

ENHANCING LLMs WITH GRAPH-BASED RETRIEVAL-AUGMENTED GENERATION
(RAG) FOR SUPERIOR ACCURACY AND CONTEXT.

OAHED NOOR FORHAD(C213056)

TOHEDUL ISLAM NIRZON(C213060)

SAIFUL ISLAM RUMI(C211080)

PROJECT SUBMITTED IN FULFILMENT FOR THE COURSE OF
MACHINE LEARNING AND DATA MINING DEPARTMENT OF COMPUTER SCIENCE
AND ENGINEERING (CSE)

INTERNATIONAL ISLAMIC UNIVERSITY CHITTAGONG (IIUC)
CHITTAGONG, BANGLADESH

DECLARATION

I/We hereby declare that the work in this Project is my/our own except for quotations and summaries which have been duly acknowledged.

19 January 2025

NAME
ID

COURSE TEACHER DECLARATION

I/We hereby declare that we have read this Project and in my/our opinion this Project is sufficient in terms of scope and quality for the award of the degree of B. Sc. in Computer Science & Engineering

19 January 2025

MR.MD.MAHIUDDIN
ASSOCIATE PROFESSOR
DEPARTMENT OF CSE

PROJECT REPORT AND COPYRIGHT

PROJECT REPORT TITLE:

ENHANCING LLMs WITH GRAPH-BASED RETRIEVAL-AUGMENTED GENERATION (RAG) FOR SUPERIOR ACCURACY AND CONTEXT.

AUTHORS:

L NO.	AUTHOR'S NAME	STUDENT ID	SIGNATURE
1	Oahed Noor Forhad	C213056	
2	Tohedul Islam Nirzon	C213060	
3	Saiful Islam Rumi	C211080	
Name OF Course Teacher : MR.MD.MAHIUDDIN			

ACKNOWLEDGEMENT

First and foremost praise be to Almighty Allah for all his blessings for giving me patience and good health throughout the duration of this thesis work.

I am very fortunate to have Associate Professor Md.Mahiuddin as a Course Instructor

Moreover, I am grateful to the faculties and staff of the Dept. of CSE, IIUC

I would like to thank all.

To my dearest ...

Last but not least, I gratefully acknowledge...

ABSTRACT

This report presents a Retrieval Augmented Generation (RAG) system designed to provide context-aware, domain-specific answers for the International Islamic University Chittagong (IIUC). By combining LightRAG, FastAPI, React, and a Knowledge Graph, the system overcomes common RAG bottlenecks—such as limited contextual awareness and slow updates—through efficient graph-based indexing and a dual-level retrieval mechanism (local and global). The backend uses LightRAG to embed and retrieve relevant textual segments, while an Ollama-based Large Language Model (LLM) generates coherent responses. A React-based interface offers real-time, streaming answers, simulating a live chat experience. Users can choose between four retrieval modes (naive, local, global, hybrid), balancing broad summaries and detail-rich context. Incremental updates ensure new data can be seamlessly integrated without re-indexing the entire corpus, and the knowledge graph provides a structured representation of university entities and relationships. Evaluations show that the hybrid mode often produces the most comprehensive and context-sensitive replies. This project underscores how graph-enhanced RAG systems can deliver fast, accurate, and scalable question-answering solutions in institutional or specialized domains.

INTRODUCTION

In the rapidly evolving field of Natural Language Processing (NLP), Retrieval Augmented Generation (RAG) has emerged as one of the most effective ways to provide contextually rich, accurate, and up-to-date answers. Traditional RAG systems, however, often face key challenges:

Loss of Global Context: Splitting knowledge into chunks can make it difficult to preserve overarching ideas.

Limited Inter-Chunk Relationships: Conventional approaches may not capture the ways different pieces of content relate to each other.

High Costs and Complexity: Updating large knowledge bases can be costly and time-consuming.

LightRAG, a novel retrieval-augmented framework, addresses these issues through more efficient indexing, dual-level retrieval (local vs. global), and incremental updates. This system can be further enhanced by incorporating a Knowledge Graph, offering a structured overview of entities (e.g., faculty, departments, courses) and their relationships.

To demonstrate a real-world application, this project focuses on providing detailed information about the International Islamic University Chittagong (IIUC)—from faculty details to admission guidelines. The final user interface is built with React (front-end) and FastAPI (back-end), using LightRAG for semantic retrieval and an Ollama-based LLM for generating text responses.

OBJECTIVE

- 1. ACCURATE DELIVERY:** PROVIDE CORRECT ANSWERS ABOUT IIUC'S STRUCTURE, FACULTY, AND PROGRAMS.
- 2. EFFICIENT UPDATES:** INTEGRATE NEW OR CHANGED DATA QUICKLY WITHOUT RE-INDEXING EVERYTHING FROM SCRATCH.
- 3. RICH USER EXPERIENCE:** UTILIZE A CHAT-LIKE FRONTEND WITH TYPED-RESPONSE STREAMING, EASY MODE SELECTION (NAIVE, LOCAL, GLOBAL, HYBRID), AND MINIMAL LATENCY.
- 4. SHOWCASE GRAPH UTILITY:** DEMONSTRATE HOW KNOWLEDGE GRAPHS CAN PROVIDE CLARITY ON RELATIONSHIPS WITHIN THE UNIVERSITY'S DATA.

METHODOLOGY

1. Data Collection and Preparation

Text Extraction:

Data (e.g., faculty names, degrees, contacts, departmental info) was scraped or copied from the IIUC official website into a text file, infobot.txt.

Cleaning and Tokenization:

Unnecessary characters and formatting artifacts were removed.

The content was segmented into manageable chunks for embedding.

Knowledge Graph Construction:

Entities: For IIUC, these might be Departments, Programs, Faculty Members, Administrative Roles, etc.

Relationships: “Teaches in,” “Chair of,” “Located at,” or “Research Focus.”

Graph-based data modeling helps in capturing interconnections more explicitly than plain text alone.

2 LightRAG Innovations

LightRAG addresses the typical limitations of RAG through graph-based text indexing and a dual-level retrieval paradigm:

Graph-Based Text Indexing:

Extract entities and relationships from the text to build an internal knowledge structure.

Use an LLM to produce descriptive key-value pairs about each entity.

Deduplicate repeated entities across chunks, preserving a cohesive global context.

Dual-Level Retrieval:

Naive directly queries the LLM without advanced retrieval adjustments.

Local (Low-Level) Retrieval focuses on precise information for detail-oriented queries.

Global (High-Level) Retrieval aggregates broader, conceptual information for queries requiring bigger-picture understanding.

Hybrid combines both, ensuring balanced coverage and depth.

IMPLEMENTATION DETAILS

1. Key Backend Code (FastAPI + LightRAG)

- Initialization of LightRAG

```
1 from lightrag import QueryParam
2
3 @app.get("/query")
4 async def query_rag(question: str, mode: str = "naive"):
5     """
6     Takes a user question and mode (naive/local/global/hybrid),
7     performs LightRAG retrieval and returns the model's response.
8     """
9     try:
10         # Validate mode
11         valid_modes = ["naive", "local", "global", "hybrid"]
12         if mode not in valid_modes:
13             return {
14                 "status": "error",
15                 "response": None,
16                 "error": f"Invalid mode: {mode}. Must be one of {valid_modes}.",
17             }
18
19         # Perform retrieval-augmented generation
20         result = rag.query(question, param=QueryParam(mode=mode))
21         return {
22             "status": "success",
23             "response": result,
24             "mode": mode
25         }
26     except Exception as e:
27         return {
28             "status": "error",
29             "error": str(e),
30             "mode": mode
31         }
```

What It Does

- Creates an instance of LightRAG using a specific LLM (qwen2.5:7b) served by Ollama.
- Configures embedding via the EmbeddingFunc, pointing to nomic-embed-text.
- Stores intermediate data in a designated WORKING_DIR (./dickens).

- Query Endpoint

```
1 import asyncio
2 import inspect
3 import logging
4 import os
5
6 from lightrag import LightRAG, QueryParam
7 from lightrag.llm import ollama_embedding, ollama_model_complete
8 from lightrag.utils import EmbeddingFunc
9
10 WORKING_DIR = "./dickens_age"
11
12 logging.basicConfig(format="%(levelname)s: %(message)s", level=logging.INFO)
13
14 if not os.path.exists(WORKING_DIR):
15     os.mkdir(WORKING_DIR)
16
17 # AGE
18 os.environ["AGE_POSTGRES_DB"] = "postgresDB"
19 os.environ["AGE_POSTGRES_USER"] = "postgresUser"
20 os.environ["AGE_POSTGRES_PASSWORD"] = "postgresPW"
21 os.environ["AGE_POSTGRES_HOST"] = "localhost"
22 os.environ["AGE_POSTGRES_PORT"] = "5455"
23 os.environ["AGE_GRAPH_NAME"] = "dickens"
24
25 rag = LightRAG(
26     working_dir=WORKING_DIR,
27     llm_model_func=ollama_model_complete,
28     llm_model_name="llama3.1:8b",
29     llm_model_max_async=4,
30     llm_model_max_token_size=32768,
31     llm_model_kwargs={"host": "http://localhost:11434", "options": {"num_ctx": 32768}},
32     embedding_func=EmbeddingFunc(
33         embedding_dim=768,
34         max_token_size=8192,
35         func=lambda texts: ollama_embedding(
36             texts, embed_model="nomic-embed-text", host="http://localhost:11434"
37         ),
38     ),
39 )
```

What It Does

- Defines a GET endpoint /query that accepts questions and mode.
- Uses LightRAG to retrieve relevant context based on the selected mode.
- Returns the generated answer or an error if something goes wrong.

2. Key Frontend Code (React)


- **Fetching Modes and Sending Queries**

```
1
2 import React, { useState, useEffect } from 'react';
3 import axios from 'axios';
4
5 const ChatApp = () => {
6   const [mode, setMode] = useState('naive');
7   const [availableModes, setAvailableModes] = useState([]);
8   const [question, setQuestion] = useState('');
9
10  useEffect(() => {
11    // Get list of modes from backend
12    axios.get('http://localhost:8000/modes')
13      .then((res) => setAvailableModes(res.data))
14      .catch((err) => console.error(err));
15  }, []);
16
17  const sendQuery = async () => {
18    try {
19      const response = await axios.get('http://localhost:8000/query', {
20        params: { question, mode }
21      });
22      // Handle the response
23      console.log(response.data.response);
24    } catch (error) {
25      console.error(error);
26    }
27  };
28
29  // ...
30 };
```

What It Does

- On component mount, fetches the retrieval modes from /modes.
- Sends a GET request to /query with the user's question and selected mode.
- Logs or handles the response for display in the chat UI.

- **Streaming the Answer (Typing Effect)**

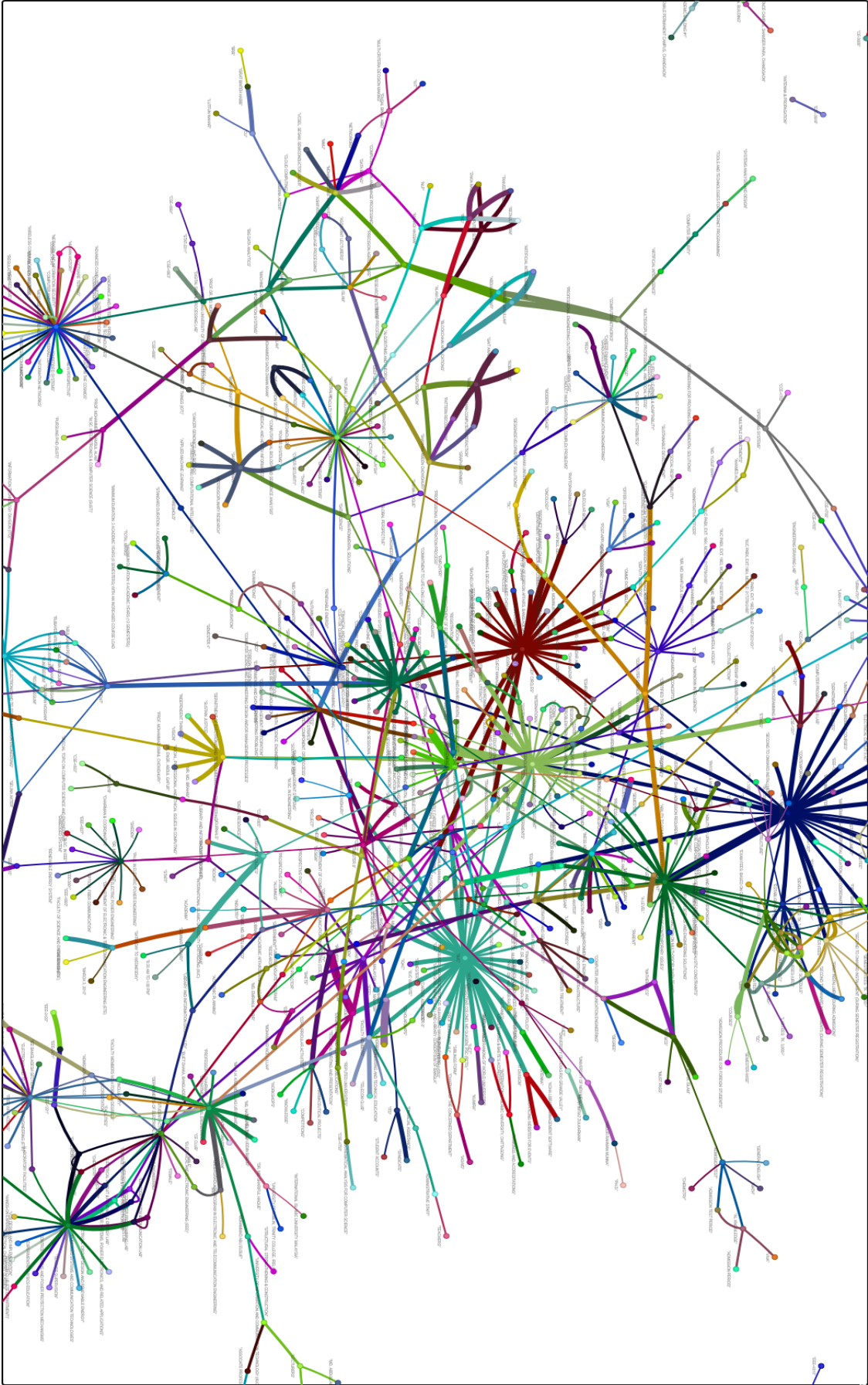


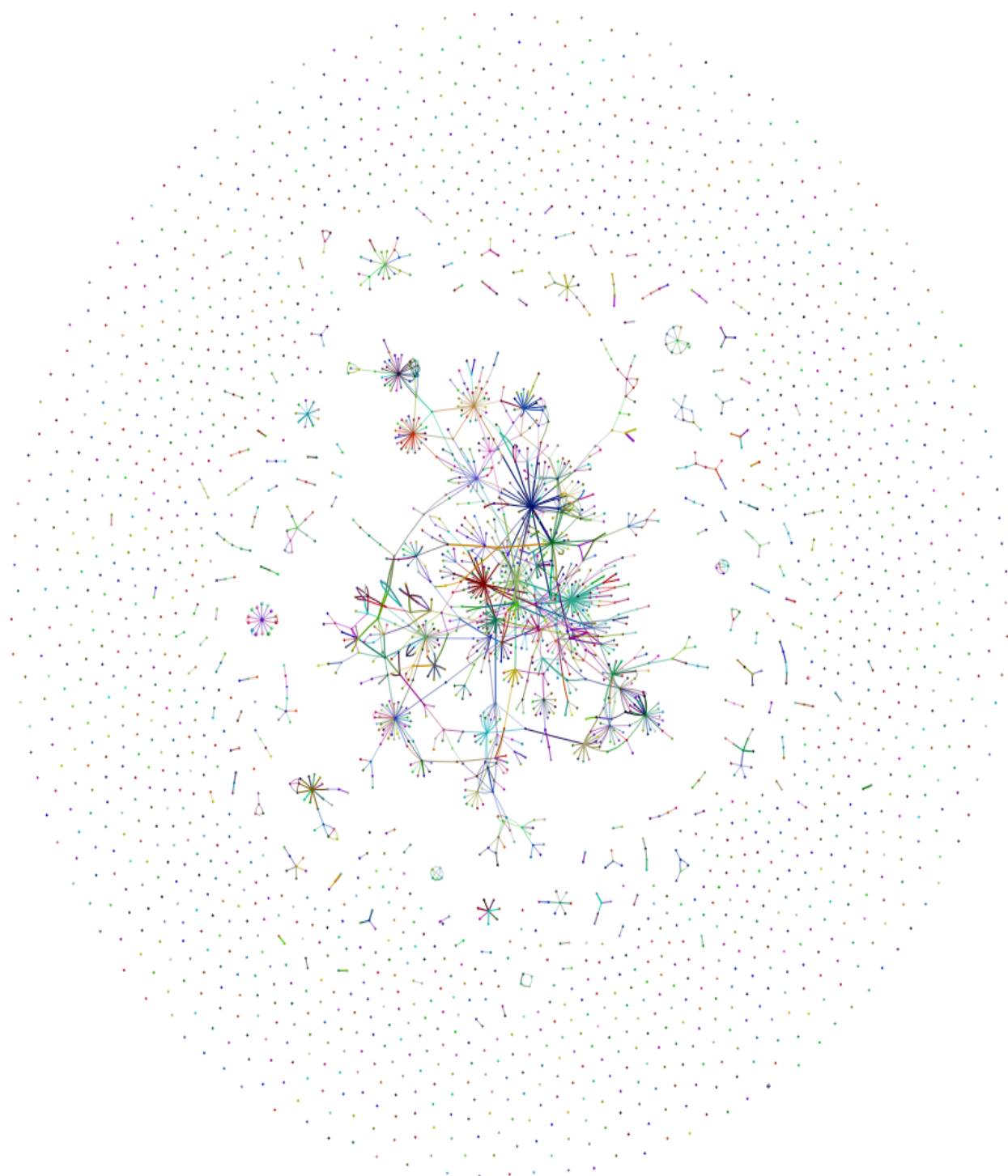
```
1 const streamText = async (text, setCurrentText) => {
2   const words = text.split(' ');
3   let accumulated = '';
4
5   for (let i = 0; i < words.length; i++) {
6     accumulated += words[i] + ' ';
7     setCurrentText(accumulated.trim());
8
9     // Delay for typing effect
10    await new Promise(resolve => setTimeout(resolve, Math.min(50, 20 + words[i].length * 2)));
11  }
12  };
```

What It Does

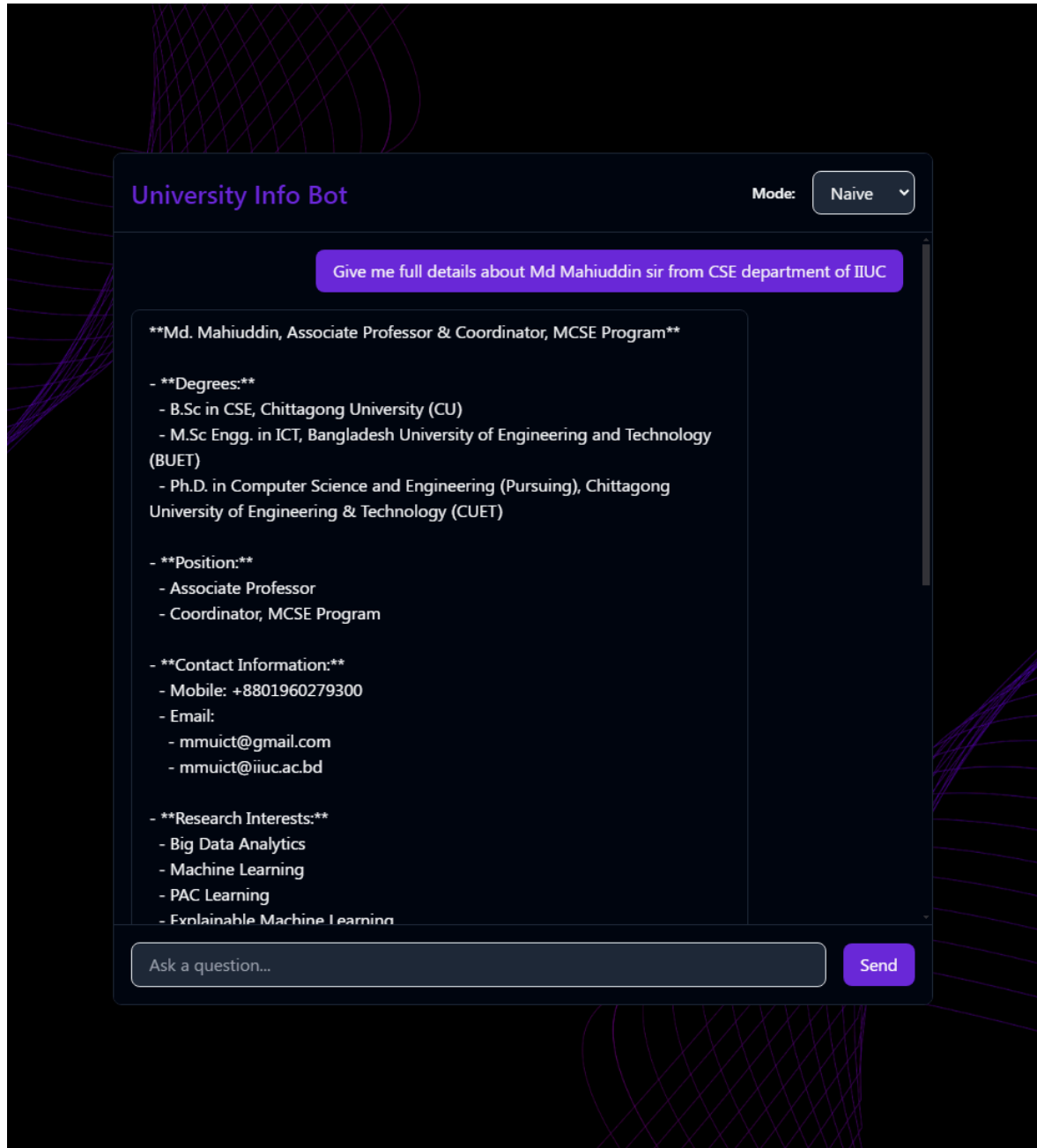
- Splits the final answer into words.
- Accumulates and updates the UI with a delayed loop, simulating a real-time typing effect.
- Provides a more interactive and conversational user experience.

VISUALIZATION OF LIGHTRAG





USER INTERFACE



CONCLUSION

By combining LightRAG, FastAPI, React, and a Knowledge Graph, we have created a powerful RAG system that:

- Handles a Range of Queries: From high-level summaries to targeted details.
- Offers Multiple Retrieval Modes: Naive, local, global, and hybrid for flexibility.
- Scales and Updates Efficiently: Incremental updates ensure the knowledge base remains current with minimal downtime.
- Provides Clear Visualization: The Knowledge Graph reveals intricate relationships, aiding developers and end users.

This approach showcases how graph-based indexing and dual-level retrieval can significantly enhance the accuracy and usability of domain-specific Q&A systems. Future enhancements may include more advanced graphs (with advanced inference rules), multi-lingual data, or integration with real-time databases.
