}")
```

two key processes are implemented to ensure the dataset is suitable for fine-tuning a language model. The first part of the code calculates the token distribution and estimates the training costs. The second part of the code checks the dataset for any format errors before proceeding with the fine-tuning process. It verifies if every entry in the dataset follows the required structure. This includes checking whether essential fields like "role" and "content" are present and ensuring that all messages include an "assistant" response. Any missing or incorrectly formatted data would generate error counts, allowing for corrections to be made before moving forward with the training process.

```
[ ] format_errors = defaultdict(int)

for ex in dataset:
    if not isinstance(ex, dict):
        format_errors["data_type"] += 1
        continue

    messages = ex.get("messages", None)
    if not messages:
        format_errors["missing_messages_list"] += 1
        continue

    for message in messages:
        if "role" not in message or "content" not in message:
            format_errors["message_missing_key"] += 1

        if any(k not in ("role", "content", "name", "function_call", "weight") for k in message):
            format_errors["message_unrecognized_key"] += 1


        if message.get("role", None) not in ("system", "user", "assistant", "function"):
            format_errors["unrecognized_role"] += 1

        content = message.get("content", None)
        function_call = message.get("function_call", None)

        if (not content and not function_call) or not isinstance(content, str):
            format_errors["missing_content"] += 1

    if not any(message.get("role", None) == "assistant" for message in messages):
        format_errors["example_missing_assistant_message"] += 1

if format_errors:
    print("Found errors:")
    for k, v in format_errors.items():
        print(f"{k}: {v}")
else:
    print("No errors found")
```

 No errors found

```

Num examples missing system message: 0
Num examples missing user message: 0

#### Distribution of num_messages_per_example:
min / max: 3, 3
mean / median: 3.0, 3.0
p5 / p95: 3.0, 3.0

#### Distribution of num_total_tokens_per_example:
min / max: 55, 527
mean / median: 172.25520988389403, 151.5
p5 / p95: 113.0, 261.0

#### Distribution of num_assistant_tokens_per_example:
min / max: 12, 478
mean / median: 125.24006400714498, 104.0
p5 / p95: 67.0, 214.0

0 examples may be over the 16,385 token limit, they will be truncated during fine-tuning

# Pricing and default n_epochs estimate
MAX_TOKENS_PER_EXAMPLE = 16385

TARGET_EPOCHS = 3
MIN_TARGET_EXAMPLES = 100
MAX_TARGET_EXAMPLES = 25000
MIN_DEFAULT_EPOCHS = 1
MAX_DEFAULT_EPOCHS = 25

n_epochs = TARGET_EPOCHS
n_train_examples = len(dataset)
if n_train_examples * TARGET_EPOCHS < MIN_TARGET_EXAMPLES:
    n_epochs = min(MAX_DEFAULT_EPOCHS, MIN_TARGET_EXAMPLES // n_train_examples)
elif n_train_examples * TARGET_EPOCHS > MAX_TARGET_EXAMPLES:
    n_epochs = max(MIN_DEFAULT_EPOCHS, MAX_TARGET_EXAMPLES // n_train_examples)

n_billing_tokens_in_dataset = sum(min(MAX_TOKENS_PER_EXAMPLE, length) for length in convo_lens)
print(f"Dataset has ~{n_billing_tokens_in_dataset} tokens that will be charged for during training")
print(f"By default, you'll train for {n_epochs} epochs on this dataset")
print(f"By default, you'll be charged for ~{n_epochs * n_billing_tokens_in_dataset} tokens")

Dataset has ~4628842 tokens that will be charged for during training
By default, you'll train for 1 epochs on this dataset
By default, you'll be charged for ~4628842 tokens

```

Preparing Amazon shoes data for embeddings

In this section of the code, the shoe product data is first loaded from the file `UpdatedShoesData.jsonl` and parsed as JSON objects. Key attributes, such as product title, rating, reviews, available colors, availability, and price, are extracted and formatted into descriptive text for embedding preparation.

```

[ ] with open(file_shoes_path, 'r') as file:
    shoes_data = [json.loads(line) for line in file]

# Prepare the text descriptions for embedding
shoe_descriptions = [
    f"Product: {item['product_title']} | Rating: {item['star_rating']} stars | "
    f"Review Headline: {item['review_headline']} | Review: {item['review_body']} | "
    f"Colors Available: {'', '.join(item['color'])} | Availability: {item['availability']} items left | "
    f"Price: £{item['price']}"
    for item in shoes_data
]

```

Embedding the Amazon shoes data using pinecone

```
[ ] import pinecone
pc=Pinecone(api_key=pinecone_api_key, environment="us-east-1")
```

Pinecone is initialized for storing vector embeddings. The Pinecone instance is connected using the provided API key and configured to operate in the us-east-1 region on AWS. If the shoe index do not already exist in Pinecone, new ones are created with a specified dimension of 1536 and the cosine similarity metric for measuring similarity between vectors. After the index is created.

```
[ ] if shoes_index_name not in pc.list_indexes().names():
    spec = ServerlessSpec(cloud='aws', region='us-east-1')
    pc.create_index(name=shoes_index_name, dimension=1536, metric="cosine",spec=spec)
if support_index_name not in pc.list_indexes().names():
    spec = ServerlessSpec(cloud='aws', region='us-east-1')
    pc.create_index(name=support_index_name, dimension=1536, metric="cosine",spec=spec)
```

A PineconeVectorStore object is instantiated to store embeddings for shoe product descriptions. Finally, the prepared text descriptions for datasets are added to the vector stores, where it will be stored as embeddings for future retrieval during chatbot interactions.

```
[ ] shoes_vectorstore = PineconeVectorStore(index_name=shoes_index_name, embedding=embeddings)
support_vectorstore = PineconeVectorStore(index_name=support_index_name, embedding=embeddings)

shoes_vectorstore.add_texts(texts=shoe_descriptions)
```

7.4 Section 4: Storing Embeddings in Pinecone

This section outlines the code and process for creating embeddings using OpenAI models and storing them in Pinecone's vector database. It covers how shoe product is embedded and stored for efficient retrieval in the RAG pipeline.

Index records

Query List/Fetch Add a record

Namespace (Default) Query by dense vector value vector 0.88,0.68,0.81,0.93,0.51,0.71,0.95,0.8,0.4,0.12,0.9,0.99,0.43,0.76,0.13,0.58,0.38,0.62,0.74,0.98,0.85,0.48,0.33,0.09,0.4 Metadata Filter Top K 10 Query

Matches: 10

	ID	VALUES	
1	e194caa5-f44...	0.00966090616, -0.0216937326, -0.00222533988, -0.0161237214, -0.0054034451, 0.0171631, -0.0105870208, -0.031074807, -0.000462640834, -0.0289...	🔍 ✎ 🗑
SCORE -0.0069	METADATA text: *Product: Robeez Sara Sandal First Walker (Toddler) Rating: 4.0 stars Review Headline: Did the job for almost as long as we needed then Review: Cut little shoes, but eventually the lining sta...		
2	a21a4969-2d...	-0.00589788798, -0.0153650269, -0.0153252212, -0.0202876739, -0.0131624406, 0.0229281224, -0.0262850765, -0.0277844258, -0.0201019123, -0.02...	
SCORE -0.0083	METADATA text: *Product: John Deere 2113 Western Boot (Toddler/Little Kid) Rating: 5.0 stars Review Headline: My 3 year old loves these. Review: My 3 year old is out on the farm with Daddy all the time. H...		

7.5 Section 5: Fine-tuning GPT-4o Mini with OpenAI

A detailed overview of how the GPT-4o Mini model was fine-tuned on the FAQ dataset, including the fine-tuning process using OpenAI. Specific hyperparameters such as epochs, batch size, and learning rates are also shown.

MODEL

ft:gpt-4o-mini-2024-07-18:personal::9ySAnm7D

Status

Succeeded

Job ID

ftjob-BZwIipxudjU2dXTo0AXsvCKI

Base model

gpt-4o-mini-2024-07-18

Output model

ft:gpt-4o-mini-2024-07-18:personal::9ySAnm7D

Created at

20 Aug 2024, 22:20

Trained tokens

13,514,133

Epochs

3

Batch size

53

LR multiplier

1.8

Seed

431097433

Checkpoints

ft:gpt-4o-mini-2024-07-18:personal::9ySAmv9e:ckpt-step-1014

ft:gpt-4o-mini-2024-07-18:personal::9ySAmrmW:ckpt-step-1521

ft:gpt-4o-mini-2024-07-18:personal::9ySAnm7D

Files

Training

customer_service_fine_tune_data.jsonl

Validation

-

Training loss

0.5181

Messages

Metrics

00:06:24

The job has successfully completed

00:06:19

Usage policy evaluations completed, model is now enabled for sampling

00:03:25

Evaluating model against our usage policies before enabling

00:03:25

New fine-tuned model created: ft:gpt-4o-mini-2024-07-18:personal::9ySAnm7D

00:03:25

Checkpoint created at step 1521 with Snapshot ID: ft:gpt-4o-mini-2024-07-18:personal::9ySAmrmW:ckpt-step-1521

00:03:25

Checkpoint created at step 1014 with Snapshot ID: ft:gpt-4o-mini-2024-07-18:personal::9ySAmv9e:ckpt-step-1014

23:29:17

Fine-tuning job started

23:29:07

Files validated, moving job to queued state

22:20:04

Validating training file: file-Kwqa8M0tR7cMCSbWgDRXZuHd

22:20:04

Created fine-tuning job: ftjob-BZwlipxudjU2dXTo0AXsvCKI

7.6 Section 6: RAG Pipeline Construction and Prompt Engineering

This section covers the implementation of the RAG pipeline using LangChain and Pinecone for vector-based retrieval. It also presents the prompt engineering strategy applied to ensure the chatbot responds accurately and maintains context in shoe-related queries.

```
prompt_template = """The customer has asked: "{question}"
Here is the recent context of the previous conversation:
{chat_history}

Instructions:
    You are a Westminster ShoeBot to help customers navigate through
the finest selections, tailored to their unique styles and needs.
    No reintroductions unless explicitly requested.
    Politely acknowledge brief replies like "ok," "nice," etc.,
without repeating your role.
    Be friendly and natural with a slight touch of style and humor.
Keep it professional and relevant to the conversation.

Your role is to assist the customer with shoe-related queries. If the
query is unrelated to shoes (e.g., clothing, accessories), kindly
acknowledge the customer's request and gently steer the conversation
back to footwear without providing unsolicited recommendations.

Handling Unrelated Queries:

    Acknowledge non-shoe-related questions (e.g., "Thanks for asking
about clothing!").
    Gently steer the conversation back to shoes (e.g., "I specialize
in shoes, but I can definitely help you find the perfect pair!").
    Do not offer recommendations for non-shoe topics.

Recommendations:

    Suggest products only if explicitly asked or if the customer
shows interest in shoe-related details (size, color, type).
    Use customer preferences and retrieved data to provide accurate,
stylish suggestions.

Use the retrieved information and the ongoing conversation context to
assist the customer with their queries. If the customer has provided
specific details like size, color, or product preferences, use that
information in your response
{context}

"""
```


Indexing and Retrieving Embeddings

```
import pinecone
pc=Pinecone(api_key=pinecone_api_key, environment="us-east-1")

if shoes_index_name not in pc.list_indexes().names():
    spec = ServerlessSpec(cloud='aws', region='us-east-1')
    pc.create_index(name=shoes_index_name, dimension=1536, metric="cosine",spec=spec)
if support_index_name not in pc.list_indexes().names():
    spec = ServerlessSpec(cloud='aws', region='us-east-1')
    pc.create_index(name=support_index_name, dimension=1536, metric="cosine",spec=spec)

shoes_index = pc.Index(shoes_index_name)
support_index = pc.Index(support_index_name)

shoes_vectorstore = PineconeVectorStore(index=shoes_index, embedding=embeddings)
support_vectorstore = PineconeVectorStore(index=support_index, embedding=embeddings)

shoes_retriever = shoes_vectorstore.as_retriever()
support_retriever = support_vectorstore.as_retriever()

combined_retriever = MergerRetriever(
    retrievers=[shoes_retriever, support_retriever],
    mode="merge",
)
```

Model Calling and memorization context

```
chat=ChatOpenAI(temperature=0, model_name=" ft:gpt-4o-mini-2024-07-18:personal::9ySAnm7D", openai_api_key="sk*****A")
@st.cache_resource
def setup_chain():
    chain = ConversationalRetrievalChain.from_llm(
        llm=chat,
        retriever=combined_retriever,
        memory=memory,
        return_source_documents=True,
        verbose=False,
        combine_docs_chain_kwargs={"prompt": PROMPT}
    )
    return chain

qa_chain = setup_chain()
```

7.7 Section 7: Streamlit Interface Development

Describes the user interface built using Streamlit for the Westminster ShoeBot. The section includes code for building the chatbot's interactive web interface, with detailed customization for chat history, input fields, and product recommendations.

```
# Streamlit Interface
st.markdown("""
<style>
```

```

body {
    font-family: 'Montserrat', sans-serif;
    background-color: #F3F4EF;
}
header {
    text-align: center;
}
.stApp {
    background-color: #F3F4EF;
    padding: 2rem;
}
.title {
    font-size: 2.5rem;
    color: #33312D;
    font-weight: bold;
}
.subtitle {
    font-size: 1.25rem;
    color: #33312D;
    margin-bottom: 2rem;
}
.st-chat-message-assistant {
    background-color: #F3F4EF;
    color: #33312D;
}
.stButton button {
    background-color: #002147;
    color: white;
    font-size: 1rem;
    padding: 10px 20px;
    border-radius: 5px;
    border: none;
}
.stButton button:hover {
    background-color: #014ea7;
}
</style>
"""', unsafe_allow_html=True)

# Header Section
st.markdown("<div class='title'>Westminster ShoeBot</div>",
unsafe_allow_html=True)
st.markdown("<div class='subtitle'>Elevating E-Commerce Experiences
with Akram's Westminster Flair</div>", unsafe_allow_html=True)
st.markdown("""<div class='subtitle'>Welcome to the Westminster
ShoeBot, a state-of-the-art e-commerce assistant designed exclusively
for shoe enthusiasts. Powered by GPT-4O Mini, this chatbot combines
the latest in AI technology with a deep understanding of the footwear
market to help you find the perfect pair of shoes. Whether you're

```

```

looking for the latest trends or the perfect fit, Akram's ShoeBot is
here to enhance your shopping experience with intelligent, context-
aware conversations.
</div>""", unsafe_allow_html=True)

# Initialize chat history in session state if not already done
if "chat_history" not in st.session_state:
    st.session_state.chat_history = []

for i, chat in enumerate(st.session_state.chat_history):
    if chat["role"] == "user":
        with st.chat_message("user"):
            st.markdown(chat['content'])
    else:
        with st.chat_message("assistant"):
            st.markdown(f"<div class='st-chat-message-
assistant'>{chat['content']}</div>", unsafe_allow_html=True)

user_input = st.chat_input(placeholder="Ask me anything about
shoes!")

if user_input:
    # Display user message immediately in the chat UI
    with st.chat_message("user"):
        st.markdown(user_input)
    st.session_state.chat_history.append({"role": "user", "content":
user_input})

    # Get the response from the model using qa_chain
    result = qa_chain.invoke({"question": user_input})
    response = result["answer"]
    # Display the assistant's response immediately
    with st.chat_message("assistant"):
        st.markdown(f"<div class='st-chat-message-
assistant'>{response}</div>", unsafe_allow_html=True)
    # Append both user and assistant messages to chat history
    st.session_state.chat_history.append({"role": "assistant",
"content": response})

```

Running Application on Streamlit

```
Overwriting app.py

!npm install localtunnel

added 22 packages, and audited 23 packages in 2s

3 packages are looking for funding
  run `npm fund` for details

2 moderate severity vulnerabilities

To address all issues, run:
  npm audit fix

Run `npm audit` for details.

[ ] !streamlit run app.py & npx localtunnel --port 8501

Collecting usage statistics. To deactivate, set browser.gatherUsageStats to false.


You can now view your Streamlit app in your browser.


Local URL: http://localhost:8501
Network URL: http://172.28.0.12:8501
External URL: http://34.48.105.137:8501
your url is: https://famous-maps-sneeze.loca.lt
```

Westminster ShoeBot

Elevating E-Commerce Experiences with Akram's Westminster Flair

Welcome to the Westminster ShoeBot, a state-of-the-art e-commerce assistant designed exclusively for shoe enthusiasts. Powered by GPT-4O Mini, this chatbot combines the latest in AI technology with a deep understanding of the footwear market to help you find the perfect pair of shoes. Whether you're looking for the latest trends or the perfect fit, Akram's ShoeBot is here to enhance your shopping experience with intelligent, context-aware conversations.

 Hi my friend!

 Hey there! It's always a pleasure to chat with a friend. How can I assist you today? If you're on the hunt for some stylish shoes, I'm here to help you find the perfect pair!

7.8 Section 8: Model Inference and Evaluation

This section provides the approach used to evaluate the model performance, including BLEU, ROUGE, and Semantic Similarity scores. It includes the test dataset used for evaluation and the key metrics generated for assessing model responses.

```

results_df = pd.DataFrame({
    "BLEU": bleu_scores,
    "ROUGE-1": rouge_scores_1,
    "ROUGE-L": rouge_scores_L,
    "Semantic Similarity": semantic_similarities
})

average_bleu = results_df['BLEU'].mean()
average_rouge1 = results_df['ROUGE-1'].mean()
average_rougeL = results_df['ROUGE-L'].mean()
average_semantic_similarity = results_df['Semantic Similarity'].mean()

print(f"Average BLEU Score: {average_bleu:.4f}")
print(f"Average ROUGE-1 Score: {average_rouge1:.4f}")
print(f"Average ROUGE-L Score: {average_rougeL:.4f}")
print(f"Average Semantic Similarity Score: {average_semantic_similarity:.4f}")

```

Average BLEU Score: 0.1917
 Average ROUGE-1 Score: 0.5749
 Average ROUGE-L Score: 0.4080
 Average Semantic Similarity Score: 0.8486

Model Evaluation Metrics

