"MoodiBot": RAG Based Chatbot Providing Intra-Organizational Information In Various Mood's

Dana Mikulinsky Saar David Nitzan Manor
Technion
{mdana@, saardavid@, nitzan.manor@}
campus.technion.ac.il

Github link

Abstract

Chatbots play a crucial role in improving customer service, yet traditional models often struggle to provide accurate and contextually relevant responses, particularly for complex queries related to specific personal matters as users rights within organizations. This project develops a Retrieval-Augmented Generation (RAG) chatbot designed to address these challenges by integrating retrieval and generative capabilities. Focusing on the "Maccabi Health Services" organization data, "MoodiBot" utilizes semantic chunking and Gemini embeddings to efficiently represent data, ensuring precise retrieval of information. To enhance user experience, a MongoDB database tracks conversation history, allowing the chatbot to provide personalized interactions tailored to individual user needs. By combining large language models with a structured, vectorized knowledge base, "MoodiBot" synthesizes relevant information to deliver contextually accurate responses. It is designed for adaptive interaction, offering various conversational styles (a.k.a moods), from formal guidance to youth-friendly tones, ensuring user engagement across diverse demographics. The evaluation process measures the chatbot's accuracy and contextual relevance by comparing its responses against established ground-truth answers and assessing retrieval performance. This chatbot's unique integration of real-time retrieval, the several interaction styles, and adaptability makes it a practical tool for rights-based support in healthcare and beyond.

1 Introduction

Chatbots became an essential tool for organizations to enhance customer service and provide instant information. However, traditional chatbot models often struggle with delivering accurate, contextually relevant answers to complex queries, as they rely on static responses and have difficulty retrieving information from external sources. This challenge is particularly significant when users seek specific information, such as their rights within an organization.

A promising solution is the implementation of Retrieval-Augmented Generation (RAG) systems, which blend information retrieval and generative models. Unlike conventional chatbots, RAG enables retrieval of relevant information from external knowledge bases, leading to more informed and personalized responses. However, existing chatbot solutions often rely on rule-based systems, which lack flexibility, or generative models, which can be inaccurate. Additionally, large language models (LLMs) can sometimes produce responses that are overly confident yet inaccurate, making them prone to delusional outputs. Neither method effectively integrates external knowledge, resulting in outdated or irrelevant responses.

"Moodibot" [1] aims to address the limitations of RAG models by developing a chatbot that boosts contextual accuracy while retrieving information. Focusing on customers' rights within "Maccabi Health Services", utilizing semantic chunking, Gemini embeddings, and a MongoDB retriever for efficient handling the data. It incorporates conversation history to improve personalization and coherence. With the ability to switch between several conversation moods, the "MoodiBot" delivers a tailored user experience and features a user-friendly front-end for smooth interaction. An evaluation framework has been implemented to assess the chatbot's precision, relevance, correctness, and faithfulness based on a predefined ground truth.

2 Methodology

Developing "MoodiBot" system required a multi-stage process that involved data collection and preprocessing, maintaining and manage an organized database of embedded chunks and chat histories, comparison of several LLM's, embedding and re-

trieval methods, prompt engineering and creation of the chatbot itself.

2.1 Datasets and Pre-processing Steps

The dataset used was sourced from "Maccabi Health Services" organization's public website and contained information on customer rights. Since the original data written in Hebrew, we first translated it to English in order to ensure accessibility for a broader audience and the option to use of readily available pre-trained models, embeddings and other sources.

We initially scraped the information from Maccabi website, saving each page as a separate document. For the translation process, we used AWS's API services to handle the task, translating each document individually. Afterwards, we manually reviewed the translations documents to identify and correct common errors, particularly issues with Hebrew names (such as "My Maccabi" instead of "Maccabi Shelli") that were mistranslated, and made the necessary adjustments.

To improve the relevance and accuracy of responses, we applied a technique called semantic chunking to the translated documents - method of dividing text into manageable, meaningful parts that retain context rather than using arbitrary boundaries like sentences or paragraphs. The process involves sliding a window over the document text, generating embeddings for text segments within this window, comparing the similarity of adjacent segments using a score threshold and creating splits when the similarity drops below the threshold, indicating a shift in semantic content. This approach ensures that chunks maintain semantic coherence, typically ranging from 100 to 400 tokens. To preserve context, the method also includes a small portion of the preceding and following chunks (200 characters) in each split. This technique results in more meaningful and context-aware text segments, enhancing the effectiveness of the RAG pipeline in retrieving relevant information for user queries.

We used 50 documents that were chunked to approximately 300 chunks. Once the documents were chunked, they were passed through an embedding model to generate embeddings—dense vector representations of the text. These embeddings allowed for semantic search and retrieval based on user queries. We have tried three embedding models - two of them provided by "Google" (text-embedding-004 and embedding-001) [8] and the last provided by "Cohere" [8], all of them are free, pretrained embeddings models. A key difference between the two providers is the length of the embedding (768 elements in "Google" embeddings vs. 384 in "Cohere"'s).

2.2 Database Management

The project implements a robust database management system using MongoDB. Additionally, we employ MongoDB Atlas's indexing and querying mechanisms to handle a vectorized DB [8]. A dedicated DBHandler class is responsible for handling all database operations, ensuring efficient storage and retrieval of data.

This class manages two primary collections:

An embeddings collection stores document embeddings for each model that was evaluated while a chat history collection maintains conversation histories for individual users.

The DBHandler utilizes a connection string to establish a secure connection with the database. It provides methods to retrieve chat history and update both embeddings and chat history collections. This approach allows for scalable and organized data management across different organizations and users.

2.3 "MoodiBot" Pipeline

The project's pipeline is managed by the Chatbot class, which coordinates the various components required to deliver a seamless question-answering experience. This class serves as the central interface through which all key operations are conducted - initialization, question processing, context retrieval, and response generation. Key Aspects of the Pipeline are:

Initialization: The chatbot is initialized with an instance of the DBHandler class, ensuring that it can interact with the database for retrieving embeddings and chat histories. Additionally, optional style parameters can be passed during initialization, allowing for customization of response formats based on user preferences.

Language Model Integration: The system is designed with flexibility in mind, supporting multiple large language models (LLMs). Currently, it supports interactions with Google's Gemini, Meta's llama and Mistral models [8]. This modular design allows the pipeline to switch between different LLMs, a thing that was critical in order to perform evaluation and the ability to choose the ideal model.

Question Processing: The pipeline begins by rephrasing the user's question to ensure it is stan-

dalone and context-independent. This step is crucial for improving the accuracy of the response, as it helps the system frame the query in a way that can be understood without relying on previous conversational context. The rephrasing itself occurs using the same LLM chosen for the answering part.
Context Retrieval: Once the question is processed, the system uses semantic search to retrieve relevant information from the database. This involves querying the embeddings collection to find the most contextually relevant documents or knowledge snippets, which are then used to build the answer. This was done by ANN and KNN methods.
Answer Generation: After retrieving the relevant context, the system combines it with the rephrased question to form a prompt. This prompt is sent to the chosen LLM, which generates an appropriate and informed response. If any style instructions were provided during initialization, these are added to the generated answer, ensuring that the response aligns with the desired tone and structure. "MoodiBot" instructed to answer based solely on the provided context and information as the health care provider's assistant it is.
History Management: Following the response generation, the system updates the chat history collection in the database. This ensures that each interaction is logged, enabling the system to reference past conversations and maintain a coherent flow of dialogue across multiple interactions. When a user restart the chat, the history is deleted in order to create a context-free new chat.
The pipeline delivers a smooth and consistent question-answering experience, backed by reliable data management and powerful language models.

2.4 Moods

"MoodiBot" is designed to adapt its conversation style based on the user's needs or preferences, offering different modes or "moods" that influence both tone and response style. The available moods include Elderly-friendly, Kids-friendly, Emojis, and Rhymes. We chose to implement this feature in order to give a unique conversation experience. The first two mood - elderly and kids are set to address these two segments of population in more appealing way. As example, in kids mood the chat is prompted to answer questions by simplifying complex concepts, using analogies and avoiding harsh health topics. In elderly mood, the bot should provide clear explanations to ensure understanding particularly for technology-related issues. The other two moods, Emojis and Rhymes, are designed to add a fun and memorable twist to the conversation.

2.5 API Integration

The project utilizes a Flask-based API to expose its functionality, facilitating smooth integration with frontend applications. This RESTful API provides endpoints for core operations: Chatbot initialization, history retrieval, Question answering through the RAG pipeline and history reset. Key benefits of this API approach include modularity, scalability and cross-platform compatibility.
The API uses JSON for data exchange, ensuring easy parsing and handling of information between the frontend and backend. The frontend is built using React and Material-UI, creating a responsive and user-friendly chat interface. This implementation focuses on component-based architecture and separates concerns between chat functionality and overall management. The frontend communicates with the backend API using fetch requests, handling both synchronous and asynchronous operations seamlessly.

3 Experiments

The evaluation process aims to assess both the retriever component and the overall chatbot response quality across a variety of configurations. This dual approach ensures that "MoodiBot" answers are not only accurate but also grounded in the correct information retrieved from the database. Each configuration varies by language model, embedding type, search method, and response style (a.k.a mood). The different moods, language models and embedding types tested were mentioned earlier. The ground truth dataset, consisting of manually created question-answer pairs, was prepared by the team members to serve as the benchmark for evaluation. We checked and documented the chunks id of each document that we based a question-answer pair according to.

3.1 Answer Quality Evaluation

The evaluation process of the chatbot employs a comprehensive set of metrics to assess its accuracy, faithfulness, and relevance in providing information on behalf of the organization. Each question-answer pair is assessed using the following metrics:
Cosine Similarity: This metric evaluates the semantic similarity between the chatbot's response and the ground truth answer. Both responses are vectorized using a natural language processing model

(e.g., Spacy embeddings), and the cosine of the angle between these vectors is computed. A higher cosine similarity indicates a closer semantic match between the chatbot's response and the ground truth.
Correctness Score: This metric assesses the factual accuracy of the chatbot's response by considering three factors: the overlap of key words, the overlap of named entities, and the cosine similarity between the chatbot's answer and the ground truth. The correctness score is calculated as the average of these three components, ensuring that the evaluation captures both the precise content (keywords and entities) and the broader semantic meaning (cosine similarity).
Faithfulness Score: The faithfulness score measures how closely the chatbot's answer aligns with the information retrieved from the organization's knowledge base. This metric ensures that the chatbot provides answers that are grounded in the source material, preventing the introduction of inaccurate or extraneous information. The faithfulness score is computed by determining the cosine similarity between the embedding of the retrieved context (i.e., the relevant information from the knowledge base) and the chatbot's response to the question.

3.2 Retriever Evaluation

In order to measure the effectiveness of the retriever, we employ a scoring system that assesses how well the retriever identifies and returns pertinent chunks of information from the database. The evaluate function compares the chunks retrieved by the chatbot with the relevant chunks identified in the ground truth data. The scoring system includes three primary metrics:
Precision: Measures the proportion of retrieved chunks that are relevant to the query.
Recall: Measures the proportion of relevant chunks that were successfully retrieved.
F1 Score: Provides a harmonic mean of precision and recall, offering a balanced evaluation of the retriever's performance.

3.3 Results

Retriever experiments showed that the exact and approximate search gained the same results for all relevant metrics (precision, recall and f1) [2]. This could be due to the fact that our dataset is not very large, which allowed the retriever to preform with identical results for both ANN and KNN search methodologies supported by MongoDB's Atlas Vector Search [8].
Moreover, it seems that Google's 004 embedding got the highest scores in all metrics [2]. Yet , overall results from testing the ground truth shows that the highest F1 score was only 0.72. We zoomed in on the 10 question average score [3] and managed to see that most of recall scores were mostly 1, beside one question that scores very low (A question referencing to a document chunked over more than 20 chunks). On the other hand, precision score is between 0.4 to 0.8 for almost all of the question. Each document spread on an average of 2-3 chunks (with exception of extra large documents, containing long lists of detailed information of phones and addresses), thus a document that was split to 2 chunks for example can get a maximal precision score of 0.66 when retrieving n=3 chunks (as per our configured logic), although the retriever succeeded to retrieve all of the related chunks the score stays low.
As per the evaluation of MoodiBot's answers quality we were able to determine a notable supremacy of Google's embedding models compared to Cohere's model (that provides with a much shorter embedding vector) [4]. With the LLM models we didn't see substantial differences when comparing the cosine similarity scores across models [6], with a slight preference for Mistral-72 model. When looking at correctness and faithfulness scores, Mistral-72 model was also the leading choice. This result is aligned with the current public benchmarks [8] which show models comparison, featuring this reletably small model as one of the leading models among the available free possibilities.
Comparing results in different moods [7], it appears that the moods did not significantly impact the performance across the three metrics (Cosine Similarity, Correctness Score, and Faithfulness Score), indicating that style variation does not noticeably affect the quality or reliability of the metrics. This result is important for determining whether each model is robust to stylistic changes in the output.
Keeping in mind that the embedding model affects both retrieving and answer-quality scores, but the LLM only affects the last one, we wished to choose the embedding model that preforms best on both mission, for which we looked at the F1 and cosine similarity scores per embedding type. Seemingly Mistral-7B and Gemini-004 achieving high cosine similarity score [5].

4 Discussion and Conclusions

4.1 Conclusions

After reviewing the results, we concluded that the best-performing configuration was Mistral-7B as LLM model due to it's absolute leadership showed in the last section. Choosing the embedding model, it seems that question-answering quality is lead by Gemini-001 but due to it's poorer results in retriever evaluation [4], we chose Gemini-004 that has better retrieving scores and similar cosine similarity score with Mistral-7B combination [5]. When comparing the approximate and exact retrieval methods, we observed no difference in performance [2]. This can likely be attributed to the limited variety of documents in the knowledge base. Additionally, we found that MoodiBot's responses remained consistent and reliable even when switching between different styles [7], which is crucial for maintaining the product's credibility.

4.2 Challenges

Throughout the project, we faced several challenges that impacted our workflow and decision-making. The most significant obstacle was our inability to incorporate the LangChain library due to conflicting environment requirements, which we were unable to resolve within the project's timeframe. LangChain would have provided valuable support for integrating multiple LLMs and evaluating them with the RAGAS framework. As a result, we had to manually create functions to interact with the APIs and build the evaluation pipeline from scratch.

A follow-up challenge arose during the evaluation pipeline development, specifically in assessing the retriever's performance. We needed to determine the relevant chunks in the knowledge base for each question in the test set. Our goal was to use an LLM to iterate through all the chunks and rate their relevance for each question, but we were unable to achieve this with the available free APIs. These either produced low-quality answers (e.g., grading the entire knowledge base as irrelevant) or had usage rate limitations. To overcome this, we manually created the ground truth for chunk relevance, which in turn limited the number of questions we were able to test.

4.3 Future work

To address some of the challenges we encountered, as discussed in the previous section, we would have liked the ability to work with larger LLMs, which could have improved the evaluation process. This could've been done by using a paid LLM service or alternatively installing an LLM locally for direct interaction. However, both solutions require additional resources and technical support. Another potential step is expanding the database size, which would test the retriever's performance under greater demands and allow us to explore alternative retrieval methodologies.

A key aspect of MoodiBot's development was experimenting with various instruction prompts, including those for answering, context, and stylistic guidance. We believe that with an LLM offering a larger context window, and more testing time, we could have achieved better results. Additionally, experimenting with summarizing the context before passing it to the final LLM might improve answer accuracy, although it could also affect the runtime.

To better support knowledge bases containing documents of highly variable lengths, we might consider more sophisticated methods for determining the number of chunks to retrieve. This is currently unsupported in our system, as we've set a fixed retrieval of three documents.

8 References

1. Mongo DB Atlas Vector Search - link
2. Embeddings in the Gemini API - link
3. Embeddings with Cohere - link
4. Mistral-7B - link
5. all-MiniLM-L6 - link
6. llama - link
7. LLM Benchmarks Overview - link

Figure 1: "MoodiBot - Follow READ ME note in the Github repository to use it on your local environment"

9 Plotted Results

embedding_type	search_method	precision	recall	f1
Gemini-001	approximate	0.566666	0.651042	0.529524
	exact	0.599999	0.701042	0.569524
Gemini-004	approximate	0.733334	0.884375	0.722857
	exact	0.733334	0.884375	0.722857
all-MiniLM-L6-v2	approximate	0.633334	0.792708	0.627619
	exact	0.633334	0.792708	0.627619

Figure 2: Retriever evaluation scores for different embedding type and search methods

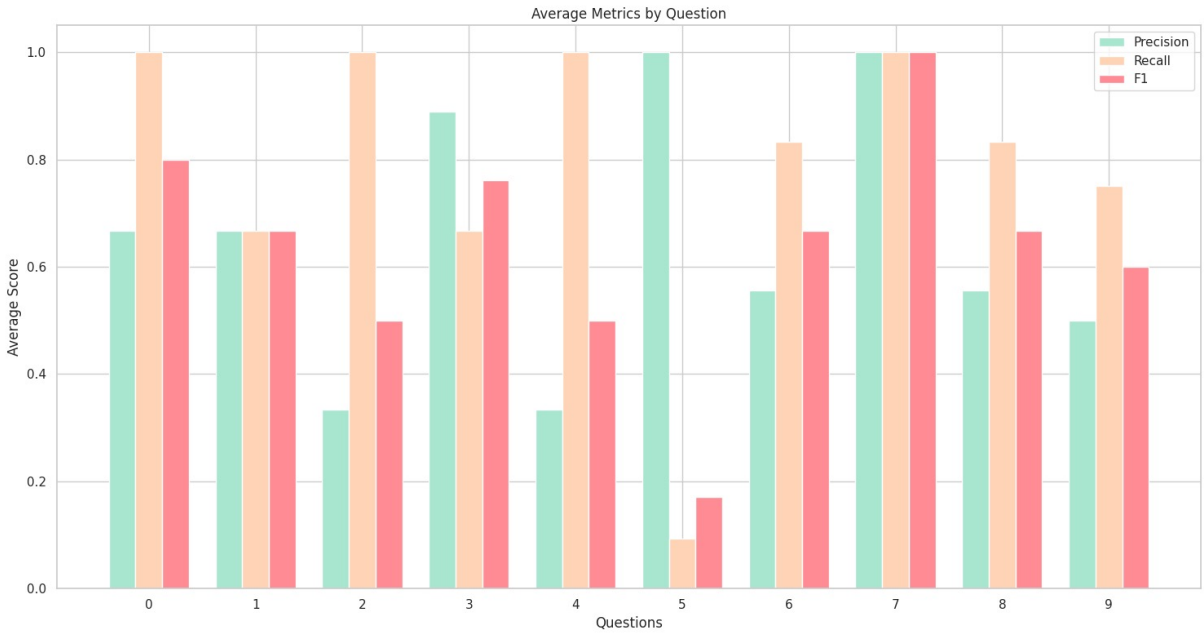

Figure 3: Average Metric for each ground-truth question checked

embedding_type	cosine_similarity	f1
Gemini-001	0.844528	0.549524
Gemini-004	0.815760	0.722857
all-MiniLM-L6-v2	0.652236	0.627619

Figure 4: Embedding Models scores on retrieving and answering

llm_name	embedding_type	cosine_similarity
Mistral-7B	Gemini-001	0.849103
gemini-1.5-flash	Gemini-001	0.844910
LLaMa 3.1 8B	Gemini-001	0.839570
Mistral-7B	Gemini-004	0.826347
LLaMa 3.1 8B	Gemini-004	0.815832
gemini-1.5-flash	Gemini-004	0.805102
Mistral-7B	all-MiniLM-L6-v2	0.673156
LLaMa 3.1 8B	all-MiniLM-L6-v2	0.646548
gemini-1.5-flash	all-MiniLM-L6-v2	0.637004

Figure 5: Cosine similarity evaluation of LLM and Embedding model combinations

llm_name	cosine_similarity	correctness_score	faithfulness_score
Mistral-7B	0.782868	0.551512	0.818641
LLaMa 3.1 8B	0.767317	0.503411	0.743110
gemini-1.5-flash	0.762339	0.459252	0.676431

Figure 6: Embedding Models scores on retrieving and answering

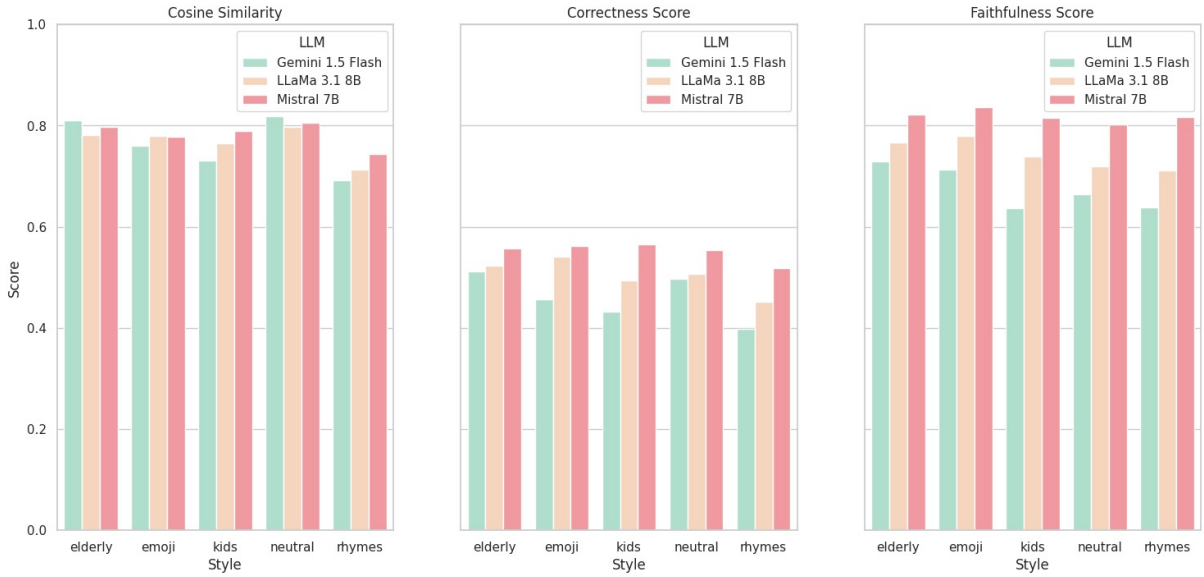

Figure 7: Cosine Similarity, Correctness and Faithfulness scores compared between the different LLM's and moods