ESP-JETA
ESP Journal of Engineering & Technology Advancements
ISSN: 2583-2646 / Volume 5 Issue 1 January 2025 / Page No: 12-29
Paper Id: JETA-V5I1P103 / Doi: 10.56472/25832646/JETA-V5I1P103

Original Article

Integrating Large Language Models: Enhancing Chatbot Capabilities for Training on Diverse Data Sources

Ranjith Gopalan[1], Abhishek Sen[2], Vishal S[3]
[1] *Principal Consultant, Cognizant technology solutions, USA.*
[2,3] *Consultant, Cognizant technology solutions, USA.*

Received Date: 12 November 2024 Revised Date: 20 December 2024 Accepted Date: 06 January 2025

Abstract: *The paper discusses the fundamentals of chatbot design, including key components, types of chatbots, and the importance of user experience and interaction design. It then delves into the structure and function of Large Language Models, their training, and their use cases in chatbot development. The paper explores the process of integrating LLMs with chatbot frameworks, highlighting the key steps and the services available for building chatbots, such as Amazon Bedrock, Microsoft LUIS, and Google Bard.*

Paper explains the fundamentals of chatbot design, provides an overview of large language models, discusses chatbot architecture for handling unstructured and structured data, and highlights the roles of AWS Titan and Anthropic Claude models in the development of retrieval-augmented generation systems. It includes a comparison of execution results, examines testing and evaluation of chatbot performance, considers ethical aspects of chatbot development, and suggests future research directions such as optimizing RAG-based and knowledge graph-enhanced models to handle larger datasets and more complex queries without compromising performance.

Furthermore, the paper provides a detailed comparison of the foundational models integrated with the Amazon Bedrock service, focusing on the Amazon Titan model used for knowledge base creation.

Keywords: *LLM, Chatbots Generative AI, Meta data, Knowledge graph, Structured and unstructured data, AWS Titan, FAISS index, Anthropic Claude*

I. INTRODUCTION

Chatbots leverage algorithms and machine learning to mimic human conversation through voice commands and text chats. Traditionally, these systems relied on predefined scripts and limited natural language processing capabilities, which constrained their ability to engage in fluid, meaningful interactions. However, the integration of large language models (LLMs) has revolutionized chatbot technology. LLMs, trained on vast datasets and utilizing deep learning techniques, have significantly advanced chatbots, enabling them to understand context, sentiment, and humor, and generate coherent, engaging responses.

This advancement transforms chatbots from basic interactive tools into sophisticated virtual assistants. By incorporating LLMs, chatbots can now provide relevant, context-aware responses that enhance user satisfaction. This integration also streamlines the development process, allowing developers to focus on creating unique functionalities rather than building foundational language capabilities from scratch.

Training LLMs on diverse datasets ensures that chatbots can cater to a wide range of demographics and industries, delivering accurate and contextually appropriate information while maintaining a professional tone. Furthermore, the ethical considerations in developing these systems, such as addressing bias and ensuring user privacy, are crucial for creating trustworthy and reliable chatbots. Thus, the integration of LLMs not only enhances the technical capabilities of chatbots but also aligns with the ethical standards necessary for widespread adoption and trust in AI-driven communication technologies.

II. RESEARCH METHODOLOGY

A. Fundamentals of Chatbot Design

a) Key Components of Chatbots:

Chatbots simulate human conversation using voice commands or text chats. Key components include natural language processing (NLP), machine learning algorithms, dialogue management, and integration with external data sources. NLP helps chatbots understand human language, while machine learning enables them to learn and improve. Dialogue management ensures coherent conversations, and external data integration allows for real-time, personalized responses.

b) Types of Chatbots:

Chatbots are categorized into rule-based, retrieval-based, and generative types. Rule-based chatbots follow predefined rules for simple applications. Retrieval-based chatbots use a database of responses to handle more complex interactions. Generative chatbots create responses from scratch using large language models (LLMs), allowing for fluid conversations. Integrating LLMs enhances all chatbot types, making them more effective and engaging.

c) *User Experience and Interaction Design:*

User experience (UX) and interaction design are crucial for effective chatbots. UX design focuses on user needs and intuitive interactions. Interaction design structures dialogue, times responses, and provides feedback. Integrating LLMs helps chatbots handle various contexts and user intents. Consistency in tone builds trust, and continuous testing refines performance, creating user-friendly and engaging chatbots.

B. Large Language Models Explained

a) Structure and Function of LLMs:

Large Language Models (LLMs) are neural networks designed to understand and generate human-like text. Built on transformer architectures, they use self-attention mechanisms to weigh the significance of words in context, capturing nuanced language relationships. LLMs predict the next word in a sequence, enabling them to engage in conversations and provide information. Trained on diverse datasets, they learn patterns, grammar, and cultural contexts, making them adaptable and informative.

b) Training Large Language Models:

Training LLMs involves selecting diverse text data, preprocessing it, and using sophisticated algorithms to optimize the model's parameters. Techniques like supervised and unsupervised learning, along with transfer learning, are crucial. Training requires significant computational resources, often using cloud-based platforms. Continuous evaluation with metrics like perplexity and accuracy ensures the model meets desired benchmarks and evolves based on user feedback.

c) Use Cases of LLMs in Chatbot Development:

i) LLMs Enhance Chatbots in Various Fields:

- Customer Service: Automating inquiries and support, reducing wait times, and improving satisfaction.
- Personal Assistants: Managing schedules and providing tailored interactions.
- Education: Answering queries, providing resources, and facilitating interactive learning.
- Healthcare: Offering medical information, scheduling, and symptom checking.
- Entertainment and Gaming: Creating dynamic interactions and immersive storylines.

C. Integrating LLMs with Chatbot Frameworks

API integration connects chatbots with powerful LLMs, enhancing their capabilities. Major platforms like OpenAI, Google, and Microsoft offer APIs for their LLMs. Key steps include selecting the right provider, defining diverse data sources, and setting up a robust backend. Continuous evaluation and iteration keep the chatbot effective and relevant.

D. Services To Build Chatbots:

This below table (Fig. 1) compares the key features and capabilities of services to build chatbots: Amazon Bedrock, Microsoft LUIS, and Google Bard.

Feature	Amazon Bedrock	Microsoft LUIS	Google Bard
Provider	Amazon	Microsoft	Google
Primary Use Case	Generative AI, text and image generation	Language understanding, intent recognition	Conversational AI, text generation
Strengths	Multiple foundation models, serverless deployment, integration with AWS ecosystem	Robust intent recognition, easy integration with Azure services	Advanced conversational capabilities, large context window
Weaknesses	Can be resource-intensive, higher cost for advanced models	Limited to language understanding, not generative	Still evolving, may lack some enterprise features
Context Window	Standard to up to 200,000 tokens	Standard (typically a few thousand tokens)	Large (specifics not disclosed)
Integration	Seamless with AWS services	Seamless with Azure services	Seamless with Google Cloud services
Customization	Fine-tuning available, multiple foundation models	Customizable intents and entities	Fine-tuning available, adaptable to various tasks
Deployment	Serverless, scalable	Cloud-based, scalable	Cloud-based, scalable
Example Models	Amazon Titan, Anthropic Claude, Cohere Command & Embed, AI21 Labs' Jurassic, Meta Llama 2, Stability AI's Stable Diffusion	Custom models for intent recognition and entity extraction	Custom models for conversational AI and text generation

Figure 1: Comparison of Chatbot Development Services

E. Amazon Bedrock Foundational Models:

This below table (Fig. 2) compares the key features and capabilities of various foundational models integrated with the Amazon Bedrock service.

Model	Amazon Titan	Anthropic Claude	Cohere Command & Embed	AI21 Labs' Jurassic	Meta Llama 2	Stability AI's Stable Diffusion
Provider	Amazon	Anthropic	Cohere	AI21 Labs	Meta	Stability AI
Primary Use Case	General-purpose text generation	Conversational AI, text generation	Text embeddings, semantic search	Large-scale language generation	Various NLP tasks	Image generation
Strengths	Robust performance, safety features	Advanced reasoning, large context window	High-quality text representations	Creative writing, content generation	Versatile, translation, summarization	Advanced diffusion techniques
Weaknesses	May lack advanced reasoning capabilities	Higher cost, may require more computational resources	Limited to text embeddings, not ideal for generation	Can be resource-intensive, may produce verbose outputs	May not be as specialized for certain tasks	Limited to image generation, not suitable for text tasks
Context Window	Standard	Up to 200,000 tokens	Standard	Standard	Standard	N/A
Pricing	On-demand, provisioned	On-demand, provisioned	On-demand, provisioned	On-demand, provisioned	On-demand, provisioned	On-demand, provisioned

Figure 2: Comparison of Foundational Models in Amazon Bedrock

F. Model Used – Knowledge Base Creation:

a) Knowledge Base Creation: Amazon Titan (Embedding)

Amazon Titan Embeddings provide high-quality, dense vector representations of text, which are critical for understanding the semantic meaning of user inputs. Embeddings help identify patterns, relationships, and similarities in textual data, enhancing the training process for chatbots.

b) Key Features:

- Provides rich semantic text representations.
- Custom embedding models for domain-specific applications.
- Improves search, matching, and recommendation capabilities.

G. Model Used – Generative AI

a) Response Generation: Anthropic Claude (Integrated With Amazon Bedrock Service)

Claude is a generative AI model that excels in creating natural, human-like responses in conversations. It's perfect for handling context-aware, meaningful interactions.

b) Key Features:

- Strong contextual understanding for better response relevance.
- Customizable through fine-tuning on specific datasets.
- Scalable to manage large user queries efficiently.
- Advanced reasoning, large context window of up to 200,000 tokens.

H. Architecture

In this work, we explore three architectural approaches to building a sophisticated chatbot using the Retrieval-Augmented Generation (RAG) paradigm, knowledge graph, leveraging Amazon Bedrock services for embedding generation and Anthropic Claude for response generation. Each approach is tailored to optimize information retrieval and response coherence while addressing specific scalability and use-case requirements.

a) Unstructured Data:

Unstructured data includes information that doesn't follow a specific format. Following is the example of unstructured data sources

- **Emails**: Communication content without a predefined structure.
- **Social Media Posts**: Tweets, Facebook updates, Instagram captions, etc.
- **Webpages**: Blog posts, articles, and other text-heavy online content.
- **Multimedia Content**: Images, videos, and audio files.
- **Customer Feedback**: Reviews, comments, and survey responses.
- **Documents**: PDFs, Word documents, and presentations.
- **Chat Logs**: Conversations from messaging apps and customer service chats.

i) Simple RAG-Based Architecture

The first architecture is based solely on the RAG architecture, which integrates an information retrieval component with a generative language model. The workflow includes the following steps:

Data Preprocessing and Embedding Generation:

- Raw textual data is pre-processed to remove noise, normalize content, and tokenize text.
- Using Amazon Titan Embeddings, the processed data is converted into high-dimensional vector representations. Each vector captures semantic information, enabling similarity-based retrieval.
- The embeddings are stored in a FAISS index, a highly efficient and scalable vector database for similarity searches.

Query Handling and Similarity Search:

- Upon receiving a user query, the input text is also transformed into a vector using Amazon Titan.
- The query vector is compared against the FAISS index using similarity to identify the most relevant stored vectors (documents).

Contextual Response Generation:

- The retrieved documents (top-k results based on similarity score) are used as context for generating responses.
- Anthropic Claude, a large language model, processes the query along with the retrieved context to produce a response that aligns with the input's intent.

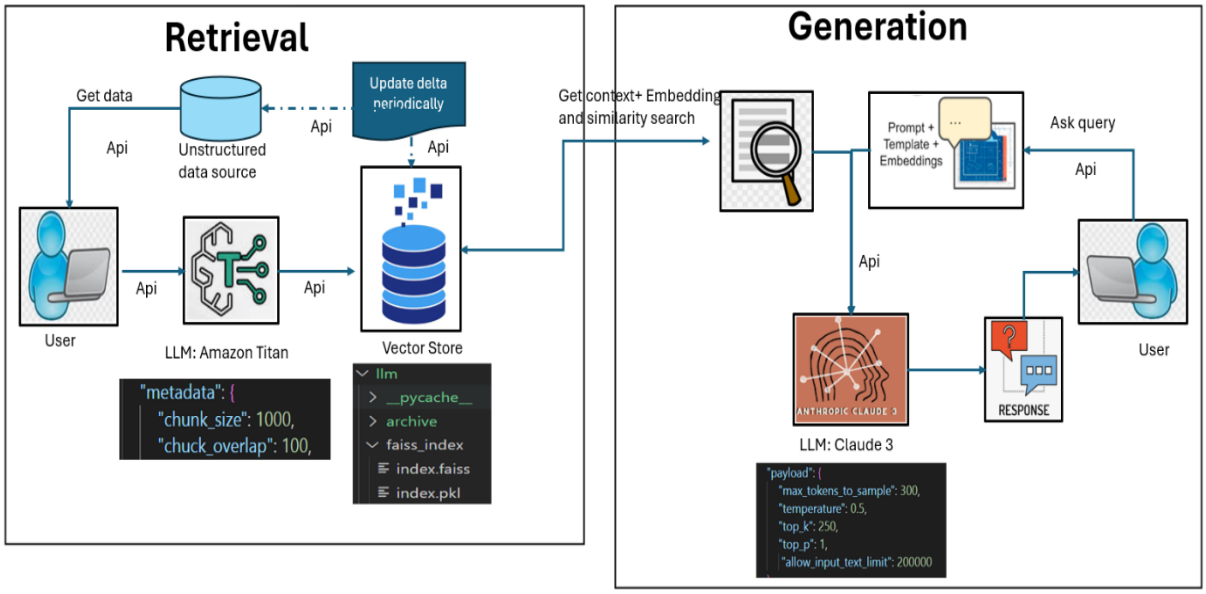


Figure 3: Simple RAG-Based Architecture

Advantages:

- Simplicity: Easy to implement and manage, with clear modular components.
- Efficiency: Works well for static datasets with straightforward relationships between data points.
- Scalability: Capable of handling medium-to-large datasets with minimal computational overhead.

Limitations:

- Lacks deep semantic reasoning or understanding of hierarchical relationships within the data.
- As the dataset grows, managing the FAISS index and ensuring retrieval accuracy becomes challenging.

b) Structured Data:

Structured data is organized and easily searchable, often stored in databases with rows and columns. Following is the example of structured data sources

- Databases: SQL databases (e.g., MySQL, PostgreSQL) and NoSQL databases (e.g., MongoDB, Cassandra).
- Spreadsheets: Excel files and Google Sheets.
- CSV Files: Comma-separated values files for tabular data.
- ERP Systems: Enterprise Resource Planning systems that manage business processes.
- CRM Systems: Customer Relationship Management systems with customer information.
- Sensor Data: Data collected from IoT devices and sensors, often in a tabular format.
- Financial Records: Transactions, balance sheets, and financial statements.

i) Knowledge Graph-Enhanced Architecture

The approach integrates a metadata-driven knowledge graph to improve the chatbot's contextual understanding and response generation capabilities. This architecture extends beyond basic vector similarity by incorporating graph-based reasoning.

Data Enrichment and Metadata Creation:

- The dataset is enriched with additional metadata, including tags, hierarchical attributes, and relationships between entities.
- Metadata examples include entity properties (e.g., categories, keywords) and explicit connections between documents or topics.

Knowledge Graph Construction:

- Using the enriched metadata, a knowledge graph is constructed, representing data as nodes (entities) and edges (relationships).
- The graph enables efficient traversal and querying, allowing for retrieval not only based on content similarity but also on semantic relationships.

Query Processing with Cosine Similarity:

- Queries are matched against both the knowledge graph and embeddings generated by Amazon Titan.
- The graph's traversal capabilities are combined with cosine similarity to identify the most relevant entities, nodes, or documents.

Contextual Response Generation:

- The information retrieved from the graph and embeddings is consolidated to provide rich, relevant context.
- This context is sent to Anthropic Claude, which generates a final response based on both explicit relationships (from the graph) and semantic relevance (from embeddings).

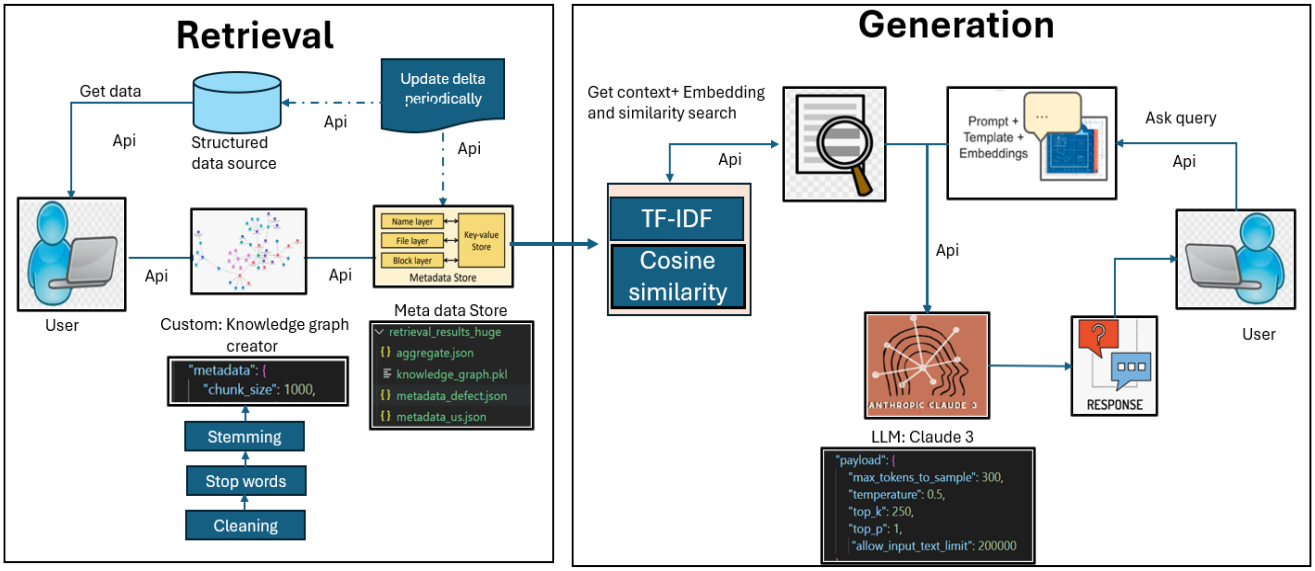


Figure 4: Knowledge Graph-Enhanced Architecture

Advantages:

- Captures complex relationships and hierarchical structures in the dataset.
- Suitable for use cases requiring semantic reasoning, such as multi-step queries or related data exploration.
- Improves the relevance of responses for datasets with interconnected entities.

Limitations:

- Graph construction and traversal require additional computational resources and maintenance.
- Query response times can be slower compared to RAG-only models, especially with large, densely connected graphs.

ii) Hybrid RAG and Knowledge Graph Architecture

The hybrid approach combines the RAG-based architecture with the knowledge graph-enhanced methodology, creating a robust framework capable of leveraging both approaches' strengths.

Data Preparation and Dual Indexing:
The dataset is processed in two ways:

- Vector Embeddings: Data is converted into vector representations using Amazon Titan and stored in a FAISS index.
- Knowledge Graph: Metadata is used to construct a knowledge graph, capturing entity relationships and semantic structures.

Dual Retrieval Mechanism:

- Queries are simultaneously matched against the FAISS vector store and the knowledge graph.
- From the FAISS index, documents are retrieved based on cosine similarity.
- From the knowledge graph, relevant nodes and their connected entities are identified through graph traversal and metadata filtering.

Contextual Data Integration:

- The results from both retrieval systems are merged and prioritized to form a consolidated context.
- This combined context ensures that the most relevant information—both semantically and structurally—is available for response generation.

Response Generation:

- The consolidated context is passed to Anthropic Claude, which generates a response leveraging both embedding-based context and the knowledge graph's insights.

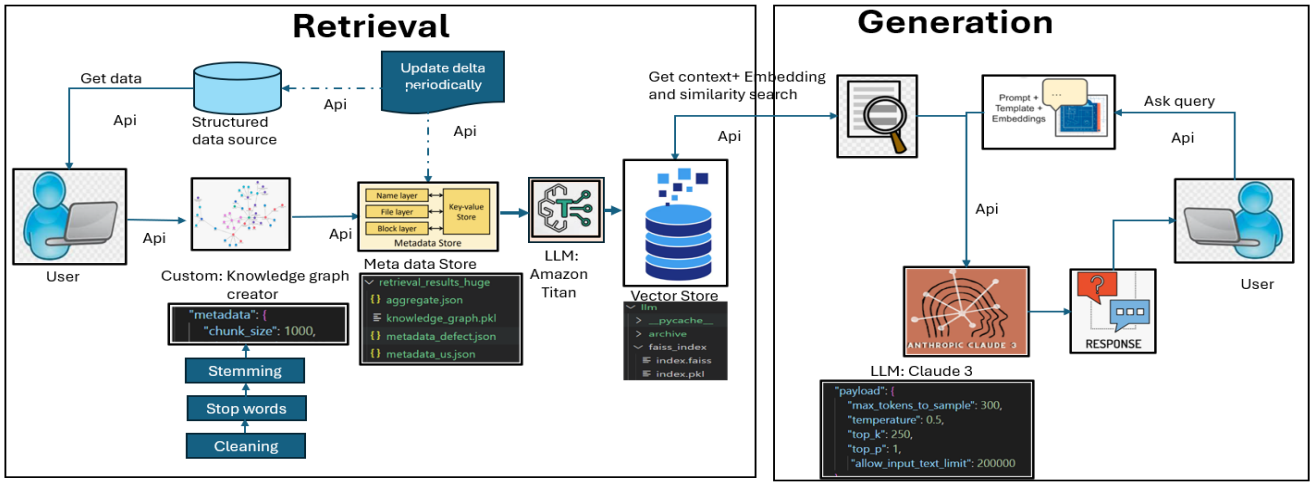


Figure 5: Hybrid RAG and Knowledge Graph Architecture

Advantages:

- Maximizes the accuracy and relevance of responses by utilizing complementary retrieval mechanisms.
- Robust for complex datasets where both hierarchical relationships and semantic relevance are critical.
- Addresses scalability issues of RAG-only models by leveraging graph reasoning for focused data subsets.

Limitations:

- Increased system complexity, requiring efficient integration of both retrieval mechanisms.
- Higher computational overhead compared to standalone RAG or knowledge graph architectures.

c) Comparative Analysis

The three approaches offer a range of options depending on the requirements:

- Simple RAG-Based Architecture is suitable for lightweight, quick deployments.
- Knowledge Graph-Enhanced Architecture adds depth for datasets with complex interrelations.
- Hybrid Architecture achieves the best balance of scalability and contextual accuracy but comes with additional complexity.

III. RESULTS

Below are the results, including screenshots of the user interface, response examples, and key code snippets. These elements demonstrate the critical data and functionality areas within the analysis.

A. Unstructured

a) Simple RAG-Based Architecture:

This architecture integrates information retrieval with a generative language model.

i) Key Steps:

- Data Preprocessing: Clean and tokenize data, convert to vectors, store in FAISS index.
- Query Handling: Transform queries into vectors, compare with FAISS index.
- Response Generation: Use top-k documents as context for Anthropic Claude to generate responses.

ii) Data:

This below image (Fig 6) shows an example of unstructured data in a text document format.

Comprehensive Guide to Common Health Issues, Diseases, Remedies, and First Aid

Health is a vital aspect of our lives, and understanding common health issues, diseases, remedies, and first aid can significantly enhance our well-being. This guide provides an extensive overview of various health topics, including common health issues, diseases, remedies, primary medications, and first aid techniques.

Common Health Issues:

Diabetes is a chronic condition where the body either doesn't produce enough insulin or can't effectively use the insulin it produces, leading to high blood sugar levels. Symptoms include increased thirst, frequent urination, extreme fatigue, blurred vision, slow-healing sores, and unexplained weight loss. Managing diabetes involves regular blood sugar monitoring, a healthy diet, regular physical activity, and medications such as insulin or oral hypoglycaemics. Prevention includes maintaining a healthy weight, eating a balanced diet rich in fibre and low in refined sugars, regular exercise, and avoiding tobacco use.

Hypertension, or high blood pressure, is a condition where the force of the blood against the artery walls is too high, which can lead to heart disease and stroke. It often presents no symptoms but can cause headaches, shortness of breath, and nosebleeds. Management includes lifestyle changes such as a healthy diet (low in salt), regular exercise, maintaining a healthy weight, and medications like ACE inhibitors, beta-blockers, and diuretics. Prevention involves regular physical activity, a balanced diet low in salt, maintaining a healthy weight, limiting alcohol intake, and avoiding smoking.

Asthma is a chronic respiratory condition characterized by inflammation and narrowing of the airways, leading to difficulty breathing. Symptoms include wheezing, shortness of breath, chest tightness, and coughing, especially at night or early in the morning. It is managed with inhalers (bronchodilators and corticosteroids), avoiding triggers (such as allergens and pollutants), and regular medical check-ups. Prevention includes avoiding known triggers, maintaining a healthy lifestyle, and following a prescribed asthma action plan.

Figure 6: Unstructured Data – Text Document

iii) User Interface:

This below image (Fig 7) displays the user interface with the "Unstructured Data Copilot" option selected.

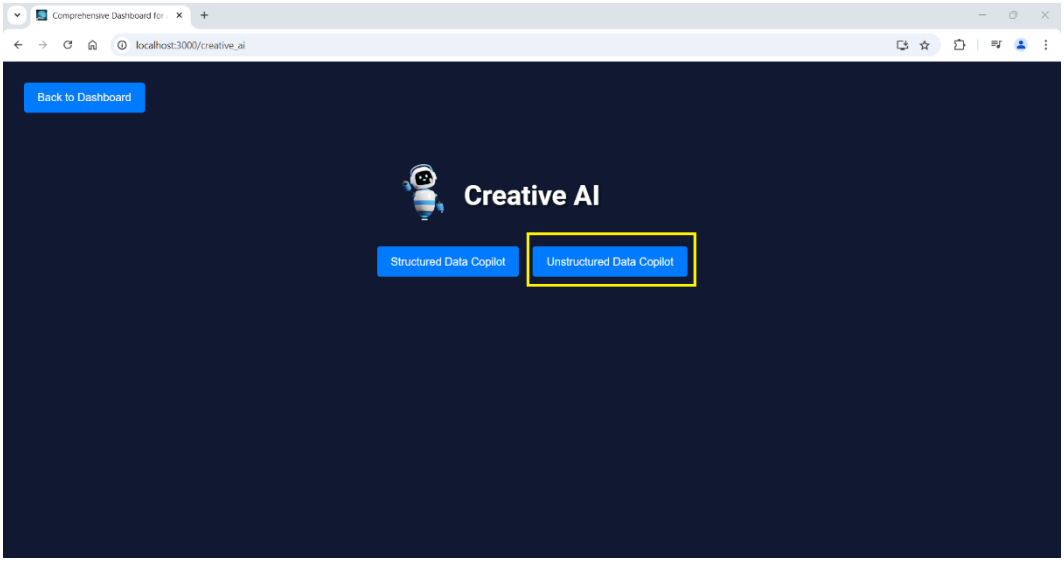


Figure 7: User Interface - "Unstructured Data Copilot" Option Selected

iv) *Response:*

This below image (Fig 8) displays the response for the query: "**how to manage and prevent Diabetes?**"

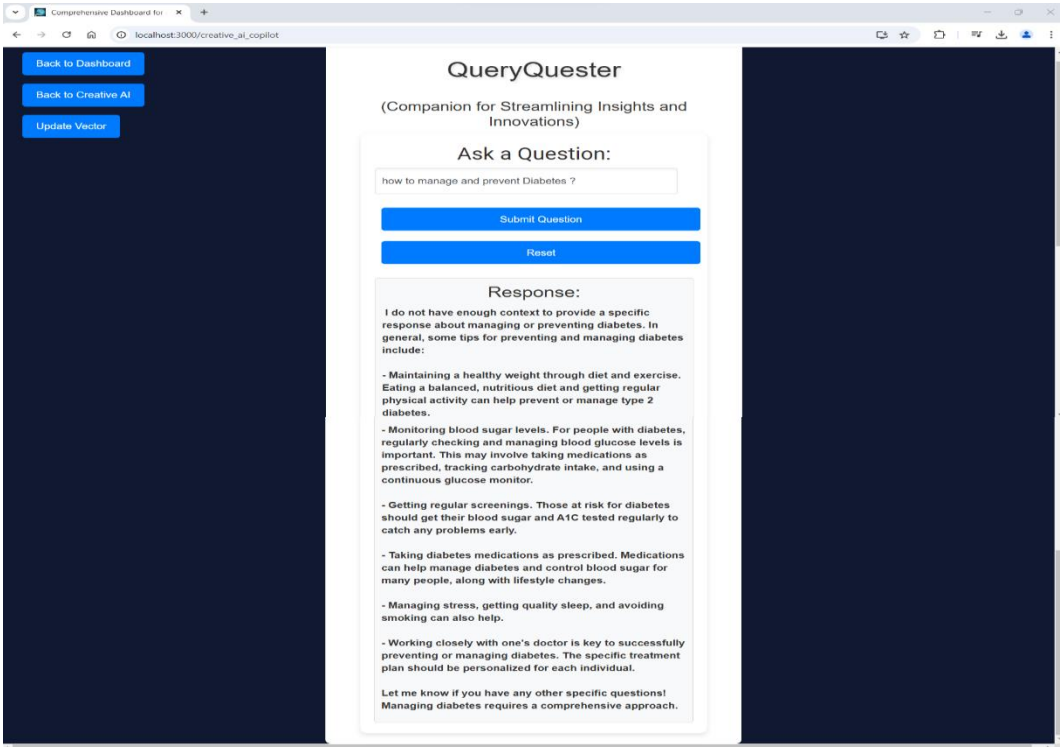


Figure 8: Chatbot Response

v) Code Snippet:

The below code snippet shows the generation of the vector knowledge base.

```
## vector embedding and vector store - Generate vector embeddings and store them using FAISS
def get_vector_store(docs, current_copilot, faiss_path):
    try:

        existing_vectorstore = load_vector_store(faiss_path)

        if existing_vectorstore:
            docs_embedding = bedrock_embeddings.embed_documents([doc.page_content for doc in docs])
            existing_vectorstore.add_documents(docs)
            # existing_vectorstore.add_embeddings(docs_embedding)
            print(f"New documents have been embedded and added to the existing Faiss index for {current_copilot}")
        else:
            existing_vectorstore = FAISS.from_documents(docs, bedrock_embeddings)
            print(f"A new index has beed created for {current_copilot}")

        existing_vectorstore.save_local(faiss_path)
    except Exception as e:
        raise ValueError(f"Error while executing get_vector_store: {str(e)}")
```

Figure 9: Vector Knowledge Base Creation

The below code snippet shows the prompt augmentation part as well as the response generation part.

```
# Prompt Template
prompt_template = """
Human: Use the following context to provide a concise and accurate answer to the question. If the question requires a short answer, keep it brief. If a detailed explanati
If you don't know the answer, just say that you don't know, don't try to make up an answer.
<context>{context}</context>

Question: {question}

Assistant:
"""

PROMPT = PromptTemplate(template=prompt_template, input_variables=["context", "question"])
##LLM Model - Create the Anthropic Claude-v2 model
def get_claude_llm():
    try:
        llm = BedrockLLM(model_id="anthropic.claude-v2", client=bedrock, model_kwargs={'max_tokens_to_sample': 500})

        return llm
    except Exception as e:
        raise ValueError(f"Error while connecting  claude_llm: {str(e)}")

##Generate Response - Generate a response from LLM based on user query
def get_response_llm(llm, vectorstore_faiss, query):
    try:
        if vectorstore_faiss is None:
            raise ValueError("vectorstore_faiss is None. Cannot proceed with as_retriever.")

        qa = RetrievalQA.from_chain_type(
            llm=llm,
            chain_type="stuff",
            retriever=vectorstore_faiss.as_retriever(search_type="similarity", search_kwargs={"k": 55}),
            return_source_documents=True,
            chain_type_kwargs={"prompt": PROMPT}
        )

        answer = qa.invoke({"query": query})

        return answer['result']
    except Exception as e:
        raise ValueError(f"Error while generating response from claude llm: {str(e)}")
```

Figure 10: Response Generation

B. Structured

a) Knowledge Graph-Enhanced Architecture

This approach integrates a metadata-driven knowledge graph to improve the chatbot's contextual understanding and response generation.

i) Key Steps:

- Data Enrichment: Add metadata (tags, attributes, relationships) to the dataset.
- Knowledge Graph Construction: Build a knowledge graph with nodes and edges for efficient querying.
- Query Processing: Match queries against the knowledge graph and embeddings using cosine similarity.
- Response Generation: Consolidate information and use Anthropic Claude to generate responses based on relationships and relevance.

ii) Data:

This below image (Fig 11) shows an example of structured data in rows and columns as CSV file.

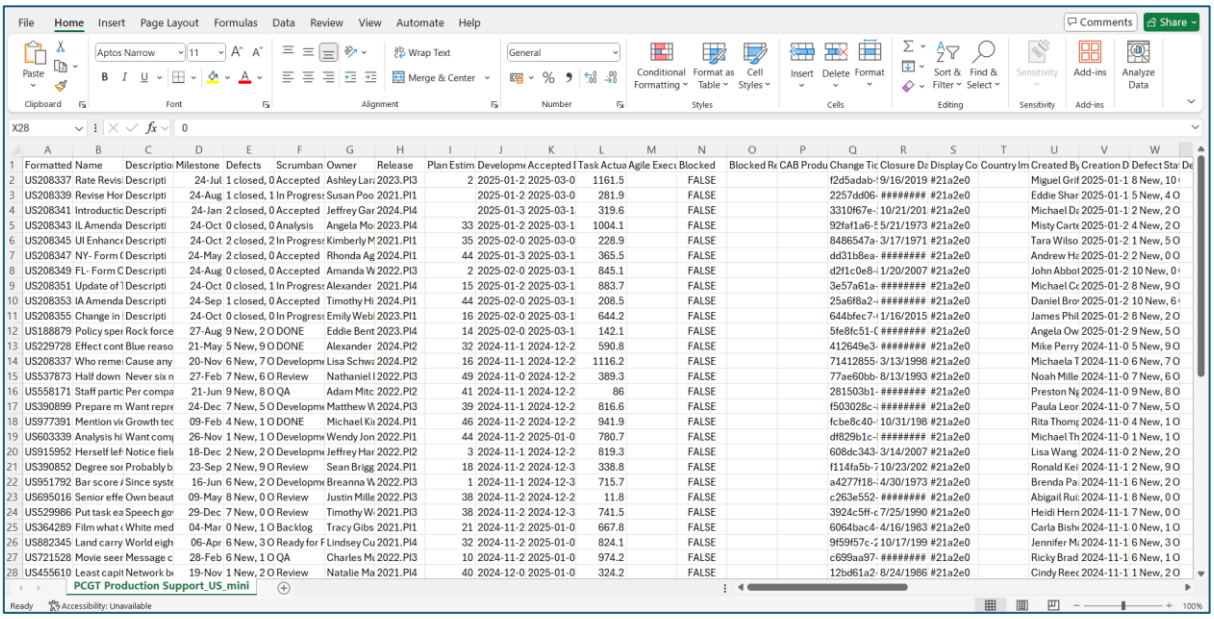


Figure 11: Structured Data – CSV File

iii) User Interface:

This below image (Fig 12) displays the user interface with the "Structured Data Copilot" option selected.

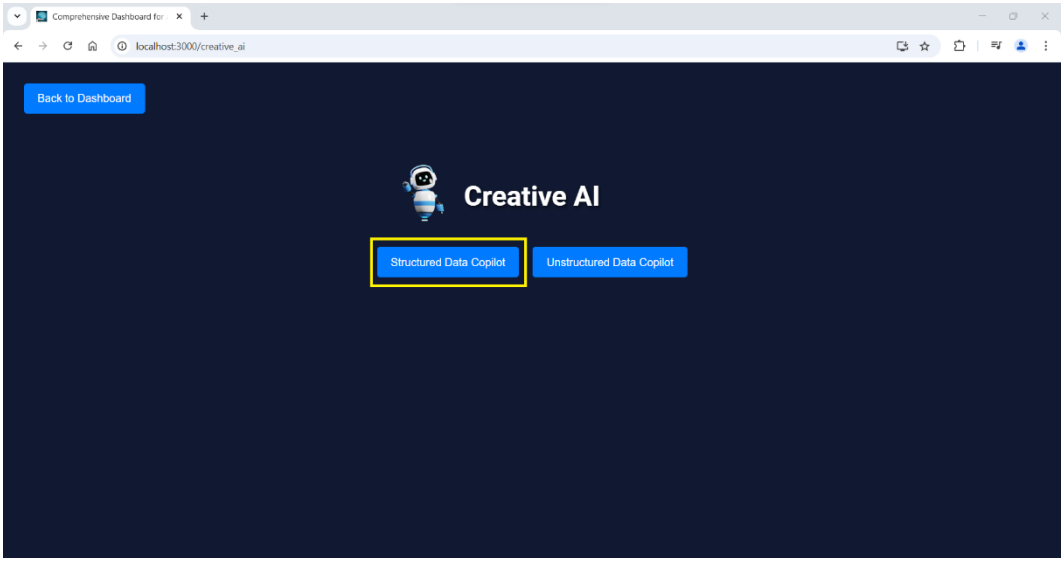


Figure 12: User Interface - "Structured Data Copilot" Option Selected

iv) *Response:*

This below image (Fig 13) displays the response for the query: "Based on current trends, which functional areas are likely to encounter the most defects in the next milestone?"

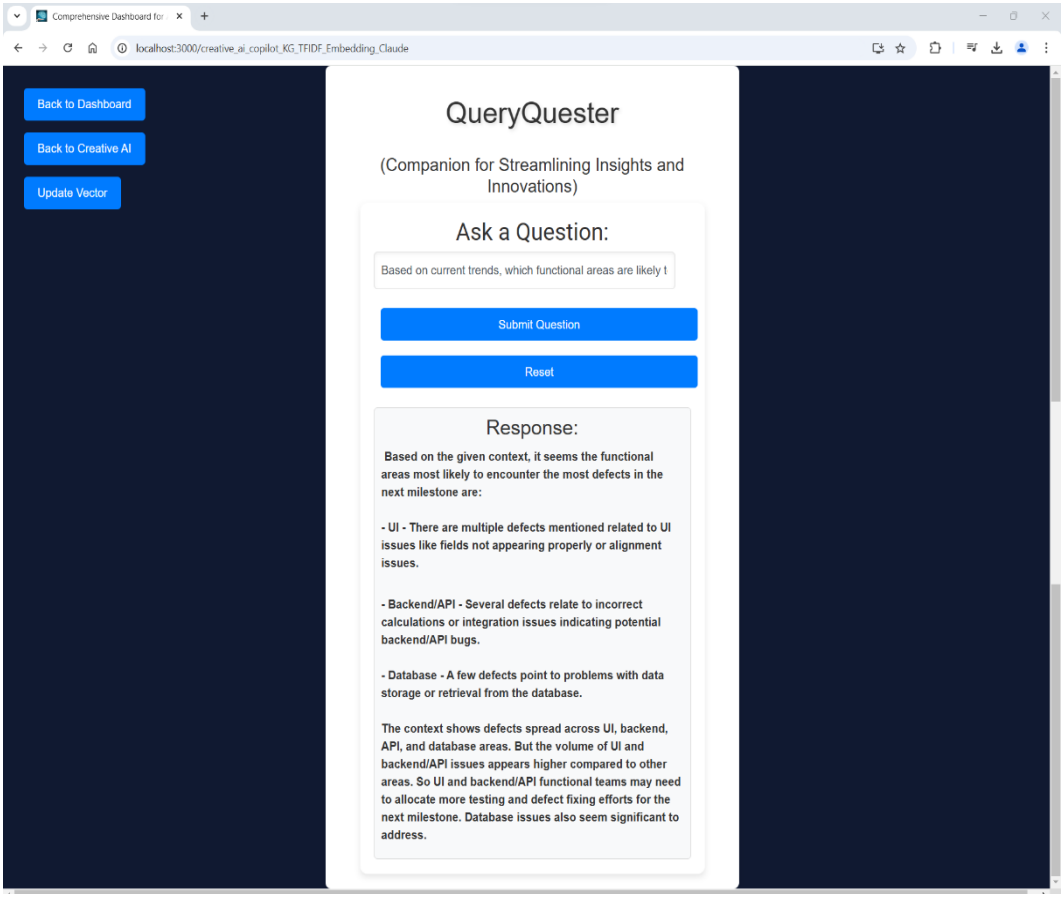


Figure 13: Chatbot Response

v) Code Snippet:

The below code snippet shows the Meta Data creation.

```
def extract_metadata_from_Defect(df):
    start_time = time()
    logging.info(f"Meta data creation for Defects is started")
    metadata = []
    for _, row in df.iterrows():
        metadata.append({
        "doc_id": str(uuid.uuid4()),  # Generate a unique doc_id
        "Formatted ID": row.get('Formatted ID', 'default_formatted_id'),
        "Name": row.get('Name', 'default_name'),
        "Description": row.get('Description', 'default_description'),
        "Requirement": row.get('Requirement', 'default_requirement'),
        "Milestones": row.get('Milestones', 'default_milestones'),
        "Defect State": row.get('Defect State', 'default_defect_state'),
        "Functional Area": row.get('Functional Area', 'default_functional_area'),
        "Owner Name": row.get('Owner Name', 'default_owner_name'),
        "Severity": row.get('Severity', 'default_severity'),
        "Priority": row.get('Priority', 'default_priority'),
        "State": row.get('State', 'default_state'),
        "Fixed In Build": row.get('Fixed In Build', 'default_fixed_in_build'),
        "Submitted By": row.get('Submitted By', 'default_submitted_by'),
        "Owner": row.get('Owner', 'default_owner'),
        "Milestones.1": row.get('Milestones.1', 'default_milestones_1'),
        "Creation Date": row.get('Creation Date', 'default_creation_date'),
        "Accepted Date": row.get('Accepted Date', 'default_accepted_date'),
        "Task Actual Total": row.get('Task Actual Total', 'default_task_actual_total'),
        "Addressed By": row.get('Addressed By', 'default_addressed_by'),
        "Affects Doc": row.get('Affects Doc', 'default_affects_doc'),
        "Blocked": row.get('Blocked', 'default_blocked'),
        "Blocked Reason": row.get('Blocked Reason', 'default_blocked_reason'),
        "Change Ticket": row.get('Change Ticket', 'default_change_ticket'),
        "Closed Date": row.get('Closed Date', 'default_closed_date'),
        "Closure Date": row.get('Closure Date', 'default_closure_date'),
        "Display Color": row.get('Display Color', 'default_display_color'),
        "Country Impacted": row.get('Country Impacted', 'default_country_impacted'),
        "Created By": row.get('Created By', 'default_created_by'),
        "Defect Type": row.get('Defect Type', 'default_defect_type'),
```

Figure 14: Meta Data Creation

The below code snippet shows the Knowledge Graph creation.

```python
def build_knowledge_graph(metadata):
    start_time = time()
    logging.info(f"knowledge graph creation  is started")
    G = nx.Graph()
    for data in metadata:
        doc_id = data['doc_id']

        if doc_id is None:
            continue  # Skip this entry if doc_id is None

        entities = data['entities']
        keywords = data['keywords']

        # Add document node
        G.add_node(doc_id, type='document')

        # Add entity nodes and edges
        for entity in entities:
            entity_text, entity_type = entity
            G.add_node(entity_text, type=entity_type)
            G.add_edge(doc_id, entity_text, relationship='contains')

        # Add keyword nodes and edges
        for keyword in keywords:
            G.add_node(keyword, type='keyword')
            G.add_edge(doc_id, keyword, relationship='contains')
    end_time = time()
    logging.info(f"knowledge graph creation  is completed")
    logging.info(f"knowledge graph creation  time: {end_time - start_time} seconds")

    return G
```

Figure 15: Knowledge Graph Creation

The below code snippet shows the prompt augmentation part as well as the response generation part.

```python
# Prompt template
prompt_template = """
Human: Use the following context to provide a concise and accurate answer to the question. If the question requires a short answer, keep it brief. If a detailed explanati
If you don't know the answer, just say that you don't know, don't try to make up an answer.
<context>{context}</context>

Question: {question}

Assistant:
"""
PROMPT = PromptTemplate(template=prompt_template, input_variables=["context", "question"])

def get_claude_llm():
    try:
        llm = BedrockLLM(model_id="anthropic.claude-v2", client=client, model_kwargs={'max_tokens_to_sample': 500})
        return llm
    except Exception as e:
        raise ValueError(f"Error while connecting to Claude LLM: {str(e)}")
        # Request payload, formatted to match Bedrock's structure
        payload = {
            "prompt": formatted_prompt,
            "max_tokens_to_sample": config['payload']['max_tokens_to_sample'],
            "temperature": config['payload']['temperature'],
            "top_k": config['payload']['top_k'],
            "top_p": config['payload']['top_p'],
        }
        # Send request to Amazon Bedrock
        response = client.invoke_model(
            modelId="anthropic.claude-v2",
            body=json.dumps(payload),
            contentType="application/json",
            accept="application/json"
        )
        # Parse and return response
        response_body = json.loads(response['body'].read().decode('utf-8'))
```

Figure 16: Response Generation

b) Hybrid RAG and Knowledge Graph Architecture

This approach combines RAG-based architecture with knowledge graph methodology to leverage the strengths of both.

i) Key Steps:

- Data Preparation: Create vector embeddings and a knowledge graph.
- Dual Retrieval: Match queries against both the FAISS index and the knowledge graph.
- Context Integration: Merge results from both retrieval systems.
- Response Generation: Use the combined context with Anthropic Claude to generate responses.

ii) Data:

This below image (Fig 17) shows an example of structured data in rows and columns from a SQL Database.

No limit

Data Output | Messages | Notifications

	id [PK] bigint	item text	request_number text	ticket_number text	requested_for text	opened_date text	opened_by text	stage text	state text	quantity integer	service_desk text	sla_due text
1	1	Access to Repository	REQ0000121	RTIM60175	Alice Johnson	10-10-2024	Alice Johnson	Open	In Progress	1	User	10-16-2024
2	2	System Error - Unable to login	REQ0000122	RTIM41186	Bob Smith	08-14-2024	Bob Smith	Closed	Closed -Complete	[null]	User	08-20-2024
3	3	Password Reset Request	REQ0000123	RTIM17341	Charlie Adams	07-15-2024	Charlie Adams	Open	In Progress	1	User	07-19-2024
4	4	Hardware Issue - Laptop not booting	REQ0000124	RTIM66952	Diana Rose	06-20-2024	Diana Rose	Closed	Closed -Complete	[null]	User	06-26-2024
5	5	VPN Connection Failure	REQ0000125	RTIM97608	Eve Davis	10-09-2024	Eve Davis	Open	In Progress	1	User	10-14-2024
6	6	Software Installation - MS Office	REQ0000126	RTIM35311	Frank Martin	06-25-2024	Bob Smith	Closed	Closed -Complete	[null]	User	07-01-2024
7	7	System Maintenance - Software Update	REQ0000127	RTIM94096	George King	07-25-2024	George King	Closed	Closed -In Complete	[null]	User	08-01-2024
8	8	Hardware Request - Keyboard	REQ0000128	RTIM43850	Hannah Lee	06-10-2024	Hannah Lee	Closed	Closed -Complete	[null]	User	06-17-2024
9	9	Email Issue - Unable to send/receive	REQ0000129	RTIM87834	Isaac Harris	10-10-2024	Eve Davis	Open	In Progress	2	User	10-16-2024
10	10	Access to Shared Drive	REQ0000130	RTIM29947	Jack White	07-20-2024	Jack White	Closed	Closed -In Complete	[null]	User	07-27-2024
11	11	Request for VPN Access	REQ0000131	RTIM10908	Karen Black	07-14-2024	Karen Black	Closed	Closed -Complete	[null]	User	07-20-2024
12	12	Software Crash - Adobe Photoshop	REQ0000132	RTIM55826	Liam Young	06-05-2024	Liam Young	Open	In Progress	2	User	06-11-2024
13	13	Performance Issue - Slow System	REQ0000133	RTIM47518	Mia Scott	08-15-2024	Mia Scott	Closed	Closed -Complete	[null]	User	08-20-2024
14	14	Hardware Request - Mouse	REQ0000134	RTIM52193	Noah Green	10-06-2024	Karen Black	Open	In Progress	2	User	10-13-2024
15	15	Access to HR Portal	REQ0000135	RTIM42745	Olivia Lewis	06-23-2024	Olivia Lewis	Open	In Progress	1	User	06-30-2024
16	16	Request for New Email Alias	REQ0000136	RTIM13671	Paul Walker	10-09-2024	Paul Walker	Open	In Progress	1	User	10-14-2024
17	17	Data Backup Request	REQ0000137	RTIM10769	Quinn Hill	07-20-2024	Quinn Hill	Open	In Progress	1	User	07-26-2024
18	18	Access to Financial Reports	REQ0000138	RTIM15979	Rachel Moore	10-08-2024	Rachel Moore	Open	In Progress	2	User	10-15-2024
19	19	Upgrade to New Laptop	REQ0000139	RTIM43634	Sophia Turner	06-05-2024	Sophia Turner	Open	In Progress	1	User	06-10-2024
20	20	Printer Connection Issue	REQ0000140	RTIM69759	Tom Baker	10-05-2024	Tom Baker	Open	In Progress	1	User	10-12-2024
21	21	Chat Support - VPN Access Issue	REQ0000141	INC54734	Sam Walker	09-22-2024	Jane Taylor	Closed	Closed -In Complete	[null]	Agent	09-28-2024

Figure 17: Structured Data – SQL Database

iii) User Interface:

This below image (Fig 18) displays the user interface with the "Structured Data Copilot" option selected.

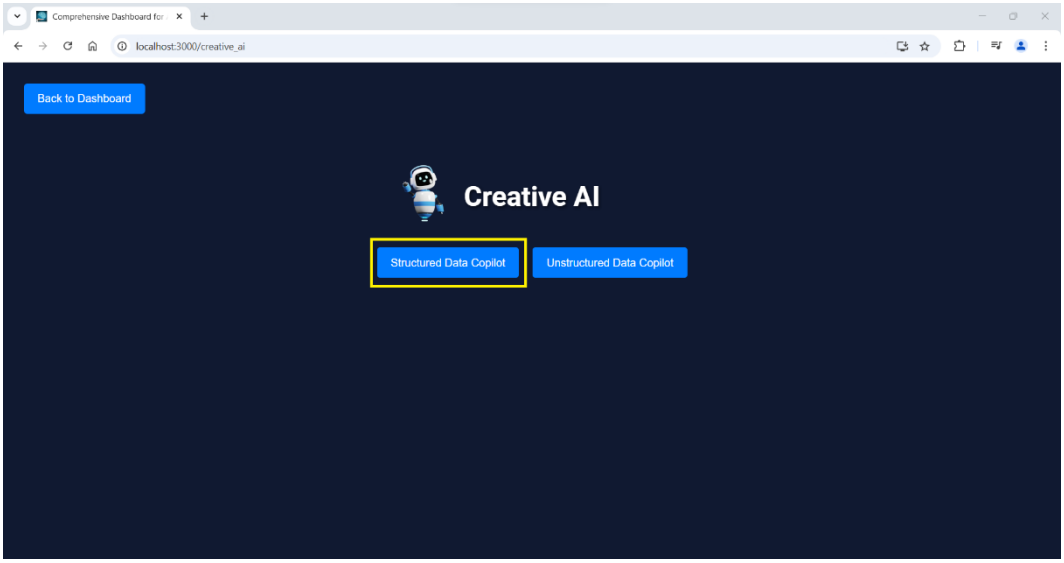


Figure 18: User Interface - "Structured Data Copilot" Option Selected

iv) *Response:*

This below image (Fig 19) displays the response for the query: "Based on current trends, which functional areas are likely to encounter the most defects in the next milestone?"

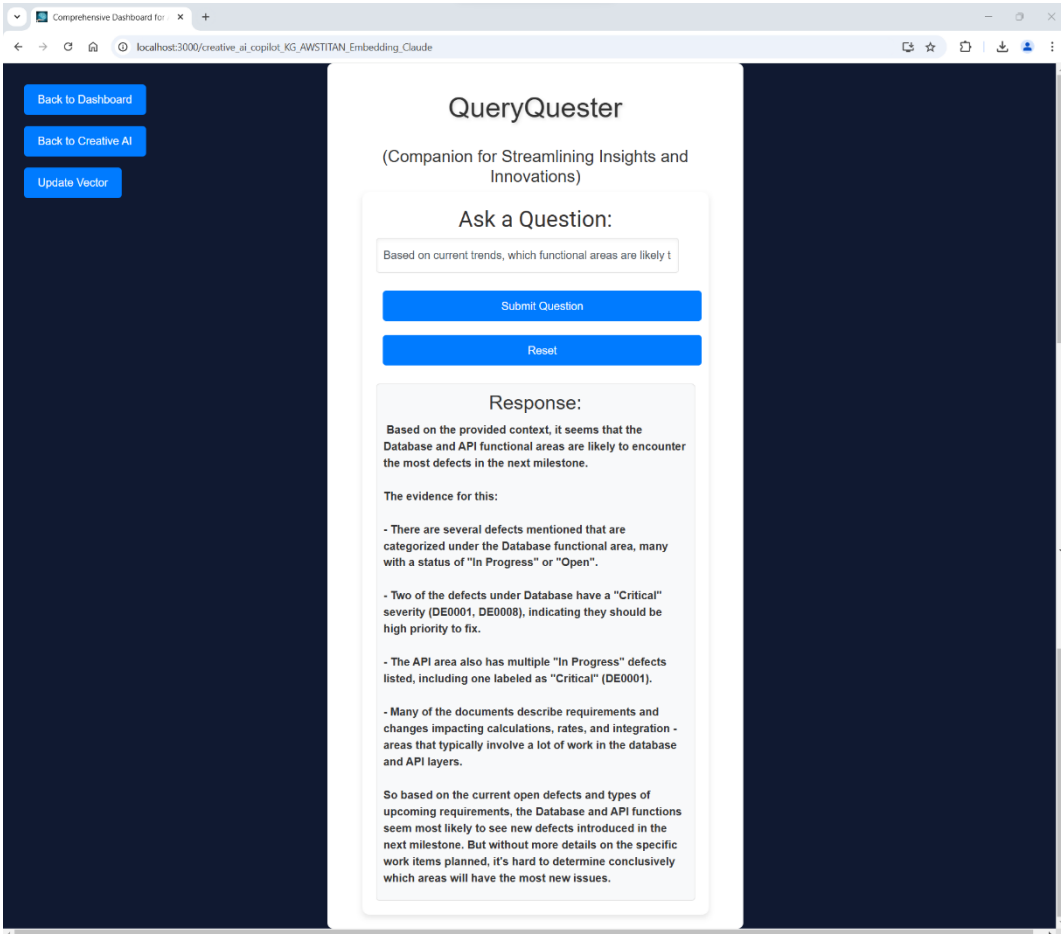


Figure 19: Chatbot Response

v) Code Snippet:

The below code snippet shows the Meta Data creation.

```python
def extract_metadata_from_Defect(df):
    start_time = time()
    logging.info(f"Meta data creation for Defects is started")
    metadata = []
    for _, row in df.iterrows():
        metadata.append({
        "doc_id": str(uuid.uuid4()),  # Generate a unique doc_id
        "Formatted ID": row.get('Formatted ID', 'default_formatted_id'),
        "Name": row.get('Name', 'default_name'),
        "Description": row.get('Description', 'default_description'),
        "Requirement": row.get('Requirement', 'default_requirement'),
        "Milestones": row.get('Milestones', 'default_milestones'),
        "Defect State": row.get('Defect State', 'default_defect_state'),
        "Functional Area": row.get('Functional Area', 'default_functional_area'),
        "Owner Name": row.get('Owner Name', 'default_owner_name'),
        "Severity": row.get('Severity', 'default_severity'),
        "Priority": row.get('Priority', 'default_priority'),
        "State": row.get('State', 'default_state'),
        "Fixed In Build": row.get('Fixed In Build', 'default_fixed_in_build'),
        "Submitted By": row.get('Submitted By', 'default_submitted_by'),
        "Owner": row.get('Owner', 'default_owner'),
        "Milestones.1": row.get('Milestones.1', 'default_milestones_1'),
        "Creation Date": row.get('Creation Date', 'default_creation_date'),
        "Accepted Date": row.get('Accepted Date', 'default_accepted_date'),
        "Task Actual Total": row.get('Task Actual Total', 'default_task_actual_total'),
        "Addressed By": row.get('Addressed By', 'default_addressed_by'),
        "Affects Doc": row.get('Affects Doc', 'default_affects_doc'),
        "Blocked": row.get('Blocked', 'default_blocked'),
        "Blocked Reason": row.get('Blocked Reason', 'default_blocked_reason'),
        "Change Ticket": row.get('Change Ticket', 'default_change_ticket'),
        "Closed Date": row.get('Closed Date', 'default_closed_date'),
        "Closure Date": row.get('Closure Date', 'default_closure_date'),
        "Display Color": row.get('Display Color', 'default_display_color'),
        "Country Impacted": row.get('Country Impacted', 'default_country_impacted'),
        "Created By": row.get('Created By', 'default_created_by'),
        "Defect Type": row.get('Defect Type', 'default_defect_type'),
```

Figure 20: Meta Data Creation

The below code snippet shows the Knowledge Graph creation.

```python
def build_knowledge_graph(metadata):
    start_time = time()
    logging.info(f"knowledge graph creation  is started")
    G = nx.Graph()
    for data in metadata:
        doc_id = data['doc_id']

        if doc_id is None:
            continue  # Skip this entry if doc_id is None

        entities = data['entities']
        keywords = data['keywords']

        # Add document node
        G.add_node(doc_id, type='document')

        # Add entity nodes and edges
        for entity in entities:
            entity_text, entity_type = entity
            G.add_node(entity_text, type=entity_type)
            G.add_edge(doc_id, entity_text, relationship='contains')

        # Add keyword nodes and edges
        for keyword in keywords:
            G.add_node(keyword, type='keyword')
            G.add_edge(doc_id, keyword, relationship='contains')
        end_time = time()
    logging.info(f"knowledge graph creation  is completed")
    logging.info(f"Knowldege graph creation  time: {end_time - start_time} seconds")

    return G
```

Figure 21: Knowledge Graph Creation

The below code snippet shows the Knowledge Graph and Meta Data.

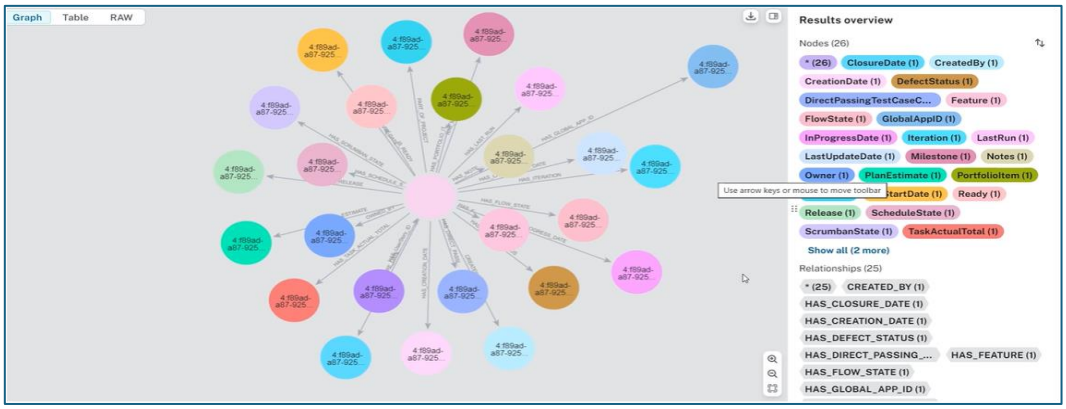


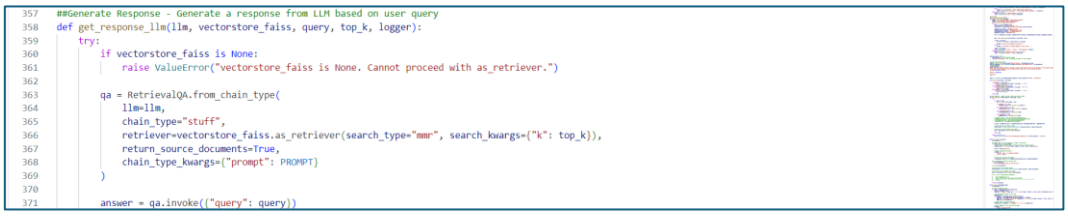
Figure 22: Knowledge Graph and Meta Data

The below code snippet shows the generation of the vector knowledge base.

```python
## vector embedding and vector store - Generate vector embeddings and store them using FAISS
def get_vector_store(docs, current_copilot, faiss_path):
    try:

        existing_vectorstore = load_vector_store(faiss_path)

        if existing_vectorstore:
            docs_embedding = bedrock_embeddings.embed_documents([doc.page_content for doc in docs])
            existing_vectorstore.add_documents(docs)
            # existing_vectorstore.add_embeddings(docs_embedding)
            print(f"New documents have been embedded and added to the existing Faiss index for {current_copilot}")
        else:
            existing_vectorstore = FAISS.from_documents(docs, bedrock_embeddings)
            print(f"A new index has beed created for {current_copilot}")

        existing_vectorstore.save_local(faiss_path)
    except Exception as e:
        raise ValueError(f"Error while executing get_vector_store: {str(e)}")
```

Figure 23: Vector Knowledge Base Creation

The below code snippet shows response generation part.



```
##Generate Response - Generate a response from LLM based on user query
def get_response_llm(llm, vectorstore_faiss, query, top_k, logger):
    try:
        if vectorstore_faiss is None:
            raise ValueError("vectorstore_faiss is None. Cannot proceed with as_retriever.")

        qa = RetrievalQA.from_chain_type(
            llm=llm,
            chain_type="stuff",
            retriever=vectorstore_faiss.as_retriever(search_type="mmr", search_kwargs={"k": top_k}),
            return_source_documents=True,
            chain_type_kwargs={"prompt": PROMPT}
        )

        answer = qa.invoke({"query": query})
```

Figure 24: Response Generation

C. Response Comparison Architecture (Unstructured and Structured):

Criteria	Simple RAG-Based	Knowledge Graph-Enhanced	Hybrid RAG and Knowledge Graph
Clarity	Clear and concise, with structured details.	Clear but slightly more verbose.	Clear and concise, with structured details.
Detail	Includes additional project details like release, owner, and milestone.	Focuses on the user story and defect details without extra project information.	Includes acceptance criteria and specific project details like release, owner, and milestone.
Summary	Provides a brief summary at the end.	Provides a brief summary at the end.	Provides a brief summary at the end.
Format	Uses bullet points and structured format for easy reading.	Uses paragraphs, making it slightly harder to scan quickly.	Uses bullet points and structured format for easy reading.
Project Information	Includes specific project details (release, owner, plan estimate, milestone).	Mentions the project name but lacks specific details like release and owner.	Includes specific project details (release, owner, plan estimate, milestone).
Defect State	Specifies the state of the defect (resolved).	Does not specify the state of the defect.	Specifies the number of closed and open defects.
User Story Details	Provides a concise description of the user story.	Provides a detailed description of the user story.	Provides a detailed description of the user story with acceptance criteria.

IV. DISCUSSION

A. Testing and Evaluating Chatbot Performance

a) Metrics for Chatbot Evaluation:

Metrics are crucial for assessing chatbot performance and user satisfaction. Accuracy measures how correctly a chatbot interprets user inputs and generates appropriate responses, using precision, recall, and F1 scores. User satisfaction is gauged through surveys and ratings like Net Promoter Score (NPS) and Customer Satisfaction Score (CSAT), providing insights into the user experience. Response time measures the speed at which a chatbot processes user queries, which is critical for user retention and satisfaction. Engagement metrics assess the depth and richness of conversations, indicating how well a chatbot maintains ongoing dialogue with users.

b) User Testing Techniques

User testing ensures chatbots meet user needs. The think-aloud protocol involves users verbalizing their thoughts during interaction, helping researchers understand the user's thought process and identify areas of confusion. A/B testing compares two versions of a chatbot to determine which performs better in terms of user engagement and satisfaction. Surveys and questionnaires collect quantitative and qualitative feedback from users, providing valuable insights into their experience. Observational studies involve real-time analysis of user interactions, offering context that quantitative data may overlook.

c) Iterative Improvement Strategies

Iterative improvement involves continuous refinement through several phases. **Evaluation** involves analysing interactions to identify strengths and weaknesses. **Modification** entails implementing changes based on insights gathered during evaluation. **Testing** involves rigorous evaluation of these modifications to measure improvements in user satisfaction, accuracy, and engagement. Finally, **deployment** marks the transition of the improved chatbot into a live setting, with ongoing monitoring and adjustments to ensure sustained performance.

B. Ethical Considerations in Chatbot Development

a) Bias and Fairness in LLMs

Bias and fairness in large language models (LLMs) are critical concerns. These models, trained on vast datasets, can reflect societal biases, perpetuating stereotypes and discrimination. Bias can stem from training data, model fine-tuning, and the model's architecture. To mitigate bias, developers should use diverse training datasets, employ bias detection algorithms,

and implement user feedback mechanisms. Addressing bias is an ongoing challenge that requires ethical AI practices to create equitable chatbot experiences.

b) Privacy Concerns

Privacy concerns are paramount in chatbot development using LLMs. These models require vast amounts of data, raising questions about data collection, storage, and use. Developers must curate training data to exclude sensitive information and implement data minimization practices. Transparency in data handling and robust security measures, such as encryption and regular audits, are essential. Compliance with regulations like GDPR and CCPA is crucial for protecting user privacy and enhancing chatbot reputation.

c) Responsible AI Practices

Responsible AI practices ensure that LLMs are used fairly, transparently, and beneficially. Mitigating bias in training data, maintaining transparency about chatbot functions, and prioritizing user privacy are key aspects. Developers should select diverse datasets, communicate clearly about AI processes, and implement strong data protection measures. Engaging in interdisciplinary discussions about AI ethics fosters a culture of responsibility, ensuring that AI advancements align with societal values and contribute positively to the community.

V. FUTURE SCOPE

The integration of Large Language Models (LLMs) in chatbot development presents numerous opportunities for future research and advancements. Enhancing Natural Language Understanding (NLU) capabilities by leveraging more advanced LLMs can lead to more accurate and human-like interactions. Additionally, ensuring scalability and efficiency is crucial as the demand for chatbots increases. Future work can optimize architectures, such as RAG-based and Knowledge Graph-enhanced models, to handle larger datasets and more complex queries without compromising performance. Personalization and adaptability are also promising directions, where chatbots can tailor interactions based on individual user preferences through adaptive learning algorithms.

Another exciting area for future research is the integration of multimodal data, incorporating text, voice, and visual inputs to provide a richer user experience. Ethical and responsible AI usage is becoming increasingly important, and future research should focus on developing frameworks for transparency, fairness, and accountability in chatbot interactions. Hybrid architectures combining RAG-based and Knowledge Graph-enhanced models offer a robust framework for handling both unstructured and structured data and refining these models can improve their accuracy and efficiency. Expanding the application of chatbots to various domains, such as healthcare, finance, and education, can enhance their effectiveness in specialized fields. Finally, implementing continuous learning mechanisms will ensure that chatbots remain up to date with the latest information and user trends, enabling them to evolve and improve over time.

By addressing these areas, future research can significantly advance the capabilities of chatbots, making them more intelligent, efficient, and user-friendly. The integration of LLMs with innovative architectures holds great promise for the next generation of conversational agents.

VI. CONCLUSION

This paper presented a comprehensive overview of the integration of Large Language Models (LLMs) to enhance chatbot capabilities. It covered various facets, including the design and architecture of chatbots for managing both unstructured and structured data. The significant contributions of AWS Titan and Anthropic Claude models in developing retrieval-augmented generation (RAG) systems were also highlighted.

The results and discussion sections provided an in-depth comparison of execution outcomes from both unstructured and structured data sets, offering a detailed analysis of their performance differences. The paper meticulously evaluated and evaluated chatbot performance, covering several critical aspects:

- Testing and Evaluation: The performance metrics examined ensured that the chatbots were not only effective but also efficient in their operations.
- Ethical Considerations: The ethical implications of chatbot development were discussed, emphasizing the necessity of creating AI systems that respect user privacy and data security.
- Future Research Directions: The paper proposed future research avenues, such as optimizing RAG-based models and enhancing knowledge graph-integrated systems. These suggestions aim to improve the models' scalability and efficiency, enabling them to manage larger datasets and more complex queries without compromising performance.

The comprehensive analysis and thoughtful recommendations provided in this paper underscore the transformative potential of LLMs in advancing chatbot functionalities. By addressing current challenges and suggesting pathways for future research, the paper lays a robust foundation for continued innovation in this field.

A. Interest Conflicts

The authors declare that there are no conflicts of interest related to this paper.

B. Funding statement

Research and publication of this paper are self-funded.

VII. REFERENCES

[1] Abbasian, M., Khatibi, E., Azimi, I., Oniani, D., Abad, Z. S. H., Thieme, A., Yang, Z., Wang, Y., Lin, B., Gevaert, O., Li, L.-J., Jain, R., & Rahmani, A. M. (2023). Foundation Metrics: Quantifying Effectiveness of Healthcare Conversations powered by Generative AI. In arXiv (Cornell University). Cornell University. https://doi.org/10.48550/arXiv.2309.

[2] Abedi, M., Alshybani, I., Shahadat, M., & Murillo, M. S. (2023). Beyond Traditional Teaching: The Potential of Large Language Models and Chatbots in Graduate Engineering Education. https://doi.org/10.32388/md04b0

[3] Adolphs, L., Shuster, K., Urbanek, J., Szlam, A., & Weston, J. (2021). Reason first, then respond: Modular Generation for Knowledge-infused Dialogue. In arXiv (Cornell University). Cornell University. https://doi.org/10.48550/arxiv.2111.05204

[4] Arun, A., Batra, S., Bhardwaj, V., Challa, A., Dönmez, P., Heidari, P., Inan, H., Jain, S., Kumar, A., Mei, S., Mohan, K., & White, M. (2020). Best Practices for Data-Efficient Modeling in NLG: How to Train Production-Ready Neural Models with Less Data (p. 64). https://doi.org/10.18653/v1/2020.coling-industry.7

[5] Ashfaq, M., Yun, J., Yu, S., & Loureiro, S. M. C. (2023). I, Chatbot: Modeling the determinants of users' satisfaction and continuance intention of AI-powered service agents. https://www.sciencedirect.com/science/article/pii/S0736585320301325

[6] Bastola, A., Wang, H., Hembree, J., Yadav, P., McNeese, N. J., & Razi, A. (2023). LLM-based Smart Reply (LSR): Enhancing Collaborative Performance with ChatGPT-mediated Smart Reply System. In arXiv (Cornell University). Cornell University. https://doi.org/10.48550/arXiv.2306.

[7] Cantador, I., Viejo-Tardío, J., Cortés-Cediel, M. E., & Bolívar, M. P. R. (2021). A Chatbot for Searching and Exploring Open Data: Implementation and Evaluation in E-Government. https://doi.org/10.1145/3463677.3463681

[8] Castillo-Bolado, D., Davidson, J. K., Gray, F., & Rosa, M. K. A. (2024). Beyond Prompts: Dynamic Conversational Benchmarking of Large Language Models. In arXiv (Cornell University). Cornell University. https://doi.org/10.48550/arxiv.2409.20222

[9] Chair, Y., Liu, A., Rogers, A., Magdy, W., Preoţiuc-Pietro, D., Akhtar, S., Αλέτρας, N., Bontcheva, K., Darwish, K.-R., ElSherief, M., Garimella, K., Guerini, M., Jaidka, K., McGillivray, B., Mejova, Y., Naseem, U., Roß, B., Thorne, J., Viviani, M., … Huang, C.-Y. (2023). Findings of the Association for Computational Linguistics: ACL 2023. In Findings of the Association for Computational Linguistics: ACL 2022. https://doi.org/10.18653/v1/2023.findings-acl

[10] Chaves, A. P., & Gerosa, M. A. (2020). How Should My Chatbot Interact? A Survey on Social Characteristics in Human–Chatbot Interaction Design. In International Journal of Human-Computer Interaction (Vol. 37, Issue 8, p. 729). Taylor & Francis. https://doi.org/10.1080/10447318.2020.1841438

[11] https://dblp.uni-trier.de/db/conf/lrec/lrec2016.html#ChiarainC16

[12] Dai, S., Wang, G., Park, S., & Lee, S. (2021). Dialogue Response Generation via Contrastive Latent Representation Learning (p. 189). https://doi.org/10.18653/v1/2021.nlp4convai-1.18

[13] Douze, M., Guzhva, A., Deng, C., Johnson, J., Szilvasy, G., Mazaré, P.-E., Lomelí, M., Hosseini, L., & Jégou, H. (2024). The Faiss library. In arXiv (Cornell University). Cornell University. https://doi.org/10.48550/arxiv.2401.08281

[14] Dowlagar, S., & Mamidi, R. (2021). CMSAOne@Dravidian-CodeMix-FIRE2020: A Meta Embedding and Transformer model for Code-Mixed Sentiment Analysis on Social Media Text. In arXiv (Cornell University). Cornell University. https://doi.org/10.48550/arXiv.2101.

[15] Fadhil, A. (2018). Domain Specific Design Patterns: Designing For Conversational User Interfaces. In arXiv (Cornell University). Cornell University. https://doi.org/10.48550/arxiv.1802.09055

[16] Finch, S. E., Paek, E. S., & Choi, J. D. (2023). Leveraging Large Language Models for Automated Dialogue Analysis. In arXiv (Cornell University). Cornell University. https://doi.org/10.48550/arxiv.2309.06490

[17] Følstad, A., & Brandtzæg, P. B. (2020). Users' experiences with chatbots: findings from a questionnaire study. In Quality and User Experience (Vol. 5, Issue 1). Springer Science+Business Media. https://doi.org/10.1007/s41233-020-00033-2

[18] Friedman, L., Ahuja, S., Allen, D. T., Tan, Z., Sidahmed, H., Long, C., Xie, J., Schubiner, G., Patel, A., Lara, H., Chu, B., Chen, Z., & Tiwari, M. K. (2023). Leveraging Large Language Models in Conversational Recommender Systems. In arXiv (Cornell University). Cornell University. https://doi.org/10.48550/arxiv.2305.07961

[19] Hofmann, V., Pierrehumbert, J. B., & Schütze, H. (2020). Dynamic Contextualized Word Embeddings. In arXiv (Cornell University). Cornell University. https://doi.org/10.48550/arXiv.2010.

[20] Huang, F., & Strigini, L. (2021). HEDP: A Method for Early Forecasting Software Defects based on Human Error Mechanisms. In arXiv (Cornell University). Cornell University. https://doi.org/10.48550/arxiv.2110.06758

[21] Isa, N. A. N. M., Jawaddi, S. N. A., & Ismail, A. (2024). Experimental Evaluation of Machine Learning Models for Goal-oriented Customer Service Chatbot with Pipeline Architecture. In arXiv (Cornell University). Cornell University. https://doi.org/10.48550/arxiv.2409.18568

[22] Jacky Casas, jacky.casas@hes-so.ch, Marc-Olivier Tricot, Omar Abou Khaled, omar.aboukhaled@hes-so.ch, Elena Mugellini, elena.mugellini@hes-so.ch, Philippe Cudré-Mauroux, philippe.cudre-mauroux@unifr.ch. (2023). Trends & Methods in Chatbot Evaluation. https://dl.acm.org/doi/10.1145/3395035.3425319

[23] Khan, A., Hasan, M. T., Kemell, K. K., Rasku, J., & Abrahamsson, P. (2024). Developing Retrieval Augmented Generation (RAG) based LLM Systems from PDFs: An Experience Report. In arXiv (Cornell University). Cornell University. https://doi.org/10.48550/arxiv.2410.15944

[24] Konreddy, S. D. R. (2021). The Impact of NLP on Software Testing. In Journal of University of Shanghai for Science and Technology (Vol. 23, Issue 8, p. 295). https://doi.org/10.51201/jusst/21/08380

[25] Li, C., Zhang, M., Mei, Q., Wang, Y., Hombaiah, S. A., Liang, Y., & Bendersky, M. (2023). Teach LLMs to Personalize -- An Approach inspired by Writing Education. In arXiv (Cornell University). Cornell University. https://doi.org/10.48550/arXiv.2308.

[26] Liu, B., & Mazumder, S. (2020). Lifelong Learning Dialogue Systems: Chatbots that Self-Learn On the Job. In arXiv (Cornell University). Cornell University. https://doi.org/10.48550/arxiv.2009.10750

[27] Mao, K., Dou, Z., Chen, H., Mo, F., & Qian, H. (2023). Large Language Models Know Your Contextual Search Intent: A Prompting Framework for Conversational Search. In arXiv (Cornell University). Cornell University. https://doi.org/10.48550/arXiv.2303.

[28] Pahune, S., & Manoj, C. (2023). Several Categories of Large Language Models (LLMs): A Short Survey. In International Journal for Research in Applied Science and Engineering Technology (Vol. 11, Issue 7, p. 615). International Journal for Research in Applied Science and Engineering Technology (IJRASET). https://doi.org/10.22214/ijraset.2023.54677

[29] Pantano, E., & Pizzi, G. (2020). Forecasting artificial intelligence on online customer assistance: Evidence from chatbot patents analysis. In Journal of Retailing and Consumer Services (Vol. 55, p. 102096). Elsevier BV. https://doi.org/10.1016/j.jretconser.2020.102096

[30] Parnin, C., Soares, G., Pandita, R., Gulwani, S., Rich, J. A. J., & Henley, A. Z. (2023). Building Your Own Product Copilot: Challenges, Opportunities, and Needs. In arXiv (Cornell University). Cornell University. https://doi.org/10.48550/arxiv.2312.14231

[31] Pozdniakov, S., Brazil, J., Abdi, S., Bakharia, A., Sadiq, S., Gašević, D., Denny, P., & Khosravi, H. (2024). Large Language Models Meet User Interfaces: The Case of Provisioning Feedback. In arXiv (Cornell University). Cornell University. https://doi.org/10.48550/arXiv.2404.

[32] Pyatkin, V., Roit, P., Michael, J., Goldberg, Y., Tsarfaty, R., & Dagan, I. (2021). Asking It All: Generating Contextualized Questions for any Semantic Role. In arXiv (Cornell University). Cornell University. https://doi.org/10.48550/arxiv.2109.04832

[33] Engineering (Vol. 29, Issue 11, p. 1673). World Scientific. https://doi.org/10.1142/s0218194019400163

[34] Saad-Falcon, J., Barrow, J., Siu, A., Nenkova, A., Rossi, R. A., & Dernoncourt, F. (2023). PDFTriage: Question Answering over Long, Structured Documents. In arXiv (Cornell University). Cornell University. https://doi.org/10.48550/arxiv.2309.08872

[35] Shanahan, M., McDonell, K., & Reynolds, L. (2023). Role play with large language models [Review of Role play with large language models]. Nature, 623(7987), 493. Nature Portfolio. https://doi.org/10.1038/s41586-023-06647-8

[36] Silva, G. R. S., & Canedo, E. D. (2022). Towards User-Centric Guidelines for Chatbot Conversational Design. In International Journal of Human-Computer Interaction (Vol. 40, Issue 2, p. 98). Taylor & Francis. https://doi.org/10.1080/10447318.2022.2118244

[37] Song, C., & Raghunathan, A. (2020). Information Leakage in Embedding Models. In Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security (p. 377). https://doi.org/10.1145/3372297.3417270

[38] Tang, X., Shin, R., Inan, H. A., Manoel, A., Mireshghallah, F., Lin, Z., Gopi, S., Kulkarni, J., & Sim, R. B. (2023). Privacy-Preserving In-Context Learning with Differentially Private Few-Shot Generation. In arXiv (Cornell University). Cornell University. https://doi.org/10.48550/arxiv.2309.11765

[39] Tanioka, H. (2019). A Fast Content-Based Image Retrieval Method Using Deep Visual Features. In arXiv (Cornell University). Cornell University. https://doi.org/10.48550/arxiv.1908.01505

[40] Tennenholtz, G., Chow, Y., Hsu, C., Jeong, J., Shani, L., Tulepbergenov, A., Ramachandran, D., Mladenov, M., & Boutilier, C. (2023). Demystifying Embedding Spaces using Large Language Models. In arXiv (Cornell University). Cornell University. https://doi.org/10.48550/arXiv.2310.

[41] Vijayaraghavan, V., Cooper, J. B., & J., R. L. (2020). Algorithm Inspection for Chatbot Performance Evaluation. In Procedia Computer Science (Vol. 171, p. 2267). Elsevier BV. https://doi.org/10.1016/j.procs.2020.04.245

[42] Villalba, A. C., Brown, E. M., Scurrell, J. V., Entenmann, J., & Daepp, M. I. G. (2023). Automated Interviewer or Augmented Survey? Collecting Social Data with Large Language Models. In arXiv (Cornell University). Cornell University. https://doi.org/10.48550/arxiv.2309.10187

[43] Wang, K., Ramos, J., & Lawrence, R. (2024). ChatEd: A Chatbot Leveraging ChatGPT for an Enhanced Learning Experience in Higher Education. In arXiv (Cornell University). Cornell University. https://doi.org/10.48550/arXiv.2401.

[44] Wang, K., Ramos, J., & Lawrence, R. (2024). ChatEd: A Chatbot Leveraging ChatGPT for an Enhanced Learning Experience in Higher Education. In arXiv (Cornell University). Cornell University. https://doi.org/10.48550/arxiv.2401.00052

[45] Wynn, D. C., & Maier, A. (2022). Feedback systems in the design and development process. In Research in Engineering Design (Vol. 33, Issue 3, p. 273). Springer Science+Business Media. https://doi.org/10.1007/s00163-022-00386-z

[46] Xia, P., Zhu, K., Li, H., Zhu, H., Li, Y., Li, G., Zhang, S., & Yao, H. (2024). RULE: Reliable Multimodal RAG for Factuality in Medical Vision Language Models. In arXiv (Cornell University). Cornell University. https://doi.org/10.48550/arxiv.2407.05131

[47] Xiong, H., Bian, J., Yang, S., Zhang, X., Kong, L., & Zhang, D. (2023). Natural Language based Context Modeling and Reasoning for Ubiquitous Computing with Large Language Models: A Tutorial. In arXiv (Cornell University). Cornell University. https://doi.org/10.48550/arxiv.2309.15074

[48] Zamfirescu-Pereira, J. D., Hartmann, B., & Yang, Q. (2023). Conversation Regression Testing: A Design Technique for Prototyping Generalizable Prompt Strategies for Pre-trained Language Models. In arXiv (Cornell University). Cornell University. https://doi.org/10.48550/arxiv.2302.03154