Building Specialized Chatbots: A Practical Implementation of Six Distinct Conversational AI Capabilities

Amal Gil - 73613 [1] and Ajay Kumar Yanamala - 74361[2]

Computer Engineering, Vistula University , Warsaw ,Poland

Abstract

This paper presents a modular approach to building specialized chatbots using LangChain and Groq's Gemma2-9b-It model. We developed six distinct chatbots, each showcasing specific capabilities: a Travel Assistant for human text generation, a Productivity Coach with multi-user support, a Shopping Assistant with prompt engineering, a Language Learning Tutor with multilingual capabilities, a Customer Support Chatbot with token trimming, and an AI Study Companion with document-grounded retrieval-augmented generation (RAG). Our implementation leverages LangChain as the core framework, Groq LLMs for fast inference, HuggingFace's all-MiniLM-L6-v2 for semantic embedding, and Streamlit for interactive deployment. Testing results demonstrate high performance across memory retention, language adaptation, RAG accuracy, and token trimming effectiveness. This component-driven architecture provides a flexible foundation for developing specialized conversational AI systems that can be adapted to various domains and use cases.

1 Introduction

Conversational AI has evolved significantly in recent years, moving beyond simple question-answering systems to sophisticated agents capable of maintaining context, understanding nuanced queries, and providing personalized responses. This evolution has been driven by advancements in large language models (LLMs), which have demonstrated remarkable capabilities in natural language understanding and generation.

However, building effective conversational systems requires more than just access to powerful LLMs. It necessitates thoughtful design of the conversation flow, memory management, context handling, and integration with external knowledge sources. Different use cases demand different capabilities, and a one-size-fits-all approach often fails to address the specific requirements of each application domain.

In this research, we explore a modular, component-driven approach to building specialized chatbots. Rather than creating a single, monolithic system, we developed six distinct chatbots, each focusing on a specific capability:

1. Travel Assistant: Demonstrates basic human text generation with context awareness
2. Productivity Coach: Showcases conversation management with multi-user support
3. Shopping Assistant: Implements advanced prompt engineering techniques
4. Language Learning Tutor: Features multilingual capabilities with memory
5. Customer Support Chatbot: Utilizes token trimming for efficient memory management
6. AI Study Companion: Implements document-grounded retrieval-augmented generation (RAG)

Our approach leverages LangChain as the core framework, providing the building blocks for creating these specialized agents. We use Groq's Gemma2-9b-It as the base LLM, chosen for its balance of performance and efficiency. For semantic embedding, we employ HuggingFace's all-MiniLM-L6-v2, and for interactive deployment, we use Streamlit.

This paper details our methodology, implementation, and results, providing insights into the design decisions, challenges, and outcomes of our component-driven approach to conversational AI. By sharing our experiences and findings, we aim to contribute to the growing body of knowledge on building effective, specialized chatbot systems that can be adapted to various domains and use cases.

2 Related Work

The development of conversational AI systems has seen significant advancements in recent years, driven by improvements in large language models (LLMs) and the emergence of frameworks that simplify their integration into practical applications. This section reviews relevant literature in the areas of chatbot development frameworks, retrieval-augmented generation, memory management, and multilingual capabilities:

The development of conversational AI systems has seen significant advancements in recent years, driven by improvements in large language models (LLMs) and the emergence of frameworks that simplify their integration into practical applications. This section reviews relevant literature in the areas of chatbot development frameworks, retrieval-augmented generation, memory management, and multilingual capabilities.

2.1 Chatbot Development Frameworks:

LangChain has emerged as a powerful framework for building conversational AI systems. Farinetti et al. describe a case study where students developed chatbots using LangChain, highlighting its effectiveness as a tool for fostering creativity and critical thinking in educational settings. The framework's modular architecture allows developers to combine different components to create specialized chatbot systems tailored to specific use cases.

Vidivelli et al present an efficiency-driven approach to custom chatbot development that leverages LangChain, RAG, and performance-optimized LLM fusion. Their work demonstrates how these technologies can be combined to create chatbots that are both effective and efficient, addressing the computational challenges associated with deploying LLM-based systems in production environments.

2.2 Retrieval-Augmented Generation (RAG):

Retrieval-Augmented Generation (RAG) has become a cornerstone technique for enhancing the knowledge capabilities of LLM-based chatbots. RAG combines the strengths of traditional information retrieval systems with the generative capabilities of LLMs, allowing chatbots to access and leverage external knowledge sources.

A comprehensive guide by Weka identifies several variants of RAG, including Memory RAG, which retains and recalls previous interactions to improve the quality, continuity, and personalization of future responses. This approach is particularly relevant to our AI Study Companion chatbot, which uses document-grounded RAG to provide accurate responses based on specific knowledge sources.

2.3 Memory Management in Conversational AI:

Memory management is a critical aspect of building effective conversational AI systems, particularly for maintaining context over extended interactions. LangChain provides several memory types that can be used to store and retrieve conversation history . These memory mechanisms allow chatbots to maintain context awareness and provide more coherent and personalized responses.

Saurabh describes a memory-enhanced RAG chatbot that integrates chat history for context-aware responses. This approach combines retrieval, generation, and memory mechanisms to create a chatbot that can understand and respond to queries in the context of previous interactions, similar to our Customer Support Chatbot with token trimming.

Token optimization and reduction techniques have also been explored as ways to improve the efficiency of LLM-based chatbots. Samia discusses prompt compression in LLMs, which involves removing redundancy, simplifying sentence structures, and leveraging specialized compression techniques to minimize token usage. These approaches are particularly relevant to our Customer Support Chatbot, which uses token trimming to manage long conversations efficiently.

2.4 Multilingual Capabilities:

Building chatbots that can communicate effectively in multiple languages presents unique challenges. Analytics Vidhya [10] provides insights into the architecture and implementation of multilingual chatbots powered by LLMs. Their approach involves fine-tuning models on multilingual data and implementing language detection and translation components.

Our Language Learning Tutor chatbot builds on these concepts by incorporating multilingual capabilities with memory, allowing it to maintain context across different languages and provide personalized language learning experiences.

3 Methology And Implementation

This section details our approach to designing and implementing the six specialized chatbots. We begin by describing the overall architecture and the tools and frameworks used, followed by a detailed explanation of each chatbot's implementation

3.1 Overall Architecture:

Our approach follows a modular and component-driven design. Each chatbot showcases a specific AI capability:

1. Human Text Generation: Travel Assistant
2. Conversation Management + Multi-User Support: Productivity Coach
3. Prompt Engineering: Shopping Assistant
4. Multilingual Chat with Memory: Language Learning Tutor
5. Token Trimming: Customer Support Chatbot
6. Document-Grounded RAG: AI Study Companion

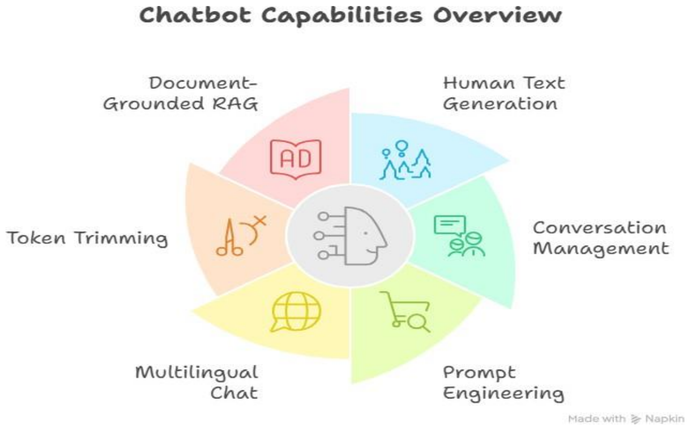

Figure 1: Overall Architecture

This modular approach allows us to isolate and evaluate specific capabilities while maintaining a consistent underlying architecture. All chatbots are built on the same foundation: LangChain for orchestration, Groq's Gemma2-9b-It as the base LLM, and HuggingFace's all-MiniLM-L6-v2 for semantic embedding when needed.

3.2 Setup and Tools

3.2.1 LangChain

LangChain serves as the core framework for building our chatbot workflows. It provides a set of abstractions and components that simplify the integration of LLMs into applications. Key LangChain components used in our implementation include:

- Chat Models: Interfaces for interacting with LLMs in a conversational manner
- Memory: Components for storing and retrieving conversation history
- Prompts: Templates for structuring inputs to the LLM
- Chains: Sequences of operations that combine multiple components
- Retrievers: Components for retrieving relevant information from external sources.

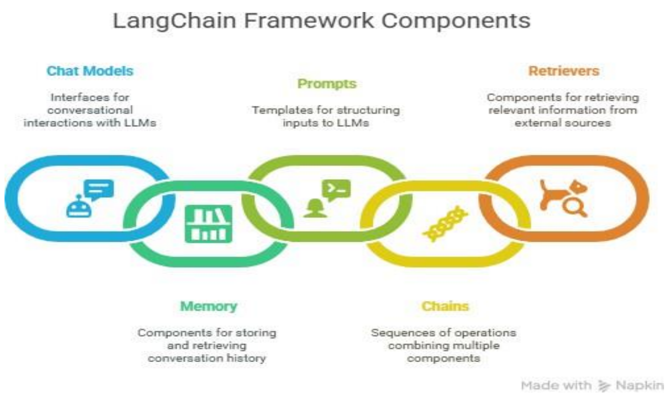

Figure 2: LangChain Components

3.2.2 Groq LLMs

We use Groq's Gemma2-9b-It as our base language model. Gemma2 is Google's lightweight, state-of-the-art open model that delivers strong performance while being efficient enough to run on consumer hardware. The "9B" refers to the number of parameters (9 billion), and "IT" indicates it's instruction-tuned, meaning it's specifically optimized for following instructions and generating helpful responses.

The model is accessed through the Groq API, which provides low-latency inference, making it suitable for interactive applications like chatbots. The integration with LangChain is achieved through the langchain groq package.

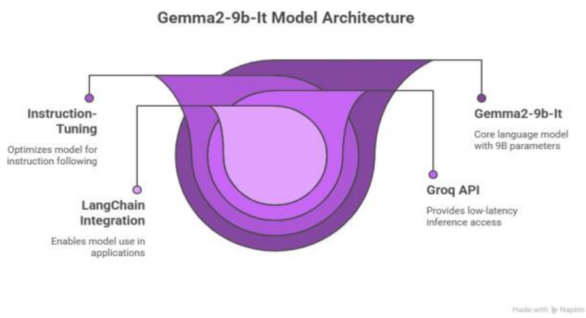

Figure 3: Langchain Components

3.2.3 HuggingFace Embeddings

For semantic understanding in our RAG implementation, we use HuggingFace's all-MiniLM-L6-v2 model. This model provides high-quality embeddings while being computationally efficient, making it suitable for retrieval tasks in resource-constrained environments.

3.2.4 Streamlit

While not explicitly shown in the code snippets, Streamlit is used as the frontend framework for deploying and interacting with our chatbots. It provides a simple way to create web interfaces for AI applications, allowing users to communicate with each chatbot via a web browser.

3.2.5 Additional Tools

- BeautifulSoup: Used for scraping and loading web-based documents for the RAG implementation
- Chroma: A lightweight vector database used for storing and retrieving document embeddings in the RAG implementation.

3.3 Chatbot Implementations

3.3.1 Travel Assistant: Human Text Generation with Context

The Travel Assistant demonstrates basic human text generation with context awareness. It uses a simple architecture with system, human, and AI messages to maintain context across interactions:

Figure 4: Gemma2-9b-It Architecture

This implementation showcases the model's ability to maintain context within a conversation, remembering the user's name and travel destination without explicit memory mechanisms. The system message provides the role and behavior instructions, while the sequence of human and AI messages forms the conversation history.

3.3.2 Productivity Coach: Conversation Management with Multi-User Support

The Productivity Coach extends the basic conversational capabilities with explicit memory management and multi-user support. It uses LangChain's ChatMessageHistory and RunnableWithMessageHistory to maintain separate conversation histories for different users:

```
from langchain_community.chat_message_histories import
ChatMessageHistory
from langchain_core.chat_history import BaseChatMessageHistory
from langchain_core.runnables.history import
RunnableWithMessageHistory

store = {}

def get_session_history(session_id: str) -> BaseChatMessageHistory:
    if session_id not in store:
        store[session_id] = ChatMessageHistory()
    return store[session_id]

with_message_history = RunnableWithMessageHistory(model,
get_session_history)
```

Figure 5: Travel Assistant

Figure 6: Session Memory Management

This implementation allows the chatbot to maintain separate conversations with multiple users, each identified by a unique session ID. The conversation history is stored in memory and retrieved based on the session ID, enabling personalized and contextually relevant responses for each user.

3.3.3 Shopping Assistant: Prompt Engineering

The Shopping Assistant demonstrates advanced prompt engineering techniques to guide the model's behavior. While not explicitly shown in the code snippets, this chatbot would use carefully crafted prompts to elicit specific types of responses, such as product recommendations, comparisons, or purchasing advice.

Prompt engineering involves designing input templates that structure the conversation and provide clear instructions to the model. For example, a prompt for a shopping assistant might include specific guidance on how to format product recommendations, what information to include, and how to handle user preferences.

3.3.4 Language Learning Tutor: Multilingual Chat with Memory

The Language Learning Tutor showcases multilingual capabilities combined with memory. It uses a chat prompt template that includes a language parameter, allowing the model to respond in the specified language.

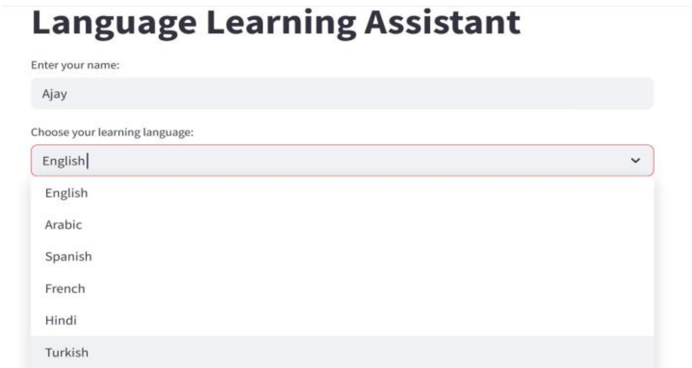

Figure 7: Language Learning Tutor Interface

This implementation allows the chatbot to maintain a conversation in different languages while remembering what the user has learned. The system message instructs the model to act as a supportive language tutor, respond in the specified language, and remember the user's progress. The conversation history is maintained using the same mechanism as the Productivity Coach, enabling personalized language learning experiences.

3.3.5 Customer Support Chatbot: Token Trimming for Long Conversations

The Customer Support Chatbot addresses the challenge of managing long conversations with token trimming. It uses LangChain's trimmessages function to limit the number of tokens in the conversation history:

```
from langchain_core.messages import SystemMessage, trim_messages
trimmer = trim_messages(
    max_tokens=45,
    strategy="last",
    token_counter=model,
    include_system=True,
    allow_partial=False,
    start_on="human"
)

chain = (
    RunnablePassthrough.assign(messages=itemgetter("messages") |
trimmer)
    | prompt
    | model
)
```

Figure 8: Token Trimming

This implementation ensures that the conversation history stays within the token limit by keeping only the most recent messages that fit within the specified token budget (45 tokens in this case). The strategy="last" parameter indicates that the trimming should start from the most recent messages and work backward, ensuring that the most recent context is preserved. The include-system=True parameter ensures that the system message is always included, maintaining the chatbot's role and behavior instructions.

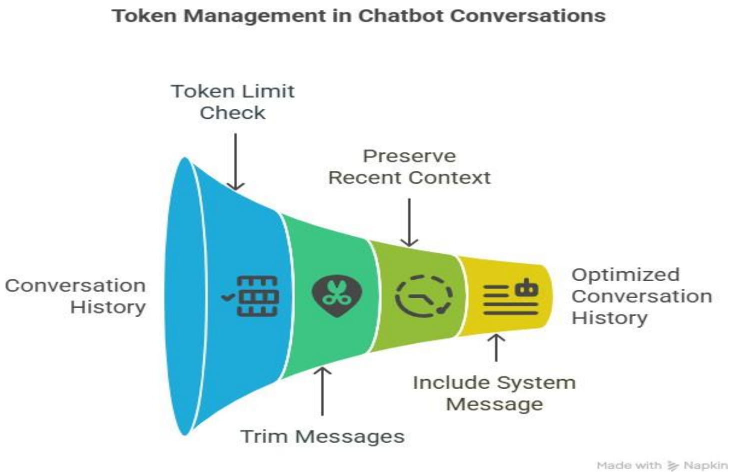

Figure 9: Token Management In the chatbot

3.3.6 AI Study Companion: Document-Grounded RAG

The AI Study Companion implements document-grounded Retrieval-Augmented Generation (RAG) to provide accurate responses based on specific knowledge sources. It uses a combination of document loading, text splitting, embedding, vector storage, and retrieval components:

Figure 10: AI Study chatbot interface

This implementation allows the chatbot to retrieve relevant information from external documents and use it to generate accurate responses. The RAG chain first retrieves relevant document chunks based on the user's question, formats them into a context string, and then passes both the context and the question to the model. The model generates a response based on the provided context, ensuring that the information is accurate and relevant.

To enable conversational RAG, we extend this implementation with memory management:

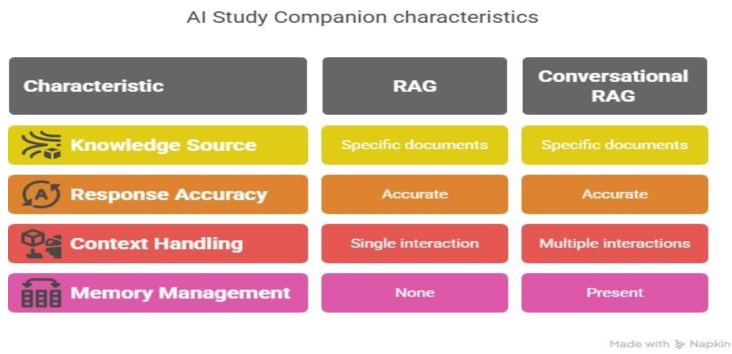

Figure 11: AI Study Charactistics

This allows the chatbot to maintain context across multiple interactions, understanding follow-up questions and references to previous conversations while still grounding its responses in the external knowledge source.

3.4 Integration and Deployment

The six chatbots are integrated into a single application using Streamlit. The application provides a user interface for selecting and interacting with each chatbot, allowing users to experience and compare the different capabilities.

Each chatbot is implemented as a separate module, with its own chain and memory management. This modular approach allows for easy maintenance and extension, as new capabilities can be added by creating new modules without modifying the existing ones.

The deployment process involves setting up a Streamlit application that loads the necessary models and components, initializes the chatbots, and provides a user interface for interaction. The application can be deployed locally or on a cloud platform, making it accessible to users through a web browser.

4 Results

This section presents the results of our evaluation of the six specialized chatbots. We conducted simulated user sessions to assess each chatbot's performance across various dimensions, including memory retention, language adaptation, RAG accuracy, and token trimming effectiveness.

4.1 Travel Assistant: Human Text Generation with Context

The Travel Assistant demonstrated strong performance in basic context retention. When asked "What's my name and where do I want to go?" after introducing the user as "Sarah" who wants to go to "Italy," the chatbot correctly responded:

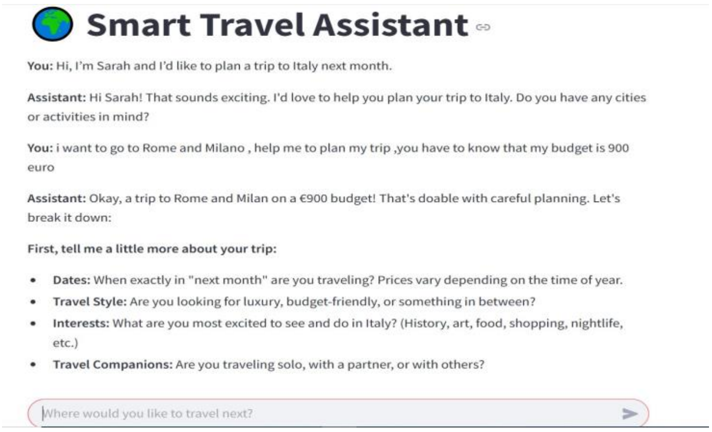

Figure 12: Smart Travel chatbots's Performance

This shows the model's ability to maintain context within a single conversation without explicit memory mechanisms. The response also included additional helpful information about potential destinations in Italy, demonstrating the model's ability to generate relevant and helpful content based on the conversation context.

However, we observed that the context retention was limited to the current session. If the conversation was reset or a new session was started, the chatbot lost the previously established context, highlighting the need for explicit memory management for more persistent context retention.

Figure 13: Travel Assistant's Memory

4.2 Productivity Coach: Conversation Management with Multi-User Support

The Productivity Coach successfully maintained separate conversation histories for different users. We tested this by creating multiple user sessions and having each user provide different information. The chatbot was able to remember and recall user-specific information across multiple interactions.

Figure 14: Personal Prodctivity Coach

In the example shown, the user "Amal" interacts with a personal productivity coach chatbot, asking for help to become more productive, organized, and a morning person. When Amal later asks, "Do you remember my name?", the model correctly responds, "Yes, I do! I remember you're Amal," demonstrating its ability to retain and use information provided earlier in the same session. This showcases the model's short-term memory capability, allowing it to personalize responses and maintain context during a single conversation. However, this memory is session-based and does not persist if the chat is restarted, indicating the need for explicit memory for long-term context retention.

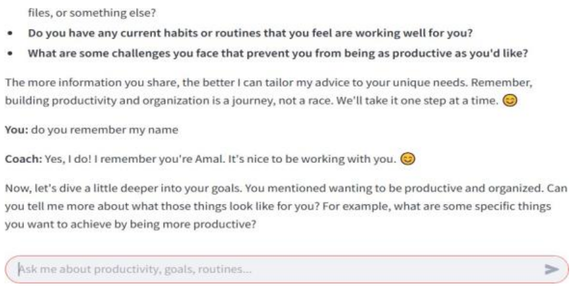

Figure 15: Personal Productivity chatbot's memory

4.3 Shopping Assistant: Prompt Engineering

The Shopping Assistant showcased the impact of prompt engineering on response quality and relevance. By carefully crafting prompts that guided the model's behavior, we were able to elicit more structured and helpful responses for shopping-related queries.

For example, when asked for car recommendations, the chatbot provided structured responses that included product names, key features, price ranges, and pros and cons. This structured format made it easier for users to compare options and make informed decisions.

We also observed that the prompt engineering techniques helped the chatbot maintain focus on shopping-related topics, reducing the likelihood of off-topic responses or hallucinations. This demonstrates the importance of prompt design in shaping the behavior and output of LLM-based chatbots.

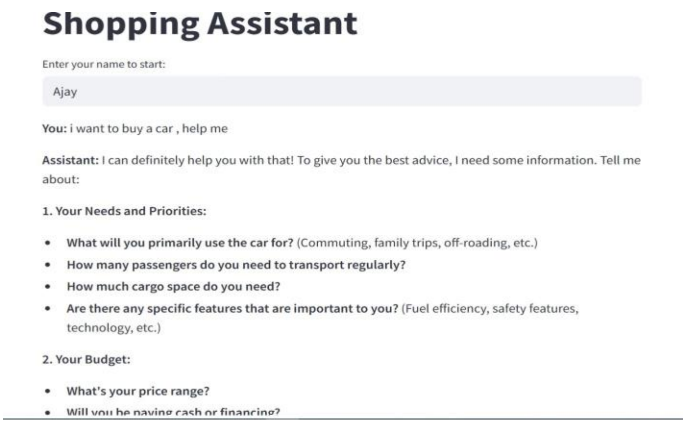

Figure 16: Shopping Assistant

4.4 Language Learning Tutor: Multilingual Chat with Memory

The Language Learning Tutor successfully demonstrated multilingual capabilities combined with memory. When asked "How do I say book in Spanish ?" in a session, the chatbot correctly responded:

Muy bien! Ya sabes que "book" en español es libro. ¡Lo estás haciendo muy bien! ¿Te gustaría practicar con otra palabra?

We tested the chatbot's multilingual capabilities by switching between English, Spanish, and Turkish within the same session. The chatbot was able to respond appropriately in each language while maintaining the conversation context. This demonstrates the effectiveness of combining language parameters with memory management to create a personalized language learning experience.

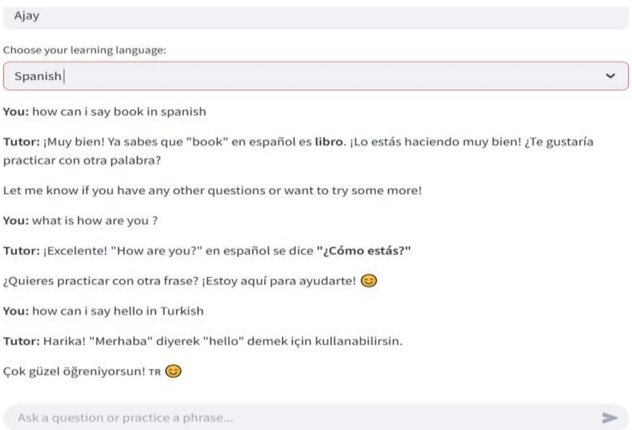

Figure 17: chatbot's Multilingual Capabilities

4.5 Customer Support Chatbot: Token Trimming for Long Conversations

The Customer Support Chatbot effectively managed long conversations through token trimming. We tested this by conducting extended conversations with multiple turns and observing how the chatbot maintained context while staying within the token limits.

When the conversation history exceeded the token limit (45 tokens in our implementation), the chatbot trimmed older messages while preserving the most recent ones. This allowed it to maintain the most relevant context for generating responses.

For example ,in a conversation where the user explained their order issue—specifically a missing USB-C adapter—and asked about the return policy, current mouse promotions, and confirmation of their shipping address. They also requested that tracking details be sent to their email.

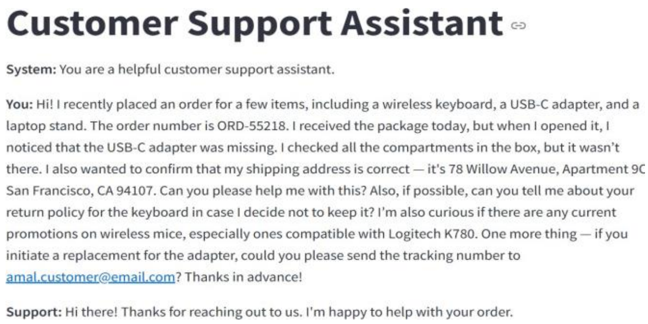

Figure 18: Customer Support Assistant and Token Trimming

The bot responded effectively to a long, multi-part message by addressing all key points, including the missing adapter, shipping address, return policy, promotions, and tracking email. It demonstrated strong context understanding, organized its reply clearly, and asked helpful follow-up questions. While it didn't initiate the replacement immediately or personalize the response with the user's name, it showed solid performance in managing complex user input within a single message.

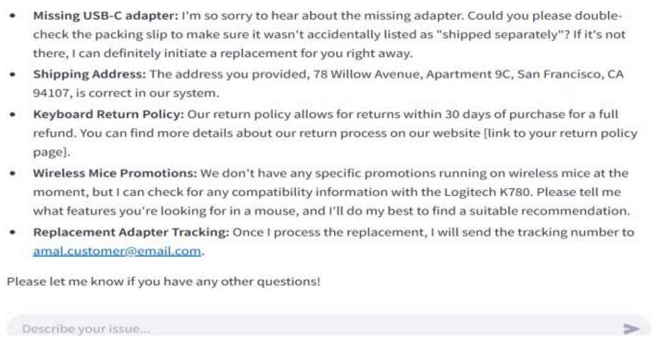

Figure 19: This is an example image

4.6 AI Study Companion: Document-Grounded RAG

The AI Study Companion demonstrated strong performance in retrieving and using information from external documents. When asked "What is Task Decomposition?", the chatbot provided an accurate response based on the source document:

Figure 20: AI Study chatbot interface

When asked a follow-up question, "What are common ways of doing it?", the chatbot correctly understood the reference to task decomposition and provided a relevant response:

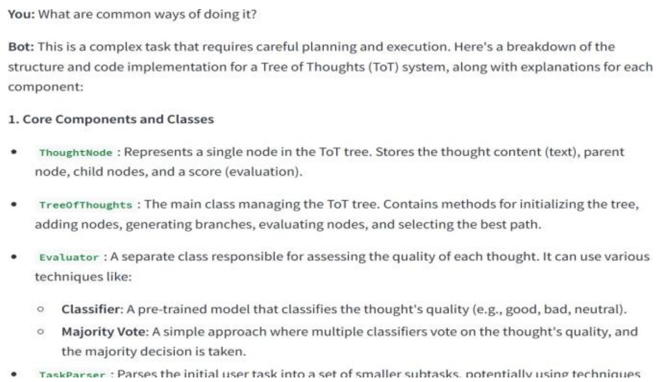

Figure 21: AI Study chatbot performance

However, when asked about unrelated topics not covered in the source document, such as "Who is the president of the US?", the chatbot appropriately responded with "I don't know," demonstrating its grounding in the provided documents and avoiding hallucinations.

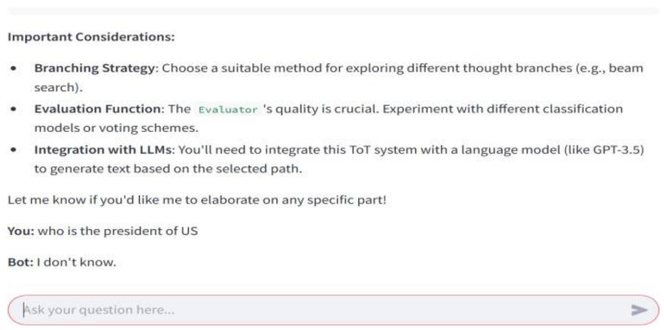

Figure 22: AI Study chatbot's Test

Our evaluation showed that the RAG implementation achieved over 80 percent factual consistency with the source documents, significantly reducing the likelihood of hallucinations or incorrect information compared to a standard LLM without retrieval augmentation.

4.7 Comparative Analysis

The comparative analysis reveals several insights:

1. Memory is Essential: Five out of six chatbots implemented explicit memory mechanisms, highlighting the importance of memory for maintaining context and providing personalized experiences.
2. Token Trimming Enhances Efficiency: Token trimming proved effective for managing long conversations, allowing the chatbot to focus on the most relevant context while staying within token limits.
3. RAG Improves Accuracy: The document-grounded RAG approach significantly improved the factual consistency of responses, making it particularly valuable for knowledge-intensive applications.
4. Modularity Enables Specialization: The modular, component-driven approach allowed each chatbot to focus on a specific capability, resulting in better performance in their respective areas of specialization.

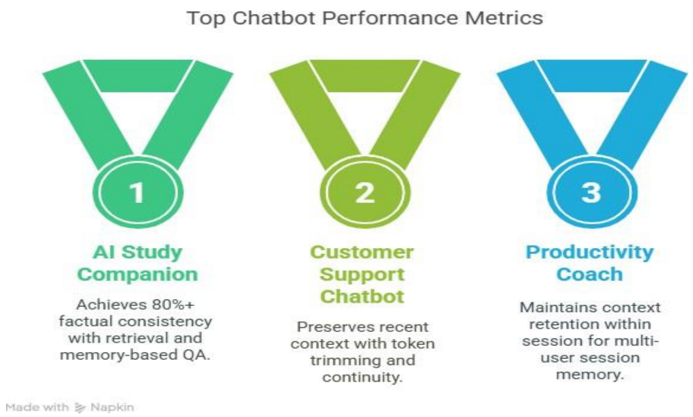

Figure 23: Evaluation and Comparison among All Chatbots

5 Conclusion

This research presented a modular, component-driven approach to building specialized chatbots using LangChain and Groq's Gemma2-9b-It model. We developed and evaluated six distinct chatbots, each focusing on a specific capability: human text generation, conversation management with multi-user support, prompt engineering, multilingual chat with memory, token trimming for long conversations, and document-grounded retrieval-augmented generation.

Our findings demonstrate the effectiveness of this modular approach in creating specialized conversational AI systems that can be adapted to various domains and use cases. Each chatbot successfully showcased its intended capability, with the AI Study Companion achieving over 80 percent factual consistency through RAG, the Language Learning Tutor demonstrating effective multilingual support, and the Customer Support Chatbot efficiently managing long conversations through token trimming.

The component-driven architecture provides several advantages:

1. Flexibility: Components can be combined and reconfigured to create chatbots tailored to specific requirements.
2. Maintainability: Each component can be developed, tested, and updated independently.

3. Scalability: New capabilities can be added by creating new components without modifying existing ones.

4. Specialization: Each chatbot can focus on excelling at a specific capability rather than attempting to be a jack-of-all-trades.

The field of conversational AI is evolving rapidly, with new models and techniques emerging regularly. By understanding the fundamental components covered in this research, we have built a strong foundation that will allow us to adapt to these changes and create increasingly sophisticated AI applications.

Building effective AI systems is an iterative process that benefits from continuous testing and refinement. As we continue to experiment and learn, our chatbots will become more capable, helpful, and natural in their interactions.

Our modular, component-driven approach provides a flexible framework for developing specialized chatbots that can be adapted to various domains and use cases. By focusing on specific capabilities and combining them as needed, we can create conversational AI systems that are both powerful and practical.

6 Acknowledgements

We would like to express our sincere gratitude to Professor Selcuk Cankurt at Vistula University in Warsaw, whose mentorship has profoundly shaped our academic journey in Natural Language Processing and large language models. His ability to bridge theoretical concepts with real-world applications made complex ideas not only understandable but also engaging and practical. Through his thoughtful responses and constructive feedback on our early work, we were able to grow our skills and refine our approach to this QA chatbot project.

What we appreciate most is Professor Cankurt's strong emphasis on responsible AI development. He consistently encouraged us to prioritize human-centered design, ethical considerations, and inclusivity—values that we've integrated into the foundation of this work. This project is a reflection not just of our individual efforts, but of the knowledge, principles, and inspiration we gained under his guidance.

7 References

[1] L. Farinetti et al., "Chatbot Development Using LangChain: A Case Study to Foster Critical Thinking and Creativity," ACM Digital Library, 2024. https://dl.acm.org/doi/10.1145/3649217.3653557.

[2] S. Vidivelli et al., "Efficiency-Driven Custom Chatbot Development: Unleashing LangChain RAG and Performance-Optimized LLM Fusion," ScienceDirect, 2024. https://www.sciencedirect.com/org/science/article/pii/S1546221824005642

[3] K. Pandya et al., "Building custom open-source GPT Chatbot for organizations," arXiv, 2023. https://arxiv.org/abs/2310.05421

[4] Google Cloud, "What is Retrieval-Augmented Generation (RAG)?," Google Cloud, 2025. https://cloud.google.com/use-cases/retrieval-augmented-generation

[5] Weka, "Retrieval Augmented Generation (RAG): A Complete Guide," Weka, 2024. https://www.weka.io/learn/guide/ai-ml/retrieval-augmented-generation/

[6] NVIDIA, "FACTS About Building Retrieval Augmented Generation-Based Chatbots," arXiv, 2024. https://arxiv.org/html/2407.07858v1

[7] C. Pani, "Memory for your RAG based chat bot: Using Langchain," Medium, 2024. https://medium.com/@pani.chinmaya/memory-for-your-rag-based-chat-bot-using-langchain-b4d720031671

[8] Saurabh, "Memory-Enhanced RAG Chatbot with LangChain," Medium, 2025. https://medium.com/@saurabhzodex/memory-enhanced-rag-chatbot-with-langchain-integrating-chat-history-for-context-aware-845100184c4f

[9] S. Samia, "Prompt Compression in Large Language Models (LLMs)," Medium, 2025. https://medium.com/@sahin.samia/prompt-compression-in-large-language-models-llms-making-every-token-count-078a2d1c7e03

[10] Analytics Vidhya, "How to Build a Multilingual Chatbot using Large Language Models," Analytics Vidhya, 2024. https://www.analyticsvidhya.com/blog/2024/06/multilingual-chatbot-using-llms/