Incorporate a Large Language Model to assist users

Mentors: Alex Richards, Mark Smith, Ulrik Egede, Aryabhatta Dey

Contributor: Sidharth Sinhasane

Personal Information:

Name : Sidharth Sinhasane
Email : sidharth.sinhasane@gmail.com
Date of Birth : 26/08/2004
Github Id : sidharth-sinhasane
Contact number : +91 9322270517
Country : India
Time Zone : UTC+05:30 (Asia/Kolkata)
University : Savitribai Phule Pune University
LinkedIn : sidharth-sinhasane

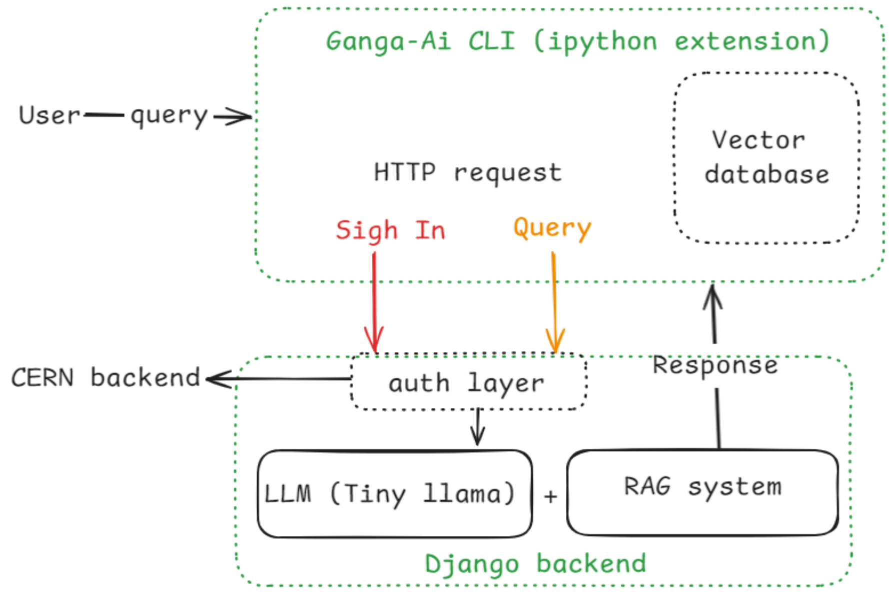
Abstract

This project integrates a Large Language Model (LLM) into Ganga's CLI to assist users with syntax and error handling through natural language input. It incorporates a Retrieval-Augmented Generation (RAG) system for context-aware responses and enhances usability with JWT-based authentication, chat context management, and LLM response streaming. By automating assistance and improving accessibility, this integration simplifies workflows for both new and experienced users.

Project Details

Overview:

To address users' syntax difficulties, this project aims to integrate a Large Language Model (LLM) into Ganga's command prompt. This integration would allow users to describe their goals in natural language and receive usable examples, while also intercepting exceptions to provide explanations and solutions. The project, which already has an ollama-based interface for building a Retrieval Augmented Generation (RAG) system with Ganga-specific information, involves several tasks: integrating the LLM and RAG into Ganga, incorporating CLI context from past inputs and outputs,developing continuous integration tests.



Goals:

1 Implement a JWT-based authentication layer within the Ganga-AI backend, integrating Single Sign-On (SSO) with CERN servers.

2 Develop a sign-in functionality at cli and sign-in endpoint at server side.

3 Write HTTP request handlers to enable CLI interactions with the backend.

4 Integrate a chat context management system in Ganga-AI to maintain ongoing chat history.

5 Rag data enhancement : currently, we are providing the whole documentation.

6 Develop continuous integration tests that ensure that LLM integration will keep working.

7 Create comprehensive project documentation.
Stretched goal : Enable streaming of LLM-generated responses.

Implementation details:

1 Authentication layer:

Sign up mechanism (Inspired by this example link)

1. Device authorization will be initiated by sending a request to the /auth/device endpoint with my client_id, a state parameter for CSRF protection.
2. We will display the verification_uri and user_code to the user, with instructions to complete verification on another device/browser.
3. We will set up a polling mechanism that checks the /token endpoint every few seconds with users client_id, device_code, and code_verifier until the user successfully logs in.
4. Once authentication is complete, the server will retrieve and return the JWT token that can be used for future API requests.

Authentication middleware:

Sample code :

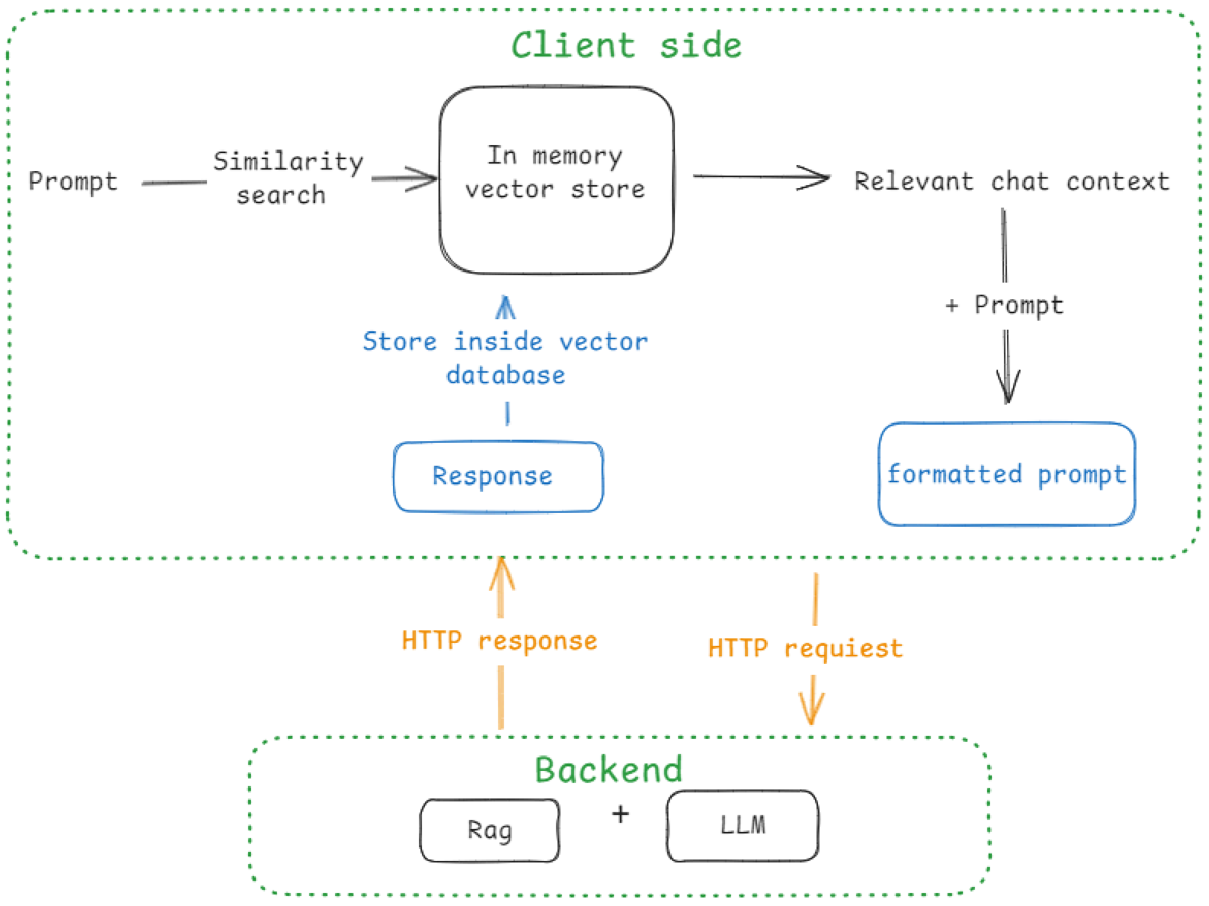
```
class JWTAuthenticationMiddleware:
    def __init__(self, get_response):
        self.get_response = get_response

    def __call__(self, request):
        auth_header = request.headers.get('Authorization')

        try:
            token = auth_header.split(' ')[1]
            decoded=jwt.decode(token,setings.JWT_SECRET,algorithm=["HS256"])
            request.user = decoded
        except (IndexError, jwt.ExpiredSignatureError, jwt.InvalidTokenError):
            return JsonResponse({'message': 'Invalid or expired token.'},
status=401)

        response = self.get_response(request)
        return response
```

2 Chat context management system



- A vector database is created in memory for each user using **langchain ollama embeddings** , and a similarity search is performed over the database to retrieve only the relevant chat context based on the query.

- **Advantage of this approach:**
  There is no need to maintain a backend database solely for storing chat history. Additionally, session management is handled automatically, (maintaining context for multiple users at a time) as context is sent through http request .

- **Challenge**:
  we need to take care about the token size of LLM.

- **To tackle this** :
  We will be selecting top n results from similarity search based on token limit .

- **Dependencies required**:
  Langchain MemoryVectorStore
  Langchain OllamaEmbeddings
  Langchain ChatPromptTemplate

Sample code:

Initialise vector store:

```
def initialiseVectorStore():
    embeddings = OllamaEmbeddings(
            base_url='http://localhost:11434',
            model='nomic-embed-text' # general purpose embedding model
    )
    vector_store = MemoryVectorStore.from_documents([], embeddings)
    return vector_store
```

Store Conversations:

```
def store_conversation(self, user_prompt, llm_response):
        prompt_doc = Document(
            page_content=user_prompt,
            metadata={
                "type": "prompt",
                "timestamp": datetime.utcnow().isoformat()
            }
        )


        response_doc = Document(
            page_content=llm_response,
            metadata={
                "type": "response",
                "timestamp": datetime.utcnow().isoformat()
            }
        )
        self.vector_store.add_documents([prompt_doc, response_doc])
```

Query vector database:

```
def query_vector(self, query, top_k):
        if not self.vector_store:
            return []

        results = self.vector_store.similarity_search(query, top_k)
        return results
```

3 LLM response streaming:

- Streaming of response means getting LLM output in chunks. This feature will definitely improve the user experience of our project.
- For that i need to make changes to query_vector_store function.
- After properly configuring both the LLM and the query engine, calling query now returns a StreamingResponse object. Reference

Timeline

Period	Objective
Pre Work Program	1 Work on issues in Ganga. 2 Do analysis of MCP based Ganga AI. 3 Community Bonding Period. 4 Project setup.
Week 1 [2 June - 8 June]	1 Get access from CERN to their database. 2 start working on the sign up endpoint at backend. 3 Remove Ollama installations from frontend.
Week 2 [9 June - 15 June]	1 Complete sign up endpoint. 2 Document the schema. 3 Start working on sign up HTTP request from CLI.
midterm evaluations 1 st milestone	
Week 3 [16 June - 22 June]	1 Complete sign up request . 2 Test backend authentication. 3 Start working on auth middleware for generating endpoints.
Week 4 [23 June - 29 June]	1 Complete middleware. 2 Complete and Test authentication functionality .
Week 5 [30 June - 6 July]	1 Start context management work. 2 implement a vector store database.
Week 6 [7 July - 13 July]	1 Implement history context retrieval. 2 Write HTTP request to access backend generated endpoint. 3 Write text formatter to format prompt.
Week 7 [14 July - 20 July]	1 Test context management system. 2 Handle token length limit problem.
Week 8 [21i July - 27 July]	1 Explore CERN docs and host our backend at CERN server. 2 Develop continuous integration tests.
2nd milstore (Auth + context management completion)	
Week 9	1 Data collection for RAG to get better results.

[28 July - 3 August]	2 Work with tokenizers and retrieval chains for better results.
Week 10 [4 August - 10 August]	1 Working on Streaming of LLM results. 2 Modify the handler at frontend to display streamed results.
Week 11 [11 August - 17 August]	1 Add comments and improve code readability 2 Continue working on any pending tasks. 3 Write detailed documentation.
Week 12 [18 August - 24 August]	1 Work on the final GSOC report. 2 continue working on any pending tasks. 3 Thoroughly test all features for bugs.
25 August - 1 September Submission week	

Experience:

Backend Developer Intern at I2i Specialist pvt limited. Jan 2025 - March 2025

- Leveraged the MERN stack and TypeScript to build robust backend solutions
- Developed and optimized REST APIs to enhance system performance and scalability
- Integrated WebSocket functionalities into the backend, enabling efficient real-time communication.

Related Work:

Supplier Compliance Monitor and Insights Dashboard

live link GitHub

- Developed a full-stack monitoring system to track compliance with contract terms (e.g., delivery times, quality standards). And supplier compliance record as input.
- Integrated Llama 3 (8B) model locally with CUDA for pattern detection inside the data and provide improvement suggestions.
- Preprocessing of data in csv file is done and leveraged as a RAG system.
- Built a FastAPI backend with PostgreSQL to store supplier-related data and historical compliance records.
- Developed a React.js front-end using Tailwind CSS, offering an intuitive dashboard to manage multiple suppliers,their reports and view AI-generated insights.
- Tech stack: FastAPI, PostgreSQL, React.js, Tailwind CSS, Llama 3 (8B), CUDA

Other commitments during GSoC

My university exams are scheduled for mid-May. Apart from that, I have no other commitments at the moment. After my summer break, I will still be able to dedicate 30 hours per week to the project.

Post GSoC:

I am eager to continue contributing to this project and remain an active part of the community. Additionally, I want to further explore Ganga and deepen my understanding of its capabilities.

Thank you!