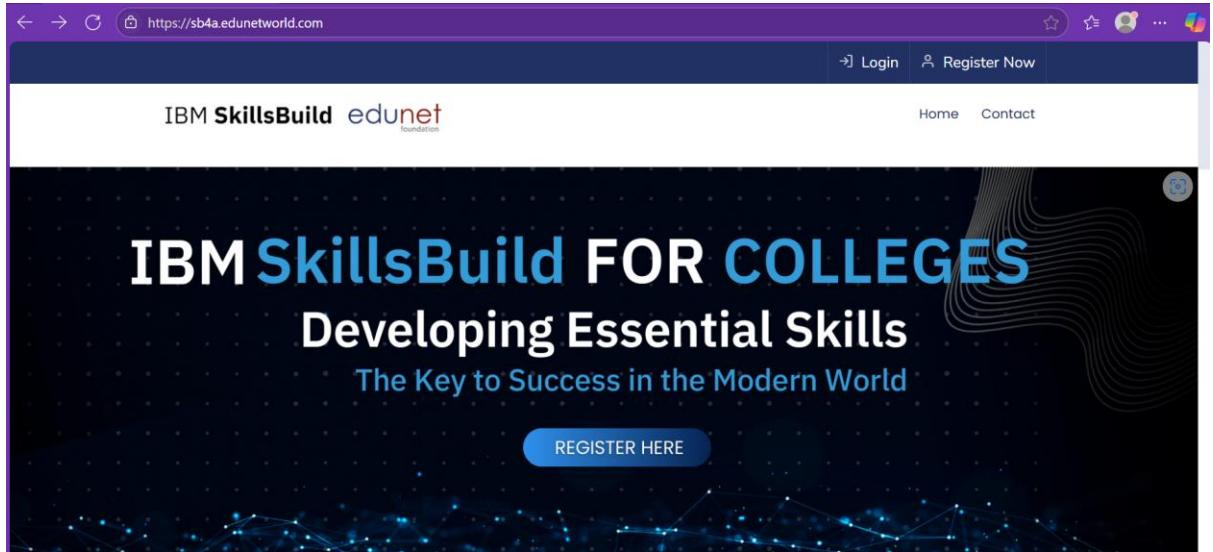


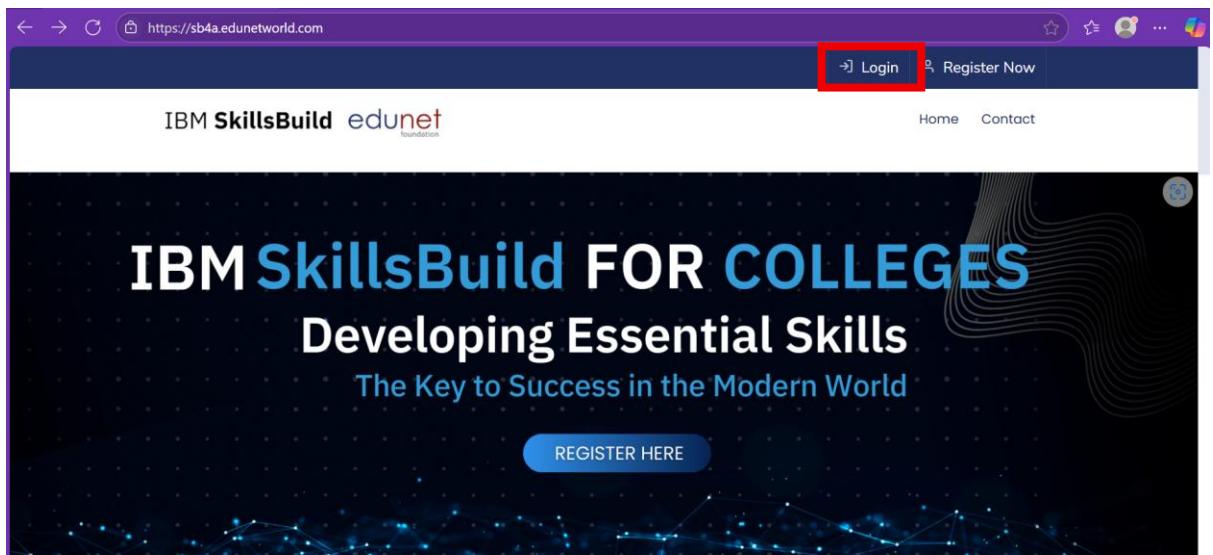
Introduction to RAG Lab

1. Open the browser and type <https://sb4a.edunetworkworld.com/> URL on address bar. Then it directs to the following page.



Welcome to IBM SkillsBuild For Colleagues with

2. Click on Login option from top right corner.



Welcome to IBM SkillsBuild For Colleagues with

3. Login page will be opened.

The screenshot shows a web browser displaying the Edunet foundation login page at <https://sb4a.edunetworkworld.com/user/login>. The page features a logo for 'edunet foundation' and two buttons: 'Login.' and 'Signup'. Below these are fields for 'Email' and 'Password', followed by a 'Captcha' field containing the text '7jCnp' with a reCAPTCHA logo. There is a 'Remember Me' checkbox and a 'Login' button. A 'Forgot Password' link is also present. At the bottom, a copyright notice reads 'Copyright © 2025 Edunet Foundation. All rights reserved.' To the right of the browser window, a photograph of a young woman with dark hair, wearing a light blue patterned dress, is seated at a desk in a computer lab, smiling while working on a computer. The desktop background shows other students and computer monitors.

4. Enter your valid Login Credentials, enter captcha and click on login button

The screenshot shows the same Edunet login page as above, but with the 'Email' and 'Password' fields filled. The 'Email' field contains a redacted email address, and the 'Password' field contains a redacted password. The 'Captcha' field now displays the text 'uetm4' with a reCAPTCHA logo. The 'Login' button is highlighted with a red rectangular box. The copyright notice at the bottom remains the same. To the right of the browser window, the same young woman is shown working on her computer in the lab, with the redboxed 'Login' button overlaid on the original image.

5. You will be logged in successfully. You can see the dashboard like this.

The screenshot shows the student dashboard of edunet foundation. On the left, there's a dark sidebar with navigation links: Dashboard, Recommended Courses (which is currently selected and highlighted in blue), Download Offer Letter, and Raise a Ticket. Below the sidebar, it says "Follow us on:" with icons for Facebook, Twitter, LinkedIn, and Instagram. The main content area has a purple header bar with the URL https://sb4a.edunetwork.com/student-dashboard?p=1. Below the header, a dark blue box displays a message: "Please update your profile." To the right of the message, the word "Dear" is followed by a redacted name. There are three cards in the center: a blue card for "Total Courses" (5), a red card for "Courses Enrolled" (4), and a green card for "Courses Completed" (2). The background of the dashboard is light grey.

6. Click on Recommended Courses from left menu and then select Guided Learning Experience (GLE)

The screenshot shows the "Recommended Courses" page. The left sidebar is identical to the dashboard, with "Recommended Courses" selected. In the main content area, the title "Study Material" is displayed above a course card for "Getting Started with Artificial Intelligence" by IBM SkillsBuild. The course card features the IBM logo and a small image of a brain. To the right of the card, there's a detailed description of the course, including its title, a brief about section, what you can expect, and what you'll learn. At the bottom, it says "After completing Getting Started with AI, you should be able to:". The background is white, and the overall layout is clean and professional.

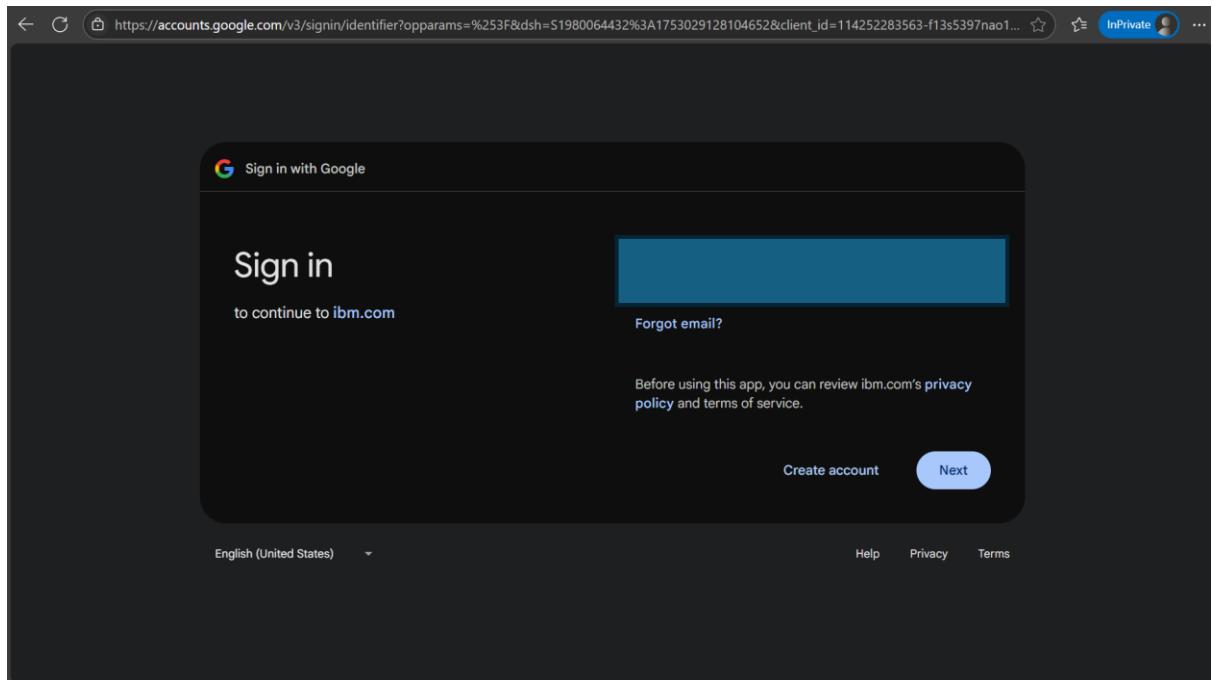
7. Scroll down and find Lab on Retrieval Augmented Generation with Langchain and click on **Enrol Now** button

The screenshot shows a web browser window for the edunet foundation website. On the left, there's a sidebar with options like 'Dashboard', 'Recommended Courses' (which is expanded to show 'Guided Learning Experience (GLEs)'), 'Download Offer Letter', and 'Raise a Ticket'. Below the sidebar, it says 'Follow us on:' with links to Facebook, Twitter, LinkedIn, and YouTube. The main content area has a blue header 'Lab: Retrieval Augmented Generation with LangChain' with a sub-section 'About this learning activity'. Below this, there's a paragraph about RAG, followed by sections for 'What you'll learn' and 'After completing this lab, you should be able to:'. At the bottom of this section is a blue button with white text that says 'Enroll Now →'. This button is highlighted with a red rectangular box. At the very bottom of the page, it says 'Copyright © 2025 Edunet Foundation. All rights reserved.'

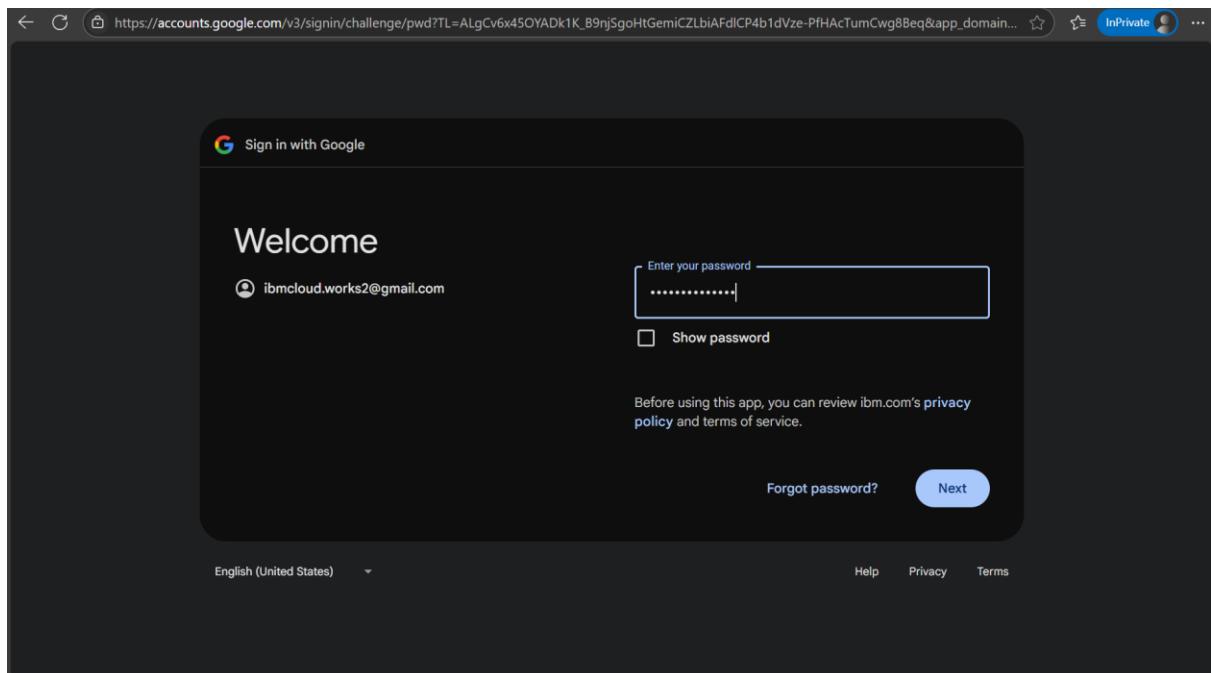
8. It takes you to IBM SkillsBuild Login page. Click on **Accept all** button to allow all cookies and click on **Log in with Google** button.

The screenshot shows the IBM SkillsBuild login page. At the top, there's a navigation bar with the IBM logo and the text 'SkillsBuild'. On the left, there's a list of login options: 'Log in with Google' (highlighted with a red box), 'Log in with Email', 'Log in with LinkedIn', and 'Log in with IBM ID'. Below this, a link says 'Don't have an account with IBM SkillsBuild? Sign up'. On the right, there's a large photo of a woman smiling while working on a laptop. At the bottom right, there's a cookie consent banner with two buttons: 'Accept all' (highlighted with a red box) and 'More options'.

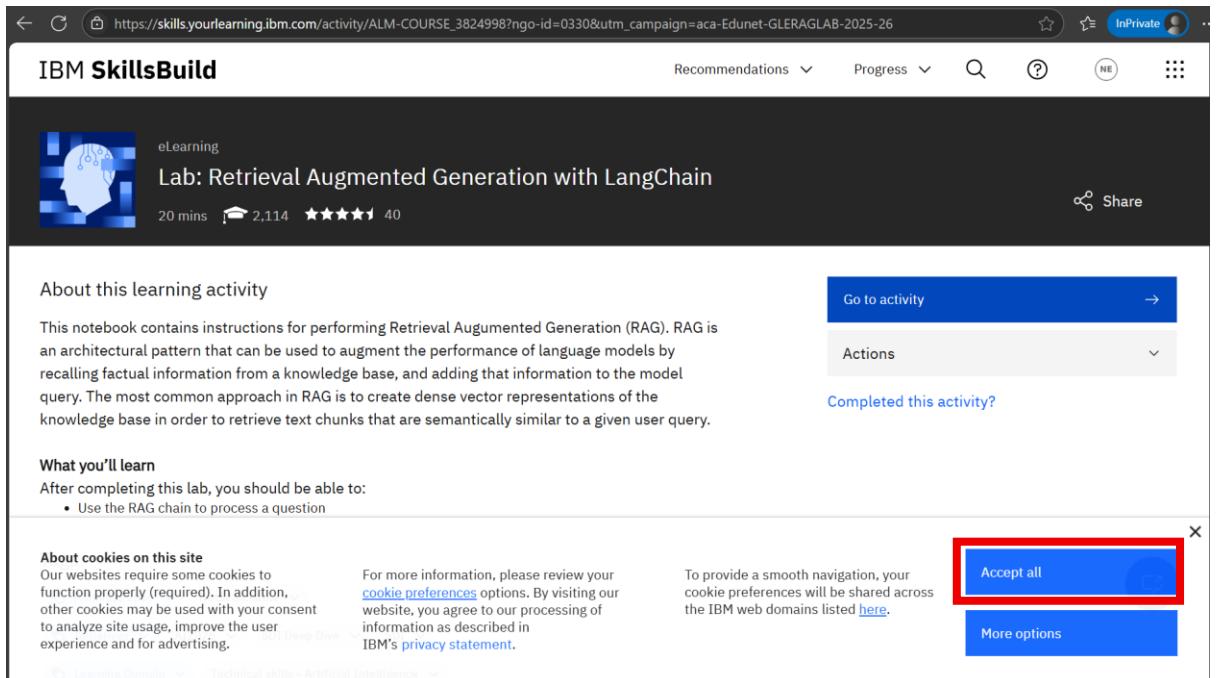
9. It asks you to enter your LMS Gmail id to login (or it prompts you to select your LMS Gmail id from the list) and click on Next



10. Enter Password of your Gmail and click on Next.

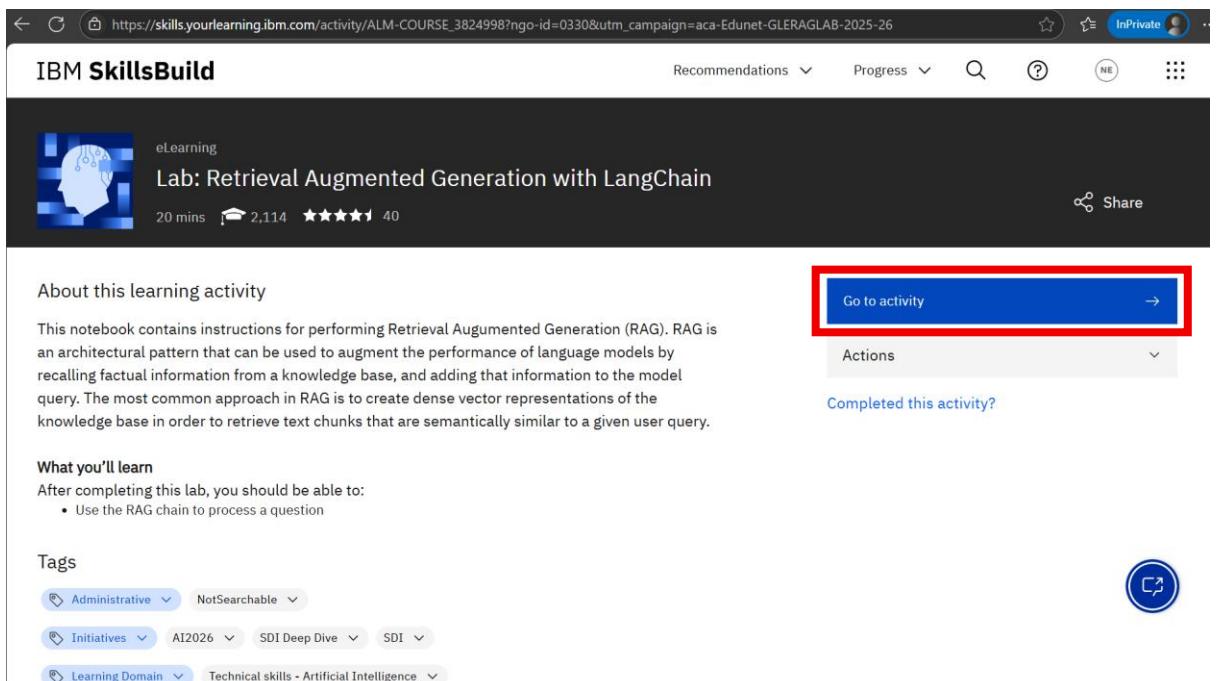


11. Now you are logged in to IBM SkillsBuild platform for the Lab. Click on Accept all button.



The screenshot shows the IBM SkillsBuild interface. At the top, there's a navigation bar with links for Recommendations, Progress, and search. Below the header, there's a banner for an eLearning activity titled "Lab: Retrieval Augmented Generation with LangChain". The activity has a duration of 20 mins, 2,114 views, and a rating of 4 stars. To the right of the activity title are "Share" and "Actions" buttons. A modal window is open at the bottom right, asking "Completed this activity?". Inside this modal, there are two buttons: "Accept all" (highlighted with a red box) and "More options".

12. Click on Go to Activity button



This screenshot shows the same learning activity page as the previous one. The "Go to activity" button is highlighted with a red box. The rest of the page content, including the activity details, "About this learning activity" section, and the cookie consent overlay, remains the same.

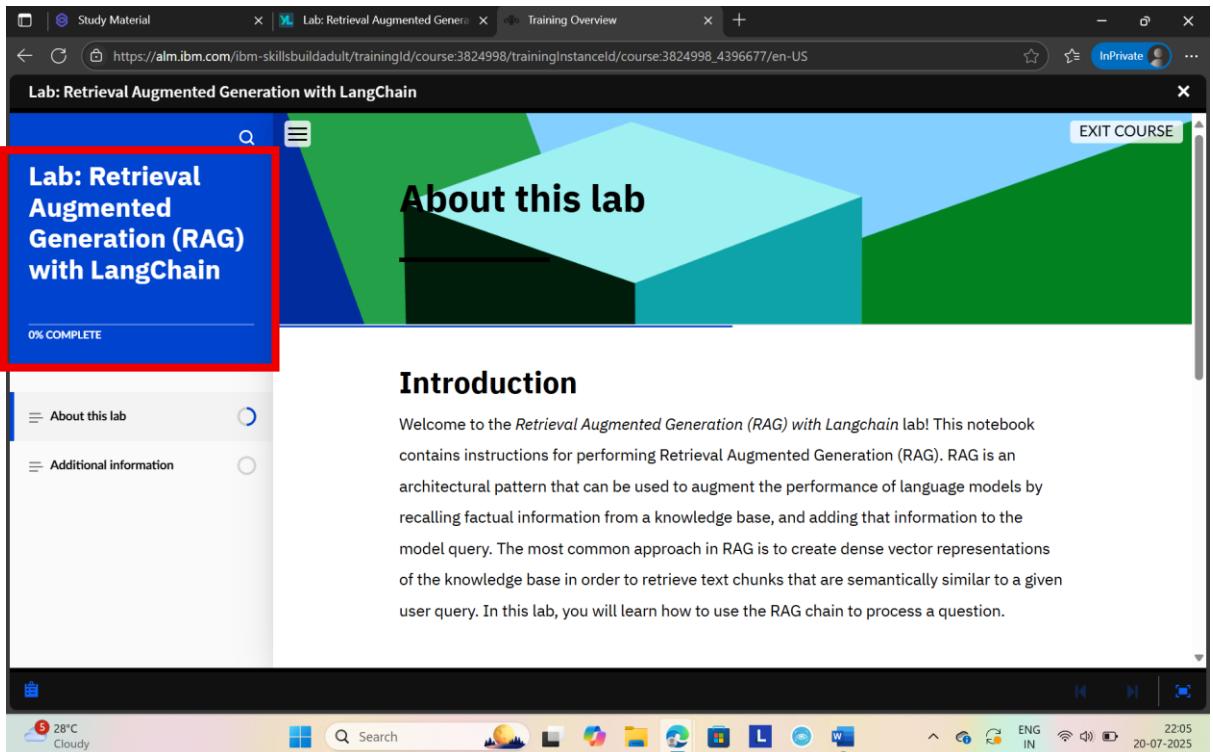
13. In the next page, scroll down and click on Lab: Retrieval Augmented Generation

The screenshot shows a web browser window with three tabs: 'Study Material', 'Lab: Retrieval Augmented Genera', and 'Training Overview'. The main content area is titled 'Lab: Retrieval Augmented Generation with LangChain' and is labeled 'Self-paced'. It includes sections for 'About this learning activity' (describing RAG as an architectural pattern), 'What you'll learn' (mentioning the ability to process a question using the RAG chain), and 'Modules'. A 'Core content' section is shown with a red box around the first item: 'Lab: Retrieval Augmented Generation with LangChain' (estimated time: 20 mins). A progress bar indicates '0/1 Core content completed'.

14. It will show start Course button. Click on it to start the Lab.

The screenshot shows the same web browser window as the previous one, but the content has changed. The title is now 'Lab: Retrieval Augmented Generation with LangChain'. The main area features a large blue background with white text: 'Lab: Retrieval Augmented Generation (RAG) with LangChain' and a prominent 'START COURSE' button. To the right is a 3D geometric graphic composed of various colored cubes (blue, green, cyan, black). The bottom of the screen shows a Windows taskbar with icons for weather (28°C Cloudy), search, file explorer, and other system tools, along with a date and time stamp (20-07-2025).

15. The Lab has started. This is the first interface for the lab. Scroll down to find the requirements to complete this lab.



Lab: Retrieval Augmented Generation (RAG) with LangChain

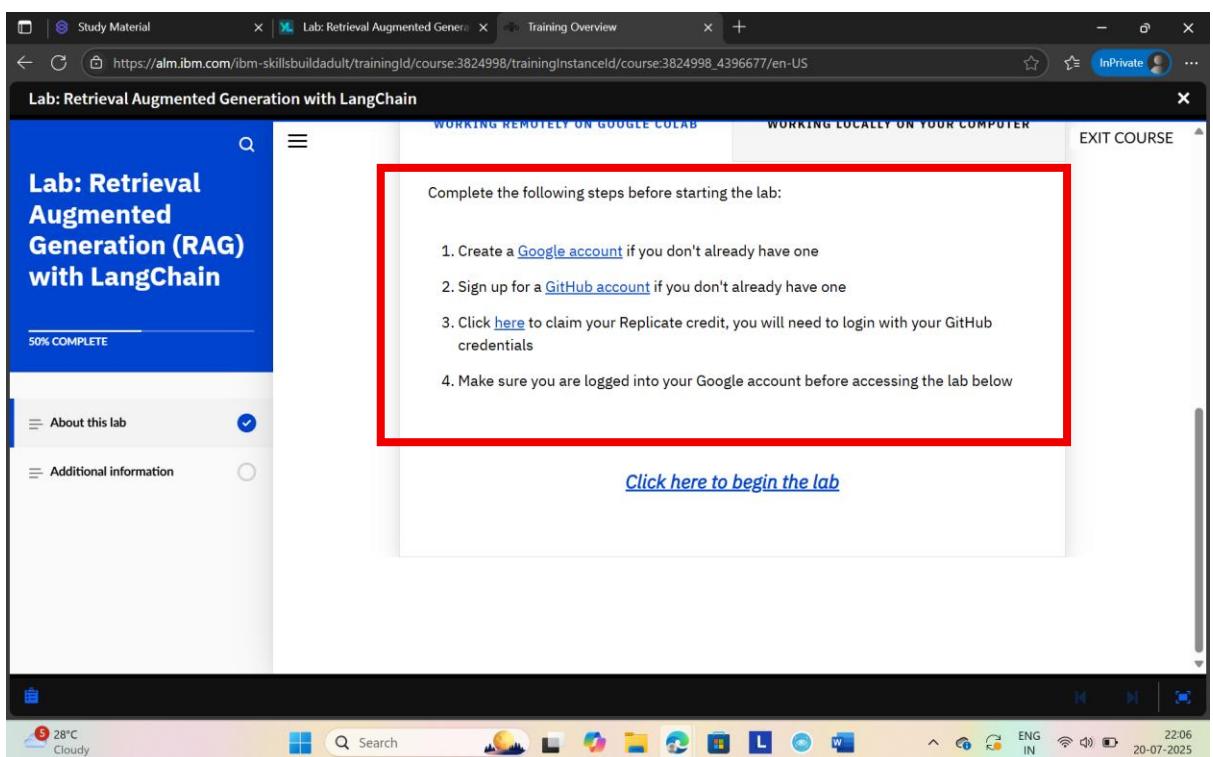
0% COMPLETE

About this lab

Introduction

Welcome to the *Retrieval Augmented Generation (RAG) with Langchain* lab! This notebook contains instructions for performing Retrieval Augmented Generation (RAG). RAG is an architectural pattern that can be used to augment the performance of language models by recalling factual information from a knowledge base, and adding that information to the model query. The most common approach in RAG is to create dense vector representations of the knowledge base in order to retrieve text chunks that are semantically similar to a given user query. In this lab, you will learn how to use the RAG chain to process a question.

16. The requirements are shown at the bottom of the page like this.



WORKING REMOTELY ON GOOGLE COLAB WORKING LOCALLY ON YOUR COMPUTER EXIT COURSE

Lab: Retrieval Augmented Generation (RAG) with LangChain

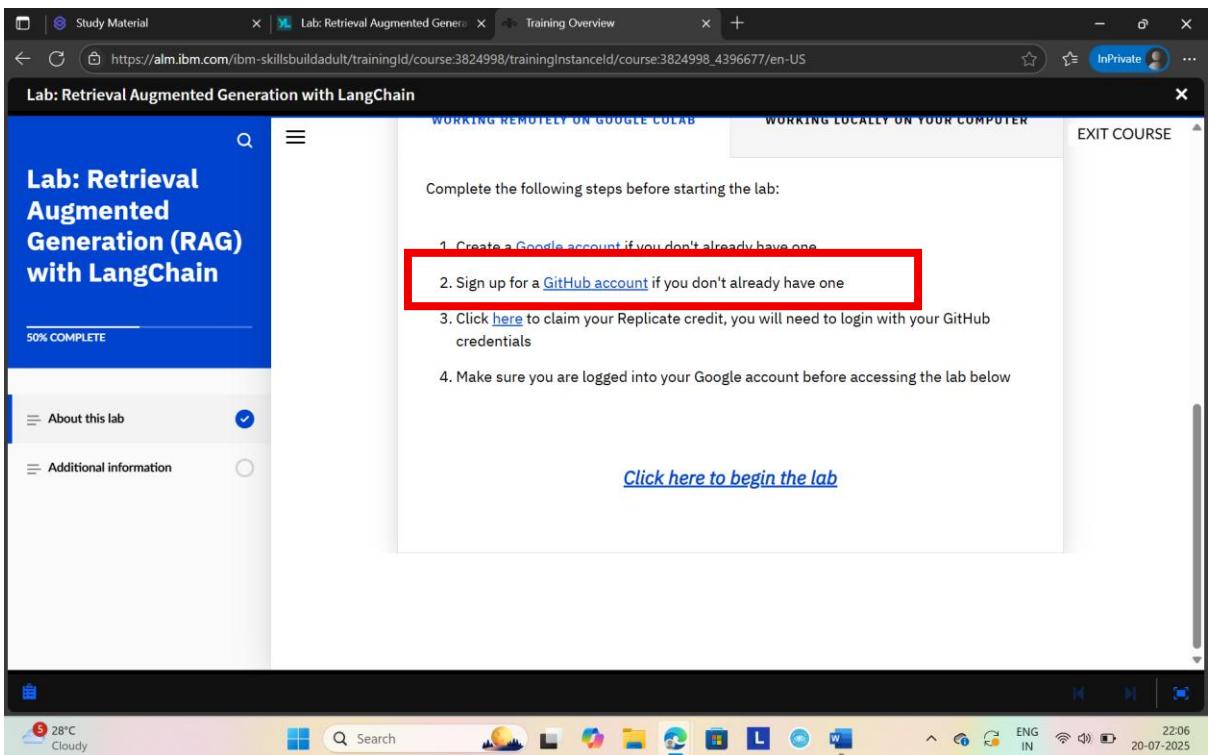
50% COMPLETE

Complete the following steps before starting the lab:

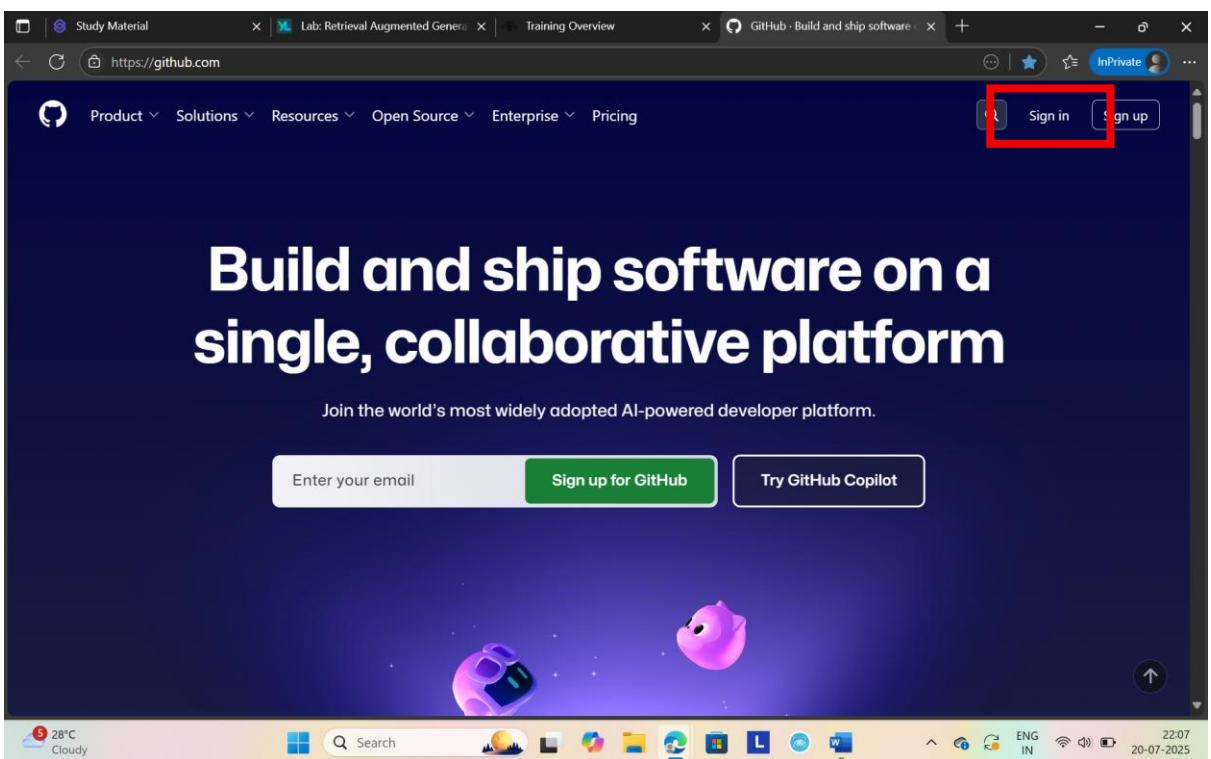
1. Create a [Google account](#) if you don't already have one
2. Sign up for a [GitHub account](#) if you don't already have one
3. Click [here](#) to claim your Replicate credit, you will need to login with your GitHub credentials
4. Make sure you are logged into your Google account before accessing the lab below

[Click here to begin the lab](#)

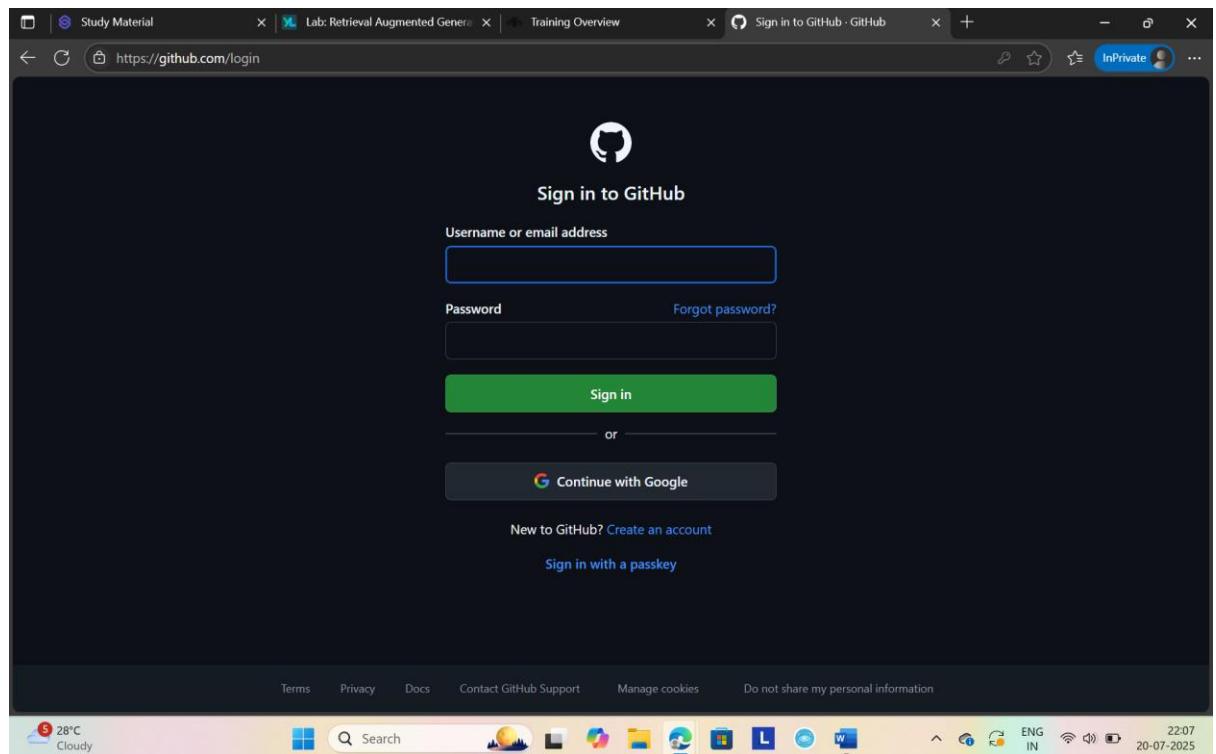
17. Since Gmail already created for you, click on Github account link.



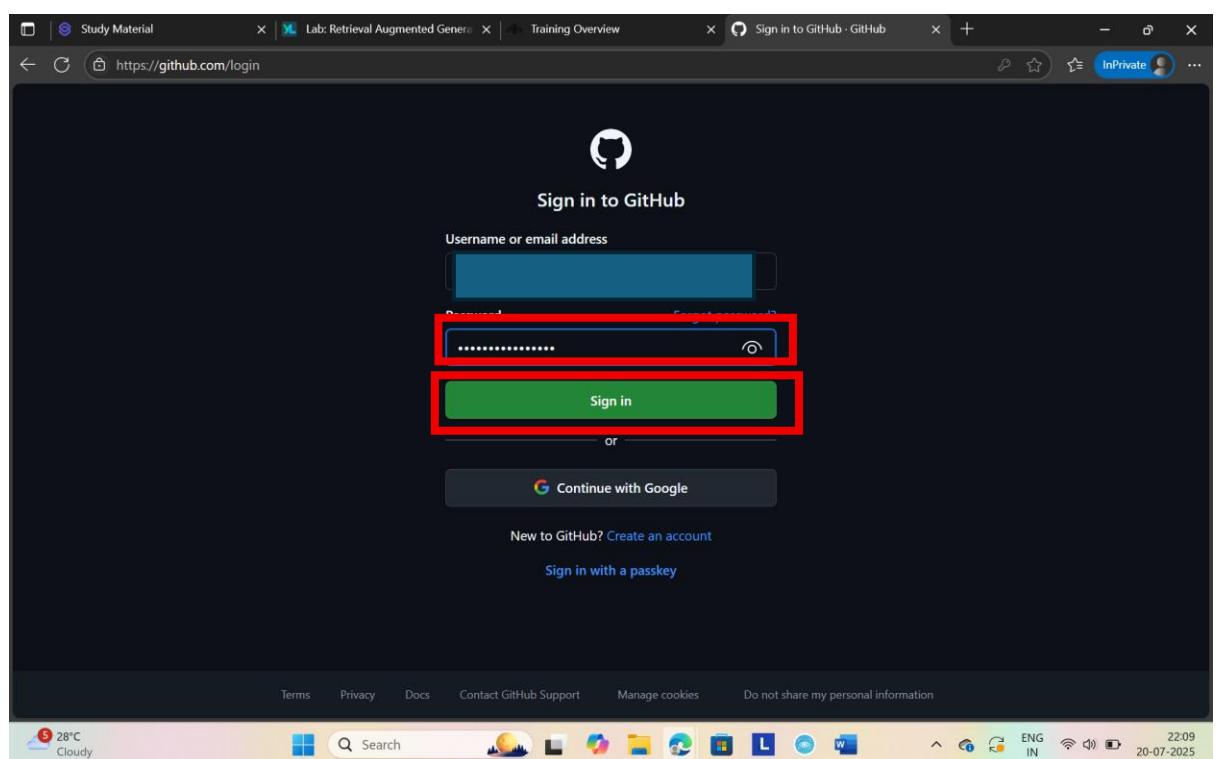
18. Sign in to Github account, if you have. Otherwise create a Github account on the same Gmail id and sign in. You can find the procedure for creating Github account with Gmail document, if you not yet created your Github account.



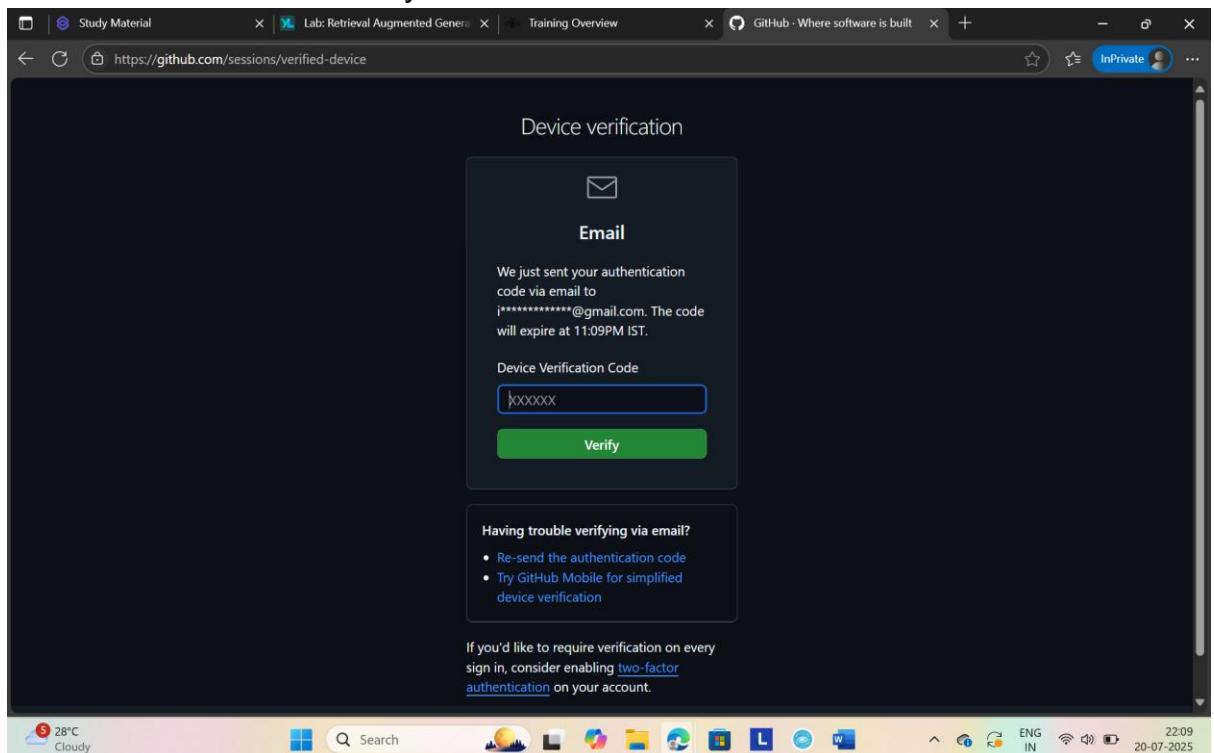
19. Provide your username for the Github account to sign in and hit enter.



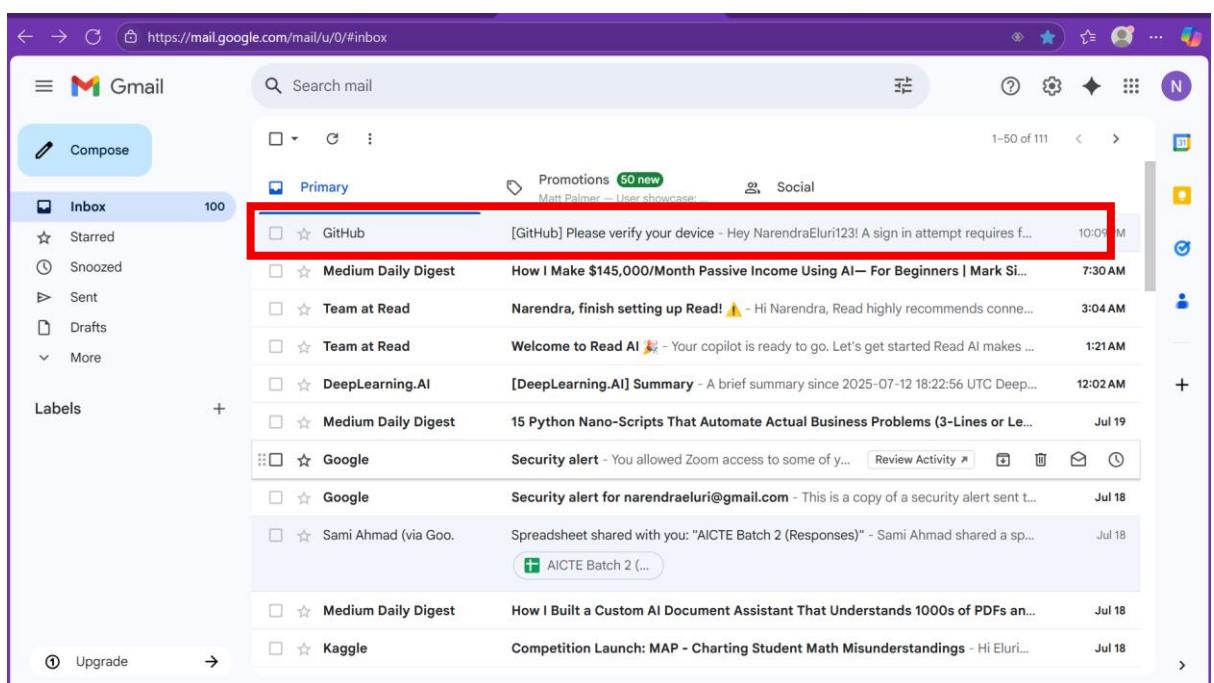
20. Provide password for the Github account and click on Sign in button.



21. It sends verification code to your Gmail.



22. Go to Your Gmail inbox and find a mail from Github.



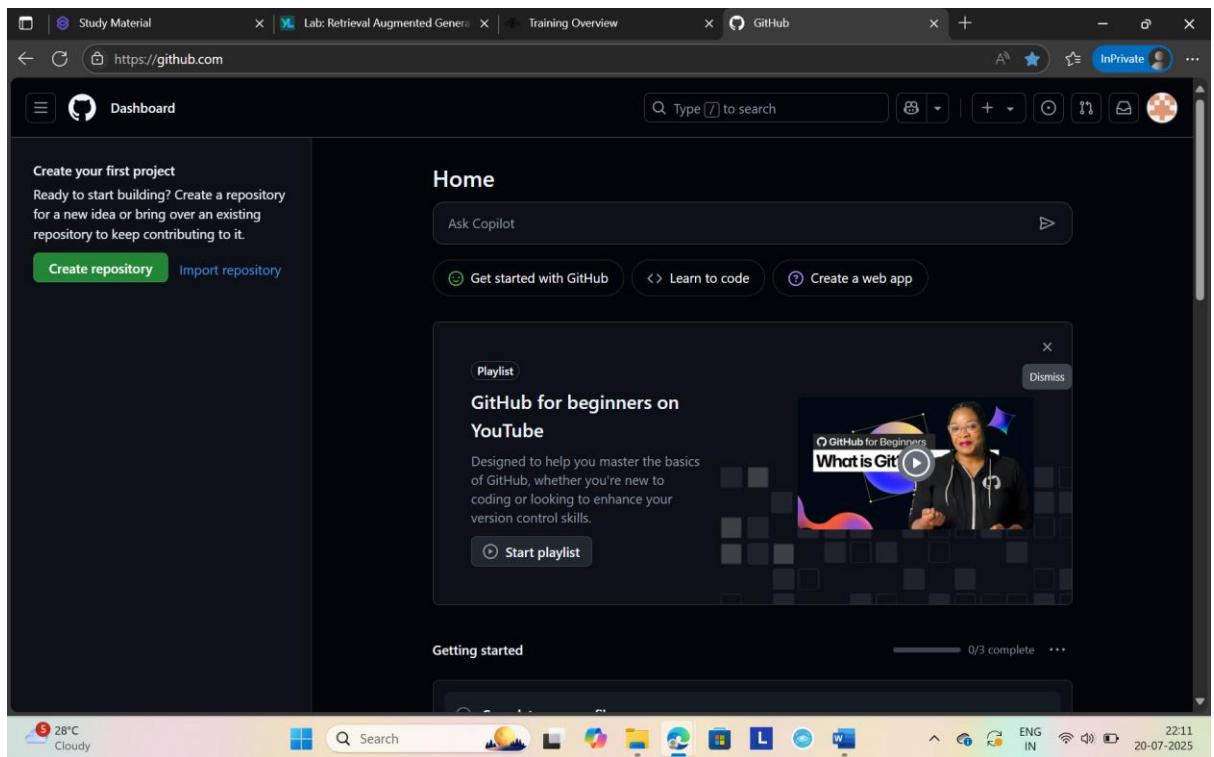
23. Open the mail received from Github and find the verification code. Copy the code.

The screenshot shows a Gmail inbox with 100 messages. An email from GitHub is selected, titled "[GitHub] Please verify your device". The message body contains a verification code: "Verification code: 610266". This line of text is highlighted with a red rectangle. Below it, there is a note about password security and links for two-factor authentication setup. The message ends with thanks from the GitHub Team.

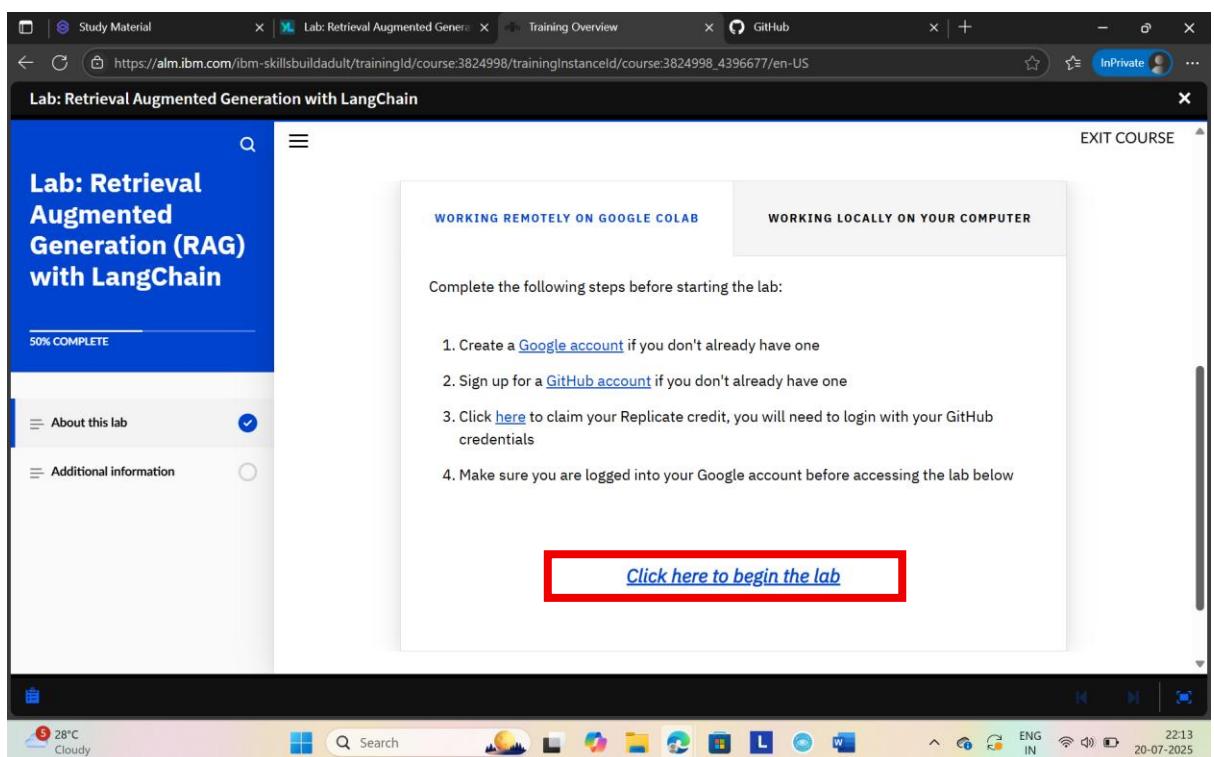
24. Return to the Github login page and paste the verification code in the text box and click on **Verify** button.

The screenshot shows a browser window with the URL <https://github.com/sessions/verified-device>. The page is titled "Device verification" and features an "Email" section. It informs the user that an authentication code was sent via email to their account. A "Device Verification Code" input field is present, with a red rectangle highlighting the "Verify" button below it. A link for "Having trouble verifying via email?" is shown, along with instructions for re-sending the code or using GitHub Mobile. At the bottom, there is a note about enabling two-factor authentication for all logins.

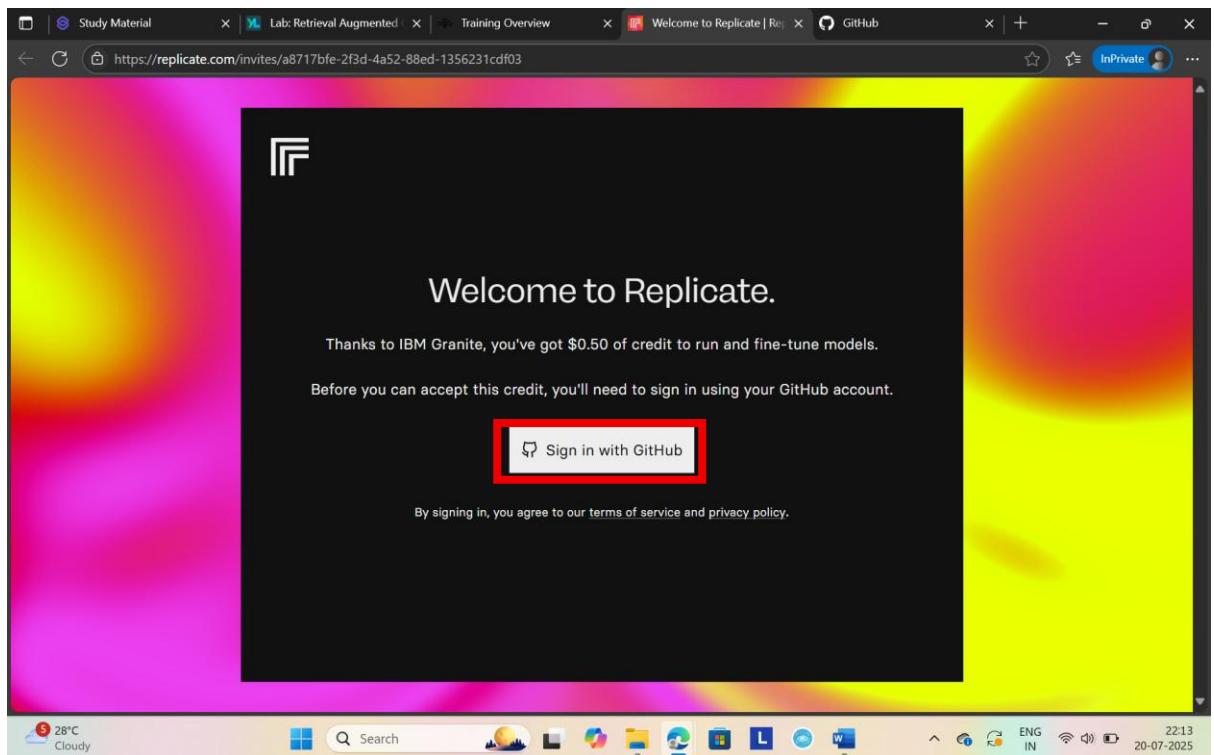
25. After successful verification of the code, you will be logged into your Github.



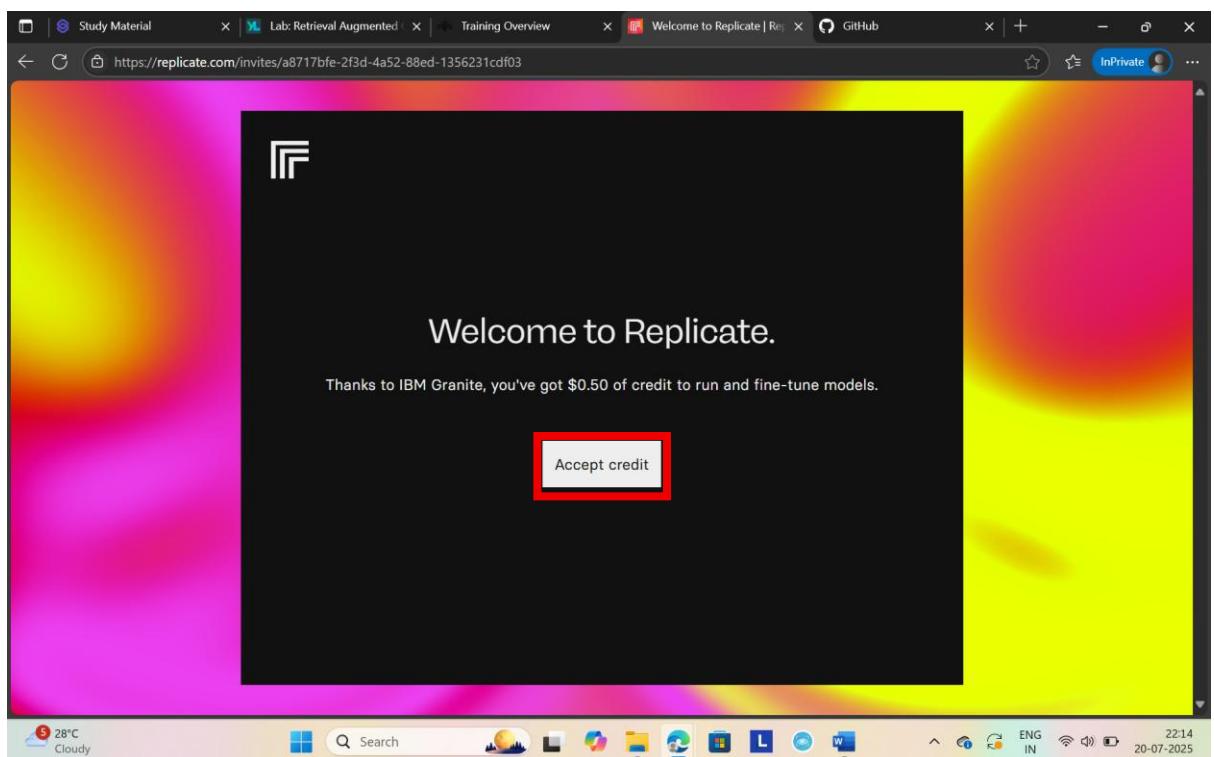
26. Keep your Github account open, go back to the Lab. Click on **Click here to begin the lab**.



27. It shows to login with Github. Click on **Sign in with GitHub** button.



28. Click on **Accept Credit** button to accept 0.5 dollar credit.



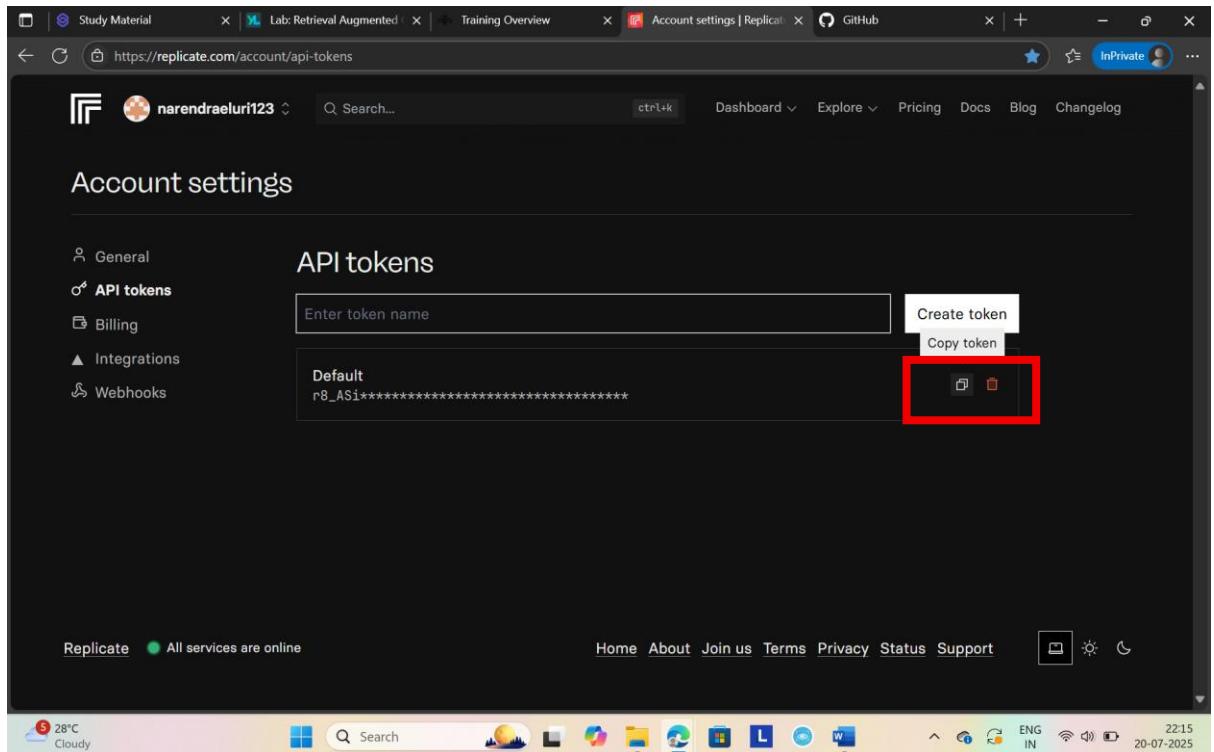
29. You can see this interface from where you need to get API Token.

The screenshot shows a dark-themed web browser window with multiple tabs open. The active tab is 'Dashboard' on the Replicate website at https://replicate.com. A success message 'Success! You've been granted a \$0.50 credit.' is displayed. Below it, there's a 'Welcome to Replicate!' section with a 'Create an account' button. To the right, there's a sidebar titled 'Try these models for free' featuring three cards: 'Imagen-4' by Google, 'flux-kontext-pro' by black-forest-labs, and 'flux-dev' by black-forest-labs. The bottom of the screen shows a Windows taskbar with various icons and system status.

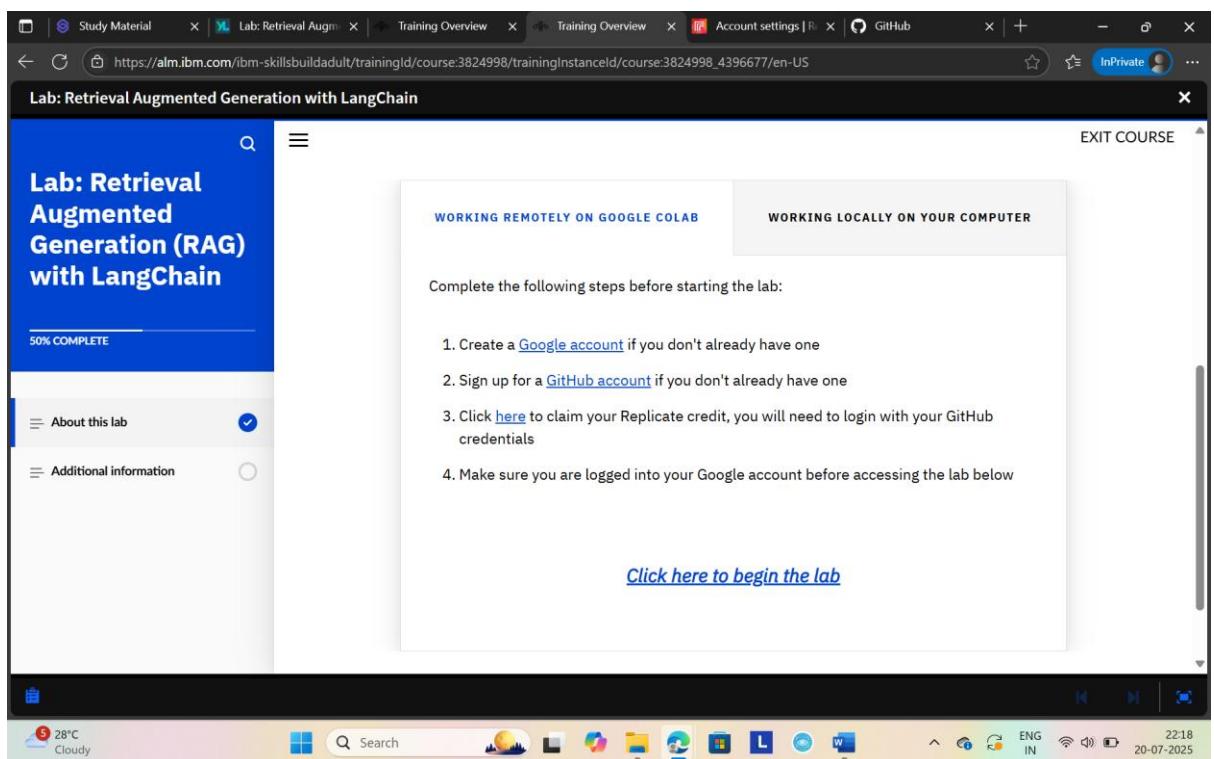
30. Click on Profile button on top left corner, select API tokens.

The screenshot shows the same Replicate.com dashboard as above, but with a red box highlighting the 'API tokens' button in the 'Account settings' dropdown menu. This indicates the step to click on the profile button and select API tokens. The rest of the interface and sidebar are identical to the previous screenshot.

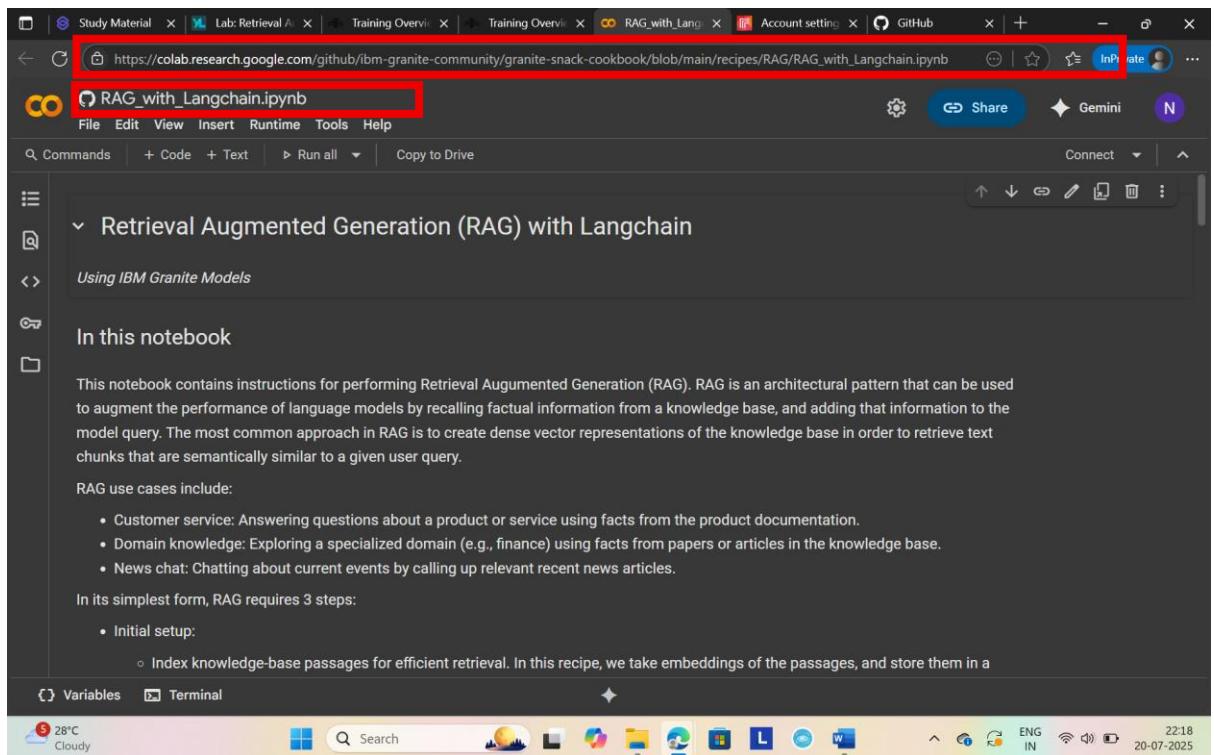
31. Copy the API token by clicking on copy icon. Save it in notepad for future reference.



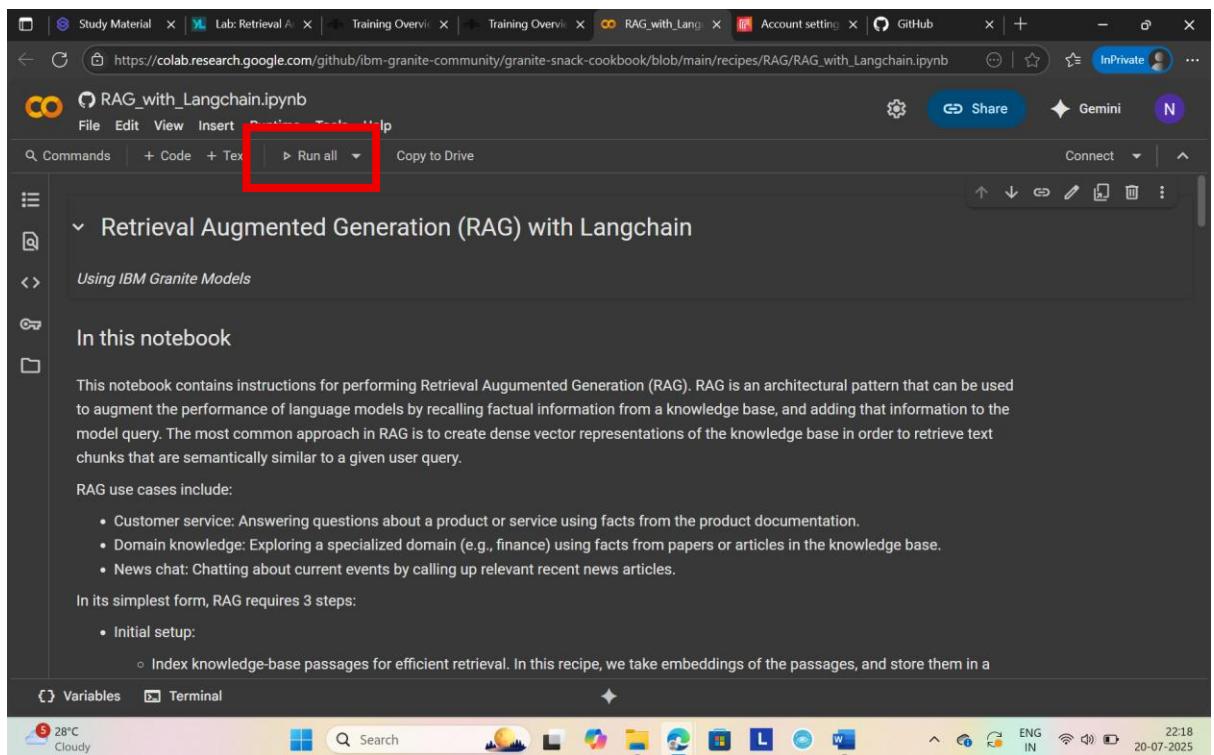
32. Come back to the Lab page, click on **Click here to begin the lab**.



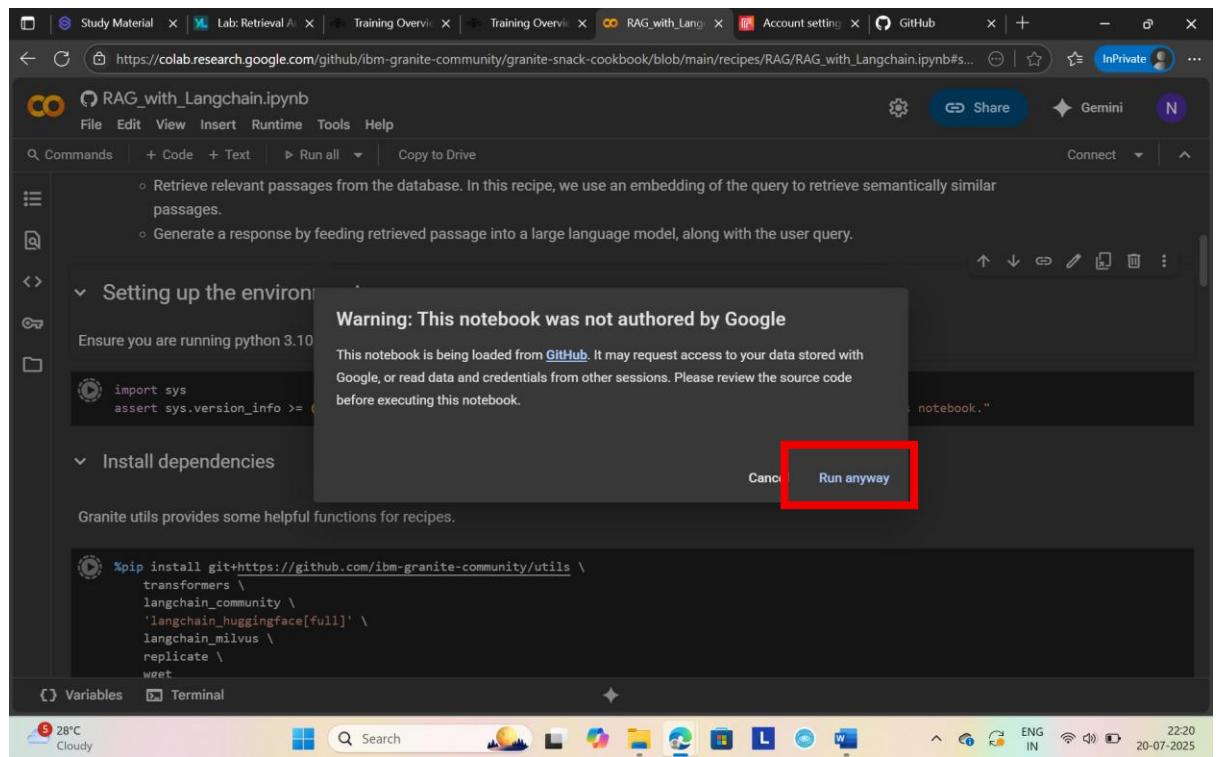
33. A Jupyter notebook with Python coding for the Lab will be opened in colab.



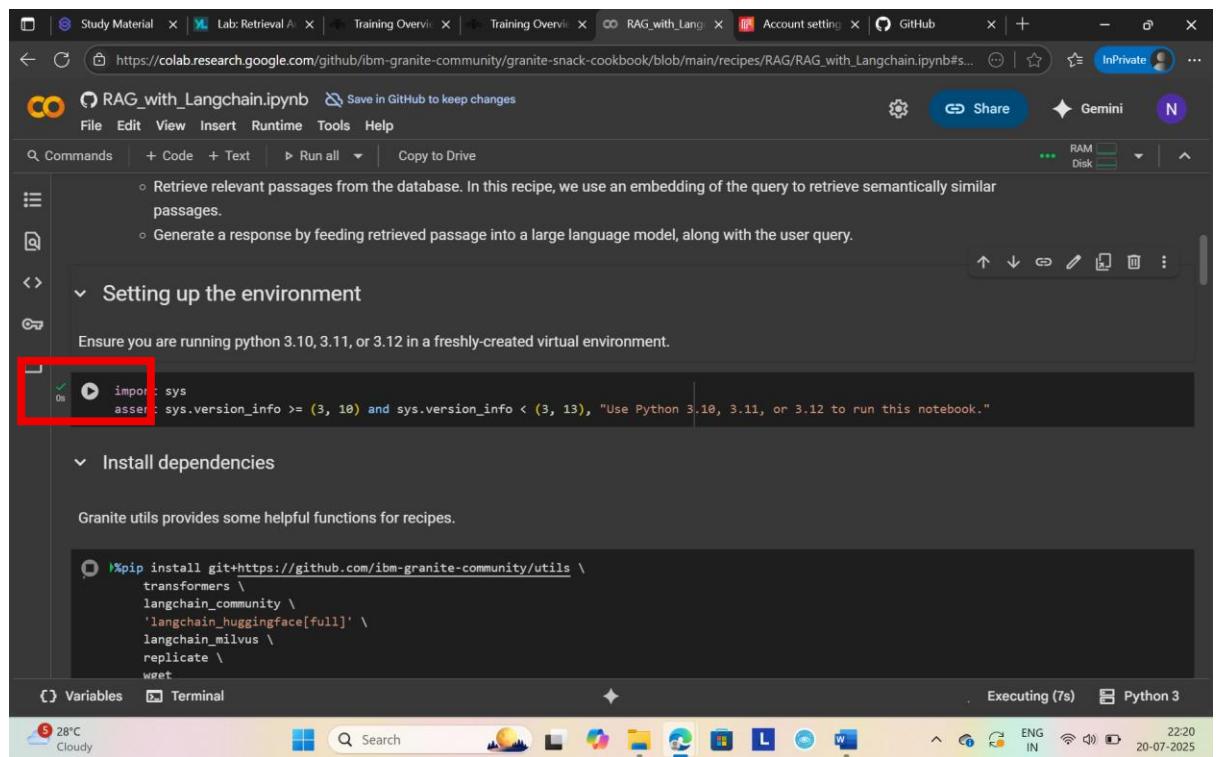
34. Find an option **Run all** and click on it.



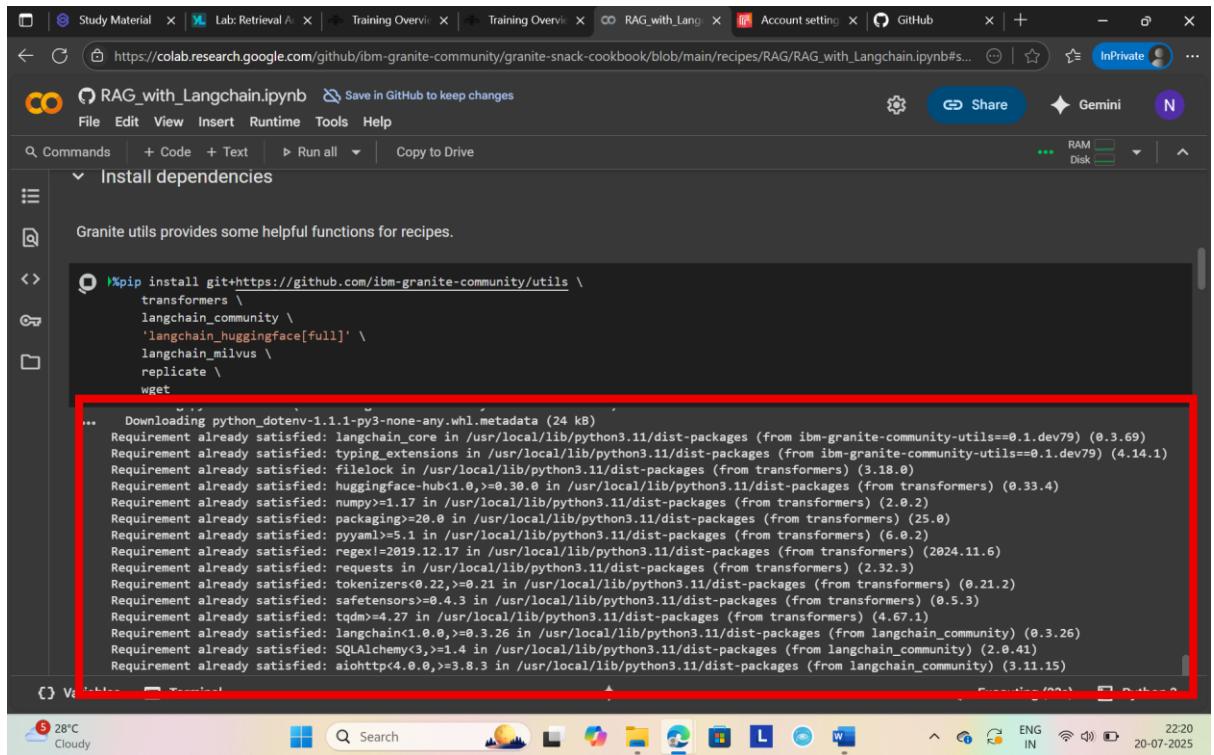
35. click on Run anyway



36. Your code will be start executing cell by cell.



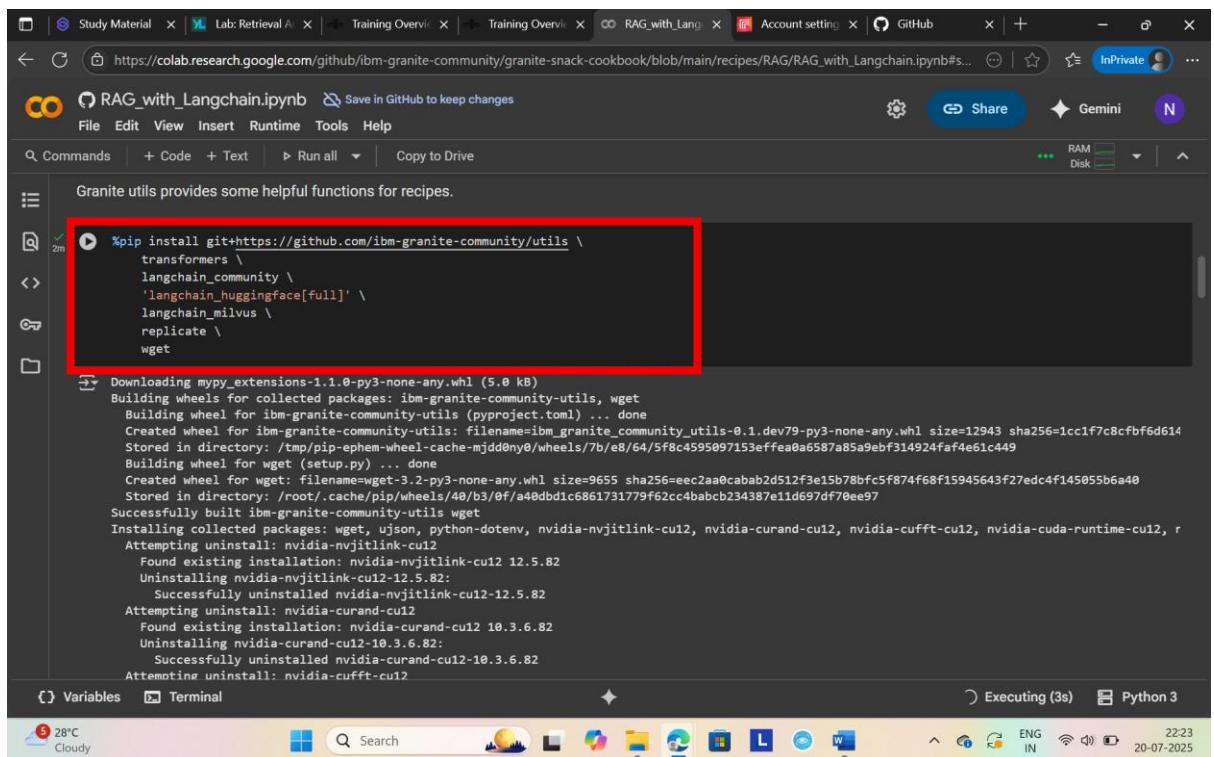
37. Your environment is setup, dependencies are installed. It will take 2-3min time.



```
%pip install git+https://github.com/ibm-granite-community/utils \
    transformers \
    langchain_community \
    'langchain_huggingface[full]' \
    langchain_milvus \
    replicate \
    wget
```

```
... Downloading python_dotenv-1.1.1-py3-none-any.whl.metadata (24 kB)
Requirement already satisfied: langchain_core in /usr/local/lib/python3.11/dist-packages (from ibm-granite-community-utils==0.1.dev79) (0.3.69)
Requirement already satisfied: typing_extensions in /usr/local/lib/python3.11/dist-packages (from ibm-granite-community-utils==0.1.dev79) (4.14.1)
Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-packages (from transformers) (3.18.0)
Requirement already satisfied: huggingface-hub<1.0,>=0.30.0 in /usr/local/lib/python3.11/dist-packages (from transformers) (0.33.4)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.11/dist-packages (from transformers) (2.0.2)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist-packages (from transformers) (25.0)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.11/dist-packages (from transformers) (6.0.2)
Requirement already satisfied: regex>=2019.12.17 in /usr/local/lib/python3.11/dist-packages (from transformers) (2024.11.6)
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-packages (from transformers) (2.32.3)
Requirement already satisfied: tokenizers<0.22,>=0.21 in /usr/local/lib/python3.11/dist-packages (from transformers) (0.21.2)
Requirement already satisfied: safetensors>=0.4.3 in /usr/local/lib/python3.11/dist-packages (from transformers) (0.5.3)
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.11/dist-packages (from transformers) (4.67.1)
Requirement already satisfied: langchain<1.0.0,>=0.3.26 in /usr/local/lib/python3.11/dist-packages (from langchain_community) (0.3.26)
Requirement already satisfied: SQLAlchemy<3,>=1.4 in /usr/local/lib/python3.11/dist-packages (from langchain_community) (2.0.41)
Requirement already satisfied: aiohttp<4.0.0,>=3.8.3 in /usr/local/lib/python3.11/dist-packages (from langchain_community) (3.11.15)
```

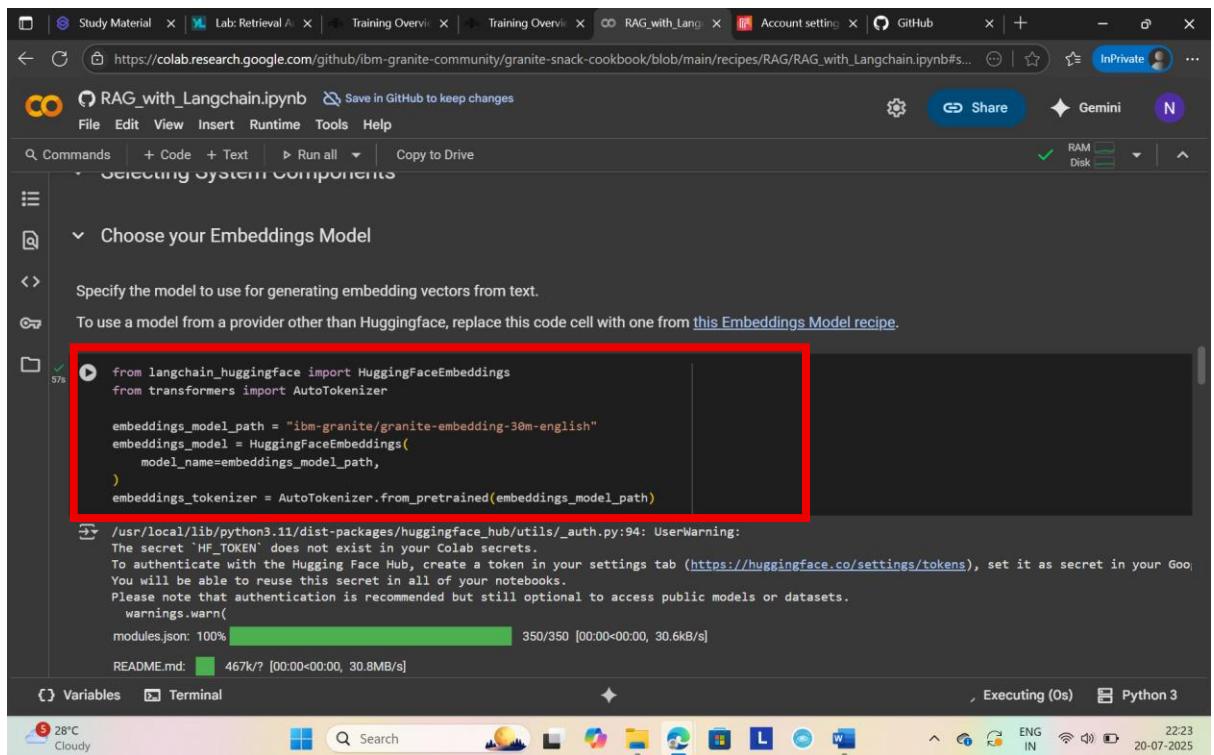
38. Executing cell by cell continues... installing all libraries, modules required for the lab activity.



```
%pip install git+https://github.com/ibm-granite-community/utils \
    transformers \
    langchain_community \
    'langchain_huggingface[full]' \
    langchain_milvus \
    replicate \
    wget
```

```
2m ⏪ Downloading mypy_extensions-1.1.0-py3-none-any.whl (5.0 kB)
Building wheels for collected packages: ibm-granite-community-utils, wget
  Building wheel for ibm-granite-community-utils (pyproject.toml) ... done
  Created wheel for ibm-granite-community-utils: filename=ibm_granite_community_utils-0.1.dev79-py3-none-any.whl size=12943 sha256=1cc1f7c8cfbf6d614
  Stored in directory: /tmp/pip-ephem-wheel-cache-mjdd0nye/wheels/7b/e8/64/5f8c4595097153effea0a6587a85a9ebf314924ffaef461c449
  Building wheel for wget (setup.py) ... done
  Created wheel for wget: filename=wget-3.2-py3-none-any.whl size=9655 sha256=eec2aa0cabab2d512f3e15b78bfc5f874f68f15945643f27edc4f145055b6a40
  Stored in directory: /root/.cache/pip/wheels/40/b3/0f/a40bd1c6861731779f62cc4babcb234387e11d697df70ee97
  Successfully built ibm-granite-community-utils wget
Installing collected packages: wget, ujson, python-dotenv, nvidia-nvjitlink-cu12, nvidia-curand-cu12, nvidia-cufft-cu12, nvidia-cuda-runtime-cu12, r
Attempting uninstall: nvidia-nvjitlink-cu12
  Found existing installation: nvidia-nvjitlink-cu12 12.5.82
  Uninstalling nvidia-nvjitlink-cu12-12.5.82:
    Successfully uninstalled nvidia-nvjitlink-cu12-12.5.82
Attempting uninstall: nvidia-curand-cu12
  Found existing installation: nvidia-curand-cu12 10.3.6.82
  Uninstalling nvidia-curand-cu12-10.3.6.82:
    Successfully uninstalled nvidia-curand-cu12-10.3.6.82
Attempting uninstall: nvidia-cufft-cu12
```

39. Here model for the lab will be chosen.



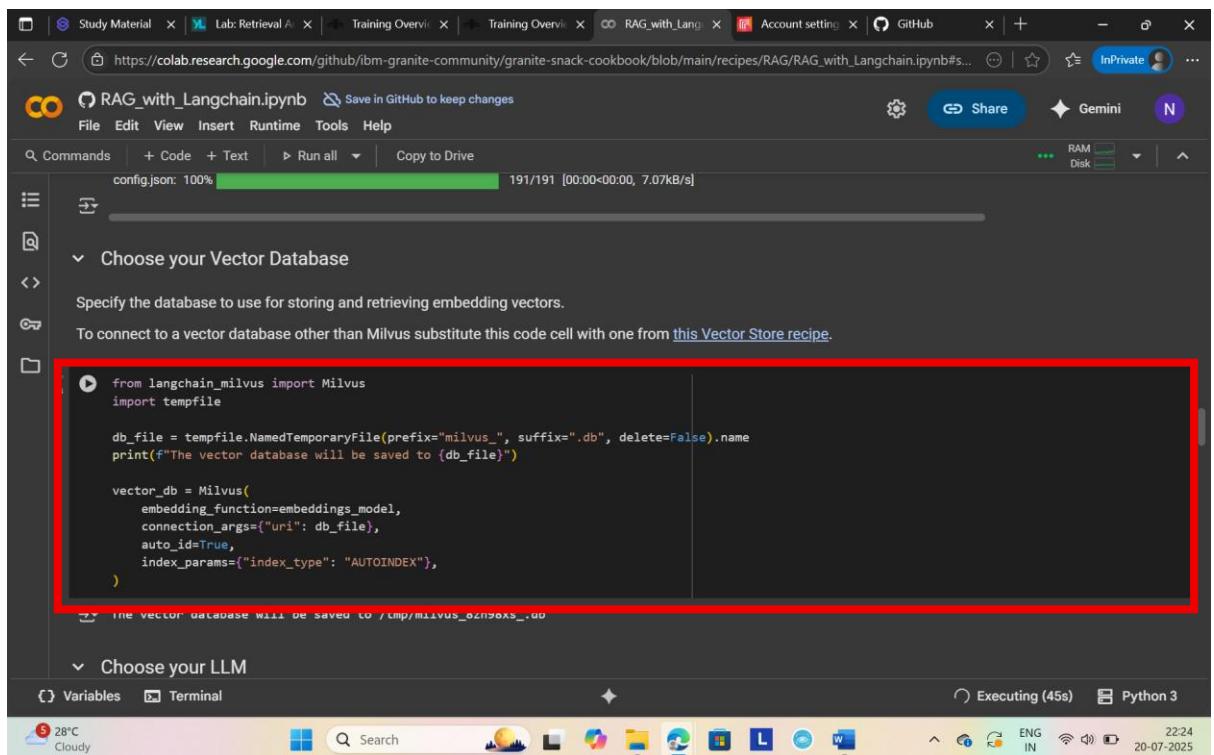
The screenshot shows a Google Colab notebook titled "RAG_with_Langchain.ipynb". The code cell under "Choose your Embeddings Model" is highlighted with a red box:

```
from langchain_huggingface import HuggingFaceEmbeddings
from transformers import AutoTokenizer

embeddings_model_path = "ibm-granite/granite-embedding-30m-english"
embeddings_model = HuggingFaceEmbeddings(
    model_name=embeddings_model_path,
)
embeddings_tokenizer = AutoTokenizer.from_pretrained(embeddings_model_path)
```

The code imports necessary libraries and defines an embeddings model and tokenizer using the specified path. A warning message about the absence of a token secret is visible below the code cell.

40. Vector Database to store text chunks will be created.



The screenshot shows a Google Colab notebook titled "RAG_with_Langchain.ipynb". The code cell under "Choose your Vector Database" is highlighted with a red box:

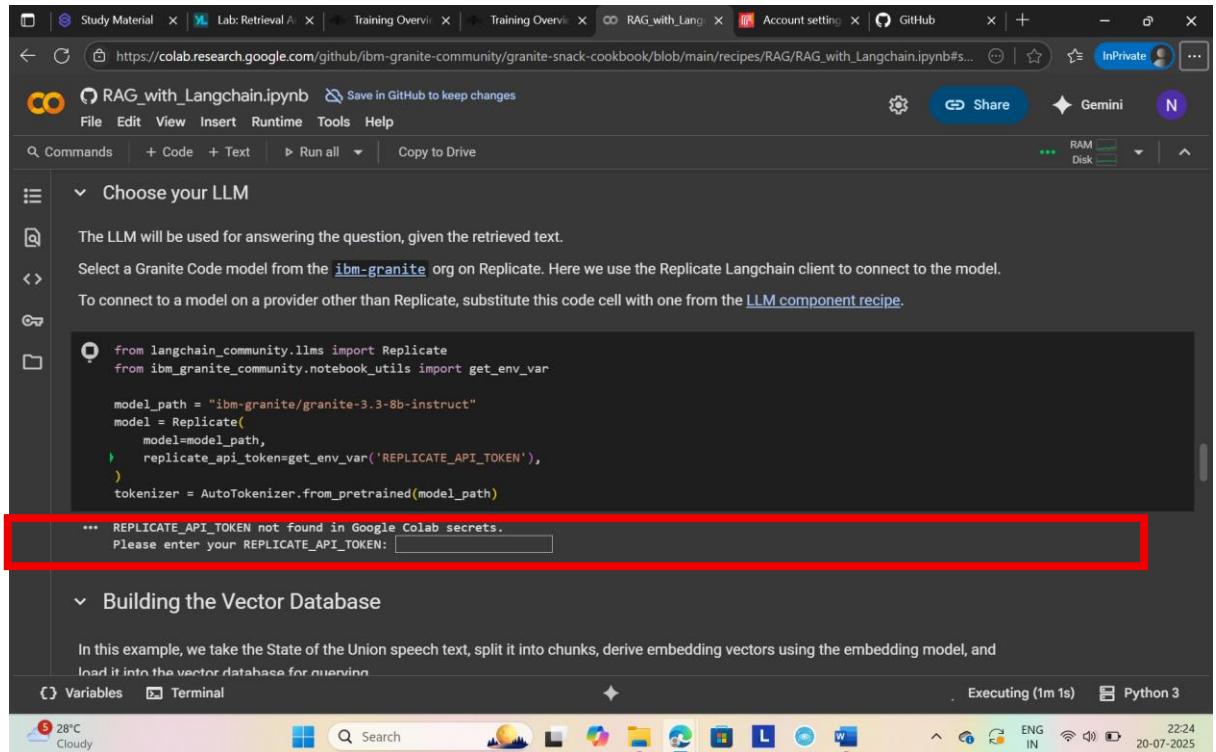
```
from langchain_milvus import Milvus
import tempfile

db_file = tempfile.NamedTemporaryFile(prefix="milvus_", suffix=".db", delete=False).name
print(f"The vector database will be saved to {db_file}")

vector_db = Milvus(
    embedding_function=embeddings_model,
    connection_args={"uri": db_file},
    auto_id=True,
    index_params={"index_type": "AUTOINDEX"},
)
```

The code imports the Milvus library and creates a temporary database file. It then initializes a Milvus vector database using the specified model and configuration.

41. This code trying to load the language model from **Replicate** using the LangChain and ibm_granite_community libraries. It requires Replicate API token. Since API token is available on colab, manually it is asking you to provie the token.



```
from langchain_community.llms import Replicate
from ibm_granite_community.notebook_utils import get_env_var

model_path = "ibm-granite/granite-3.3-8b-instruct"
model = Replicate(
    model=model_path,
    replicate_api_token=get_env_var('REPLICATE_API_TOKEN'),
)
tokenizer = AutoTokenizer.from_pretrained(model_path)

*** REPLICATE_API_TOKEN not found in Google Colab secrets.
Please enter your REPLICATE_API_TOKEN: [ ]
```

▼ Choose your LLM

The LLM will be used for answering the question, given the retrieved text.

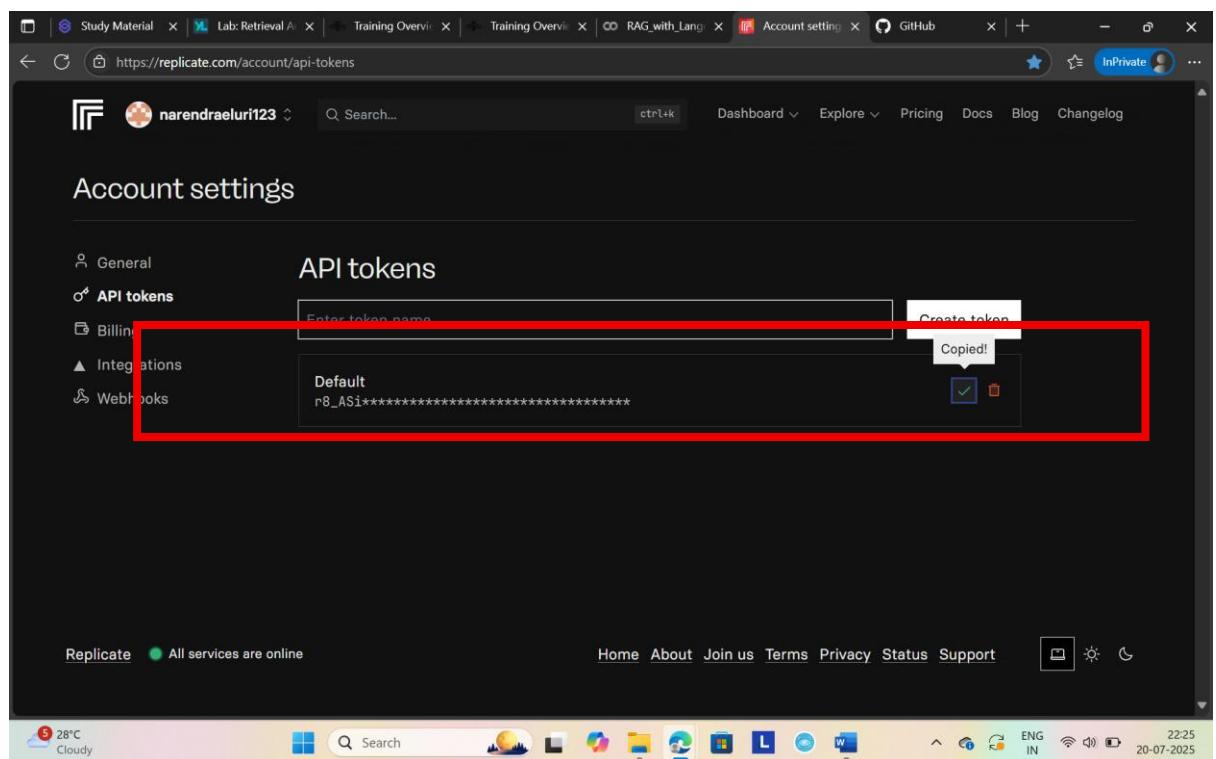
Select a Granite Code model from the [ibm-granite](#) org on Replicate. Here we use the Replicate Langchain client to connect to the model.

To connect to a model on a provider other than Replicate, substitute this code cell with one from the [LLM component recipe](#).

▼ Building the Vector Database

In this example, we take the State of the Union speech text, split it into chunks, derive embedding vectors using the embedding model, and load it into the vector database for querying.

42. You can visit replicate page, copy the token.(or copy from notepad also)



43. Replicate API token is taken, and starts creating the model using Langchain and IBM Granite community models.

The screenshot shows a Google Colab notebook titled "RAG_with_Langchain.ipynb". The code cell contains Python code for initializing a Replicate model from the "ibm-granite-community" org. A red box highlights the code block. An error message at the bottom of the cell reads: "REPLICATE_API_TOKEN not found in Google Colab secrets. Please enter your REPLICATE_API_TOKEN:". The status bar at the bottom right indicates "Executing (2m 47s)" and "Python 3".

```
from langchain_community.llms import Replicate
from ibm_granite_community.notebook_utils import get_env_var

model_path = "ibm-granite/granite-3.3-8b-instruct"
model = Replicate(
    model=model_path,
    replicate_api_token=get_env_var('REPLICATE_API_TOKEN'),
)
tokenizer = AutoTokenizer.from_pretrained(model_path)

*** REPLICATE_API_TOKEN not found in Google Colab secrets.
Please enter your REPLICATE_API_TOKEN: .....
```

44. It starts creating model

The screenshot shows the same Google Colab notebook. The code cell now has a green checkmark icon and a "3m" duration indicator. The error message is still present. The status bar at the bottom right indicates "Executing (2s)" and "Python 3".

```
from langchain_community.llms import Replicate
from ibm_granite_community.notebook_utils import get_env_var

model_path = "ibm-granite/granite-3.3-8b-instruct"
model = Replicate(
    model=model_path,
    replicate_api_token=get_env_var('REPLICATE_API_TOKEN'),
)
tokenizer = AutoTokenizer.from_pretrained(model_path)

REPLICATE_API_TOKEN not found in Google Colab secrets.
Please enter your REPLICATE_API_TOKEN: .....
```

45. Starts creating vector databases

The screenshot shows a Google Colab notebook titled "RAG_with_Langchain.ipynb". The code cell [6] contains Python code to download a file from GitHub and then split it into chunks using LangChain's CharacterTextSplitter.

```
[6] import os  
import wget  
  
filename = 'state_of_the_union.txt'  
url = 'https://raw.githubusercontent.com/IBM/watson-machine-learning-samples/master/cloud/data/foundation_models/state_of_the_union.txt'  
  
if not os.path.isfile(filename):  
    wget.download(url, out=filename)  
  
[6]: wget.download(url, out=filename)
```

The notebook also includes sections for "Building the Vector Database" and "Download the document", with a note about using President Biden's State of the Union address from March 1, 2022.

At the bottom, the status bar shows "Variables" and "Terminal", the date "20-07-2025", and the time "22:32".

46. Splitting the text document into small chunks

The screenshot shows a Google Colab notebook titled "RAG_with_Langchain.ipynb". The code cell [6] contains Python code to download a file from GitHub and then split it into chunks using LangChain's CharacterTextSplitter.

```
[6] wget.download(url, out=filename)  
  
[6]: wget.download(url, out=filename)  
  
[7] from langchain.document_loaders import TextLoader  
from langchain.text_splitter import CharacterTextSplitter  
  
loader = TextLoader(filename)  
documents = loader.load()  
text_splitter = CharacterTextSplitter.from_huggingface_tokenizer(  
    tokenizer=embeddings_tokenizer,  
    chunk_size=embeddings_tokenizer.max_len_single_sentence,  
    chunk_overlap=0,  
)  
texts = text_splitter.split_documents(documents)  
doc_id = 0  
for text in texts:  
    text.metadata["doc_id"] = (doc_id:=doc_id+1)  
    print(f"{len(texts)} text document chunks created")  
  
[7]: 19 text document chunks created
```

The notebook also includes sections for "Split the document into chunks" and "Populate the vector database".

At the bottom, the status bar shows "Variables" and "Terminal", the date "20-07-2025", and the time "22:27".

47. Populating the vector databases with the chunks

The screenshot shows a Google Colab notebook titled "RAG_with_Langchain.ipynb". The code cell [7] contains:for text in texts:
 text.metadata["doc_id"] = (doc_id:=doc_id+1)
 print(f"{len(texts)} text document chunks created")

The output shows: 19 text document chunks created.

The code cell [8] contains:ids = vector_db.add_documents(texts)
print(f"{len(ids)} documents added to the vector database")

The output shows: 19 documents added to the vector database.

The notebook structure includes sections like "Populate the vector database" and "Querying the Vector Database". A note states: "NOTE: Population of the vector database may take over a minute depending on your embedding model and service."

At the bottom, the status bar shows: 10:27PM Python 3. The system tray shows: 28°C Cloudy, ENG IN, 22-27, 20-07-2025.

48. Performing similarity search basing on the input query will be done.

The screenshot shows a Google Colab notebook titled "RAG_with_Langchain.ipynb". The code cell [9] contains:query = "What did the president say about Ketanji Brown Jackson?"
docs = vector_db.similarity_search(query)

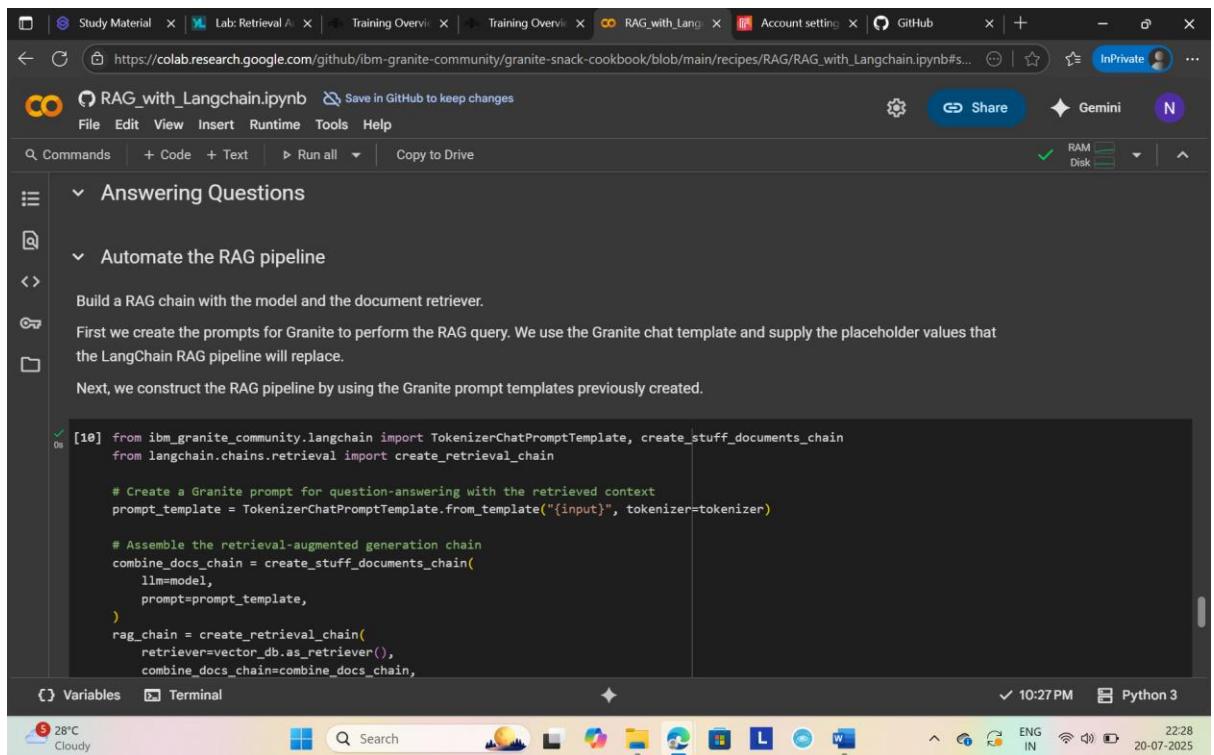
The output shows: 19 documents returned. The first few results are:

```
And the costs and the threats to America and the world keep rising.  
That's why the NATO Alliance was created to secure peace and stability in Europe after World War 2.  
The United States is a member along with 29 other nations.  
It matters. American diplomacy matters. American resolve matters.  
Putin's latest attack on Ukraine was premeditated and unprovoked.  
He rejected repeated efforts at diplomacy.
```

The notebook structure includes sections like "Querying the Vector Database" and "Conduct a similarity search". A note states: "Search the database for similar documents by proximity of the embedded vector in vector space."

At the bottom, the status bar shows: 10:27PM Python 3. The system tray shows: 28°C Cloudy, ENG IN, 22-28, 20-07-2025.

49. Prompt augmentation will be done.



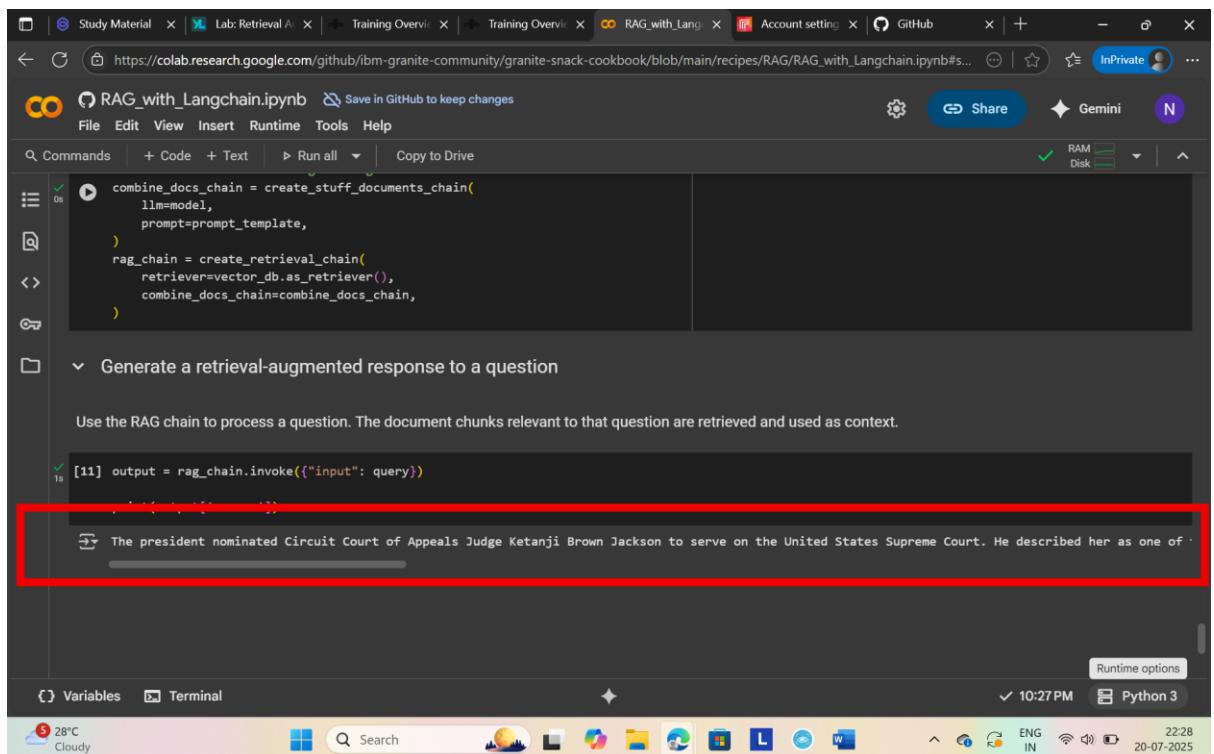
The screenshot shows a Google Colab notebook titled "RAG_with_Langchain.ipynb". The code cell [10] contains the following Python code:

```
[10] from ibm_granite_community.langchain import TokenizerChatPromptTemplate, create_stuff_documents_chain
      from langchain.chains.retrieval import create_retrieval_chain

      # Create a Granite prompt for question-answering with the retrieved context
      prompt_template = TokenizerChatPromptTemplate.from_template("{input}", tokenizer=tokenizer)

      # Assemble the retrieval-augmented generation chain
      combine_docs_chain = create_stuff_documents_chain(
          llm=model,
          prompt=prompt_template,
      )
      rag_chain = create_retrieval_chain(
          retriever=vector_db.as_retriever(),
          combine_docs_chain=combine_docs_chain,
      )
```

50. The model will take the refined prompt and generates best suited answer from its knowledge as output.



The screenshot shows the same Google Colab notebook. The code cell [11] contains the command:

```
[11] output = rag_chain.invoke({"input": query})
```

The output of the cell is a text response:

```
The president nominated Circuit Court of Appeals Judge Ketanji Brown Jackson to serve on the United States Supreme Court. He described her as one of .
```

A red rectangle highlights the output text.

Congratulations, you have successfully executed RAG lab with Langchain.