For the **Skill India - Personalized Learning Pathways** project, where an LLM uses **RAG (Retrieval-Augmented Generation)** to retrieve information and generate personalized learning pathways, you’ll need data related to:

* Job opportunities
* Government-sponsored training programs
* Skill gaps
* Professions
* Skills
* Courses

Here's where you can obtain relevant data for each of these categories:

**1. Government Data Sources**

* **Skill India Portal (NSDC)**: The **National Skill Development Corporation (NSDC)** provides information on various government-sponsored training programs, skill development courses, and career pathways. You can explore:
  + **Skill India Mission**: [Skill India Official Website](https://www.skillindia.gov.in/),

<https://www.skillindiadigital.gov.in/home>

* + **NSDC Open Data**: Some reports and datasets might be available for public use. https://nsdcindia.org/
  + **PMKVY (Pradhan Mantri Kaushal Vikas Yojana)**: Provides information on various skill development initiatives and available courses. https://www.pmkvyofficial.org/
* **National Career Service (NCS)**: Offers job opportunities, career guidance, and skill development programs. Data on job vacancies and training can be retrieved through their portal: [NCS Portal](https://www.ncs.gov.in/).

**2. Open Data from Government APIs**

* **India Open Government Data Platform**: The Government of India’s data portal provides a wide range of datasets related to employment, training programs, and skill development. Relevant datasets may include labor market trends, skill development programs, job openings, etc.
  + **Open Data Platform**: [data.gov.in](https://data.gov.in/)
* **APIs**: You can explore and integrate open APIs like those from NSDC and NCS to retrieve real-time data on training programs and job opportunities.

**3. International Open Datasets**

* **World Bank Open Data**: The World Bank provides datasets on global employment trends, skill gaps, and economic indicators, which can be helpful in understanding skill gaps and trends in various professions.
  + **World Bank Data**: [World Bank Open Data](https://data.worldbank.org/)
* **UNESCO Institute for Statistics (UIS)**: For data on education and training programs globally, UNESCO’s datasets may offer insights on skill development and vocational education.
  + **UNESCO UIS**: [UIS Data](http://uis.unesco.org/)

**4. Job Opportunity Data**

* **Job Boards and Aggregators**:
  + **LinkedIn Jobs API**: You can scrape or use LinkedIn job posts related to different industries and skill sets.
  + **Indeed API**: For retrieving job postings related to specific skills or industries.
  + **Naukri API**: (If available) for retrieving job postings in the Indian context.

**5. MOOCs and Online Courses**

* **Coursera, edX, and Udemy APIs**: Many online learning platforms provide APIs to access their course catalogs. This could help retrieve data on courses that align with the skills needed for different professions.
* **SWAYAM (India's MOOC Platform)**: The Government of India’s **SWAYAM** platform provides numerous skill development courses. Data can be scraped from the platform or obtained through their API (if available).
  + **SWAYAM Courses**: [SWAYAM Portal](https://swayam.gov.in/)

**6. Skill Gap Reports and Research**

* **FICCI-KPMG Reports**: The Federation of Indian Chambers of Commerce and Industry (FICCI) and KPMG periodically publish reports on skill gaps and the future of jobs in India, which provide valuable insights into the skills required by different sectors.
* **McKinsey Global Institute**: McKinsey publishes reports on skill gaps, reskilling needs, and the future of work.
* **National Skill Development Reports**: NSDC publishes reports on industry-wise skill gaps in India. These reports could provide the insights needed to identify skills gaps for various professions.

**7. Knowledge Graph Creation**

* **Open Skills API (Workforce Data Initiative)**: Provides access to structured data on job skills, job roles, and qualifications. This can be helpful in building your knowledge graph of professions, skills, and courses.
  + [Open Skills API](https://github.com/openskills)
* **ESCO (European Skills, Competences, Qualifications, and Occupations)**: ESCO provides a structured classification of skills, competences, qualifications, and occupations. This data can be integrated into your knowledge graph.
  + [ESCO Portal](https://ec.europa.eu/esco/portal/home)

**8. Partnerships with Educational Institutions or Training Providers**

* **National Skill Training Institutes (NSTIs)**: Collaborating with NSTIs might provide access to specific training program data, courses, and assessments aligned with the Skill India Mission.
* **Industry-Sponsored Skill Development Programs**: Many industries, especially in sectors like IT, manufacturing, and healthcare, run their own skill development programs that could be integrated into your system.

**Data Preparation and Integration**

Once you gather data from the above sources, you can organize it into a **knowledge graph** of:

* **Professions** (e.g., IT Developer, Electrician)
* **Skills** (e.g., Python programming, wiring circuits)
* **Courses** (e.g., Online Python Course, Electrical Engineering certification)
* **Job Opportunities** (linking jobs to required skills)

The LLM (Ollama or another model) can then use **RAG** to retrieve relevant information and suggest personalized learning pathways to users based on their skillset, career goals, and available opportunities.

Would you like assistance in building or integrating any part of this system?

To build a **Streamlit app using Ollama and a Knowledge Graph** for the **Skill India - Personalized Learning Pathways** use case, here's a step-by-step guide to help you proceed:

**Steps to Build the Application**

**1. Define Project Structure**

Set up a folder structure for your project:

bash

Copy code

skill-india-pathways/

├── app.py # Streamlit app

├── data/ # Folder to store data (datasets, knowledge graph)

├── kg\_construction.py # Script for building the Knowledge Graph

├── ollama\_integration.py # Script for integrating Ollama LLM

├── requirements.txt # List of required libraries

└── utils.py # Utility functions for data retrieval, etc.

**2. Set Up the Environment**

Since you're already using Anaconda, create a new environment to keep the dependencies isolated.

bash

Copy code

conda create -n skill\_india\_env python=3.9

conda activate skill\_india\_env

Install necessary dependencies for Streamlit, Graph Database, and Ollama:

bash

Copy code

pip install streamlit neo4j ollama

**3. Knowledge Graph Construction**

You'll need to build a **Knowledge Graph** that connects skills, professions, courses, and job opportunities. You can use **Neo4j** as the graph database.

1. **Set Up Neo4j**:
   * Install Neo4j Desktop or use **Neo4j AuraDB** (cloud-based option).
   * Create a database where nodes represent:
     + **Professions** (e.g., Data Scientist, Electrician)
     + **Skills** (e.g., Python, Electrical Wiring)
     + **Courses** (e.g., Coursera, SWAYAM courses)
     + **Jobs** (linked to the required skills and professions)
2. **Construct the Graph**: Create relationships like:
   * **HAS\_SKILL**: Connect professions to skills
   * **OFFERS\_COURSE**: Connect courses to skills
   * **JOB\_REQUIRES\_SKILL**: Connect job opportunities to the required skills

Here's a sample script (kg\_construction.py) to build the knowledge graph in **Neo4j**:

python

Copy code

from neo4j import GraphDatabase

# Connect to the Neo4j database

uri = "bolt://localhost:7687"

driver = GraphDatabase.driver(uri, auth=("neo4j", "password"))

def create\_node(tx, label, properties):

query = f"CREATE (n:{label} {{name: $name, description: $description}})"

tx.run(query, properties)

def create\_relationship(tx, node1, node2, relationship):

query = f"MATCH (a {{name: $name1}}), (b {{name: $name2}}) CREATE (a)-[:{relationship}]->(b)"

tx.run(query, name1=node1, name2=node2)

with driver.session() as session:

session.write\_transaction(create\_node, "Profession", {"name": "Data Scientist", "description": "Data analysis profession"})

session.write\_transaction(create\_node, "Skill", {"name": "Python", "description": "Programming language"})

session.write\_transaction(create\_relationship, "Data Scientist", "Python", "HAS\_SKILL")

**4. Integrating Ollama LLM (Retrieval-Augmented Generation)**

**Ollama** can be used for **Retrieval-Augmented Generation (RAG)** to answer user queries by retrieving knowledge from the graph and generating personalized learning pathways.

1. **Install Ollama**:

bash

Copy code

pip install ollama

In ollama\_integration.py, define a function to query Ollama:

python

Copy code

import ollama

def query\_ollama(prompt):

response = ollama.query(model="llm-model-name", prompt=prompt)

return response['text']

1. **Use RAG for Personalized Pathways**:
   * The LLM can be prompted to retrieve information from the graph.
   * For example, ask it to suggest learning pathways by matching users’ current skills with available courses from the knowledge graph.

**5. Building the Streamlit App**

Now, create a **Streamlit app** (app.py) to interface with the user, retrieve data from the knowledge graph, and interact with Ollama LLM.

1. **Basic Streamlit Setup**:

python

Copy code

import streamlit as st

from neo4j import GraphDatabase

from ollama\_integration import query\_ollama

# Connect to Neo4j

driver = GraphDatabase.driver("bolt://localhost:7687", auth=("neo4j", "password"))

def get\_skills\_for\_profession(profession):

with driver.session() as session:

result = session.run("MATCH (p:Profession {name: $profession})-[:HAS\_SKILL]->(s:Skill) RETURN s.name", profession=profession)

return [record["s.name"] for record in result]

st.title("Skill India - Personalized Learning Pathways")

user\_profession = st.text\_input("Enter your profession")

if user\_profession:

# Retrieve related skills from Neo4j

skills = get\_skills\_for\_profession(user\_profession)

st.write(f"Skills required for {user\_profession}: {', '.join(skills)}")

# Query Ollama to generate learning pathway

ollama\_prompt = f"Generate a personalized learning pathway for someone working as a {user\_profession} who needs to learn {', '.join(skills)}."

response = query\_ollama(ollama\_prompt)

st.write("Suggested Learning Pathway:")

st.write(response)

1. **Visualizing Learning Pathways**: You can enhance the app by adding visualization features for the knowledge graph (using **pyvis** or **networkx** libraries) to show relationships between professions, skills, and courses.

**6. Testing and Deployment**

* Run the app locally to test:

bash

Copy code

streamlit run app.py

* Deploy the app on **Streamlit Cloud** or **Heroku** for easy access.

**7. Future Enhancements**

* **User Authentication**: Implement user profiles to track their learning progress.
* **Recommendation System**: Use graph algorithms to suggest relevant skills and courses based on the user’s profession and current skill set.
* **Interactive Queries**: Allow users to ask Ollama personalized queries about specific professions, training programs, or job opportunities.

This approach combines **Neo4j** for storing and retrieving knowledge, **Ollama** for generating personalized insights, and **Streamlit** to provide an interactive interface, making the solution comprehensive for Skill India's mission. Let me know if you need help with any specific part of this implementation!

**ChatGPT said:**

ChatGPT

Using LinkedIn to gather data for a Skill India project is feasible, but LinkedIn's data access is restricted and requires adherence to their API usage policies. Here’s a step-by-step approach to using LinkedIn for data on professions, skills, and courses:

**1. LinkedIn API Access (Preferred Method)**

LinkedIn provides an official **REST API**, but access to its full data (such as skills required for specific professions) is restricted and typically requires approval for use cases like data analysis or skill-matching applications. Here’s how to get started:

* **Step 1: Apply for LinkedIn API Access**
  + Go to the [LinkedIn Developers Platform](https://developer.linkedin.com/).
  + Create a LinkedIn Developer account and register a new app.
  + Submit a detailed application explaining your use case (e.g., research on skill development aligned with Skill India).
* **Step 2: API Key and OAuth Authentication**
  + Once LinkedIn approves your application, you’ll receive an API Key and Secret.
  + Use **OAuth 2.0** for authentication, as LinkedIn requires it for API access.
* **Step 3: Use the People and Skills API Endpoints**
  + LinkedIn’s API offers endpoints such as People, Jobs, and Skills to retrieve data.
  + Example of using LinkedIn's API to retrieve skills for a given job title:

python

Copy code

import requests

# Sample API endpoint (Replace with the actual LinkedIn API endpoint)

url = "https://api.linkedin.com/v2/jobSkills?q=jobTitle&title=Data Scientist"

headers = {

"Authorization": "Bearer ACCESS\_TOKEN" # Replace with your OAuth access token

}

response = requests.get(url, headers=headers)

if response.status\_code == 200:

data = response.json()

for skill in data.get("skills", []):

print(skill["name"]) # Print skill names

else:

print("Error:", response.status\_code, response.text)

* **Step 4: Data Processing and Storage**
  + Process the API data, store it in CSV or JSON format, and use it for Neo4j import.

**2. LinkedIn Learning Data**

LinkedIn Learning (formerly Lynda) offers courses, skills assessments, and certification information which can be valuable for building a skills knowledge graph. However, LinkedIn Learning's data isn’t publicly accessible via API, so your options are:

* **Browse Manually**:
  + Browse LinkedIn Learning courses that align with specific skills or professions and manually record relevant data points (course name, duration, skills taught).
* **Web Scraping (With Caution)**:
  + Scraping LinkedIn Learning is discouraged due to LinkedIn’s strict anti-scraping policies. However, if permitted by LinkedIn, tools like **BeautifulSoup** or **Selenium** could automate data extraction.
  + Check LinkedIn’s robots.txt file and terms of service before attempting any scraping.

**3. Alternative Sources for LinkedIn Data: Public Datasets and Aggregators**

* **Kaggle**:
  + LinkedIn-related datasets on Kaggle might offer data on popular job roles, skill requirements, and job descriptions.
  + Search Kaggle with terms like “LinkedIn job roles dataset” or “LinkedIn skills data.”
* **External Job Portals** (Similar to LinkedIn):
  + Consider aggregators or job portals like **Indeed**, **Glassdoor**, or **Monster** for open data related to professions and skills.

**Summary**

LinkedIn is a robust source for profession and skills data, but access is controlled. LinkedIn's API offers the most reliable method, but requires approval, while LinkedIn Learning provides a broad range of course data for skill development. Supplementing this with external sources like Kaggle or job aggregators can provide a well-rounded dataset for your Skill India project.

Since you have successfully imported the CSV data and verified that your graph with nodes and relationships is created in Neo4j, here are the **next steps** you can take to continue building your Skill India project:

**1. Analyze the Graph Data**

Start by running some basic Cypher queries to understand the structure and relationships in your graph:

* **Get all Industries and their required Skills**:

cypher

Copy code

MATCH (industry:Industry)-[:HAS\_CATEGORY]->(category:SkillGroupCategory)-[:REQUIRES\_SKILL]->(skill:SkillGroup)

RETURN industry.name AS Industry, skill.name AS Skill LIMIT 10;

* **List top industries based on skill requirements**:

cypher

Copy code

MATCH (industry:Industry)-[:HAS\_CATEGORY]->(category:SkillGroupCategory)-[:REQUIRES\_SKILL]->(skill:SkillGroup)

RETURN industry.name AS Industry, COUNT(skill) AS NumberOfSkills

ORDER BY NumberOfSkills DESC LIMIT 5;

* **Skills grouped by ISIC Sections**:

cypher

Copy code

MATCH (isic:ISICSection)<-[:FROM\_isiccat]-(industry:Industry)-[:HAS\_CATEGORY]->(category:SkillGroupCategory)-[:REQUIRES\_SKILL]->(skill:SkillGroup)

RETURN isic.name AS ISIC\_Section, skill.name AS Skill LIMIT 10;

**2. Build a Streamlit App for Visualization**

You can create a simple **Streamlit app** to visualize and interact with your Neo4j graph.

**Steps**:

1. **Install the Streamlit and Neo4j Python driver** if not already installed:

bash

Copy code

pip install streamlit neo4j

1. **Create a Python file** (e.g., skill\_india\_app.py) with the following content:

python

Copy code

import streamlit as st

from neo4j import GraphDatabase

# Connect to Neo4j

uri = "bolt://localhost:7687" # Adjust if using a different host

driver = GraphDatabase.driver(uri, auth=("neo4j", "your\_password"))

# Query function

def fetch\_data(query):

with driver.session() as session:

result = session.run(query)

return [record.data() for record in result]

# Streamlit UI

st.title("Skill India Knowledge Graph")

# Show Industry and Skills

st.subheader("Top 5 Industries by Skill Requirements")

data = fetch\_data("MATCH (industry:Industry)-[:HAS\_CATEGORY]->(category)-[:REQUIRES\_SKILL]->(skill) RETURN industry.name AS Industry, COUNT(skill) AS NumberOfSkills ORDER BY NumberOfSkills DESC LIMIT 5")

st.table(data)

# Interactive Skill Query

industry\_name = st.text\_input("Enter an Industry Name:")

if industry\_name:

query = f"""

MATCH (industry:Industry {{name: '{industry\_name}'}})-[:HAS\_CATEGORY]->(category)-[:REQUIRES\_SKILL]->(skill)

RETURN skill.name AS RequiredSkills

"""

skills = fetch\_data(query)

if skills:

st.write("Skills Required:", skills)

else:

st.write("No skills found for this industry.")

1. **Run the Streamlit app**:

bash

Copy code

streamlit run skill\_india\_app.py

**3. Enhance the Graph Data with LLM Integration using Ollama**

You can integrate **Ollama** or another LLM to provide natural language insights or recommendations based on the graph data.

**Example Use Case**:

* Allow users to ask questions like "What skills are required for the IT industry?" and generate insights using an LLM.

**4. Perform Graph Algorithms for Deeper Analysis (Optional)**

Leverage Neo4j's built-in graph algorithms to find patterns or insights:

* **PageRank** to identify the most influential skills.
* **Community Detection** to identify clusters of similar industries based on skill requirements.

**5. Document Your Project**

Prepare documentation for your project, including:

* **Problem Statement**.
* **Data Source** (e.g., Kaggle dataset, Skill India data).
* **Graph Structure** (nodes, relationships).
* **Key Cypher Queries**.
* **How to Run the Streamlit App**.

**Summary**

* **Explore Data**: Run Cypher queries to understand your graph.
* **Build App**: Create a Streamlit app for interactive visualization.
* **Enhance Features**: Add LLM integration or graph algorithms for deeper insights.
* **Documentation**: Document your project for better understanding and presentation.

This way, you will have a complete end-to-end project that includes data ingestion, graph creation, visualization, and interactive exploration.

4o

Top of Form

how to integrating real-time queries to job portals or using APIs for fetching the latest skill demands in industries.

**ChatGPT said:**

ChatGPT

Integrating real-time queries to job portals or using APIs for fetching the latest skill demands in industries involves several steps. Here's a comprehensive guide:

**1. Identify Suitable Job Portals or APIs**

Some popular job portals and platforms provide APIs to access their data:

* **LinkedIn API** (requires access approval for job data).
* **Indeed API** (Job Search API for accessing job postings).
* **Monster API** (job and skill data via their APIs).
* **Glassdoor API** (company reviews, salaries, and job insights).

**2. Obtain API Access**

* **Register for API Access**:
  + Visit the job portal's developer page (e.g., LinkedIn Developer Portal, Indeed API documentation).
  + Create an account or application and request access.
* **Generate API Key**:
  + Once approved, obtain an API key or token for authentication.

**3. Explore the API Documentation**

* Understand the endpoints provided by the API.
* Look for relevant endpoints for:
  + Job postings.
  + Industry-specific skills or trends.
  + Required skills for job roles.

For example:

* **LinkedIn**: GET /jobs/{job\_id}
* **Indeed**: GET /jobs?q={keyword}

**4. Build the Integration**

**a. Install Required Libraries**

Install libraries for making HTTP requests and processing JSON responses:

bash

Copy code

pip install requests

**b. Write Python Code to Fetch Data**

Here’s an example for integrating with a generic API:

python

Copy code

import requests

class JobPortalAPI:

def \_\_init\_\_(self, api\_key, base\_url):

self.api\_key = api\_key

self.base\_url = base\_url

def fetch\_skill\_demands(self, industry, location):

endpoint = f"{self.base\_url}/jobs"

headers = {"Authorization": f"Bearer {self.api\_key}"}

params = {"industry": industry, "location": location}

response = requests.get(endpoint, headers=headers, params=params)

if response.status\_code == 200:

return response.json() # Process and return JSON data

else:

print("Error:", response.status\_code, response.text)

return None

# Example usage

api\_key = "your\_api\_key\_here"

base\_url = "https://api.jobportal.com"

job\_api = JobPortalAPI(api\_key, base\_url)

data = job\_api.fetch\_skill\_demands(industry="Information Technology", location="India")

print(data)

**c. Parse and Store Data**

* Extract relevant fields (e.g., skills, roles, locations).
* Store the data in:
  + A local database (e.g., SQLite, PostgreSQL).
  + Your Neo4j knowledge graph as new nodes and relationships.

**5. Integrate with the Knowledge Graph**

Store the fetched skills and demands into your Neo4j graph:

**Example Cypher Query:**

cypher

Copy code

MERGE (job:Job {title: $job\_title})

MERGE (skill:Skill {name: $skill\_name})

MERGE (industry:Industry {name: $industry\_name})

MERGE (job)-[:REQUIRES\_SKILL]->(skill)

MERGE (job)-[:IN\_INDUSTRY]->(industry)

Update the Python integration:

python

Copy code

def store\_data\_in\_graph(driver, job\_data):

with driver.session() as session:

for job in job\_data:

session.run("""

MERGE (job:Job {title: $title})

MERGE (skill:Skill {name: $skill})

MERGE (industry:Industry {name: $industry})

MERGE (job)-[:REQUIRES\_SKILL]->(skill)

MERGE (job)-[:IN\_INDUSTRY]->(industry)

""", title=job["title"], skill=job["skill"], industry=job["industry"])

**6. Schedule Real-Time Updates**

Use task schedulers like **Celery** or **cron jobs** to fetch and update the skill demands periodically.

* **Example with schedule library**:

python

Copy code

import schedule

import time

def fetch\_and\_store\_job\_data():

data = job\_api.fetch\_skill\_demands(industry="Information Technology", location="India")

if data:

store\_data\_in\_graph(driver, data)

schedule.every(6).hours.do(fetch\_and\_store\_job\_data)

while True:

schedule.run\_pending()

time.sleep(1)

**7. Display Data in Streamlit App**

Update the Streamlit app to show real-time skill demands:

* Add a button to fetch the latest skills.
* Display the data in a table or graph using st.dataframe or st.plotly\_chart.

**8. Challenges and Solutions**

* **Rate Limits**: APIs often have usage limits; cache results to avoid hitting limits.
* **Data Consistency**: Clean and normalize data before storing it in the knowledge graph.
* **API Costs**: Some APIs charge for access. Use open APIs or negotiate access.

By implementing these steps, you can effectively fetch real-time data from job portals, integrate it into your Neo4j knowledge graph, and display it in your Streamlit app.

Bottom of Form

i have create streamlit app using neo4j and ollama. i want deploy it using git hoe to do it

**ChatGPT said:**

ChatGPT

To deploy a Streamlit app that uses **Neo4j** and **Ollama**, using **GitHub**, follow these steps:

**Step 1: Prepare Your Project**

1. **Organize Project Files**:
   * Ensure all files are in a single folder, including:
     + app.py: Your Streamlit app script.
     + requirements.txt: A list of all dependencies.
     + .streamlit/: Configuration files for Streamlit.
2. **Create a requirements.txt**:
   * List all Python dependencies in this file. For example:

Copy code

streamlit

neo4j

ollama

pandas

requests

* + Generate automatically using:

bash

Copy code

pip freeze > requirements.txt

1. **Create a .streamlit/config.toml File** (optional):
   * Add configurations for your app, e.g.:

toml

Copy code

[server]

headless = true

port = $PORT

enableCORS = false

1. **Environment Variables**:
   * If your app uses sensitive keys (e.g., Neo4j credentials, Ollama API keys), create a .env file and include:

makefile

Copy code

NEO4J\_URI=bolt://<your-database-uri>

NEO4J\_USERNAME=<your-username>

NEO4J\_PASSWORD=<your-password>

OLLAMA\_API\_KEY=<your-ollama-key>

* + Use the python-dotenv library to load these in your app:

python

Copy code

from dotenv import load\_dotenv

import os

load\_dotenv()

NEO4J\_URI = os.getenv("NEO4J\_URI")

**Step 2: Push to GitHub**

1. **Initialize Git**:
   * Navigate to your project folder:

bash

Copy code

git init

git add .

git commit -m "Initial commit"

1. **Create a GitHub Repository**:
   * Go to [GitHub](https://github.com/) and create a new repository.
2. **Link Your Project to GitHub**:
   * Add the remote repository:

bash

Copy code

git remote add origin https://github.com/username/repository-name.git

git branch -M main

git push -u origin main

**Step 3: Deploy the App**

You can deploy the Streamlit app using one of the following platforms:

**Option 1: Streamlit Community Cloud**

1. **Go to Streamlit Cloud**: Streamlit Cloud.
2. **Connect GitHub**:
   * Log in and click **New App**.
   * Select your GitHub repository and branch (main).
3. **Set Environment Variables**:
   * Go to the **Advanced Settings** and add your variables:

bash

Copy code

NEO4J\_URI=bolt://<your-database-uri>

NEO4J\_USERNAME=<your-username>

NEO4J\_PASSWORD=<your-password>

OLLAMA\_API\_KEY=<your-ollama-key>

1. **Deploy**:
   * Click **Deploy**, and your app will be live.

**Option 2: Deploy on Render or Heroku**

**Render:**

1. **Sign Up**: [Render](https://render.com/).
2. **Create a Web Service**:
   * Connect to your GitHub repo.
   * Choose Python as the environment and set the build command:

bash

Copy code

pip install -r requirements.txt

streamlit run app.py

1. **Add Environment Variables**:
   * Under **Environment Settings**, add the variables for Neo4j and Ollama.

**Heroku:**

1. **Install Heroku CLI**:

bash

Copy code

npm install -g heroku

1. **Create a Heroku App**:

bash

Copy code

heroku create

1. **Push Code to Heroku**:

bash

Copy code

git push heroku main

1. **Add Environment Variables**:

bash

Copy code

heroku config:set NEO4J\_URI=bolt://<your-database-uri>

heroku config:set NEO4J\_USERNAME=<your-username>

heroku config:set OLLAMA\_API\_KEY=<your-ollama-key>

**Step 4: Test the Deployment**

1. Access the deployed app using the provided URL (from Streamlit, Render, or Heroku).
2. Ensure Neo4j and Ollama integrations work as expected.

**Step 5: Monitor and Debug**

* **Streamlit Logs**: Access logs via Streamlit Cloud's dashboard.
* **Neo4j Logs**: Check your Neo4j database logs for connection issues.
* **Error Handling**: Use try-except blocks to catch API or database errors in your app.

With these steps, your Streamlit app using Neo4j and Ollama should be successfully deployed and accessible!