

AI Chatbot Development

Katoo Roofthoof , Francesco Chrabieh , Egemen Alkan

- Introduction to Chatbot Development
- Utilizing ChromaDB
- Groq API and Huggingface API
- Transition to Qdrant Database
- Replacing Langchain with Llamaindex
- Content Agents and Upload Materials
- QAAgent
- Crew
- Voice Chat Integration
- Long-Term Memory Capabilities
- Introduction of Study Tools
- React Integration for Quizzes

Introduction to Chatbot Development

The Personalized Learning Assistant project aimed to create an interactive chatbot to assist learners with tailored support, summaries, and explanations, using the Groq API and Huggingface API.

Utilizing ChromaDB

ChromaDB is a local vector database that stores embeddings, enabling efficient retrieval, accurate responses, and context retention by tracking user queries and interactions.

```
from langchain.chroma import Chroma
from langchain.huggingface import HuggingFaceEmbeddings
import chromadb
from chromadb.config import Settings

class VectorStore:
    def __init__(self, llm_choice="Groq API", groq_api_key=None, hf_key=None):
        self.client = chromadb.Client(Settings(persist_directory="db"))

        if llm_choice == "Groq API":
            self.embeddings = HuggingFaceEmbeddings(model_name="sentence-transformers/all-MiniLM-L6-v2")
        elif llm_choice == "HuggingFace API":
            self.embeddings = HuggingFaceEmbeddings(model_name="sentence-transformers/all-MiniLM-L6-v2")
        else:
            raise ValueError(f"Unsupported LLM choice: {llm_choice}")

        self.vector_store = Chroma(
            client=self.client,
            embedding_function=self.embeddings,
            collection_name="learning_materials"
        )

    def add_documents(self, documents):
        self.vector_store.add_documents(documents)

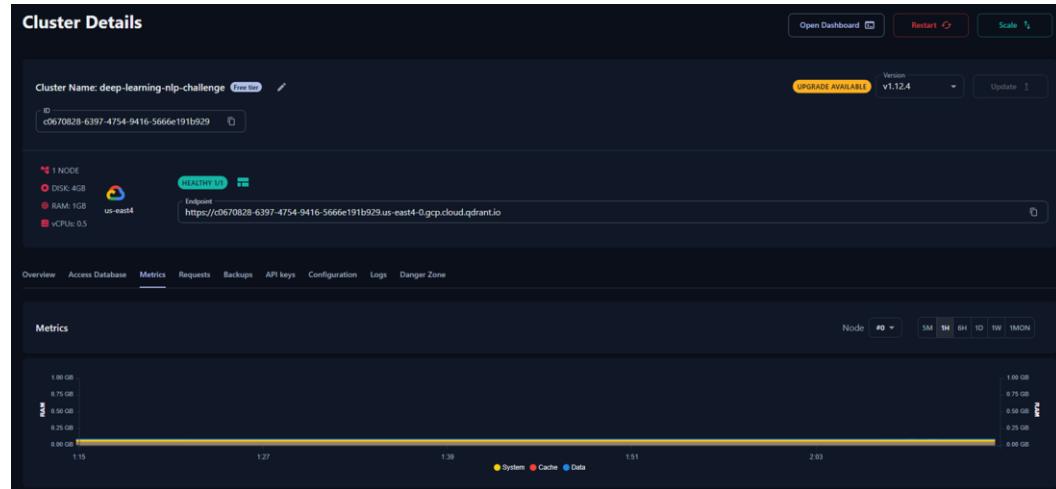
    def as_retriever(self):
        return self.vector_store.as_retriever(search_kwargs={"k": 5})
```

Transition to Qdrant Database



Benefits of Qdrant

Qdrant's advanced features and optimized indexing improved retrieval speed and performance.



Replacing Langchain with Llamaindex

We explored Llamaindex for better data organization but faced compatibility issues and we reverted to Langchain .

```
from crewai import Agent, Task
from llama_index.core import VectorStoreIndex, Document, Settings
from llama_index.readers.youtube_transcript import YoutubeTranscriptReader
from llama_index.readers.web import SimpleWebPageReader
from llama_index.readers.file import PptxReader
from qdrant_client import QdrantClient
from qdrant_client.http.models import Distance, VectorParams
import os
```

Groq API and Huggingface API

GroqAPI

- Integrates with Groq's language model for chat-based tasks.

```
class GroqAPI(BaseChatModel):
    api_key: str = Field(..., description="API key for authenticating requests to the Groq API")
    model_name: str = Field(default="mistral-8x7b-s3248", description="Model name for Groq API")
    _api_url: str = PrivateAttr(default="https://api.groq.com/openai/v1/chat/completions")

    def __init__(self, **data):
        super().__init__(**data)
        # Remove the groq/ prefix if it exists
        if self.model_name.startswith("groq/"):
            self.model_name = self.model_name[5:]

    def __call__(self, messages: List[BaseMessage], stop=None, **kwargs):
        """
        Call the Groq API with a list of messages.
        """

        # Properly format messages for the Groq API
        formatted_messages = []
        for msg in messages:
            if isinstance(msg, HumanMessage):
                formatted_messages.append({"role": "user", "content": msg.content})
            elif isinstance(msg, AIMessage):
                formatted_messages.append({"role": "assistant", "content": msg.content})
            elif isinstance(msg, SystemMessage):
                formatted_messages.append({"role": "system", "content": msg.content})
            else:
                raise ValueError(f"Unsupported message type: {type(msg)}")

        headers = {"Authorization": f"Bearer {self.api_key}"}
        payload = {
            "model": self.model_name, # No 'groq/' prefix needed
            "messages": formatted_messages,
            "temperature": kwargs.get("temperature", 0.7),
            "max_tokens": kwargs.get("max_tokens", 1000),
        }
```

HuggingFaceAPI

- Connects to Hugging Face models for text generation.

```
class HuggingFaceAPI(BaseChatModel):
    api_key: str = Field(..., description="API key for authenticating requests to the Hugging Face API")
    model_name: str = Field(..., description="Model name for Hugging Face API")
    max_tokens: int = Field(default=1000, description="Maximum number of tokens to generate")
    temperature: float = Field(default=0.7, description="Sampling temperature")
    _api_url: str = PrivateAttr(default="https://api-inference.huggingface.co/models/")
    _client: InferenceClient = PrivateAttr()

    def __init__(self, **data):
        super().__init__(**data)
        load_dotenv()
        self._api_key = self.api_key or os.getenv("HF_API_KEY")
        if not self._api_key:
            raise RuntimeError("HuggingFace API key not found.")

        # Initialize the inference client
        self._client = InferenceClient(api_key=self._api_key)
        self.model_name = self.model_name

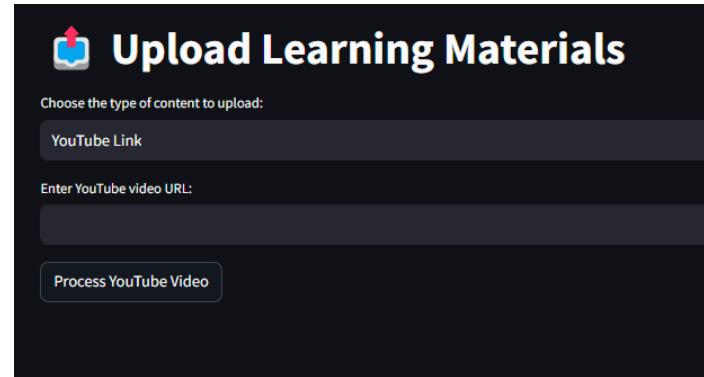
    def __call__(self, messages: List[BaseMessage], context=None, **kwargs):
        """
        Call the Hugging Face Inference API with optional context.
        """

        formatted_prompt = self._format_prompt(messages, context)
        headers = {"Authorization": f"Bearer {self._api_key}"}
        payload = {
            "inputs": formatted_prompt,
            "parameters": [
                {"name": "max_tokens", "value": kwargs.get("max_tokens", self.max_tokens)},
                {"name": "temperature", "value": kwargs.get("temperature", self.temperature)},
            ],
        }
```

Content Agents and Upload Materials

Content Agents and Upload Materials

We introduced content agents to process and index various educational materials, including PDFs, video transcripts, and presentations..



The screenshot shows a user interface titled "Upload Learning Materials". At the top, there is a logo of a blue square with a white arrow pointing upwards and to the right. Below the title, a section asks "Choose the type of content to upload:" followed by a button labeled "YouTube Link". Underneath this, there is a field labeled "Enter YouTube video URL:" with a text input field below it. At the bottom of the interface is a button labeled "Process YouTube Video".

QAAgent

The QAAgent combines document retrieval, web search using GroqAPI or HuggingFaceAPI. It dynamically selects methods, validates context relevance, and generates accurate answers by integrating multiple data sources and tailored prompts for diverse question-answering needs.

```
class QAAgent(Agent):
    _vector_store: PrivateAttr
    _llm: PrivateAttr
    _use_chain: PrivateAttr
    _retriever: PrivateAttr
    _qa_chain: PrivateAttr

    def __init__(self, vector_store, llm_choice, groq_api_key=None, hf_key=None):
        # Initialize LLM based on user choice
        if llm_choice == "Groq API":
            llm = GroqAPI(api_key=groq_api_key, model_name="mixtral-8x7b-32768")
            use_chain = True
        elif llm_choice == "HuggingFace API":
            llm = HuggingFaceAPI(api_key=hf_key, model_name="tiiuae/falcon-7b-instruct")
            use_chain = False
        else:
            raise ValueError(f"Unsupported LLM choice: {llm_choice}")

        # Initialize the Agent with CrewAI parameters
        super().__init__(
            role="Question Answering Expert",
            goal="Provide accurate and detailed answers based on available context",
            backstory="Expert at analyzing context and providing comprehensive answers",
            allow_delegation=True,
            llm=llm,
            verbose=True
        )
```

Crew

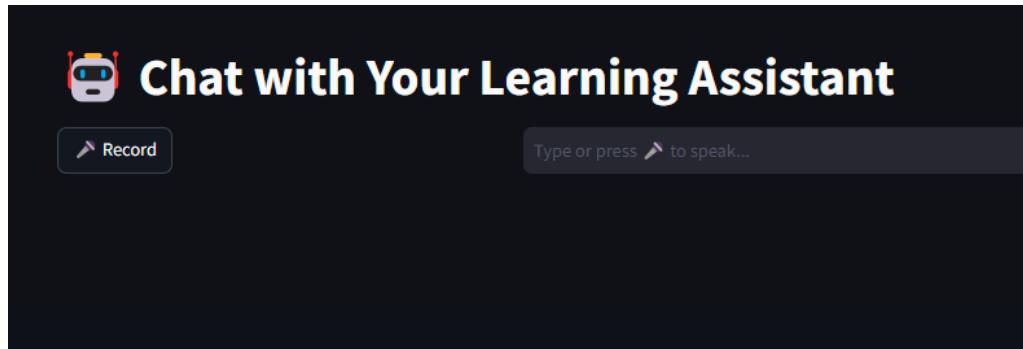
The ResearchCrew coordinates content and QA agents and it dynamically handles research-based and direct question-answering tasks, integrating multiple agents.

```
# Create crew
crew = Crew(
    agents=[self.content_agent, self.qa_agent],
    tasks=[research_task, qa_task],
    process=Process.sequential,
    verbose=True
)
```

Voice Chat

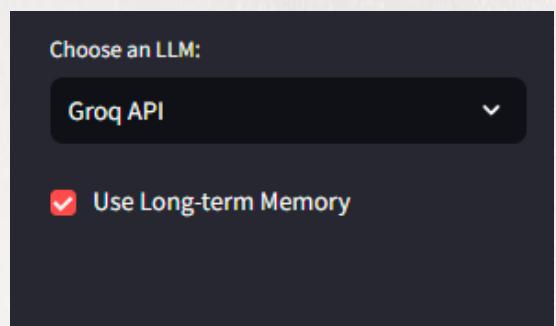
Voice Chat Integration

The project integrated OpenAI's Whisper model for accurate speech-to-text transcription and gTTS for text-to-speech.



Long-Term Memory Capabilities

The long-term memory integration enhanced the learning assistant by enabling it to retain and utilize historical context through the MemoryStore class.



Study Tools

Introduction of Study Tools

The CheatSheetAgent generates concise summaries of key concepts, while the QuizAgent creates tailored quizzes with instant feedback.

This screenshot shows the Study Tools interface with the QuizGenerator tool selected for Field Hockey. The user has input '1' for the number of questions and chosen 'multiple_choice' as the question type. The generated quiz is titled 'Field Hockey Quiz' and contains one question: 'What should you avoid when positioning yourself near the right post in field hockey?'. The correct answer is 'Positioning yourself further from the post', and there are three incorrect options: 'Deflecting the ball with heel', 'Stepping out and in front of the defender', and 'Walking to the defender with your left leg'. A note at the bottom states: 'Marking yourself further from the post can cause you to miss the goal if you get a 90-degree deflection. Stepping inside the post ensures a goal if you get a 90-degree deflection.'

This screenshot shows the Study Tools interface with the Cheat Sheet Generator selected for Field Hockey. The user has entered the topic 'Field Hockey' and clicked 'Generate Cheat Sheet'. The resulting page is titled 'Field Hockey Goal Scoring Cheat Sheet' and contains three sections: 1. Key Concepts, 2. Important Definitions, and 3. Core Principles. Each section lists specific tips and strategies for goal scoring.

Field Hockey Goal Scoring Cheat Sheet

1. Key Concepts

- Positioning inside the post for deflections
- Using entire stick for ball control
- Stepping out and in front of defender
- Making determined leads for ball deflection
- Getting low for quick stick adjustments

2. Important Definitions

- Deflection: A ball's change in direction after hitting a player's stick
- Lead: A player's movement to get in front of the defender for a deflection

3. Core Principles

- Keep your position inside the post when expecting a deflection for maximum scoring potential
- Utilise the entire stick, especially the forehand side, to control the ball and create dangerous situations
- When leading, step out and in front of the defender, then quickly deflect the ball into the goal
- Ensure good connection with the player on the ball and maintain steady footwork during leads
- Get low to the ground for quick stick adjustments and effective deflections

This screenshot shows the Study Tools interface with the QuizGenerator tool selected for Field Hockey. The user has input '1' for the number of questions and chosen 'open-ended' as the question type. The generated quiz is titled 'Field Hockey Quiz' and contains one question: 'Describe three ways to improve your awareness and scoring in front of the goal in field hockey, especially when a ball is about to be crossed from the right or left side.' A note at the bottom states: 'Strength training for explosiveness, develop a range of shooting skills, be unpredictable to create space'.

React Integration for Quizzes

Due to issues with Streamlit where quizzes disappeared upon answering the questions or submission, we used React to implement the quizzes, ensuring proper functionality and displaying results.

Question 1

What should you avoid when positioning yourself near the right post in field hockey?

- Positioning yourself further than the post
- Deflecting the ball with two hands
- Stepping in front of the defender
- Blocking the defender with your left leg

✓ Correct!

Positioning yourself further than the post can cause you to miss the goal if you get a 90 degree deflection. Staying inside the post ensures a goal if you get a 90 degree deflection.

Current Score

1 / 1 correct

Thank you.