

UNIVERSITY COLLEGE LONDON

DEPARTMENT OF COMPUTER SCIENCE

---

MULTI-DIALOG CHATBOTS: KNOWLEDGE  
BASE UPDATES FOR ENHANCED USER  
PREFERENCE SUPPORT

---

*Author*

Valerie SONG

*Academic Supervisor*

Prof. Philip TREVELEAN  
DEPARTMENT OF COMPUTER  
SCIENCE  
UNIVERSITY COLLEGE LONDON

*Industrial Supervisor*  
Prof. Andy PARDOE  
WISDOM WORKS

*September 11, 2023*

This dissertation is submitted as part requirement for the MSc Emerging Digital  
Technologies degree at UCL.

## DECLARATION

I, Valerie Song, confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the thesis.

## ABSTRACT

This thesis represents a comprehensive exploration of the capabilities of large language models (LLMs) as a conversational agent, with a specific focus on handling multiple dialog flows and updating a knowledge graph to cater to user preferences in an e-commerce context. This research is motivated by the growing adoption of LLM chatbots on websites, with a particular emphasis on enhancing the user experience through the use of natural language processing and generative AI.

This dissertation investigates various deep learning and machine learning methodologies, aiming to seamlessly unify a knowledge graph and large language model to achieve contextual awareness. This will aid in developing a chatbot capable of providing personalized assistance, question-answering capabilities, and product recommendations based on users' preferences.

The thesis consists of three investigations contributing to the development of chatbots. The research structure is presented as follows and will be further explored in section three, four, and five respectively.

1. **Experiment 1: Building Experimental LLM Chatbot Strategies.** This study involves the development and implementation of various chatbot strategies. It explores different fine-tuning methods to adapt LLMs as an e-commerce chatbot tailored to helping users find products.
2. **Experiment 2: Designing Dialog Management Strategies.** This investigation explores dialog management strategies to create a context-aware model. We look at drawing from both past and ongoing interactions between the user and model to achieve this. Additionally, techniques such as retrieval-augmented generation is explored for incorporating external data into the LLMs context, enhancing overall conversational abilities.
3. **Experiment 3: Knowledge-Base Update Mechanisms.** The final investigation in this thesis explores different approaches to update knowledge bases with relevant information from dialogues held between the user and chatbot. This research predominantly involves an extensive review of the unification of knowledge graphs and large language models.

## CONTRIBUTIONS TO SCIENCE

1. **Advancement of Natural Language Processing (NLP).** Developing experimental chatbot strategies through the use of LLMs contributes to the advancement of NLP. It explores the use of language models for understanding user queries, generating contextually relevant responses, and providing a natural conversational experience. This has implications not only for e-commerce but also for broader applications in language understanding and generation.
2. **Advancement of Knowledge Representation.** The research into the unification of knowledge graphs with LLMs to provide personalised product suggestions to users, advances the field of knowledge representation and reasoning. We explore the methods for enhancing the chatbot's knowledge base and enabling it to retrieve and use external information effectively.
3. **Dialog Management.** Our investigation into dialog management strategies contributes to the science of effective dialog systems. It explores techniques for managing conversations, recognizing user intents, and crafting contextually relevant responses. These findings can inform the development of conversational agents in various domains beyond e-commerce.

## IMPACT STATEMENT

The thesis presents the following impact statements to businesses:

1. **Enhanced Customer Satisfaction.** Implementing an LLM-powered chatbot creates more engaging and interactive customer experiences. Through a good dialog management policy, it allows businesses to converse with customers in natural language, answer their questions, and offer personalized recommendations, thereby increasing the chance of retaining customers.
2. **Data-Driven Insights.** The unification of knowledge graphs and LLMs, allow the chatbot to retrieve and update valuable customer data and insights from interactions. This data can inform marketing strategies, product development, and customer segmentation, enabling businesses to make decisions that align with current market demands and customer preferences.
3. **Cost-savings and Time Efficiency.** Incorporating chatbots into e-commerce platforms means businesses can focus less on human resources for customer support, resulting in cost savings. Additionally, as chatbots operate 24/7, customer queries are met at a faster duration.

To access the code for the following experiments, please click on this link and request access from the following email: valeriewqsong@gmail.com

## ACKNOWLEDGMENTS

I would like to overstate my gratitude to my MSc supervisors, Professor Philip Treleaven, Professor Andy Pardoe, and Fares Fawzi. They made this research possible with their friendly support, patience and technical expertise. I would also like to thank my parents for the opportunities they have given me that have lead me here.

# CONTENTS

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Background . . . . .	1
1.2	Research Motivation . . . . .	3
1.3	Research Objectives . . . . .	3
1.4	Thesis Structure . . . . .	4
<b>2</b>	<b>Literature Review</b>	<b>5</b>
2.1	Large Language Models . . . . .	5
2.1.1	Architecture . . . . .	6
2.2	Knowledge Graphs . . . . .	7
2.2.1	Structure . . . . .	7
2.2.2	Unification of KG + LLMs . . . . .	8
2.2.3	KG-enhanced LLM during inference . . . . .	8
2.2.4	LLM-Augmented Knowledge Graph Construction . . . . .	8
<b>3</b>	<b>Building Experimental LLM Chatbot Strategies</b>	<b>9</b>
3.1	Introduction . . . . .	9
3.2	Background . . . . .	10
3.2.1	Fine-Tuning . . . . .	10
3.3	Dataset . . . . .	11
3.3.1	LLM Model: MPT-7B . . . . .	11
3.3.2	LLM-Foundry . . . . .	13
3.3.3	Dataset: OpenDialKG . . . . .	13
3.4	Methodology . . . . .	13
3.5	Results . . . . .	17
3.6	Discussion . . . . .	19
3.6.1	Limitations . . . . .	22
3.7	Summary . . . . .	22

<b>4 Designing Dialog Management Strategies</b>	<b>24</b>
4.1 Introduction . . . . .	24
4.2 Background . . . . .	25
4.2.1 Retrieval-Augmented Generation . . . . .	25
4.3 Dataset . . . . .	27
4.3.1 RAG Dataset . . . . .	27
4.3.2 Embedding Model . . . . .	28
4.3.3 LLM Model . . . . .	28
4.4 Methodology . . . . .	28
4.4.1 RAG Experiment . . . . .	28
4.5 Results . . . . .	30
4.5.1 Experiment 1: Asking the conversation agent for product recommendations . . . . .	31
4.5.2 Experiment 2: Testing models ability to hold multiple dialog flows .	32
4.5.3 Experiment 3: Conversational memory . . . . .	33
4.5.4 Experiment 4: Prompt vs No prompt . . . . .	35
4.6 Discussion . . . . .	37
4.6.1 Experiment 1 . . . . .	37
4.6.2 Experiment 2 . . . . .	37
4.6.3 Experiment 3 . . . . .	38
4.6.4 Experiment 4 . . . . .	39
4.6.5 Limitations . . . . .	39
4.7 Summary . . . . .	39
<b>5 Knowledge-Base Update Mechanisms</b>	<b>41</b>
5.1 Introduction . . . . .	41
5.2 Background . . . . .	42
5.2.1 Information Extraction Pipeline . . . . .	43
5.3 Dataset . . . . .	44
5.3.1 Frameworks . . . . .	44
5.4 Methodology . . . . .	44
5.4.1 Knowledge Graph Creation . . . . .	44
5.4.2 Retrieving external data during inference . . . . .	44
5.5 Results . . . . .	47
5.5.1 Experiment 1: Knowledge Retrieval using Cypher Chain . . . . .	47
5.5.2 Experiment 2: Agent Comparison . . . . .	48
5.5.3 Experiment 3: Conversational Agent Retrieval Capabilities . . . . .	49

5.6	Discussion . . . . .	52
5.6.1	Experiment 1 . . . . .	52
5.6.2	Experiment 2 . . . . .	52
5.6.3	Experiment 3 . . . . .	52
5.7	Summary . . . . .	53
<b>6</b>	<b>Conclusion and Future Work</b>	<b>55</b>
6.1	Conclusion . . . . .	55
6.1.1	Building Experimental LLM Chatbot Strategies . . . . .	55
6.1.2	Designing Dialog Management Strategies . . . . .	56
6.1.3	Knowledge Base Update Mechanisms . . . . .	57
6.2	Future Work . . . . .	57
6.2.1	Building Experimental LLM Chatbot Strategies . . . . .	57
6.2.2	Designing Dialog Management Strategies . . . . .	58
6.2.3	Knowledge Base Update Mechanisms . . . . .	59
<b>Bibliography</b>		<b>59</b>
<b>Appendix A Code</b>		<b>64</b>
A.1	Data pre-processing OpenDialKG Dataset . . . . .	64
A.2	Experiment 2: Conversational Agent Implementation . . . . .	67
A.3	Cypher Chain Model Implementation . . . . .	68

# LIST OF FIGURES

2.1 Transformer - Model Architecture [43] . . . . .	7
3.1 LoRA reparametrization process [23] . . . . .	11
3.2 Training data sources [39] . . . . .	12
3.3 OpenDialKG dataset stats [31] . . . . .	14
3.4 OpenDialKG dataset structure [47] . . . . .	14
3.5 Dataset split for fine-tuning . . . . .	15
3.6 Training Loss graph . . . . .	18
3.7 Alpaca Dataset Data Fields . . . . .	20
3.8 Preprocessing Alpaca Dataset . . . . .	20
3.9 MPT-7B Training Loss for Alpaca Dataset . . . . .	21
3.10 MPT-7B Evaluation Loss for Alpaca Dataset . . . . .	21
4.1 Vector Store Pipeline [44] . . . . .	26
4.2 RAG Retrieval Process [5] . . . . .	27
4.3 Similarity Search Experiment 1 . . . . .	31
4.4 Similarity Search Experiment 2 . . . . .	31
4.5 Product Recommendation Dialog . . . . .	32
4.6 Multi-Dialog Experiment . . . . .	32
4.7 User Inputs . . . . .	34
4.8 Outputs with 'ConversationSummaryBuffer' Memory . . . . .	34
4.9 Outputs with 'ConversationBufferWindow' Memory . . . . .	35
4.10 Conversational History - summarised version . . . . .	35
4.11 Prompt comparison dialog . . . . .	36
5.1 Information Extraction Pipeline . . . . .	43
5.2 Product details . . . . .	45
5.3 Cypher Generation Prompt Template . . . . .	46
5.4 Question-Answering Prompt . . . . .	46

5.5	Knowledge Graph instance of Product and Colour relationship . . . . .	47
5.6	Knowledge Retrieval using Cypher Chain Result . . . . .	48
5.7	Comparison of conversational agents results . . . . .	49
5.8	Conversation Chain Execution Steps . . . . .	50
5.9	Conversational Chain Experiment 1 . . . . .	51
5.10	Conversational Chain Experiment 2 . . . . .	51

# LIST OF TABLES

3.1	Attention Configuration Parameters . . . . .	16
3.2	LoRA Hyperparameters . . . . .	16
3.3	Training Hyperparameters . . . . .	17
3.4	Total trainable parameters in LoRA . . . . .	17
3.5	Training Results . . . . .	17
3.6	Evaluation Results . . . . .	18
3.7	Training Results - Alpaca Dataset . . . . .	20
3.8	Evaluation Results - Alpaca Dataset . . . . .	21
4.1	Fashion Dataset Attributes . . . . .	27
5.1	Dataset Nodes and Relationships . . . . .	45

# CHAPTER 1

## INTRODUCTION

*The objective of this chapter is to present an overview of the thesis by discussing the motivation behind the research problem, objectives, experiments and impacts of this study. This chapter starts by briefly introducing background information on Large Language Models (LLM) and knowledge graphs while displaying the importance of contextual awareness for an improved user experience. The chapter concludes with the structure of the thesis.*

This thesis investigates ways to support chatbots with multi-dialog flows by identifying the intent of the user with the assistance of knowledge bases to support user preferences and the generation of recommendations. The majority of this study was conducted in collaboration with WisdomWorks under the supervision of Professor. Andy Pardoe, facilitating expert guidance and validation.

### 1.1 BACKGROUND

The establishment of chatbots dates back to 1966 when MIT introduced the first one, known as ELIZA. These systems started as rule-based entities, relying on predefined scripts and rules to function. However, over the past decade, the landscape of chatbot development has undergone a significant transformation driven by artificial intelligence and natural language processing tools.

More recently, the introduction of large language models into chatbots, exemplified by OpenAI's ChatGPT, has catalyzed a paradigm shift in the chatbot industry. This shift is primarily attributed to the remarkable capacity of these models to generate coherent and contextually relevant responses. Their proficiency stems from deep learning algorithms and their exposure to large amounts of data during the training process. As this tech-

nology continues to permeate various industries and influence the daily lives of users, this thesis centers on the evolutionary trajectory of chatbots within the e-commerce sector. We highlight the profound impact of these technologies reshaping operational methods within this industry.

By 2027, Garner predicts that chatbots will become the primary customer support channel for organisations [13]. This can be linked to the positive effects it produces in regards to customer experience, streamlining operations, and business growth. Through the improvement of NLP, chatbots are able to extract valuable customer data during these interactions, which aids businesses in identifying trends as well as the ability to make data-driven decisions.

However, a key limitation of chatbots are their inability to handle complex queries. In terms of rule-based chatbots, if the knowledge base does not include the users' query, it will not be able to provide an accurate response. This ties in with their inability to provide a lack of personalised experience to the user due to a lack of contextual awareness. According to an Uberall [42] survey , 43% of customers indicated that context in chatbots need to be improved. It has been observed that chatbots often struggle to handle complex dialog flows, especially where customers may have diverse intents and preferences. Additionally, knowledge bases used in e-commerce sites are predominantly static, which can lead to inaccurate information as it becomes outdated. This is especially detrimental for e-commerce platforms, where data is constantly changing.

Consequently, there is a growing need for chatbot systems that can seamlessly support numerous dialog flows and update their knowledge base dynamically from dialog policies. Knowledge bases play a crucial role in improving the performance of chatbots by providing them with structured and domain-specific knowledge.

To mitigate these limitations, techniques such as natural language processing (NLP), machine learning, and context management systems are being integrated into systems. These techniques enable chatbots to go beyond the static information in the knowledge base and incorporate real-time contextual understanding, user history, and dynamic updates for more effective and personalized interactions.

## 1.2 RESEARCH MOTIVATION

The main objective of this research is to investigate experimental chatbot strategies and propose knowledge base update techniques that support multiple dialog flows. Below are research motivations of this thesis:

1. Personalised User Experience. The use of LLM-powered chatbots have the potential to offer highly personalized recommendations based on user preferences, and past interactions. Their natural language understanding capabilities play a pivotal role in this delivery.
2. Continuous Learning and Adaptation. Using the insights gained from user interactions, the chatbot can continually adapt its responses and behavior. It can learn to provide more accurate answers, refine its product recommendations, and adjust its tone and style of communication based on individual user preferences.
3. Enhanced Decision-Making Support. The integration of knowledge graph and dialog management strategies into LLMs allow the chatbot to utilise up-to-date and accurate information to meet users' needs.

With a focus on addressing these limitations and investigating diverse techniques to facilitate multiple dialog flows and dynamic knowledge base updates, this thesis aims to make a meaningful contribution to the advancement of LLM-powered chatbots within the realm of e-commerce platforms.

## 1.3 RESEARCH OBJECTIVES

1. Building Experimental LLM Chatbot Strategies. The first experiment aims to build a chatbot capable of contextual understanding and generation of personalised product recommendations to users. This study starts by introducing large language models and its underlying architecture. We then delve into different fine-tuning strategies to adapt the LLM as a conversational agent for optimal performance. The primary focus of this study is to achieve high user satisfaction by using different statistical methods to create a personalised experience between users and the chatbot.
2. Designing Dialog Management Strategies. This study investigates different strategies to support multiple dialog flows effectively. Two strategies we will analyse are Context Management and Intent Recognition. Within these we will look at their conversational abilities, and compare which technique is able to better meet the needs

of users. This will be evaluated by performing A/B testing between the different dialog strategies.

3. Exploring Knowledge Base Update Mechanisms. This analysis explores different approaches of unifying knowledge graphs and large language models to extract and retrieve relevant information from interactions between the user and conversational agent. Additionally, we will delve into various Natural Language Processing (NLP) techniques to improve the chatbot's conversational abilities, facilitating the seamless identification of knowledge gaps or inaccuracies within the knowledge base.

## 1.4 THESIS STRUCTURE

The structure of the thesis is organised as follows:

- Chapter 1: Introduction
- Chapter 2: Background and Literature Review
- Chapter 3: Building Experimental LLM Chatbot Strategies
- Chapter 4: Dialog Management Strategies
- Chapter 5: Knowledge-Base Update Mechanisms
- Chapter 6: Conclusion and Future Work

# CHAPTER 2

## LITERATURE REVIEW

*This chapter presents background information on a number of key concepts in the areas that this research focuses on. The first part of this chapter looks at the architecture of transformers as the building block into LLM (Large Language Models), followed by a deep-dive of the LLM itself. Next, we delve into knowledge graphs, what it is and how it can be used. The chapter ends with an investigation into the limitations of large language models and how we can alleviate these issues through the integration of a knowledge graph.*

### 2.1 LARGE LANGUAGE MODELS

Large Language Models (LLMs) are neural networks with the capability to perform a wide range of tasks, including language translation, text generation, content summarisation, and sentiment analysis, among others. Their effectiveness in these tasks is attributed to their training on extensive datasets, often comprising over a billion parameters. The vast amount of data used during training allows the LLM to generate coherent and helpful responses to input prompts, rendering them highly versatile and adaptable for various applications. Some notable examples of LLMs adapted as conversational agents include: GPT-4 [33] from OpenAI, LLaMa [41] from Meta, and BERT [18] from Google.

The performance of LLMs is significantly influenced by the datasets used for training. The process starts with pre-training the model through self-supervised learning, where the model trains itself to predict the next token given a sequence of previous tokens [17]. This self-supervised learning approach eliminates the need for manually labeled data, allowing for the acquisition of large quantities of data more easily and cost-effectively than traditional supervised learning methods.

During this task, the model will come to learn the general language patterns and structure between tokens, as well as the knowledge in the dataset [19]. As the model learns from broad and diverse datasets, it may not excel in tasks that require domain-specific knowledge. To create a proficient conversational agent tailored to an organization’s specific needs, fine-tuning is necessary. Various strategies can be employed for fine-tuning, including retraining an LLM, and prompt engineering. These strategies will be explored in more depth in subsequent chapters, as there are multiple avenues to achieve effective fine-tuning for specific use cases.

### 2.1.1 ARCHITECTURE

To gain a comprehensive understanding of language models, it is essential to delve into the foundational architecture of Large Language Models (LLMs), known as Transformers. Transformers are a type of deep learning architecture that heavily relies on a self-attention mechanism, as detailed in Vaswani et al.[43]. Self-attention is what allows the model to capture long-range dependencies between different tokens in the input sequence. This is achieved by assigning attention weights, which results in the model better being able to capture relevant information during both encoding and decoding stages.

In a transformer, there are two main components known as an encoder and decoder. Seen in Figure 2.1, positioned to the left where the input starts is the encoder. It takes the role of processing the input text and producing a series of hidden states that capture the meaning of the text. However, to fully capture its meaning, the order which the words come in are crucial. Traditional neural networks do not possess this capability, therefore a positional encoder is introduced into the transformer. This provides the model with a set of numbers that are relative to the position of tokens in the input sequence, along with its corresponding input embedding.

For an LLM to predict the next word, during training the decoder is only provided with previous words from the sequence. This is achieved by shifting the output sequence to the right by one. Both the positional encoded input representation and output embeddings go through the decoder, as positioned on the right of Figure 2.1. This component takes on the job of generating the output sequence from the result of the encoder’s representation. Several layers of both components are used in the Transformer.

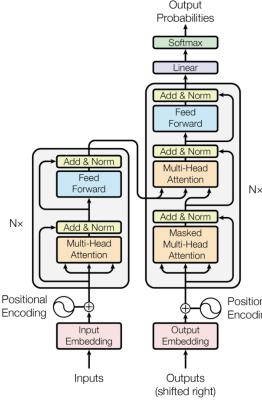


Figure 2.1: Transformer - Model Architecture [43]

## 2.2 KNOWLEDGE GRAPHS

A knowledge graph is a network representing real-world entities, and the relationship between them [25]. It is a structured way in organising information to represent data in a machine-readable format. These entities can fall anywhere between objects, events, situations, or concepts. They often store a large amount of data to enhance the knowledge of various applications. Presently, knowledge graphs are categorized into four main groups: encyclopedic, commonsense, domain-specific, and multi-modal. In the context of our research, this study will focus on domain-specific knowledge graphs.

### 2.2.1 STRUCTURE

A knowledge graph can be defined by the following equation 2.1, where  $\epsilon$  = set of entities (nodes),  $R$  = set of relations (type of edges), and  $\tau$  = relational facts (edges). An edge is what connects a pair of nodes to represent a relationship between them, and the set of relations defines the meaning of the relationship between the nodes.

$$G = \{\epsilon, R, \tau\} \quad (2.1)$$

Often, knowledge graphs are stored as triples, where each triple consists of two nodes and a relationship between them. For example, in the context of an e-commerce site, the triple (Samantha, purchased, NikeShoes) indicates that Samantha purchased some Nike shoes. A representation of this can be seen in the equation 2.2, where head entity  $h \in \epsilon$ , tail entity  $t \in \epsilon$ , relation  $r \in R$  [10].

$$\tau = \{(h, r, t)\} \quad (2.2)$$

### 2.2.2 UNIFICATION OF KG + LLMs

A common criticism with LLMs is their tendency to 'hallucinate', meaning they have the ability to produce high quality text which hold factually incorrect information. These systems are black boxes, which make it difficult to validate the knowledge obtained by these models as their reasoning for their choice of prediction is not directly accessible nor explainable to humans [34].

Another prominent issue is that LLMs trained on a general corpus might not be able to generalize well to specific domains or new knowledge. As an LLM only has information to the point it was trained on, any new information will not be known by the model. This would be a significant issue for an e-commerce product recommendation agent who would require real-time information on stock inventory, products, ever-changing trends, user preferences and more. With the use of KGs we can offer accurate explicit knowledge to the LLM. There are several ways in which we can unify the two systems. In this thesis, we will focus on 'KG-enhanced LLM during inference' and 'LLM-Augmented Knowledge Graph Construction'.

### 2.2.3 KG-ENHANCED LLM DURING INFERENCE

This method injects knowledge into our model during inference. By keeping the knowledge space and text space separate we are able to update our knowledge graph with real-world knowledge and use this in our LLMs without retraining the whole model.

### 2.2.4 LLM-AUGMENTED KNOWLEDGE GRAPH CONSTRUCTION

This method leverages large language models to aid in constructing a knowledge graph. It is a systematic process focused on structuring knowledge within a specific domain, which encompasses the identification of entities and their intricate relationships [34]. This multi-faceted undertaking typically involves several key stages, including entity discovery, coreference resolution, and relation extraction.

The effect of these strategies will allow the chatbot to interpret requests from the user and deliver accurate and contextually relevant information to users. With an added centralised repository of information, chatbots can maintain conversational context. By storing and updating information about ongoing conversations, user preferences, and previous interactions, the chatbot can provide a more personalized and consistent user experience, understanding user intents and queries in the appropriate context.

## CHAPTER 3

# BUILDING EXPERIMENTAL LLM CHAT-BOT STRATEGIES

*This chapter delves into the methodology behind the construction of the chatbot, focusing on the data used for training and a comprehensive examination of the selected model. The primary objective of this research is to fine-tune a large language model as a conversational agent. We will delve into the rationale behind the choice of dataset for model training, elucidating the reasons behind this selection. The chapter ends with an evaluation of our fine-tuned model, providing insights into its performance and effectiveness.*

### 3.1 INTRODUCTION

This study investigates various chatbot strategies geared toward delivering personalized product recommendations to users by enhancing contextual awareness within Large Language Models (LLMs). We investigate different fine-tuning approaches, comparing the process of updating the entire model’s parameter weights with freezing certain layers and only updating a small portion of the LLM.

The primary goal of this research is to develop a chatbot that exhibits a high degree of contextual awareness. It achieves this by incorporating external documents during inference and drawing insights from conversational history to generate tailored product recommendations for users. The study employs the open-source LLM model MPT-7B [39] as the foundation, fine-tuning it to specialize as an e-commerce conversational agent.

The subsequent sections are structured as follows: Section 2 presents a literature review encompassing the core topics explored in this study, with a particular focus on

fine-tuning. Section 3 introduces the open-source LLM model, detailing its training data and hyperparameters. Section 4 outlines our fine-tuning methodology. Section 5 evaluates the performance of our chatbot through various evaluation methods. Finally, Section 6 offers a conclusion, along with a discussion of the results and potential suggestions to improve our model.

## 3.2 BACKGROUND

### 3.2.1 FINE-TUNING

Fine-Tuning a foundation model is the act of specialising a model to a specific task using a smaller dataset through modification of the model parameters. The goal of this process is to achieve high performance of our pre-trained model to a specific domain, in this instance as an e-commerce chat assistant.

The most common way of fine-tuning a pre-trained model is to update the weights of the model through retraining it with a smaller dataset. The second technique we will look at is Parameter-efficient fine-tuning (PEFT). This is where only a small subset of the LLM is trained, and the remainder are left untouched. As the parameter count of LLMs have been reaching trillions, fine-tuning the entire model can become very expensive and impractical. This method allows developers to save back on computational power and cost.

A PEFT technique that has recently emerged is Low Rank Adaption, also known as LoRA [23]. This is a technique aimed at mitigating the need of retraining all the parameters in a language model. This process works by allowing some dense layers in a neural network to be trained by optimising the change of rank decomposition matrices during adaption.

The authors of LoRA recognised that pre-trained models have low 'intrinsic dimension' and have the ability to learn efficiently despite projecting to a smaller subspace. With this observation they came up with the following strategy as seen in Figure 3.1. To the left of this figure, the pre-trained weights are frozen whilst on the right, the trainable parameters A and B are used to approximate the gradient update. These gradients are then used to update linear layer, h. Forward pass h, is then modified to be defined as:

$$h = W_0x + \Delta Wx = W_0x + BAx \quad (3.1)$$

where  $W_0$  is the pre-trained weight matrix.

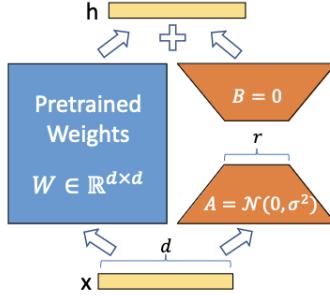


Figure 3.1: LoRA reparametrization process [23]

The most significant benefit of LoRA is the reduction in memory and storage usage. According to the author, they were able to reduce the VRAM consumption during training of GPT-3 [9] 175B from 1.2TB to 305GB. This is a significant reduction of storage by almost 3.5 times. However, based on an experiment carried out by Sun et al. [37], utilising the base model LLaMA-7B [41] they compared the full-parameter version of LLaMA-7B versus LoRA-based fine-tuning. The full-parameter version outperformed LoRA with an average score of 0.710 compared to a score of 0.686. Saying that, fine-tuning the whole model required about 3-5 times the time cost to complete the training according to [37]. Another limitation of LoRA is the inability to batch inputs to different tasks with different A and B's in a single forward pass in a manner that is straightforward.

### 3.3 DATASET

#### 3.3.1 LLM MODEL: MPT-7B

MPT-7B is a Large Language Model (LLM) model that underwent training on an extensive dataset comprising 1 trillion tokens, with a batch size of 1760. The training data was a composite of various datasets, offering a diverse range of sources, as outlined in Figure 3.2. Notably, MPT-7B adopts a decoder-only architecture, also known as an auto-regressive model. In this model, only the decoder portion of the transformer is employed, as depicted in Figure 2.1. This setup consists of a masking mechanism that restricts attention heads to consider only previous words to generate the next token. There are three key attributes that distinguish the MPT-7B architecture from a standard transformer:

1. FlashAttention [16].
2. ALiBi (Attention with Linear Biases) [36]
3. No biases are implemented

Data Source	Number of Tokens in Source		Effective Number of		
		Proportion	Tokens	Epochs	
mC4 3.1.0 - English	417.99 B	0.33	330 B	0.14	
C4 - English - SemDedup 80%	100.42 B	0.299	299 B	2.98	
RedPajama - CommonCrawl	878.45 B	0.1	100 B	0.11	
The Stack - Selected Languages	463.78 B	0.1	100 B	0.22	
RedPajama - Wikipedia - En	4.87 B	0.04	40 B	8.21	
The Stack - Markdown	107.07 B	0.035	35 B	0.33	
S2ORC	48.85 B	0.033	33 B	0.68	
RedPajama - Books	26.02 B	0.03	30B	1.15	
RedPajama - arXiv	28.10 B	0.019	19 B	0.68	
RedPajama - StackExchange	20.54 B	0.014	14 B	0.68	

Figure 3.2: Training data sources [39]

### 3.3.1.1 FLASH ATTENTION

Training transformers is a slow and memory consuming process. The time and memory complexity of self-attention are quadratic in sequence length as described by Dao et al. As we see large language models continue to develop and grow, training will only become more of a problem. The creation of FlashAttention was developed to tackle this. It is an IO-aware attention algorithm that uses tiling to reduce the number of memory reads and writes between GPU's. When evaluated on existing baselines, the author of the paper found that FlashAttention trains transformers up to 15% faster in terms of wall-clock speed up on BERT-large compared to the MLPerf [29] 1.1 training speed record. Furthermore, this algorithm also enables longer context in Transformers which ultimately develops higher quality model capabilities.

### 3.3.1.2 ALIBI

ALiBi, also known as attention with linear biases, presents an innovative approach to the attention mechanism, distinct from the original transformer architecture. Unlike the conventional use of positional embeddings, ALiBi omits this step. This alteration was motivated by the observation that while positional encodings perform well, they tend to

struggle when faced with sequences longer than the length of the pre-training data. ALiBi effectively addresses this challenge by incorporating positional information directly into the transformer’s self-attention mechanism. This is achieved by introducing an additive penalty to the key-query attention score. Notably, this penalty is a static, non-learned bias score, proportional to the distance between a pair of tokens. The distances are then scaled by a fixed scalar multiplier denoted as “ $m$ ,” which remains constant during the training process. The advantage of this approach lies in its improved ability to generalize sequences longer than those in the pre-training data, due to the fact that pairwise distance depends less on the length of the sequence in comparison to the absolute position of tokens within a sequence. [3, 46].

### 3.3.2 LLM-FOUNDRY

LLM-Foundry [39] is a codebase repository made for the purpose of training, finetuning, evaluating, and deploying Language Model Models (LLMs) for inference using Composer and the MosaicML platform. Engineered to be user-friendly, effective, and adaptable, this collection of code was constructed to ease the process of these techniques. We will be using this codebase as a guide to help fine-tune our LLM.

### 3.3.3 DATASET: OPENDIALKG

To fine-tune our model we have chosen DialogStudio’s OpenDialKG dataset [47]. This dataset is built upon conversations between two agents on a given subject. As seen in Figure 3.3, the data consists of 12,302 dialog sessions across two domains. Each dialog turn is paired with a corresponding knowledge-graph path that identifies the KG entities and relationships that arise from the dialog. As these dialogs are formed as a conversation between user and agent for the purpose of product recommendation, fine-tuning this way will allow our model to produce more coherent dialogs for the use-case of an e-commerce chatbot recommendation system. Furthermore, this dataset allows our model to learn how to identify and extract relevant entities, helping both the process of producing personalised recommendations to the user as well as updating our knowledge base; the latter will be discussed in more detail in Chapter 5.

## 3.4 METHODOLOGY

### 3.4.0.1 DATA PRE-PROCESSING

MPT-7B requires their data to be in the format of prompt and response, where the prompt is the question and the response is the systems answer. As OpenDialKG’s dataset does

### Task: Recommendation

Domain:	Movies	Books
# of dialogs	6,429	5,891
# of turns	37,838	34,035

Figure 3.3: OpenDialKG dataset stats [31]

not abide by this structure but instead uses the structure as seen in Figure 3.4, we had to extract the user utterance in the feature log as the prompt, and the system response also from the feature log, as our response. The prompt represents the input that would be provided during inference time, and the latter is the response to the input.

```
{
    "dataset_name--train/val/test--dialog_id": {
        "original dialog id": str,
        "dialog index": int,
        "original dialog info": dict,
        "log": [
            {
                "turn id": int,
                "user utterance": str,
                "system response": str,
                "dialog history": str,
                "original user side information": dict,
                "original system side information": dict,
                "dst": str,
                "dst accumulated": str
                "external knowledge non-flat": list,
                "external knowledge": str,
            },
            ...
        ]
        "prompt": [
            "This is a bot helping users to get navigation.
            Given the dialog context and external database, please
            generate a relevant system response for the user.",
            ...
        ],
        ...
    }
}
```

Figure 3.4: OpenDialKG dataset structure [47]

The dataset has been divided into three distinct segments: training, validation, and test, with an allocation of 80%, 10%, and 10% of the data, respectively. The distribution and number of rows for each set are illustrated in Figure 3.5. Utilizing MPT-7B’s tokenizer, ‘EleutherAI/gpt-neox-20b’ [7], we tokenize each prompt and corresponding response on a row-by-row basis, yielding input\_id’s and attention masks for each.

In order to ensure uniform tensor sizes, thereby enabling simultaneous processing of

all sequences within a batch, we implement two key processes: truncation and padding. Truncation involves shortening sequences to match the maximum allowable length as defined in Table 3.3, while padding entails appending special padding tokens to the end of shorter sequences to reach the same maximum length. Given our intention to leverage the PyTorch library for fine-tuning our LLM, we convert our tensors into PyTorch tensors. Lastly, we remove the original prompt and response columns from our DataFrame, retaining only the tensors for further processing [7].

```
DatasetDict({
    train: Dataset({
        features: ['original dialog id', 'new dialog id', 'dialog index', 'original dialog info', 'log', 'prompt', 'external knowledge non-flat', 'external knowledge', 'dst knowledge', 'intent knowledge'],
        num_rows: 9841
    })
    validation: Dataset({
        features: ['original dialog id', 'new dialog id', 'dialog index', 'original dialog info', 'log', 'prompt', 'external knowledge non-flat', 'external knowledge', 'dst knowledge', 'intent knowledge'],
        num_rows: 1231
    })
    test: Dataset({
        features: ['original dialog id', 'new dialog id', 'dialog index', 'original dialog info', 'log', 'prompt', 'external knowledge non-flat', 'external knowledge', 'dst knowledge', 'intent knowledge'],
        num_rows: 1230
    })
})
```

Figure 3.5: Dataset split for fine-tuning

### 3.4.0.2 FINE-TUNING

For the fine-tuning of our model, we harnessed the capabilities of the Hugging Face Transformers library [45] and LoRA (Low-Rank Adaptation) [23] to configure our model. LoRA represents a method designed to expedite the fine-tuning process for large models while conserving memory resources. It accomplishes this by introducing pairs of rank-decomposition weight matrices to the existing model weights, focusing training efforts solely on these newly added weights.

To ensure compatibility between MPT-7B and LoRA, we initially load our LLM in 8-bit precision. Subsequently, we transform this model into a Composer model using the Composer library, a tool developed by MosaicML explicitly designed to accelerate neural network training and improve generalization.

The configuration encompasses defining various LoRA hyperparameters, as delineated in Table 3.2, along with the training parameters for the fine-tuning process, detailed in Table 3.3. In our fine-tuning endeavor, we opted for a lower learning rate of 3e-4 and a batch size of 2. This choice aligns with common practice in fine-tuning, as it helps prevent overfitting and the loss of pre-trained model features and knowledge by maintaining a relatively smaller batch size compared to the training phase. To monitor the model’s

performance during training, we relied on the WandB (Weights & Biases) [6] machine learning development platform.

The attention mechanism we have chosen to use is the multi-head attention, a configuration in which multiple self-attention layers operate concurrently. This approach enables each attention head to learn a different linguistic phenomenon. Each layer would then output its own filtered value matrix based on different combinations of linguistic features. This in theory should allow the LLM to encode multiple relationships and nuances for each word [20]. We then concatenate the layers and pass them through a linear layer to get an output shape. In our model, we implement this mechanism using PyTorch’s MultiheadAttention module. Specific details of our attention mechanism configuration can be found in Table 3.1.

Attention Configurations	
Attention Type	Multihead Attention
PDrop	0.0
Implementation	Torch
qk_ln	False
clip_qkv	None
softmax_scale	$1/\sqrt{d}$ , d= dimension
prefix_lm	False
attn_uses_sequence_id	False
alibi	True
alibi_bias_max	8

Table 3.1: Attention Configuration Parameters

LoRA Hyperparameters	
Rank	8
Alpha (scaling factor)	16
Dropout	0.05
Bias	None
Target Modules	'Wqkv'
Task type	'CAUSAL_LM'

Table 3.2: LoRA Hyperparameters

Training Hyperparameters	
Batch size	2
Mini batch size	1
Epochs	3
Learning Rate	3e-4
Max Length for tokens	256
Optimiser	AdamW
Warm up step	5
Save steps	50
Logging steps	5

Table 3.3: Training Hyperparameters

### 3.5 RESULTS

Following the implementation of LoRA, Table 3.4 shows the number of parameters in MPT-7B that underwent training during the fine-tuning process. The entire fine-tuning process for MPT-7B on our dataset required a cumulative duration of 11 hours and 7 minutes and 52 seconds.

MPT-7B: Trainable Parameters	
Trainable	4,194,304
Total	6,653,480,960
Trainable (%)	0.06304 (5 s.f.)

Table 3.4: Total trainable parameters in LoRA

Training Results	
Loss	0.03768
Learning Rate	0.00000099
Samples per second	2.52
Steps per second	1.26

Table 3.5: Training Results

Evaluation Results	
Loss	NA
Runtime	283.65
Samples per second	14.60
Steps per second	1.83

Table 3.6: Evaluation Results



Figure 3.6: Training Loss graph

### 3.6 DISCUSSION

The fine-tuned model exhibited suboptimal performance overall. Our total training loss came to a value of 0.03768 (rounded to 5 significant figures). Figure 3.6 provides a visual representation of the training loss execution, which remained consistently low throughout but experienced a significant spike toward the end of the training process. Several factors may contribute to this notably low training loss.

Firstly, the learning rate might have been set too low, impeding the model’s ability to learn effectively and causing slow or negligible progress in learning. Additionally, the small batch size employed during training could be a contributing factor to the observed increase in loss towards the end. A small batch size implies that the model relies on a limited subset of data points for each training iteration. Consequently, any mislabeled or noisy samples within this small batch can disproportionately impact the loss. Utilizing a larger batch size typically mitigates the effects of individual mislabeled samples and leads to a more stable training process.

Secondly, our model encountered difficulties in generating an evaluation loss, as indicated by the absence of an evaluation loss value in Table 3.6.

To further investigate the cause of this, we decided to fine-tune the same LLM but with a different dataset. The dataset we chose to use was the Alpaca-cleaned [38]. This dataset is a clean version of the dataset used to train the Alpaca LLM (Large Language Model). It contains 52,000 instructions and demonstrations generated by OpenAI’s text-davinci-003 engine. This instruction data was used to conduct instruction-tuning for language models and make the language model follow instruction better. The data was formatted to have 3 attributes: instruction, input, and output which is depicted in Figure 3.7.

## Data Fields

The data fields are as follows:

- `instruction`: describes the task the model should perform. Each of the 52K instructions is unique.
- `input`: optional context or input for the task. For example, when the instruction is "Summarize the following article", the input is the article. Around 40% of the examples have an input.
- `output`: the answer to the instruction as generated by `text-davinci-003`.
- `text`: the `instruction`, `input` and `output` formatted with the [prompt template](#) used by the authors for fine-tuning their models.

Figure 3.7: Alpaca Dataset Data Fields

To find the root cause, we kept all configurations consistent with those defined using our initial dataset. As the structure of this dataset was already put into `instruction`, `input`, and `output`, we simply had to separate it into a prompt as depicted in the code in Figure 3.8. We then proceeded with the same tokenization process used for our original dataset.

```
def make_prompt(instruction, input_, output=""):  
    return "{0}\n\n{1}\n\n{2}\n\n{3}\n\n{4}\n\n{5}\n\n{6}".format(  
        """Below is an instruction that describes a task, paired with an input that provides further context.  
        Write a response that appropriately completes the request.""",  
        "### Instruction:",  
        instruction,  
        "### Input:",  
        input_,  
        "### Response:",  
        output  
)
```

Figure 3.8: Preprocessing Alpaca Dataset

The results for fine-tuning with the Alpaca cleaned dataset produced the following training and evaluation results represented in Table 3.7 and Table 3.8, respectively. We observe that this dataset not only managed to produce an evaluation loss but also performed significantly better results than our initial experiment. From Figure 3.9 and 3.10, we notice an overall downward trend throughout the process. This indicates that the model was able to learn and predict.

Training Results - Alpaca Dataset	
Loss	0.8969
Learning Rate	1.2075933469657202e-7,
Samples per second	2.63
Steps per second	1.32

Table 3.7: Training Results - Alpaca Dataset

Evaluation Results - Alpaca Dataset	
Loss	0.92
Runtime	308.32
Samples per second	33.58
Steps per second	4.2

Table 3.8: Evaluation Results - Alpaca Dataset



Figure 3.9: MPT-7B Training Loss for Alpaca Dataset

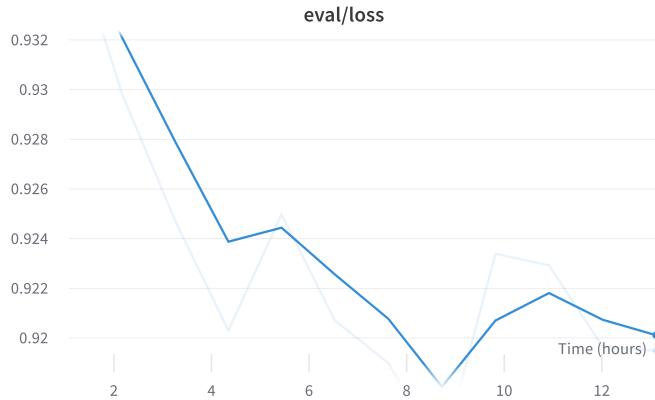


Figure 3.10: MPT-7B Evaluation Loss for Alpaca Dataset

Our investigation into this issue revealed two potential root causes for this anomaly. Firstly, we believe that the dataset was incorrectly preprocessed, which may have caused the model to train incorrectly and unable to perform evaluation. While we successfully generated `input_ids` and `attention_masks`, we inadvertently omitted the creation of corresponding labels. This oversight was identified when we applied the same fine-tuning method to the Alpaca dataset, which not only produced an evaluation score but also exhibited significantly improved performance. Secondly, in comparison to the Alpaca dataset, the OpenDialKG dataset was significantly smaller of 12,000 compared to 52,000. Therefore, it is possible our training hyperparameters were not suitable for the OpenDialKG data.

### 3.6.1 LIMITATIONS

Fine-tuning an LLM posed various limitations, with the primary constraint being insufficient processing power. This limitation promptly eliminated the option of fine-tuning by updating the entire model’s weights. While PEFT LoRA emerged as a viable solution, it introduced additional challenges, particularly because MPT-7B was not initially compatible with LoRA. Furthermore, the limitations in memory resources on our device prevented us from utilizing larger models, such as MPT-30B, which is the larger counterpart of MPT-7B.

Additionally, we encountered limitations in the availability of LLMs that were both open-source and licensed for commercial use. This restricted our choice of models and resulted in a limited variety we could consider. In comparison we found that LLMs that did not meet these criteria often possessed higher capabilities.

Lastly, we could not evaluate our fine-tuned model through adjusting hyperparameters due to insufficient processing power and time constraints. Retraining the model with adjusted parameters would have taken a large portion of time. We relied on Google Colab for running our models, which provided access to GPUs and higher RAM but had limitations on resource allocation without a subscription. However, for future work, we intend to continue refining the model by adjusting parameters to achieve higher performance.

## 3.7 SUMMARY

In summary, we attempted to fine-tune the MPT-7B large language model to possess conversational capabilities. Two primary methods were explored, fine-tuning the whole model’s weights and freezing majority of the model’s layers and updating a small subsection.

During our efforts, we encountered processing power limitations that led us to leverage PEFT LoRA as a solution. Despite successfully executing the fine-tuning process, the results were unsatisfactory. Our loss remained relatively low throughout the training process, ranging from 0 to 0.08 on average, with a significant increase in loss observed towards the end. Moreover, our model failed to produce an evaluation loss.

To investigate the reasons for the poor performance of our fine-tuned model, we con-

ducted a similar fine-tuning process using Alpaca’s clean dataset. This dataset was pre-processed with an ‘input, output, instruction’ format, requiring minimal data processing was needed. We kept all LoRA and training configurations consistent. The fine-tuning process on MPT-7B with this dataset yielded significantly better results, with a loss of 0.897 that had an overall downward trend during training. This model also successfully produced an evaluation loss of 0.92.

Two key factors contributed to these differences in performance. Firstly, incorrect pre-processing of the OpenDialKG dataset. It is possible that our oversight in generating labels caused the model to not learn effectively. The second reason being the configuration of our training hyperparameters. We kept the fine-tuning configurations the same between both datasets but the Alpaca fine-tuned model performed significantly better, we can assume that the dataset size plays a factor in the performance of our model. The OpenDialKG data consists of 12,000 rows in comparison to Alpaca which contains 52,000.

## CHAPTER 4

# DESIGNING DIALOG MANAGEMENT STRATEGIES

*This study explores various machine learning strategies that handle dialog flows to promote a coherent conversation between the user and AI. We will focus on two aspects of dialog management: Context Management and Intent Recognition. In the first strategy, we will look at conversational history and using both memory and attention mechanisms. The latter will delve into RAG (Retrieval Augmented Generation). We examine the advantages and challenges associated with each strategy, determining which is the most appropriate. The chapter ends by performing comparative analysis on the models stated above.*

### 4.1 INTRODUCTION

This chapter conducts a comprehensive examination of dialog management strategies aimed at elevating user satisfaction by effectively discerning user requirements. We delve into context management and intent recognition as mechanisms to support this endeavor. The central goal of this research is to enhance the user experience, and to this end, we aspire to employ methodologies that facilitate contextual communication. This involves identifying the ongoing dialogs subject matter and utilising this to craft the most appropriate response based on evaluation metrics.

The components of dialog management we will focus on are conversational context and conversational history. The subsequent sections of this study will follow this structure: Section 2 will provide a foundational overview of grounding LLMs, with a particular emphasis on the RAG technique as a means of introducing contextual awareness into a conversational agent. Section 3 will introduce the datasets, models, and frameworks

essential for conducting this study. Section 4 will elaborate on the methodology employed for integrating these strategies. Section 5 will present the outcomes of our experimental efforts. Lastly, in Section 6, we will discuss our findings, considering their implications and discussing any limitations faced.

## 4.2 BACKGROUND

Grounding, as defined by Microsoft [5], involves injecting domain-specific information that falls outside the purview of the LLM’s pre-trained knowledge. The primary objective of this process is to develop high-quality, precise, and contextually appropriate dialogs. At present, there exist two popular techniques for incorporating context into LLMs. The first approach entails injecting data directly into a model’s prompt, thereby equipping it to utilize this contextual information when responding to user queries. The second method involves retraining the model by presenting it with a dataset comprising of prompt and completion pairs [24].

However, these approaches have inherent limitations. Firstly, large language models are constrained by a finite context window, restricting their ability to process a large number of tokens in a single iteration. Consequently, if the context passed is larger than the context window, it may lead to sub-optimal performance. Secondly, the challenge of retraining a model to accommodate extended context is twofold, characterized by both its substantial expense and time-intensive nature. This presents a significant drawback when considering the implementation of an e-commerce chatbot, as it requires frequent updates to align with evolving user preferences, market trends, and inventory fluctuations, rendering this methodology impractical for businesses.

Therefore, the ability to access real-time information becomes imperative for the development of an e-commerce chatbot. Enabling data retrieval from external sources, which can be independently updated without requiring comprehensive model retraining, offers a solution to the issues of impracticality and cost, thereby addressing these limitations [28].

### 4.2.1 RETRIEVAL-AUGMENTED GENERATION

To address the limitations discussed earlier, we turn to Retrieval-Augmented Generation (RAG), a fine-tuning technique introduced by Lewis et al. [27]. RAG enables the retrieval of external data from sources beyond the Large Language Model (LLM) by injecting

relevant retrieved information into the LLMs context window. This approach combines both pre-trained parametric and non-parametric memory for language generation. The parametric memory component comprises our pre-trained sequence-to-sequence model, while the non-parametric memory involves a vector index store of external data, accessible through a pre-trained neural retriever.

To comprehend the process of RAG, it is essential to delve into its retrieval mechanism, which relies primarily on semantic search. This method aims to find relevant documents by identifying their similarities instead of just matching keywords between documents. To find these similarities, we convert our data into vector embeddings; these are multi-dimensional numerical representations of the meaning of our inputs. It essentially allows us to quantify how similar the objects are based on their distance in their corresponding vector spaces. Next, we utilise a vector database to store these vectors. By having vector embeddings of our documents it allows for easy indexing to retrieve similar objects to the one being queried. The pipeline of this process is depicted in Figure 4.1. Some popular databases that have emerged over the years include Pinecone [35] and Chroma [12].

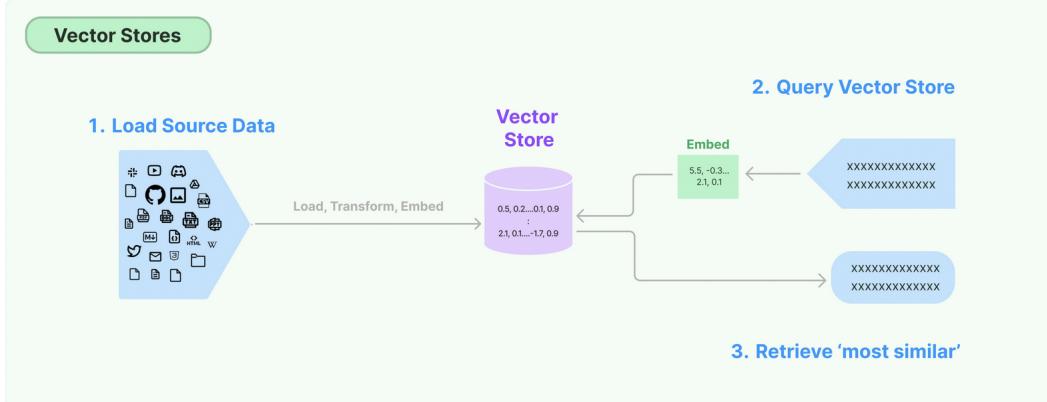


Figure 4.1: Vector Store Pipeline [44]

Furthermore, an essential factor to consider when implementing a vector indexing and retrieval method is the chunking strategy. Chunking involves dividing a large vector into smaller, more manageable chunks, with each chunk representing a subset of the data. The size of these chunks plays a crucial role in determining the quality of the retrieved information, making it imperative to strike a balance that ensures efficiency in both indexing and retrieval operations [5]. The process of using RAG with Large Language Models (LLMs) can be summarized in the pipeline depicted in Figure 4.2.

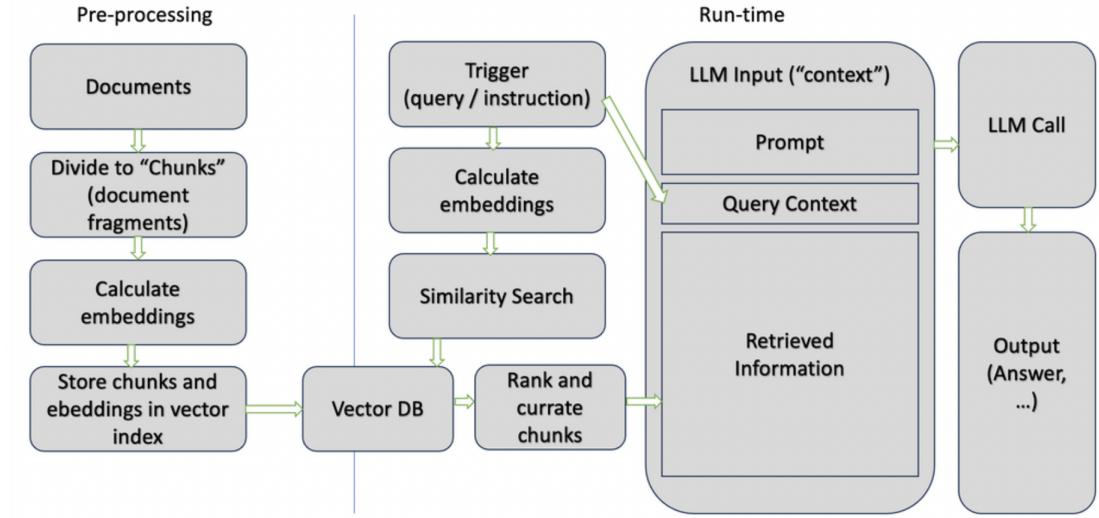


Figure 4.2: RAG Retrieval Process [5]

## 4.3 DATASET

### 4.3.1 RAG DATASET

To implement Retrieval-Augmented Generation (RAG), we have selected a dataset containing fashion products from 'ashraaq/fashion-product-images-small' [2]. This dataset comprises 44,072 fashion products, each of which is associated with multiple label attributes describing the product and includes descriptive text detailing its characteristics. Each product can be identified based on their ID.

#### 4.3.1.1 DATA PREPROCESSING

The fashion products dataset initially included an image field. However, as we have opted not to utilize it, we made the decision to remove this column. Table 4.1 displays the revised dataset attributes.

Attributes	Description
id	Unique identifier
gender	Sex the product was made for
masterCategory	Overall product category
subCategory	Niche product category
articleType	Product type
baseColour	Colour of product
season	Season product is typically worn
year	Year of product release
usage	Occasion when product is typically used
productDisplayName	Product description

Table 4.1: Fashion Dataset Attributes

### 4.3.2 EMBEDDING MODEL

To generate our vector embeddings, we chose to use the 'all-MiniLM-L6-v2' embedding model [4]. This model was specifically designed for tasks such as semantic search and clustering. It operates by mapping sentences and paragraphs to a dense vector space with dimensions of 384. The model's base architecture is derived from 'nreimers/MiniLM-L6-H384-uncased' [30], and it has undergone fine-tuning on a dataset comprising one billion sentence pairs. It was given the objective to predict which randomly sampled set of sentences a given sentence belonged to.

### 4.3.3 LLM MODEL

The LLM we have chosen to use is OpenAI's large language model - GPT-3.5-turbo [21]. This is a model made to understand and generate natural language. We have chosen this model as it has been optimised for chat using the chat completions API. Furthermore, it is also the most cost effective model coming in at a tenth of the price of model 'text-davinci-003'. In regards to specifications, it can take up to a maximum of 4096 tokens, and was pre-trained on data up to September 2021.

## 4.4 METHODOLOGY

In this experiment, we will be utilising the open-source framework Langchain [26]. It is widely known for aiding in developing applications using large language models. The components utilised from this framework include: chains, agents, and tools. In its most generic form, a chain is an end-to-end wrapper around multiple individual components [11]. An Agent is a wrapper around a model, which takes in user input and returns a response corresponding to an "action" to take and a corresponding "action input". [1]. And finally, a tool is a specific abstraction around a function that makes it easy for a language model to interact with it [40].

### 4.4.1 RAG EXPERIMENT

To perform retrieval-augmented generation, our initial step involves the initialization of our embedding model and the creation of our vector database. For the vector database, we establish an index with specific metrics and dimensions to align with our model, utilizing a cosine metric and setting dimensions to 384.

The subsequent step entails encoding our documents. We chose to use the 'productName' field from our dataset as our documents due to its richness in product

information, passing the remaining fields as metadata. To store this information in our vector database, we generate embeddings for our product ID, documents, and metadata, concatenating them together. To optimize efficiency, we process our documents in batches of one hundred at a time, thus saving both time and memory. After adding the vector embeddings to our vector database, Pinecone provides the following statistics: 'dimension': 384, 'index\_fullness': 0.2, 'total\_vector\_count': 44,072, metric: cosine.

To store our vectors, we employ LangChain's vectorStore library and integrate our vector database with it. This integration enables us to retrieve our vectors from Pinecone. To validate this, we conduct a simple similarity search on our vector store, utilizing the query "What black shoes do you recommend for women?" and specifying the return of the three most relevant documents from our database.

#### 4.4.1.1 CONVERSATION AGENT

To create a conversational agent, we use OpenAI's model GPT-3.5-Turbo [21] alongside LangChain's framework to produce an agent that acts as an e-commerce chatbot with the capability of information retrieval when required.

We have selected the 'chat-conversation-react-description' agent for our chatbot implementation [15]. Initializing this agent involves configuring three key components: the retriever, the retriever tool, and the conversational memory. The retriever serves as an interface responsible for returning documents based on unstructured queries. Within the retriever category, we have opted for a retrieval question-answer chain, tailored to assist in answering user queries based on retrieved documents, provided the query is relevant. To facilitate the retrieval process, we have designed a tool with a specific prompt, indicating when to invoke this chain, particularly when users seek product recommendations.

To incorporate conversational memory, we harness the conversational memory library offered by LangChain. This library offers multiple types of memory stores with different functionalities. The conversational memory works by interacting with the chain in two distinct phases: firstly, after the user input but prior to the core logic execution, the chain reads from its memory system and augments the user inputs. Secondly, after the core logic has been executed but before the response is generated, the chain records both the input and output of the current run to memory, enabling future reference.

For our memory store, we opted for the 'ConversationBufferWindow'. This store involves retaining each interaction in the chat history up to a specified limit, denoted as 'k' interactions. This approach ensures that the LLM has access to the maximum amount of information available.

However, this comes with limitations, including slower response times and increased costs due to the larger number of tokens being stored [14]. Alternatively, LangChain also has a 'ConversationSummaryBuffer' memory option which summarizes each new interaction and appends it to an ongoing summary of all past interactions. For our experiments, our default setting will be the 'ConversationBufferWindow' memory store. Nevertheless, we also plan to conduct an experiment to compare the two memory stores and evaluate any impact they might have on the performance of our conversational agent.

## 4.5 RESULTS

Before implementing retrieval-augmented generation in a conversational agent setting, we conducted a preliminary experiment as seen in Figure 4.3 and 4.4 involving a simple similarity search in our vector store to verify its functionality. In this experiment, we instructed the vector store to retrieve the three most relevant documents. When querying the vector store directly, we observed that all the retrieved documents were indeed relevant.

However, we encountered a challenge related to the content within the 'productDisplayName' attribute. This attribute contains very concise information, offering limited contextual details for the large language model to operate effectively.

**Query 1: "What black shoes do you recommend for women?"**

```
1. Document
(
    page content='Reebok Women Black Casual Shoes'
    metadata='articleType': 'Flats', 'colour': 'Black', 'gender': 'Women',
    'season': 'Summer', 'usage': 'Casual'
)

2. Document
(
    page content='Nike Women Black Shoes'
    metadata='articleType': 'Casual Shoes', 'colour': 'Black', 'gender': 'Women',
    'season': 'Fall', 'usage': 'Casual'
)

3. Document
(
    page content='Nike Women Wmn's Black Casual Shoes'
    metadata='articleType': 'Casual Shoes', 'colour': 'Black', 'gender': 'Women',
    'season': 'Fall', 'usage': 'Casual'
)
```

Figure 4.3: Similarity Search Experiment 1

**Query 2: "Do you have any summer clothes?"**

```
1. Document
(
    page content="Disney Kids Girl's Summer Fun Fair Yellow Teen Kidswear"
    metadata='articleType': 'Tshirts', 'colour': 'Yellow', 'gender': 'Girls',
    'season': 'Summer', 'usage': 'Casual'
)

2. Document
(
    page content="Disney Kids Girl's Yellow Summer Fun Fair Kidswear"
    metadata='articleType': 'Tshirts', 'colour': 'Yellow', 'gender': 'Girls',
    'season': 'Summer', 'usage': 'Casual'
)

3. Document
(
    page content='Little Miss Girls Sunshine Red Clothing Set'
    metadata='articleType': 'Clothing Set', 'colour': 'Red', 'gender': 'Girls',
    'season': 'Summer', 'usage': 'Casual'
)
```

Figure 4.4: Similarity Search Experiment 2

#### 4.5.1 EXPERIMENT 1: ASKING THE CONVERSATIONAL AGENT FOR PRODUCT RECOMMENDATIONS

This experiment tests the performance of the conversational agent when asked for product recommendations. We observe the process of the agent as depicted in Figure 4.5, with an output of the memory store at the end to demonstrate how the dialog history is stored.

```

'user': 'What kind of shoes do you recommend for women?',
        "action": "Knowledge Base",
        "action_input": "women's shoe recommendations"

'system': 'Based on the available options, you can consider Reebok Women Black Casual Shoes, Reebok Women White Casual Shoes, Catwalk Women Ballerina Grey Casual Shoe, or Reebok Women Navy Blue Casual Shoes for women's shoe recommendations.'

'user': 'I prefer pink shoes'
        "action": "Knowledge Base",
        "action_input": "women's pink shoes"

Observation: There are several options for women's pink shoes mentioned in the context. Some options include HM Women Pink Sandals, Newfeel Women Pink Shoes, and Catwalk Women Pink Shoes.

'system': 'Based on the information obtained, some options for women's pink shoes include HM Women Pink Sandals, Newfeel Women Pink Shoes, and Catwalk Women Pink Shoes.'

'chat_history':
[HumanMessage (content= 'What kind of shoes do you recommend for women?'),
AIMessage (content= "Based on the given context, some women's shoe recommendations include Reebok Women Black Casual Shoes, Reebok Women White Casual Shoes, Nike Women Wmns Black Casual Shoes, and ADIDAS Women White Sparta Lea Sports Shoes. However, it ultimately depends on your personal preference and the specific type of shoe you are looking for.")]

```

Figure 4.5: Product Recommendation Dialog

#### 4.5.2 EXPERIMENT 2: TESTING MODELS ABILITY TO HOLD MULTIPLE DIALOG FLOWS

In this experiment, we test the conversational agent's ability to hold a conversation unrelated to products, to then abruptly ask the agent for product recommendations to display it's ability to hold multiple dialog flows.

```

'user': 'What do you think of dogs?'
        "system": "As an AI, I don't have personal opinions or emotions, so I don't have thoughts or feelings about dogs. However, dogs are widely regarded as loyal, loving, and intelligent animals. They are often kept as pets and are known for their companionship and ability to form strong bonds with humans. Dogs come in many different breeds, each with its own unique characteristics and traits. Some people may have a preference for certain breeds based on their size, temperament, or specific needs. Overall, dogs are popular pets and are often considered to be a part of the family by their owners."
'user': 'I think Golden Retrievers are the cutest'
        "system": "Golden Retrievers are indeed considered by many people to be cute. They are known for their friendly and gentle nature, as well as their beautiful golden coats. Their expressive eyes and wagging tails often add to their charm. However, cuteness is subjective, and different people may have different opinions on what they find cute. Ultimately, it's a matter of personal preference."
'user': 'I want some casual tshirts'
        "system": "The Lee Grey Stripes Casual T-Shirts and Nike Men's Casual Blue T-shirt are both casual t-shirts."
'user': 'Do you have any green ones?'
        "action": "Knowledge Base",
        "action_input": "green casual t-shirts"
        "system": "The available options for green casual t-shirts are Basics Men Green T-shirt, Status Quo Men Green T-shirt, and Nike Men Premier Green T-shirt."

```

Figure 4.6: Multi-Dialog Experiment

#### 4.5.3 EXPERIMENT 3: CONVERSATIONAL MEMORY

Here, we investigate two distinct types of conversational memory stores: 'ConversationBufferWindow' and 'ConversationSummaryBuffer'. In the case of 'ConversationBufferWindow', we set up our memory to retain the three most recent interactions between the user and the conversational agent. It is worth noting that we set a relatively low number for the purpose of this experiment. In practice, this number would ideally be higher to provide more context.

As for the summarised memory ('ConversationSummaryBuffer'), we've configured it to store a maximum of 20 tokens. Again, this is a relatively small number selected for experimental reasons. We observe how the memory store summarises each dialog interaction as seen in Figure 4.10.

To ensure consistency throughout the experiment, we supply both agents with identical user inputs, models, and prompts. This approach allows us to precisely compare the performance of the two conversational memory stores under controlled conditions.

```

Conversational History Experiment User Inputs

'user': "Hi I would like some women's clothes",
'user': "The pink polo t-shirts sound nice. Do you have more"
'user': 'Can you tell me more about the Flying Machine t-shirt you recommended?'
'user': 'What about pants?'
'user': 'Thanks, I will keep that in mind. The blue harem pants seem suitable for me.'
'user': 'Blue harem pants are very stylish. Can you recommend something along the same lines?'
'user': 'Which tshirt did you recommend again?'

```

Figure 4.7: User Inputs

```

ConversationSummaryBuffer Memory System Outputs

"system": "Based on your previous interest in women's clothing, I recommend the following items: 1. ADIDAS Women Very Berry Lycra Track Pants 2. ADIDAS Women Wo Basic Black Track Pants 3. Puma Women Ess Pink Polo Tshirts 4. Pieces Women Black Handbag. These items are stylish, versatile, and perfect for different occasions. Let me know if you need any further assistance!""

"system": "I apologize for the confusion. Based on your previous interest in pink polo T-shirts, I would recommend checking out the Flying Machine Women Pink Polo T-shirts. They offer a similar style and colour that you might like. Let me know if you need any further assistance or if there's anything else I can help you with!""

"system": "The Flying Machine Men Grey Polo T-shirt and the Flying Machine Men Crown Purple T-shirt are both stylish options that would make a great addition to your wardrobe. They are versatile and can be worn for various occasions. Let me know if you need any further assistance or if there's anything else I can help you with!""

"system": "Based on your previous interest in women's clothing, I recommend the following items: 1. ADIDAS Women Very Berry Lycra Track Pants 2. ADIDAS Women Wo Basic Black Track Pants 3. Puma Women Ess Pink Polo Tshirts 4. Pieces Women Black Handbag. These items are stylish, versatile, and perfect for different occasions. Let me know if you need any further assistance!""

"system": "I'm glad you found the blue harem pants suitable! If you have any more questions or need further assistance, feel free to ask."

"system": "Based on your previous interest in Fabindia Women's Harem Pants, I would recommend the Fabindia Women Blue Harem Pants. They are a stylish and comfortable option that would be a great addition to your wardrobe. Is there anything else I can assist you with?""

"system": "The t-shirt I recommended earlier is the Flying Machine Women Pink Polo T-shirts."

```

Figure 4.8: Outputs with 'ConversationSummaryBuffer' Memory

```

ConversationBufferWindow Memory System Outputs

"system": "Based on your previous interest in women's clothing, I recommend the following items: 1. ADIDAS Women Very Berry Lycra Track Pants 2. ADIDAS Women Wo Basic Black Track Pants 3. Puma Women Ess Pink Polo Tshirts 4. Pieces Women Black Handbag. These items are stylish, versatile, and perfect for different occasions. Let me know if you need any further assistance!"

"system": "Sure! Here are a few more options for women's pink polo T-shirts: 1. Ralph Lauren Women's Classic Fit Pink Polo T-shirt 2. Lacoste Women's Slim Fit Pink Polo T-shirt 3. Tommy Hilfiger Women's Custom Fit Pink Polo T-shirt. These brands offer high-quality pink polo T-shirts that you might like. Let me know if you need any further assistance or if there's anything else I can help you with!"

"system": "Flying Machine offers a variety of stylish t-shirts for men, including the Grey Polo T-shirt and the Crown Purple T-shirt. These t-shirts are known for their quality and trendy designs. They would be a great addition to your wardrobe. Let me know if there's anything else I can help you with!"

"system": "Based on your previous interest in women's pants, here are a few recommendations: 1. Fabindia Women Black Harem Pants, 2. Pepe Jeans Women Black Jeans, 3. ADIDAS Women Wo Basic Black Track Pants, 4. Fabindia Women Blue Harem Pants. Let me know if you'd like more information or if there's anything else I can assist you with!"

"system": "That's great to hear! The blue harem pants from Fabindia would be a stylish and comfortable choice. I'm glad I could help you find something suitable. If you have any more questions or need further assistance, feel free to ask!"

"system": "Based on your previous interest in Fabindia Women's Harem Pants, I would recommend checking out the Fabindia Women's Black, Green, and Pink Harem Pants. These options are similar to the blue harem pants you mentioned and offer a variety of colors to choose from. Let me know if you need any further assistance!"

"system": "The recommendations I provided for harem pants were Fabindia Women Pink Harem Pants, Fabindia Women Black Harem Pants, Fabindia Women Green Harem Pants, and Fabindia Women Beige Harem Pants."

```

Figure 4.9: Outputs with 'ConversationBufferWindow' Memory

```

ConversationSummaryBuffer Memory System History

'chat_history':
[
  SystemMessage(content="

    The human asks for women's clothes and the AI recommends several items based on their previous interest in women's clothing, including stylish and versatile options for different occasions.

    The AI also suggests the Flying Machine Women Pink Polo T-shirts as a similar option to the pink polo T-shirts the human mentioned. In response to the human's inquiry about pants, the AI provides recommendations such as Fabindia Women Black Harem Pants, Pepe Jeans Women Black Jeans, ADIDAS Women Wo Basic Black Track Pants, and Fabindia Women Blue Harem Pants.

    The AI concludes by offering further assistance and asking if there's anything else it can help with. The human expresses gratitude and mentions that the blue harem pants seem suitable.

    The AI responds by expressing gladness and offering continued assistance, recommending the Fabindia Women Blue Harem Pants as a stylish and comfortable option.

    The AI asks if there is anything else it can assist with.")

]

```

Figure 4.10: Conversational History - summarised version

#### 4.5.4 EXPERIMENT 4: PROMPT VS NO PROMPT

In this experiment, we investigated the impact of providing a specific prompt to the large language model compared to not providing any prompt at all.

The prompt given to the LLM was as follows: *"Pretend you're an e-commerce AI assistant tasked with providing product recommendations to the user. Develop a fitting response or suggestion for the user, taking into account the conversation history and context."*

This prompt was designed to guide the LLM in generating responses that align with the role of an e-commerce AI assistant and consider the ongoing conversation history and context. The goal was to assess how the presence or absence of such a prompt influenced the quality and relevance of the model's responses.

**Prompt:**

```
'input': 'What type of black shoes do you have?'
{
    "action": "Knowledge Base",
    "action_input": "black shoes"
}

"system": "Based on your previous interest in Nike Men Black Shoes, I
recommend checking out our latest collection of black shoes for men. We
have a wide range of options from various brands that you might like. Is
there a specific style or brand you are looking for? Let me know, and I'll
be happy to assist you further!"
```

**No prompt:**

```
'input': 'What type of black shoes do you have?'
{
    "action": "Knowledge Base",
    "action_input": "black shoes"
}

"system": "The shoes mentioned in the context are black shoes."
```

Figure 4.11: Prompt comparison dialog

## 4.6 DISCUSSION

### 4.6.1 EXPERIMENT 1

In this experiment, we tasked the model with generating product recommendations specifically for women’s shoes. The model effectively engaged with the knowledge base tool, where it provided an action input of ‘women’s shoe recommendations.’ This action input serves as a concise summary of what the large language model (LLM) deems relevant to retrieve the necessary knowledge.

The outcomes of this experiment were favourable, as the recommendations generated by the model were both relevant and accurate. This showcases the model’s ability to understand the task at hand, communicate effectively with the knowledge base tool, and provide valuable product suggestions in the specified category of women’s shoes.

### 4.6.2 EXPERIMENT 2

In our second experiment, we assessed the systems ability to engage in a conversation and seamlessly transition to producing product recommendations. We observed that the model effectively demonstrated this ability.

Here is a summary of our findings:

1. **Transitioning from Conversation to Recommendations:** We initiated the conversation with a random question posed by the user. The model was able to recognize this as a conversational context and did not make any additional queries to external knowledge sources. Instead, it relied on its pre-training data to respond effectively.
2. **Context Retention and Relevance:** As seen in Figure 4.6, our experiment demonstrated the model’s skill in retaining the conversational history and staying contextually relevant. For instance, when the user input was ”Do you have any green ones?” it did not explicitly mention ”t-shirts.” However, based on the ongoing conversation history, the agent successfully identified the user’s intent and, queried the knowledge base with an action input of ‘green casual t-shirts.’ This showcased the model’s ability to grasp user intent and provide tailored responses.

These findings highlight the model’s competence in maintaining conversational context

and seamlessly transitioning between conversation and recommendation modes, demonstrating its ability to support multiple dialog flows.

#### 4.6.3 EXPERIMENT 3

In our third experiment, we conducted a comparison between two different memory stores: 'ConversationSummaryWindow' and 'ConversationBufferWindow.' The results of this experiment revealed some important insights:

1. **Response Time Comparison:** 'ConversationSummaryWindow' exhibited faster response times compared to 'ConversationBufferWindow.' The summarization method stores the conversational history as a concise summary, leading to quicker response generation. 'ConversationBufferWindow' stores the entire dialogue, which contributed to slower response times. On average, it took approximately 60 seconds to generate a response with this memory store, compared to 26 seconds with the summarization approach.
2. **Quality of Responses:** In comparison, 'ConversationSummaryWindow' provided faster responses alongside an output that was more concise. However, this brevity may have contributed to less comprehensive responses. Conversely, 'ConversationBufferWindow' which stored the entire conversation, generated responses that were comparatively better. It had the ability to recall past information and produce more contextually relevant responses.
3. **Limitations of 'ConversationBufferWindow':** Despite its advantages, 'ConversationBufferWindow' had limitations, especially in terms of conversation length. In this experiment, it struggled to recall specific details mentioned at the beginning of the conversation. Even with an extended memory, the system will still face limitations in assisting with inquiries related to events that occurred early in the conversation.

To conclude, this sub-experiment aimed to compare response times and response quality between the summarization-based memory store ('ConversationSummaryWindow') and the full dialogue memory store ('ConversationBufferWindow'). The results showed that while summarization led to quicker responses, the full dialogue memory store produced more detailed and contextually relevant responses in some instances. These findings provide valuable insights into the trade-offs between response speed and response quality when choosing a memory store for a conversational agent.

#### 4.6.4 EXPERIMENT 4

In our fourth experiment, we explored the impact of providing a prompt to the chatbot versus not providing one. The results of this experiment revealed a significant difference in the style and output of our conversational agent.

Here is what we observed:

1. **With Prompt:** As seen in Figure 4.11, when we provided a helpful prompt such as "*Pretend you're an e-commerce AI assistant tasked with providing product recommendations to the user*", along with a user request for black shoe recommendations, the system responded in a helpful manner. It provided a suggestion for black shoes and followed up with a question to clarify the user's specific needs. This approach demonstrated an understanding of the user's query and a commitment to providing useful information.
2. **Without Prompt:** In contrast, when we didn't provide a prompt and presented the same user request for black shoe recommendations, the conversational agent produced an unhelpful and unfavorable response. This outcome underscored the significance of prompts in guiding the large language model (LLM) to generate useful and relevant responses.

These findings highlight the importance of using prompts to shape the behavior and utility of the LLM. A well-crafted prompt can help ensure that the model understands its role and context, leading to more valuable interactions with users.

#### 4.6.5 LIMITATIONS

While more advanced models like OpenAI's 'text-embedding-ada-002' are indeed available, it's worth noting that their utilization would necessitate a paid subscription, a resource allocation beyond the purview of this experiment. Additionally, constraints related to limited memory resources prevented us from conducting RAG testing on our initial Large Language Model (LLM), namely MPT-7B.

### 4.7 SUMMARY

In summary, this experiment investigated dialog management strategies as a way to improve contextual awareness and create a coherent and engaging conversational agent. We focused on two key dialog components: intent recognition and context management.

To improve intent recognition, we explored prompt-tuning and integrated conversational history into the context of large language models. For context management, we implemented retrieval-augmented generation by creating a vector store containing external documents for the model to retrieve during inference. To trigger document retrieval, we provided specific prompts to identify when the user required assistance with product recommendations.

We conducted four sets of experiments to assess the conversational agent’s capabilities. These experiments covered the agent’s ability to retrieve documents from the vector database, handle multiple dialog flows, incorporate conversational history into decision-making, and evaluate the impact of prompt-engineering on the model’s understanding of its role as an e-commerce chatbot.

Overall, the results of this experiment were positive, leading to the development of a contextually aware conversational agent capable of generating coherent and friendly interactions with users.

# CHAPTER 5

# KNOWLEDGE-BASE UPDATE MECHANISMS

*This chapter explores various mechanisms to update knowledge bases with relevant information from the dialog between a user and conversational agent. An analysis will be performed on leveraging knowledge graphs with large language models during the point of inference. We will look at various NLP and Machine Learning techniques to aid in this experiment. This chapter ends by evaluating the performance of unifying a knowledge graph and large language model.*

## 5.1 INTRODUCTION

This study delves into a variety of strategies regarding knowledge graph updates by extracting information from large language models. We look at the different approaches to unify these objects with the goal of enhancing decision making in LLMs.

The aim of this study is to find ways to update a knowledge graph by extracting information from dialogs to track user preferences and link them to available products. By keeping a store of this information it will provide each user with a personalised experience through storing their information.

The subsequent sections of this study will follow this structure: Section 2 will provide background information on the unification of Knowledge Graphs and LLMs, highlighting the advantages of this integration. We will also examine the process of extracting unstructured data and converting it into structured data. In Section 3, we introduce the datasets and frameworks used to carry out this experiment. Section 4 discusses the methodology

to implement our information extraction pipeline, establish a knowledge graph, and subsequently update the knowledge graph with this extracted data. Our experimental findings will be presented in Section 5. Finally, Section 6 will offer a summary of this study, encompassing a discussion on our results, and the encountered limitations.

## 5.2 BACKGROUND

Knowledge Graphs excel in their ability to systematically capture and arrange information in a structured format. This not only facilitates efficient navigation, organization, querying, and exploration of connections between various entities but also underscores their commitment to upholding data quality and precision. They achieve this through the incorporation of validation and integrity mechanisms, which, in turn, fosters consistency throughout the graph. These attributes play a pivotal role in decision-making and knowledge-based applications.

However, it's essential to acknowledge that knowledge graphs are bounded by the information available at the time of their creation. While it is possible to update the graph subsequently, there is no guarantee of capturing the entirety of domain knowledge. This inherent limitation can impede the decision-making process when responding to user queries. Another constraint lies in knowledge graphs' inability to comprehend language and unstructured data.

As discussed in our literature review, Large Language Models (LLMs) exhibit a range of limitations. To begin, they are susceptible to hallucination, wherein they generate factually incorrect information. Additionally, their lack of domain-specific knowledge can be attributed to their pre-training data, rendering them of having information outside of their training data.

Considering the limitations inherent in both entities, their integration offers potential solutions to mitigate each other's shortcomings. As elucidated in Chapter 2, various strategies exist for unifying Large Language Models (LLMs) and Knowledge Graphs to create a conversational agent with enhanced capabilities and overall performance. This study concentrates on two key aspects: firstly, the utilization of Knowledge Graphs to enhance LLMs during the inference process, and secondly, LLM-Augmented Knowledge Graphs to aid in the construction and expansion of the graph itself.

In our particular use-case, leveraging a knowledge graph in conjunction with a Large Language Model (LLM) presents the opportunity to seamlessly incorporate external company documents into the LLM during inference, enabling us to proficiently address user inquiries. This encompasses responding to inquiries about product availability and delivering product recommendations derived from the external documents. Furthermore, the unification of these systems provides us the ability to extract user preferences from the conversations conducted with the conversational agent, subsequently constructing a knowledge graph that links these preferences to various products. In summary, this integrated approach yields a chatbot that is inherently contextually aware, predictably responsive, and tailored to the unique needs of users.

### 5.2.1 INFORMATION EXTRACTION PIPELINE

A strategy we explore for extracting unstructured data like a chatbot dialog to structured data is the use of an information extraction pipeline. This consists of using various NLP techniques to produce structured data to be inserted into a knowledge graph. As seen in Figure 5.1, this pipeline consists of three main steps. It begins by taking our unstructured input data and processing it through a coreference resolution model. Coreference resolution is the process of converting pronouns into referred entities [8], an example is when the user says 'I would like a pink t-shirt.'. The coreference resolution model would link the word 'I' to the user conversing with the agent.

Moving along the pipeline, the subsequent stage is named entity recognition (NER). This critical step involves the identification of all entities within the input text. The specific types of entities recognized are dependent upon the use-case at hand. In our case, the focus will be on identifying entities related to user preferences regarding products. Finally, the pipeline ends with relationship extraction, where we are able to discern and identify any relationships that may exist within the text.

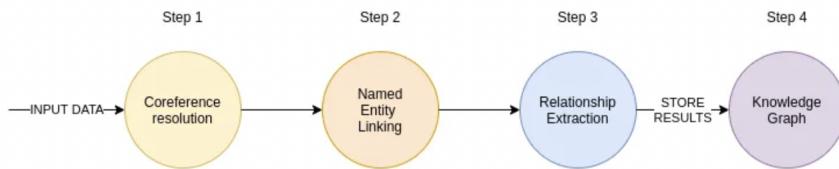


Figure 5.1: Information Extraction Pipeline

## 5.3 DATASET

This study will use the same dataset and LLM as Chapter 4: 'ashraq/fashion-product-images-small' [2] and OpenAI's large language model - GPT-3.5-Turbo [21]. For the purpose of this study, we utilised the CSV version of this dataset.

### 5.3.1 FRAMEWORKS

This section introduces the frameworks and libraries used to carry out this experiment. Neo4j [32] is a graph database management system, we will be using this as our knowledge graph. It uses a query language known as Cypher to interact with knowledge graphs. This query language was inspired from SQL.

Additionally, we will be incorporating SpaCy [22] into our toolkit. SpaCy is an open-source Python library renowned for its skills in Natural Language Processing (NLP). Among its capabilities are named entity recognition, entity linking, tokenization, and more. In this study, we will use this library to assist in the construction of our information extraction pipeline.

## 5.4 METHODOLOGY

### 5.4.1 KNOWLEDGE GRAPH CREATION

To construct our knowledge graph populated with fashion products, we've employed Cypher statements to extract each header in our dataset as an individual node. Subsequently, we've established relationships to connect these nodes. The nodes that have been extracted include: Colour, Gender, Product, Season, Year, ArticleType, SubCategory, MasterCategory, and Occasion. These nodes have been interconnected by establishing the relationships outlined in Table 5.1. For a visual example of a product node linked to its associated details, please refer to Figure 5.2.

### 5.4.2 RETRIEVING EXTERNAL DATA DURING INFERENCE

To allow the conversational agent to access our knowledge graph during inference, we make use of Langchain to create a Cypher search chain. This is a chain to help us generate Cypher statements. As this chain is relatively generic, we created a few-shot prompt providing examples of how to generate statements based on particular questions. Figure

Entity 1	Relationship	Entity 2
Product	FOR_GENDER	Gender
Product	FOR_SEASON	Season
Product	FOR_USAGE	Occasion
Product	HAS_ARTICLETYP	ArticleType
Product	HAS_COLOUR	Colour
Product	MADE_IN	Year
Product	HAS_CATEGORY	MasterCategory
MasterCategory	HAS_SUBCATEGORY	SubCategory

Table 5.1: Dataset Nodes and Relationships

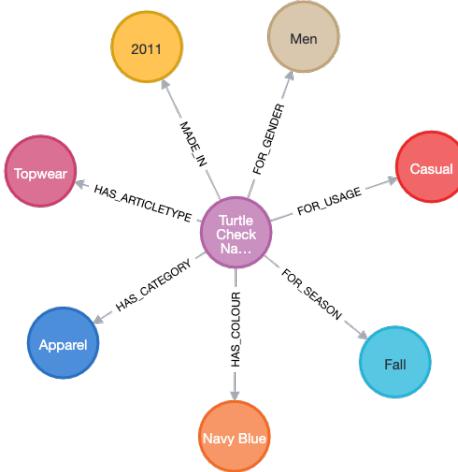


Figure 5.2: Product details

5.3, shows the prompt template provided to our chain.

To enable question-answering capabilities against our knowledge graph, we use Langchain's chain 'GraphCypherQACChain'. We initialise this chain to use the following features: GPT-3.5-turbo as it's LLM, our populated knowledge graph, the Cypher prompt template as seen in Figure 5.3, and a separate question-answering prompt as seen in Figure 5.4 to make our LLM better suited as an e-commerce chatbot assistant.

To include conversational history as context during an interaction with a user, we utilised the same conversational agent we created in Chapter 4 with a few minor adjustments. Instead of using the 'ConversationBufferWindow' memory, we changed this to 'ConversationBufferMemory' memory. We chose this memory store as it saves the whole dialog between the user and agent, whereas the other memory store would only store up to k previous interactions. Finally, to include our Cypher chain into the agent, we turned this chain into a tool with the instruction of utilising it when providing product recommendations to users'.

```

CYPHER_GENERATION_TEMPLATE = """Task:Generate Cypher statement to query a graph database.
Instructions:
Use only the provided relationship types and properties in the schema.
Do not use any other relationship types or properties that are not provided.
Schema:
{schema}
Note: Do not include any explanations or apologies in your responses.
Do not respond to any questions that might ask anything else than for you to construct a Cypher statement.
Do not include any text except the generated Cypher statement.
Examples: Here are a few examples of generated Cypher statements for particular questions:

# What black shoes do you have for women?
MATCH (p:Product)-[:HAS_COLOUR]->(:Colour {{colour: 'Black'}})
MATCH (p)-[:FOR_GENDER]->(:Gender {{gender: 'Women'}})
RETURN p.product AS Product

# Do you have clothes for fall?
MATCH (p:Product)-[:FOR_SEASON]->(:Season {{season:'Fall'}})
RETURN p.product as Product

# I would like some casual t-shirts for men.
MATCH (p:Product)-[:FOR_USAGE]->(:Occasion {{occasion: 'Casual'}})
MATCH (p)-[:FOR_GENDER]->(:Gender {{gender: 'Men'}})
MATCH (p)-[:HAS_ARTICLETYPE]->(:articleType {{}})
RETURN p.product as Product

# Blue Tshirt
MATCH (p:Product)-[:HAS_ARTICLETYPE]->(:articleType {{articleType: 'Tshirts'}})
MATCH (p)-[:HAS_COLOUR]->(:Colour {{colour:'Blue'}})
RETURN p.product as Product

The question is:
{question}"""

```

Figure 5.3: Cypher Generation Prompt Template

```

prompt_template = """Pretend you're an e-commerce AI assistant tasked with providing product recommendations to the user.
Develop a fitting response or suggestion for the user, taking into account the conversation history and context.

{context}

Question: {question}
"""

qa_prompt = PromptTemplate(
    template=prompt_template, input_variables=["context", "question"]
)

```

Figure 5.4: Question-Answering Prompt

## 5.5 RESULTS

In Figure 5.5, we provide an illustration of a graph that showcases the structure of data in a knowledge graph. This graph represents a few products and their respective associations with colours, offering a visual representation of how information is organized within the knowledge graph.

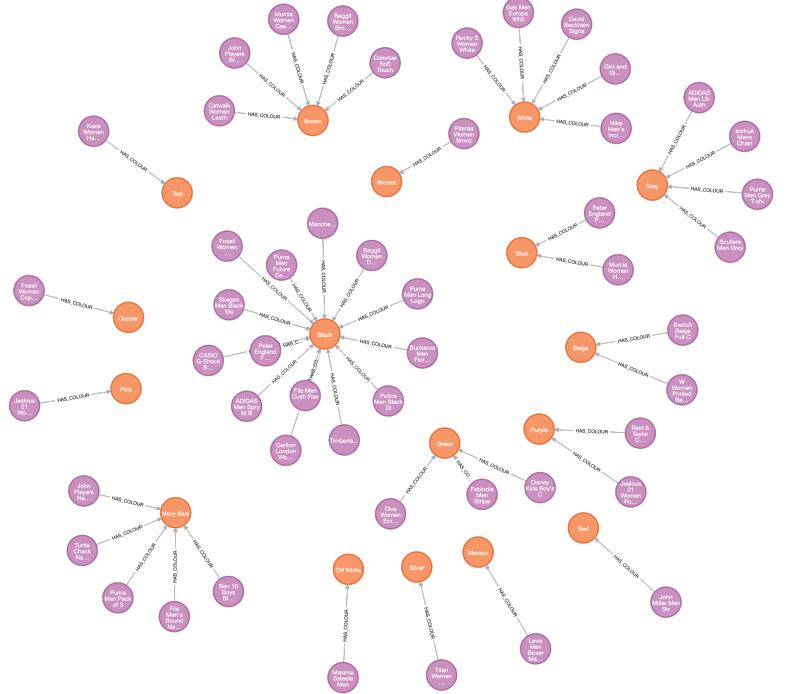


Figure 5.5: Knowledge Graph instance of Product and Colour relationship

### 5.5.1 EXPERIMENT 1: KNOWLEDGE RETRIEVAL USING CYpher CHAIN

In this experiment, we evaluated the capabilities of the Cypher Chain's model by presenting it with a user statement expressing the intent to purchase shoes. We closely observed the steps that the model underwent, including the generation of Cypher queries to retrieve relevant information. The experiment concluded with the model providing a response to fulfill the user's request for purchasing shoes. This process allowed us to assess the model's proficiency in understanding user queries, constructing meaningful Cypher queries, and delivering appropriate responses to user requests with the input of a knowledge graph.

```

Query to cypher chain:

'User': 'I would like to buy some black men shoes'

> Entering new GraphCypherQACChain chain...

Generated Cypher:

MATCH (p:Product)-[:HAS_COLOUR]->(:Colour {colour: 'Black'})
MATCH (p)-[:FOR_GENDER]->(:Gender {gender: 'Men'})
RETURN p.product AS Product

Full Context:
[{'Product': 'Nike Men Black Sweatshirt'},
 {'Product': 'Quiksilver Men Black & White Messenger Bag'},
 {'Product': 'Classic Polo Men Printed Black T-shirt'},
 {'Product': 'Fila Men Fusion Black Sports Shoes'},
 {'Product': 'ADIDAS Originals Men Chord Black Casual Shoes'}]

> Finished chain.

'Response': Based on your preference for black men's shoes, I have a few
recommendations for you. We have the "Fila Men Fusion Black Sports Shoes"
and the "ADIDAS Originals Men Chord Black Casual Shoes" available. Both of
these options are stylish and comfortable. Let me know if you would like
more information or if there's anything else I can assist you with.'

```

Figure 5.6: Knowledge Retrieval using Cypher Chain Result

### 5.5.2 EXPERIMENT 2: AGENT COMPARISON

In this experiment, depicted in Figure 5.7, we conducted a comparative analysis between two different agents. The first agent was the Cypher Chain agent, while the second agent was the conversational agent introduced in Chapter 4, which integrated the Cypher Chain agent as a tool.

The motivation behind this experiment stemmed from the observation that the Cypher Chain agent, while capable of storing memory, encountered difficulties in effectively utilizing its memory within its context when responding to user queries. By comparing the performance of these two agents, we aimed to gain insights into their abilities to leverage stored information and provide more contextually relevant responses to user inquiries.

```

Conversational History Experiment User Inputs

'User': "I'm looking for casual pink dresses"
'User': "What about blue ones?"

Cypher chain outputs

'System': "Based on your preference for casual pink dresses, I have a few recommendations for you. Here are some options you might like:
1. Doodle Girls Pink Printed Dress
2. Remanika Women Printed Pink Dress
3. Doodle Kids Girl Printed Pink Dress

Please let me know if you would like more information about any of these dresses or if there's anything else I can assist you with."

'System': "Based on your previous interest in blue products, I have a few recommendations for you: 1. W Women Blue Kurta: This kurta is a stylish and comfortable option for women who love the colour blue. 2. Nike Men's Zoom Ambassador II Blue White Shoe: These shoes are perfect for men who want a sporty and trendy blue footwear option. 3. Gini and Jony Boys Washed Blue 3/4 Length Pants: If you're looking for blue pants for boys, these washed blue 3/4 length pants are a great choice.

Let me know if you'd like more information about any of these products or if there's anything else I can assist you with!"

Conversational agent with cypher chain as a tool output

'System': "Here are some casual pink dresses that you might like: 1. Doodle Girls Pink Printed Dress 2. Remanika Women Printed Pink Dress 3. Doodle Kids Girl Printed Pink Dress.

Let me know if you need more information or if there's anything else I can assist you with!"

'System': "Based on your previous conversation, I have found some casual blue dresses that you might like: 1. Gini and Jony Girls Blue Dress 2. Arrow Woman Kaya Blue Dress 3. French Connection Women Blue Dress.

Let me know if you would like more information about any of these dresses or if there's anything else I can assist you with!"

```

Figure 5.7: Comparison of conversational agents results

### 5.5.3 EXPERIMENT 3: CONVERSATIONAL AGENT RETRIEVAL CAPABILITIES

Based on the findings from Experiment 2, where the conversational agent that utilized the Cypher Chain as a tool outperformed other configurations, it was decided to proceed with this model for the current experiment.

The objective of this experiment is to demonstrate the conversational capabilities of the agent in two distinct scenarios. Additionally, Figure 5.8 has been included to visually depict the steps involved in the process of retrieving knowledge from the knowledge graph. This graphical representation offers a clear overview of the sequence of actions the agent takes when answering a users query through accessing the knowledge in the knowledge graph, enhancing the understanding of its functioning.

```

'User': "Hi, I want to buy some jeans"
> Entering new AgentExecutor chain...
{
    "action": "Cypher Search",
    "action_input": "jeans"
}
> Entering new GraphCypherQAChain chain...
Generated Cypher:
MATCH (p:Product)-[:HAS_ARTICLETYPE]->(:articleType {articleType: 'Jeans'})
RETURN p.product as Product
Full Context: [{"Product": "Lee Men Light Blue Chicago Fit Jeans"}, {"Product": "Spykar Women Washed Blue Jeans"}, {"Product": "Flying Machine Men Blue Jeans"}]
> Finished chain.

Observation: Based on your previous conversation, I have some jeans recommendations for you. Here are a few options: 1. Lee Men Light Blue Chicago Fit Jeans 2. Spykar Women Washed Blue Jeans 3. Flying Machine Men Blue Jeans Let me know if you would like more information about any of these jeans or if there's anything else I can assist you with!

Thought:
{
    "action": "Final Answer",
    "action_input": "Based on the recommendations, you can consider purchasing the Lee Men Light Blue Chicago Fit Jeans, Spykar Women Washed Blue Jeans, or Flying Machine Men Blue Jeans. Let me know if you need more information about any of these options or if there's anything else I can help you with!"
}
> Finished chain.

'system': "Based on the recommendations, you can consider purchasing the Lee Men Light Blue Chicago Fit Jeans, Spykar Women Washed Blue Jeans, or Flying Machine Men Blue Jeans. Let me know if you need more information about any of these options or if there's anything else I can help you with!"

```

Figure 5.8: Conversation Chain Execution Steps

```

'User': "Hi, I want to buy some jeans"
Generated Cypher:
MATCH (p:Product)-[:HAS_ARTICLETYPE]->(:articleType {articleType: 'Jeans'})
RETURN p.product as Product

Full Context: [{"Product": "Lee Men Light Blue Chicago Fit Jeans"}, {"Product": "Spykar Women Washed Blue Jeans"}, {"Product": "Flying Machine Men Blue Jeans"}]

[System]: "Based on the recommendations, you can consider purchasing the Lee Men Light Blue Chicago Fit Jeans, Spykar Women Washed Blue Jeans, or Flying Machine Men Blue Jeans. Let me know if you need more information about any of these options or if there's anything else I can help you with!"

'User': 'Do you have any more for women?'
Generated Cypher:
MATCH (p:Product)-[:HAS_ARTICLETYPE]->(:articleType {articleType: 'Jeans'})
MATCH (p)-[:FOR_GENDER]->(:Gender {gender: 'Women'})
RETURN p.product as Product

Full Context: [{"Product": "Pepe Jeans Women Blue Jeans"}, {"Product": "Jealous 21 Women's Blue Jegging"}, {"Product": "Lee Women Scraped Blue Roxy Fit Jeans"}]

[System]: "Here are a few more recommendations for women's jeans: 1. Pepe Jeans Women Blue Jeans 2. Jealous 21 Women's Blue Jegging 3. Lee Women Scraped Blue Roxy Fit Jeans. Let me know if you need more information about any of these products or if there's anything else I can assist you with!"

```

Figure 5.9: Conversational Chain Experiment 1

```

'User': 'Hi, how are you?'
[System]: "I'm an AI language model, so I don't have feelings, but I'm here to help you. How can I assist you today?"
'User': 'I am looking to buy a pair of white sneakers'
Generated Cypher:
MATCH (p:Product)-[:HAS_COLOUR]->(:Colour {colour: 'White'})
MATCH (p)-[:HAS_ARTICLETYPE]->(:articleType {articleType: 'Sneakers'})
RETURN p.product AS Product

Full Context: []
[System]: "Based on your preference for white sneakers, I recommend considering popular brands like Nike, Adidas, and Converse. These brands offer a wide range of white sneakers in various styles, including athletic and casual options. You can find these sneakers at local shoe stores or online retailers such as Amazon, Zappos, or the official websites of the respective brands. When searching, make sure to specify your size and any other preferences you may have. Happy shopping!"

```

Figure 5.10: Conversational Chain Experiment 2

## 5.6 DISCUSSION

### 5.6.1 EXPERIMENT 1

In our test of the Cypher Chain implementation, we initiated a straightforward query to the chain, conveying the user’s desire to purchase black shoes. The Cypher Chain demonstrated its competence by effectively generating a Cypher statement designed to retrieve black men’s shoes from the knowledge graph. Moreover, the response provided by the Cypher Chain was characterized by its friendly and assistant-like demeanor, signifying our success in prompt tuning the large language model (LLM) to perform in a manner that aligns with the role of an assistant. This successful outcome validates the utility and adaptability of the Cypher Chain for specific tasks and showcases the positive impact of prompt tuning in achieving the desired conversational style and functionality.

### 5.6.2 EXPERIMENT 2

In our methodology section, we discussed a limitation of the Cypher Chain, which was its inability to effectively use the conversational history as context during a session with the user. To investigate this limitation further, we designed an experiment that required the user’s past interactions with the agent to correctly address the user’s query.

From the results presented in Figure 5.9, we observed that the Cypher LLM’s response to the follow-up question, ”What about blue ones?” did not recognize the meaning of the user’s request for blue dresses as opposed to just blue products.

However, when we integrated the Cypher Chain with a conversational agent, we achieved a different outcome. The system’s response included the phrase ”Based on your previous conversation...”, indicating that conversational history was used as context. Subsequently, the system provided relevant suggestions for blue dresses. This result demonstrated that by wrapping the Cypher Chain within a conversational agent, we were able to effectively leverage previous interactions as context in subsequent parts of the dialogue, addressing the limitation observed with the standalone Cypher Chain model.

### 5.6.3 EXPERIMENT 3

Based on the findings from experiment two, where it was established that wrapping the Cypher Chain model with a conversational agent allowed for the effective utilization of conversational history within the LLM and Cypher search, this method will be adopted

as the default agent for the remaining experiments.

Figure 5.8 provides a visual representation of the intermediate steps involved when a user submits a query. The agent follows a sequence that includes entering both the conversational chain and the graph query chain. It retrieves knowledge from the knowledge graph within the graph chain, then exits this chain and returns to the conversational chain to produce the final response to the user, utilizing the retrieved knowledge.

In the experiments conducted, the model showcased its conversational and retrieval capabilities through two separate dialog chains. As demonstrated in Figure 5.9, the model effectively generated a Cypher query to retrieve products based on the user’s query. This dialog also illustrated the model’s ability to incorporate conversational history into its decision-making process.

However, Figure 5.10 presented a scenario where the Cypher query failed to retrieve any ‘white sneakers’ products. In such cases, the LLM autonomously generated its own responses for the user. While this can be advantageous in some situations, it raises concerns in an e-commerce context where the agent is expected to act in the company’s best interests, making it unfavorable to recommend users to competitors’ websites. This highlights the importance of robust knowledge retrieval mechanisms to ensure meaningful and on-brand responses in all scenarios.

## 5.7 SUMMARY

This chapter focused on the integration of Large Language Models (LLMs) with Knowledge Graphs, particularly how knowledge graphs can enhance LLMs during inference.

To conduct this experiment, we constructed a knowledge graph using an open-source dataset containing fashion products. The creation of this knowledge graph involved identifying different entities and establishing their relationships with each other.

To enable the LLM to retrieve information from our knowledge graph during inference, we developed two agents. The first agent, a Cypher search agent, was responsible for generating Cypher statements to retrieve products from the knowledge graph. The second agent was designed to engage in conversations with users in the role of an e-commerce chatbot.

Three experiments were conducted to assess the performance of the LLM when unified with the knowledge graph. The first experiment tested the Cypher search chain’s ability to retrieve knowledge from the graph. The second experiment compared the two agents to determine which was more effective in meeting user needs. The final experiment examined the conversational agent’s performance in various dialog scenarios.

Overall, this study successfully demonstrated the retrieval of information from the knowledge graph during inference. However, there are instances where the large language model is unable to generate accurate Cypher statements according to the users input. Furthermore, we find that the conversational agent when unable to generate a Cypher statement for retrieving products, tended to hallucinate an answer to the user. This is potentially due to the prompt template provided to the agent, and for future work can be solved by tuning this prompt.

# CHAPTER 6

## CONCLUSION AND FUTURE WORK

*This chapter concludes the thesis and revisits the key findings of its studies, highlights the achievements and contributions of this thesis, and provides recommendations for future work to be carried on by other researchers.*

### 6.1 CONCLUSION

The significance of this research is encapsulated in the title of the thesis, "Multi-Dialog Chatbots: Knowledge Base Updates for Enhanced User Preference Support." This title aptly represents the wide range of topics and areas explored within this research.

The research conducted in this thesis is structured in a sequence that provides progressive insights and results, ultimately aimed at enhancing the capabilities of multi-dialog chatbots and the management of knowledge bases. The journey begins with the exploration of various strategies for constructing a chatbot tailored to serve as a personal assistant in an e-commerce environment. It then delves into a deeper examination of methods designed to enhance contextual awareness and improve decision-making support within Large Language Models (LLMs).

In this chapter, we conclude this thesis by summarizing the motivations behind the research, highlighting the key findings, and concluding with recommendations for future work in this dynamic and evolving field.

#### 6.1.1 BUILDING EXPERIMENTAL LLM CHATBOT STRATEGIES

The thesis commences with an exploration of various fine-tuning strategies for large language models (LLMs) with the aim of adapting them to function as a conversational agent.

The following two primary strategies were investigated:

1. **Fine-Tuning with Full Model Updates:** Initially, the approach involved fine-tuning the model using a smaller dataset and updating all of the model's weights. However, due to limitations in processing power and resource constraints, this approach quickly became impractical.
2. **Layer-Freezing and Selective Parameter Updates:** In response to the limitations faced with the first strategy, the research shifted towards an alternative approach. This involved freezing the majority of the layers within the LLM and updating only a small set of trainable parameters. While this strategy produced less favorable results, it provided valuable insights into the intricacies of fine-tuning LLMs.

Key takeaways from this exploration included:

1. **Dataset Selection Importance:** The choice of dataset for fine-tuning the LLM was recognized as a critical factor influencing success. The dataset serves as the foundation for training the model to perform well in a specific domain or task.
2. **Fine-Tuning Sensitivity:** The fine-tuning process was revealed to be highly sensitive to various factors, such as the combination of hardware and software (device, framework, LLM model). These factors played a crucial role in determining whether fine-tuning was successful or not.

This initial exploration paved the way for further research into fine-tuning strategies and highlighted the need for careful consideration of dataset selection and fine-tuning parameters in the development of conversational agents powered by LLMs.

### 6.1.2 DESIGNING DIALOG MANAGEMENT STRATEGIES

The research then transitioned to the examination of dialog management strategies, with a specific focus on context management and intent recognition within dialogs. An essential aspect of this exploration was the incorporation of external documents into the Large Language Model (LLM) through the retrieval-augmented generation process.

The experiment conducted in this phase yielded satisfactory results and successfully integrated these features into our conversational agent. The outcomes of this experiment

had a notable impact, notably in enhancing contextual awareness within dialogs. This was achieved by seamlessly integrating conversational history and external documents into the decision-making process of the conversational agent.

In summary, this phase of the research contributed to the development of more contextually aware conversational agents, improving their capacity to manage context and recognize user intent effectively.

#### 6.1.3 KNOWLEDGE BASE UPDATE MECHANISMS

The final study in the thesis centers around the integration of Large Language Models (LLMs) and Knowledge Graphs, exploring the diverse ways in which these models can enhance user preferences and retrieve external documents. A significant emphasis was placed on leveraging a knowledge graph to augment an LLM during inference.

In this experiment, the research successfully achieved the retrieval of documents for incorporation as inputs into the conversational agent during inference. However, it was also acknowledged that there are areas for further improvement in the retrieval mechanism, particularly in terms of generating Cypher statements.

Overall, this endeavor of unifying LLMs and Knowledge Graphs holds great promise. It offers the potential to create a chatbot that boasts increased contextual awareness, greater predictability of LLM output, and an enhanced capacity to cater to the individual needs of users. This fusion of technologies represents a significant step forward in advancing the capabilities of conversational agents.

## 6.2 FUTURE WORK

### 6.2.1 BUILDING EXPERIMENTAL LLM CHATBOT STRATEGIES

Experimenting with different hyperparameters is an important avenue for future work in the study of building chatbot strategies. Hyperparameters play a critical role in fine-tuning and configuring models for specific tasks. Some key directions for future work:

1. **Learning Rate Optimization:** Variations in learning rates can significantly impact model convergence and performance. Experimenting with different learning

rates, including schedules for adaptive learning rates, can lead to more favorable results.

2. **Batch Size Variation:** The choice of batch size can influence training dynamics and computational efficiency. Exploring different batch sizes and their effects on training stability and performance is worthwhile.
3. **Architectural Hyperparameters:** Researchers can investigate the impact of architectural choices, such as the number of layers, hidden units, or attention mechanisms, on model performance. Adjusting these architectural hyperparameters can help tailor models for specific tasks.

By systematically varying these hyperparameters and evaluating their effects on model performance, researchers can gain a deeper understanding of how to optimize chatbot strategies for specific tasks and domains, ultimately leading to more favorable results and improved conversational agents.

#### 6.2.2 DESIGNING DIALOG MANAGEMENT STRATEGIES

Exploring the integration of Retrieval-Augmented Generation (RAG) at different stages of the Large Language Model (LLM) is an potential avenue for future research. This approach involves introducing RAG at various points within the LLM’s architecture to understand if there are significant variations in performance.

Key areas to investigate might include:

1. **Early-Stage RAG Integration:** Examining the impact of introducing RAG at the early stages of processing within the LLM. This could involve using RAG as a source of initial context or knowledge input to guide the model’s initial understanding of user queries.
2. **Mid-Stage RAG Integration:** Implementing RAG at intermediate stages of the LLM’s processing pipeline. This could influence how the model maintains context and retrieves external information as it processes user inputs.
3. **Late-Stage RAG Integration:** Exploring the effects of introducing RAG as a final step in response generation. This might involve using RAG to refine or verify responses generated by the LLM, enhancing the model’s response quality.

By systematically experimenting with RAG integration at different stages, researchers can gain valuable insights into the optimal points of intervention for this technology and

how it impacts the overall performance of conversational agents. This exploration can lead to more effective dialog management strategies and improved conversational agent capabilities.

#### 6.2.3 KNOWLEDGE BASE UPDATE MECHANISMS

In regards to future work, there are several promising directions to pursue:

1. **Enhancing Knowledge Graph Integration:** Building upon the foundation laid in this research, further efforts can be directed towards refining the process of retrieving data from the knowledge graph. Addressing instances where the LLM struggles to generate the correct Cypher statements to obtain desired results is paramount. This involves improving the model's ability to understand and formulate queries that efficiently access the knowledge graph.
2. **Addressing Hallucination:** The challenge of hallucination, where the model generates information that is not accurate or relevant, is a persistent issue in conversational agents. Future work can involve extensive experimentation with prompt-tuning techniques to reduce hallucination in LLM-based conversational agents. Fine-tuning prompts can play a crucial role in guiding the model to generate more precise and contextually relevant responses.

# REFERENCES

- [1] *Agents*. 2023. URL: <https://docs.langchain.com/docs/components/agents/>.
- [2] Param Aggarwal. *Fashion Product Images (Small)*. 2019. URL: <https://www.kaggle.com/datasets/paramaggarwal/fashion-product-images-small>.
- [3] *ALiBi*. 2022. URL: [https://docs.mosaicml.com/projects/composer/en/stable/method\\_cards/alibi.html](https://docs.mosaicml.com/projects/composer/en/stable/method_cards/alibi.html).
- [4] *all-MiniLM-L6-v2*. 2023. URL: <https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2>.
- [5] Eleanor Berger. *Grounding LLMs*. 2023. URL: <https://techcommunity.microsoft.com/t5/fasttrack-for-azure/grounding-llms/ba-p/3843857>.
- [6] Lukas Biewald. *Experiment Tracking with Weights and Biases*. Software available from wandb.com. 2020. URL: <https://www.wandb.com/>.
- [7] Sid Black et al. *GPT-NeoX-20B: An Open-Source Autoregressive Language Model*. 2022. DOI: 10.48550/ARXIV.2204.06745. URL: <https://arxiv.org/abs/2204.06745>.
- [8] Tomaz Bratanic. *Extract knowledge from text: End-to-end information extraction pipeline with spaCy and Neo4j*. 2022. URL: <https://towardsdatascience.com/extract-knowledge-from-text-end-to-end-information-extraction-pipeline-with-spacy-and-neo4j-502b2b1e0754>.
- [9] Tom B. Brown et al. *Language Models are Few-Shot Learners*. 2020. arXiv: 2005.14165 [cs.CL].
- [10] Jiahang Cao et al. *Knowledge Graph Embedding: A Survey from the Perspective of Representation Spaces*. 2022. arXiv: 2211.03536 [cs.LG].
- [11] *Chain*. 2023. URL: <https://docs.langchain.com/docs/components/chains/chain>.
- [12] *Chroma Database*. URL: <https://www.trychroma.com/>.

- [13] Stamford Conn. *Gartner Predicts Chatbots Will Become a Primary Customer Service Channel Within Five Years*. 2022. URL: [www.gartner.com/en/newsroom/press-releases/2022-07-27-gartner-predicts-chatbots-will-become-a-primary-customer-service-channel-within-five-years](http://www.gartner.com/en/newsroom/press-releases/2022-07-27-gartner-predicts-chatbots-will-become-a-primary-customer-service-channel-within-five-years).
- [14] *Conversational Memory for LLMs with Langchain*. URL: <https://www.pinecone.io/learn/series/langchain/langchain-conversational-memory/>.
- [15] *Conversational Retrieval Agent*. URL: [https://python.langchain.com/docs/use\\_cases/question\\_answering/how\\_to/conversational\\_retrieval\\_agents](https://python.langchain.com/docs/use_cases/question_answering/how_to/conversational_retrieval_agents).
- [16] Tri Dao et al. *FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness*. 2022. arXiv: 2205.14135 [cs.LG].
- [17] Shah Deval. *Self-Supervised Learning and Its Applications*. 2023. URL: <https://neptune.ai/blog/self-supervised-learning>.
- [18] Jacob Devlin et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2019. arXiv: 1810.04805 [cs.CL].
- [19] Drost Dorian. *Different ways of training LLMs*. 2023. URL: <https://towardsdatascience.com/different-ways-of-training-langs-c57885f388ed>.
- [20] Ketan Doshi. *Transformers Explained Visually (Part 3): Multi-head Attention, deep dive*. 2021. URL: <https://towardsdatascience.com/transformers-explained-visually-part-3-multi-head-attention-deep-dive-1c1ff1024853>.
- [21] *GPT-3.5 Turbo: A Language Model by OpenAI*. 2023. URL: <https://platform.openai.com/docs/models/gpt-3-5>.
- [22] Matthew Honnibal et al. “spaCy: Industrial-strength Natural Language Processing in Python”. In: *10.5281/zenodo.1212303* (2020). URL: <https://spacy.io/>.
- [23] Edward J. Hu et al. *LoRA: Low-Rank Adaptation of Large Language Models*. 2021. arXiv: 2106.09685 [cs.CL].
- [24] Xin Huang et al. *Question answering using Retrieval Augmented Generation with foundation models in Amazon SageMaker JumpStart*. 2023. URL: <https://aws.amazon.com/blogs/machine-learning/question-answering-using-retrieval-augmented-generation-with-foundation-models-in-amazon-sagemaker-jumpstart/>.
- [25] IBM. *What is a Knowledge Graph?* URL: <https://www.ibm.com/topics/knowledge-graph>.
- [26] *Langchain Framework*. URL: <https://www.langchain.com/>.

- [27] Patrick Lewis et al. *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks*. 2021. arXiv: 2005.11401 [cs.CL].
- [28] Jiongnan Liu et al. *RETA-LLM: A Retrieval-Augmented Large Language Model Toolkit*. 2023. arXiv: 2306.05212 [cs.IR].
- [29] Peter Mattson et al. *MLPerf Training Benchmark*. 2020. arXiv: 1910.01500 [cs.LG].
- [30] *MiniLM-L6-H384-uncased*. 2023. URL: <https://huggingface.co/nreimers/MiniLM-L6-H384-uncased>.
- [31] Seungwhan Moon et al. “OpenDialKG: Explainable Conversational Reasoning with Attention-based Walks over Knowledge Graphs”. In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. 2019.
- [32] *Neo4j - Graph Database*. URL: <https://neo4j.com/>.
- [33] OpenAI. *GPT-4 Technical Report*. 2023. arXiv: 2303.08774 [cs.CL].
- [34] Shirui Pan et al. *Unifying Large Language Models and Knowledge Graphs: A Roadmap*. 2023. arXiv: 2306.08302 [cs.CL].
- [35] *Pinecone Database*. URL: <https://www.pinecone.io/>.
- [36] Ofir Press, Noah A. Smith, and Mike Lewis. *Train Short, Test Long: Attention with Linear Biases Enables Input Length Extrapolation*. 2022. arXiv: 2108.12409 [cs.CL].
- [37] Xianghui Sun et al. *A Comparative Study between Full-Parameter and LoRA-based Fine-Tuning on Chinese Instruction Data for Instruction Following Large Language Model*. 2023. arXiv: 2304.08109 [cs.CL].
- [38] Rohan Taori et al. *Stanford Alpaca: An Instruction-following LLaMA model*. [https://github.com/tatsu-lab/stanford\\_alpaca](https://github.com/tatsu-lab/stanford_alpaca). 2023.
- [39] MosaicML NLP Team. *Introducing MPT-7B: A New Standard for Open-Source, Commercially Usable LLMs*. Accessed: 2023-05-05. 2023. URL: [www.mosaicml.com/blog/mpt-7b](http://www.mosaicml.com/blog/mpt-7b) (visited on 05/05/2023).
- [40] *Tools*. 2023. URL: <https://docs.langchain.com/docs/components/agents/tool>.
- [41] Hugo Touvron et al. *LLaMA: Open and Efficient Foundation Language Models*. 2023. arXiv: 2302.13971 [cs.CL].

- [42] Uberall. *80% of Consumers Report Chatbot Experiences as Positive Yet nearly 60% still lack strong enthusiasm for the technology*. 2023. URL: <https://uberall.com/en-us/company/press-releases/study-80-of-consumers-report-chatbot-experiences-as-positive-yet-nearly-60-still-lack-strong-enthusiasm-for-the-technology-according-to-new-uberall-study>.
- [43] Ashish Vaswani et al. *Attention Is All You Need*. 2023. arXiv: 1706.03762 [cs.CL].
- [44] *Vector Stores*. 2023. URL: [https://python.langchain.com/docs/modules/data\\_connection/vectorstores/](https://python.langchain.com/docs/modules/data_connection/vectorstores/).
- [45] Thomas Wolf et al. “Transformers: State-of-the-Art Natural Language Processing”. In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Online: Association for Computational Linguistics, Oct. 2020, pp. 38–45. URL: <https://www.aclweb.org/anthology/2020.emnlp-demos.6>.
- [46] Cameron R. Wolfe. *Democratizing AI: MosaicML’s Impact on the Open-Source LLM Movement*. 2023. URL: <https://cameronrwolfe.substack.com/p/democratizing-ai-mosaicmls-impact>.
- [47] Jianguo Zhang et al. *DialogStudio: Towards Richest and Most Diverse Unified Dataset Collection for Conversational AI*. 2023. arXiv: 2307.10172 [cs.CL].

# APPENDIX A

## CODE

### A.1 DATA PRE-PROCESSING OPENDIALKG DATASET

```
class TrainODKG(TrainDataBase):  
    def __init__(self, dataset: str, val_set_size: int, tokenizer, cutoff_len)  
        -> None:  
            super().__init__(dataset, val_set_size, tokenizer,  
                           cutoff_len)  
  
    def tokenizer_inputs(self, promptResponse, use_eos_token=True, **kwargs)  
        -> Dict[str, Any]:  
            result = self.tokenizer(  
                promptResponse['prompt'],  
                promptResponse['response'],  
                truncation=True,  
                padding = "max_length",  
                max_length=self.cutoff_len + 1,  
                return_tensors = 'pt',  
            )  
            return {  
                "input_ids": result["input_ids"],  
                "attention_mask": result["attention_mask"],  
            }  
  
    def preprocessing_data(self, inp: Dict, split: str):  
        ft_prompt = []
```

```

ft_response = []

for log in inp[split]['log']:
    for user_utterance in log:
        ft_prompt.append(user_utterance['user utterance'])

for log in inp[split]['log']:
    for system_utterance in log:
        ft_response.append(system_utterance['system response'])

return {'prompt':ft_prompt,'response':ft_response}

def remove(self,inp:Dict):
    inp = inp.remove_columns(['prompt','response'])
    inp.set_format("torch")
    return inp

def prepare_data(self, use_eos_token=True, **kwargs) -> None:
    data = load_dataset('Salesforce/dialogstudio', 'OpenDialKG')

    # Split the 'train' feature into validation and test sets

    # 80% train, 20% test + validation
    train_testvalid = data['train'].train_test_split(
        test_size=0.2,
        shuffle=True,
        seed=42)

    # Split the 20% test + valid in half test, half valid
    test_valid = train_testvalid['test'].train_test_split(
        test_size=0.5,
        shuffle=True,
        seed = 42)

    main_dataset = DatasetDict({
        'train': train_testvalid['train'],

```

```

        'validation': test_valid['test'],
        'test': test_valid['train']
    })

train_prompt_response = self.preprocessing_data(main_dataset,'train')
val_prompt_response = self.preprocessing_data(main_dataset,'validation')
test_prompt_response = self.preprocessing_data(main_dataset,'test')

#Convert to Dataset

train_prompt_response = datasets.Dataset.from_pandas
(pd.DataFrame(data=train_prompt_response))
val_prompt_response = datasets.Dataset.from_pandas
(pd.DataFrame(data=val_prompt_response))
test_prompt_response = datasets.Dataset.from_pandas
(pd.DataFrame(data=test_prompt_response))

# Tokenize data with map function

self.train_data = train_prompt_response.map
(lambda x: self.tokenizer_inputs(x,use_eos_token= use_eos_token,**kwargs),
batched = True)
self.val_data = val_prompt_response.map
(lambda x: self.tokenizer_inputs(x,use_eos_token= use_eos_token),
batched = True)
self.test_data = test_prompt_response.map
(lambda x: self.tokenizer_inputs(x,use_eos_token= use_eos_token),
batched = True)

self.train_data = self.remove(self.train_data)
self.val_data = self.remove(self.val_data)
self.test_data = self.remove(self.test_data)

```

## A.2 EXPERIMENT 2: CONVERSATIONAL AGENT IMPLEMENTATION

### Knowledge Base Tool Implementation

```
from langchain.agents import Tool

tools = [
    Tool(
        name='Knowledge Base',
        func=qa.run,
        description=(
            'use this tool to provide product recommendations to users '
        )
    )
]
```

```
from langchain.chat_models import ChatOpenAI
from langchain.chains.conversation.memory import ConversationBufferWindowMemory
from langchain.chains import RetrievalQA

# chat completion llm
llm = ChatOpenAI(
    model_name='gpt-3.5-turbo',
    temperature=0.0
)
# conversational memory
conversational_memory = ConversationBufferWindowMemory(
    memory_key='chat_history',
    k=5,
    return_messages=True
)
# retrieval qa chain
qa = RetrievalQA.from_chain_type(
    llm=llm,
    chain_type="stuff",
```

```
retriever=vectorstore.as_retriever()
)
```

### A.3 CYpher CHAIN MODEL IMPLEMENTATION

```
from langchain.prompts.prompt import PromptTemplate
from langchain.chat_models import ChatOpenAI
from langchain.chains.conversation.memory
import ConversationBufferWindowMemory,
ConversationSummaryBufferMemory,
ConversationBufferMemory
from langchain.chains import RetrievalQA
```

```
CYPHER_GENERATION_TEMPLATE = """Task:Generate Cypher statement
to query a graph database.
```

Instructions:

Use only the provided relationship types and properties in the schema.

Do not use any other relationship types or properties that are not provided.

Schema:

```
{schema}
```

Note: Do not include any explanations or apologies in your responses.

Do not respond to any questions that might ask anything
else than for you to construct a Cypher statement.

Do not include any text except the generated Cypher statement.

Examples: Here are a few examples of generated Cypher
statements for particular questions:

```
# What black shoes do you have for women?
MATCH (p:Product)-[:HAS_COLOUR]->(:Colour {{colour: 'Black'}})
MATCH (p)-[:FOR_GENDER]->(:Gender {{gender: 'Women'}})
RETURN p.product AS Product
```

```
# Do you have clothes for fall?
MATCH (p:Product)-[:FOR_SEASON]->(:Season {{season:'Fall'}})
RETURN p.product as Product
```

```

# I would like some casual t-shirts for men.

MATCH (p:Product)-[:FOR_USAGE]->(:Occasion {{occasion: 'Casual'}})
MATCH (p)-[:FOR_GENDER]->(:Gender {{gender: 'Men'}})
MATCH (p)-[:HAS_ARTICLETYPE]->(:articleType {{}})
RETURN p.product as Product

# Blue Tshirt

MATCH (p:Product)-[:HAS_ARTICLETYPE]->(:articleType {{articleType: 'Tshirts'}})
MATCH (p)-[:HAS_COLOUR]->(:Colour {{colour:'Blue'}})
RETURN p.product as Product

The question is:
{question}"""

CYPHER_GENERATION_PROMPT = PromptTemplate(
    input_variables=["schema", "question"],
    template=CYPHER_GENERATION_TEMPLATE
)

# chat completion llm
llm = ChatOpenAI(
    model_name='gpt-3.5-turbo',
    temperature=0.0 # initialised to 0, so more predictable
)

# conversational memory
conversational_memory = ConversationBufferMemory(
    memory_key='chat_history',
    return_messages=True,
)

cypher_search = GraphCypherQACChain.from_llm(
    llm=llm, graph=graph, verbose=True, top_k =3,
    return_intermediate_steps=True,
    cypher_prompt=CYPHER_GENERATION_PROMPT, qa_prompt=qa_prompt
)

```

)