



TEAM MEMBERS:

Mythili.K s

Manjusri.A

Mathu sree.T

Narmadha.P

CONTENTS

Project Description	4
Objectives:	4
Pre-Requisites:	4
Project workflow:	5
Activity-1: Exploring Naan Mudhalvan Smart Interz Portal.	5
Activity-2: Choose a IBM Granite model From Hugging Face.	9
Activity-3: Running Application in Google Collab.	11
Activity-4: Upload Your Project in GitHub.	17

SmartSDLC – AI-Enhanced Software Development Lifecycle



Project Description

The SmartSDLC project is an AI-enhanced platform designed to automate and streamline the Software Development Lifecycle (SDLC) using advanced technologies like IBM Watsonx, FastAPI, LangChain, and Streamlit. It integrates generative AI to handle key SDLC phases, including requirement analysis, code generation, test case creation, bug fixing, and documentation. The platform features a user-friendly interface that allows users to upload PDFs, generate structured requirements, and transform natural language prompts into functional code. It also includes an AI-powered chatbot for real-time assistance and support. The backend, built with FastAPI, efficiently processes API requests, while the frontend, developed using Streamlit, offers a visually appealing and interactive dashboard. The system is modular, scalable, and secure, featuring a robust authentication mechanism and seamless integration between AI models and user inputs.

Objectives:

- Automatically generate and run test cases.
- Identify defects in code through AI-powered code reviews.
- Predict possible bugs before deployment.
- Ensure continuous monitoring after release to catch issues quickly.
- Adapt quickly to changing requirements
- Ensure compliance and security checks automatically
- Auto-generate documentation and test cases

Pre-Requisites:

1. Gradio Framework Knowledge: [Gradio Documentation](#)
2. IBM Granite Models (Hugging Face): [IBM Granite models](#)

3. Python Programming Proficiency: [Python Documentation](#)
4. Version Control with Git: [Git Documentation](#)
5. Google Collab's T4 GPU Knowledge: [Google collab](#)

Project workflow:

Activity-1: Exploring Naan Mudhalvan Smart Interz Portal.

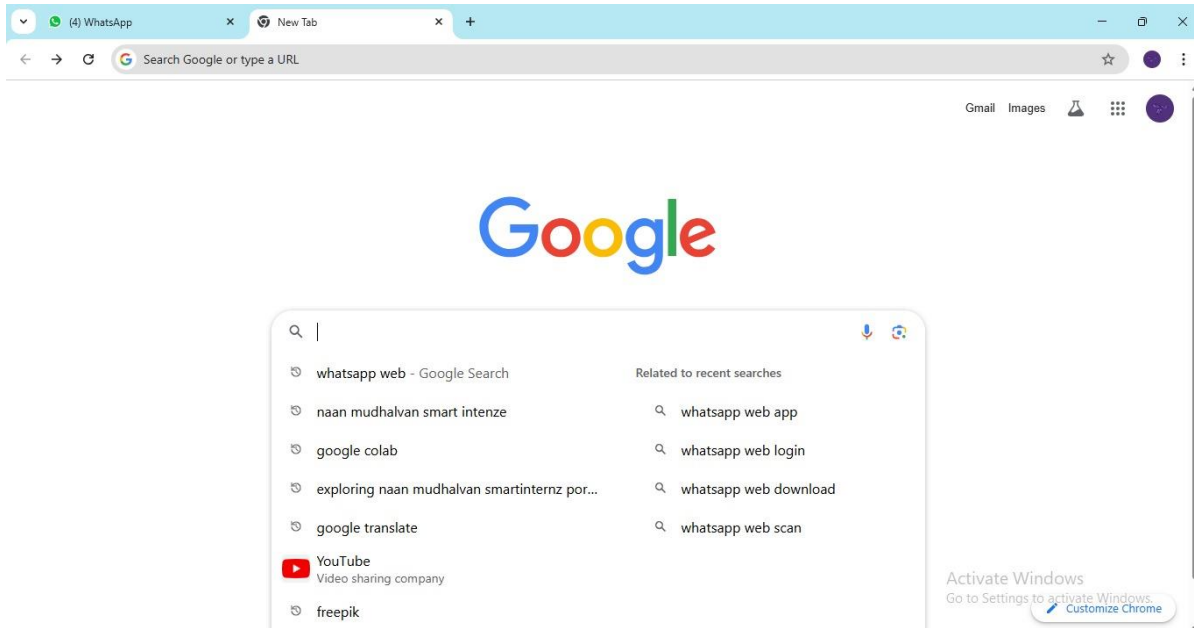
Activity-2: Choosing a IBM Granite Model From Hugging Face.

Activity-3: Running Application in Google Colab.

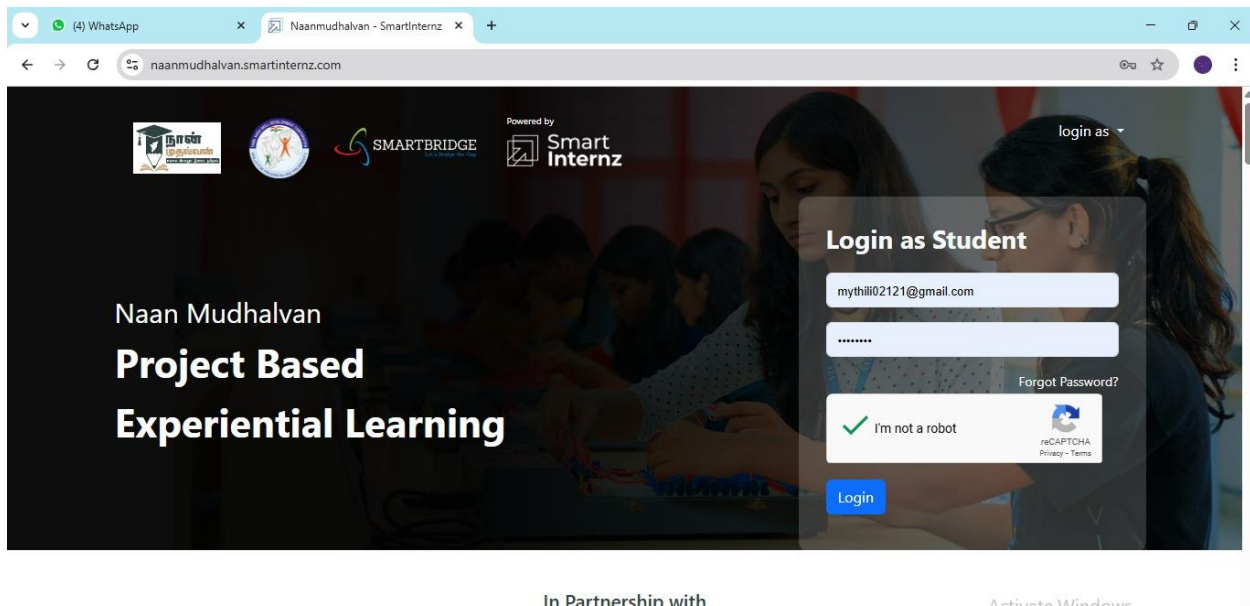
Activity-4: Upload your Project in GitHub

Activity-1: Exploring Naan Mudhalvan Smart Interz Portal.

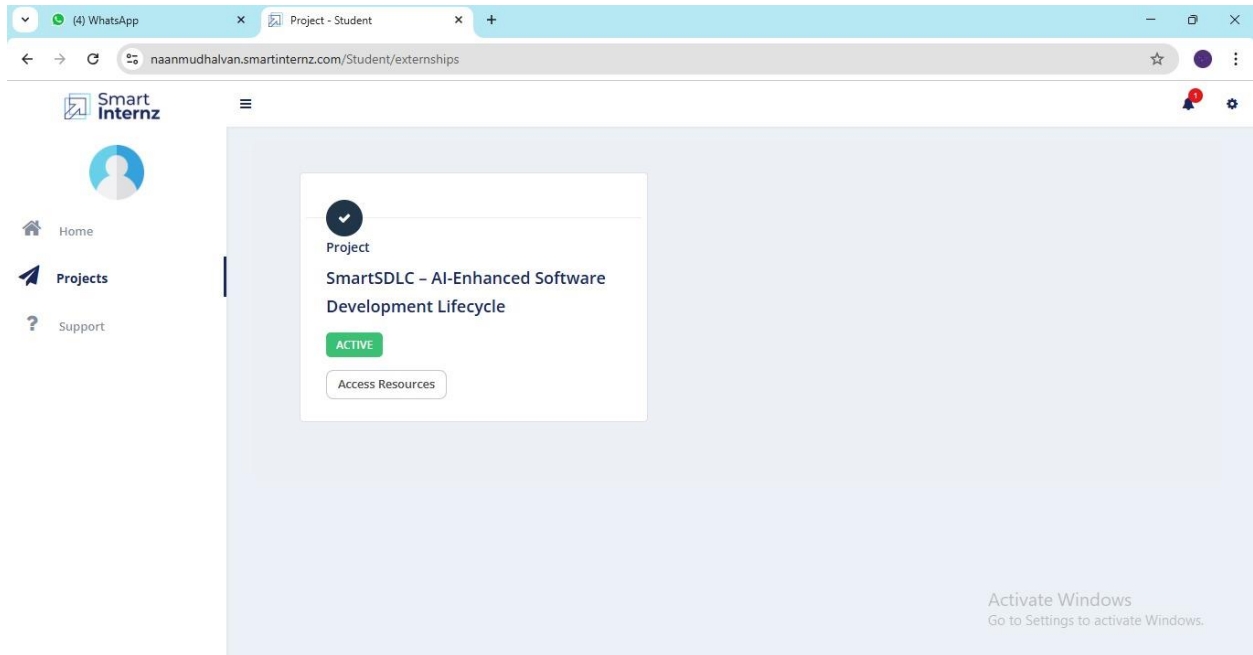
- Search for “Naan Mudhalvan Smart Interz” Portal in any Browser



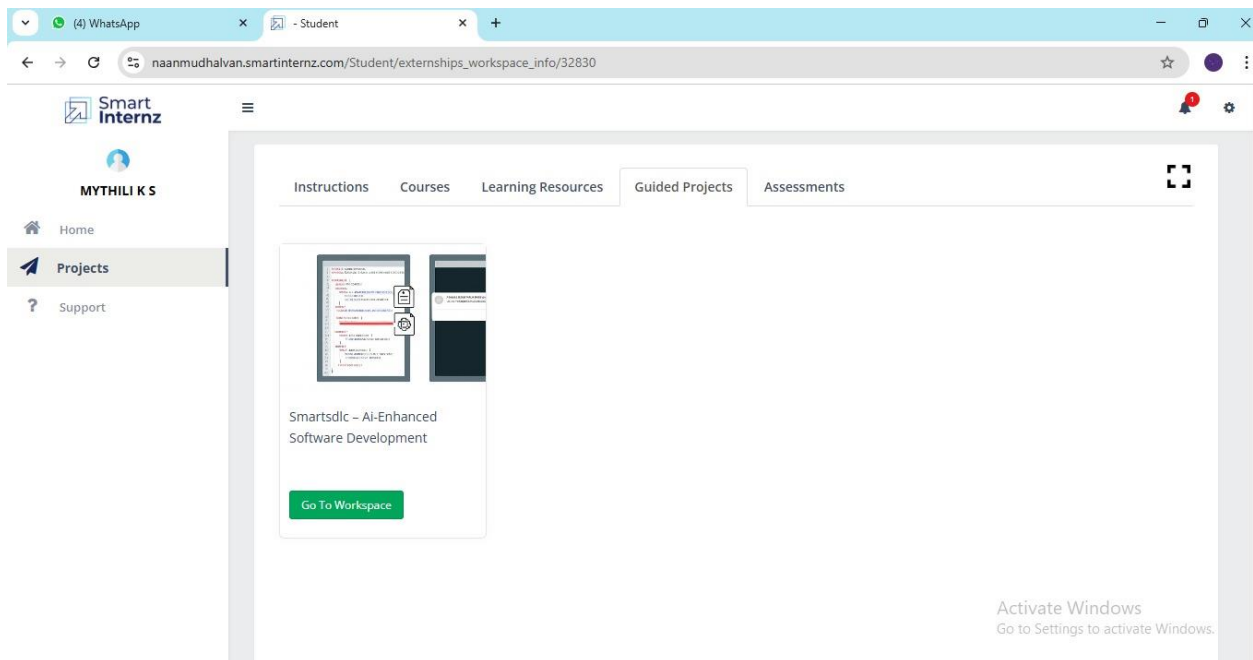
- Then Click on the first link. (Naanmudhalvan Smartinternz) Then login with your details



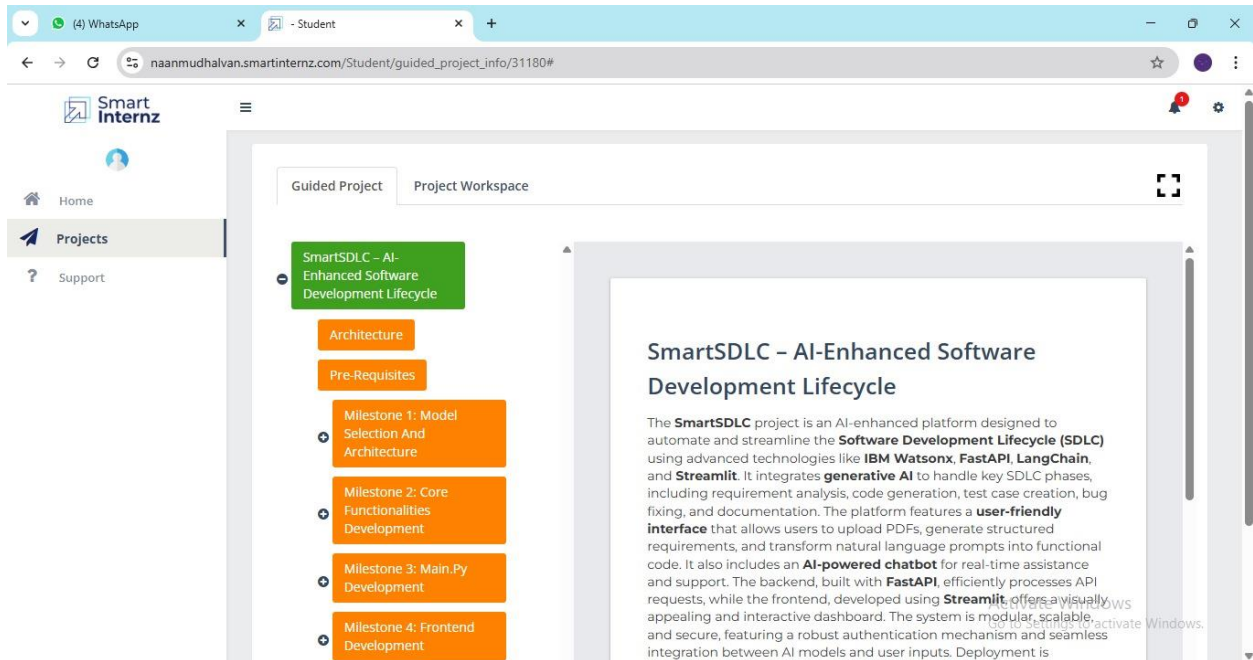
- Then you will be redirected to your account then click on “Projects” Section. There you can see which project you have enrolled in here it is “SmartSDLC – AI-Enhanced Software Development Lifecycle”.



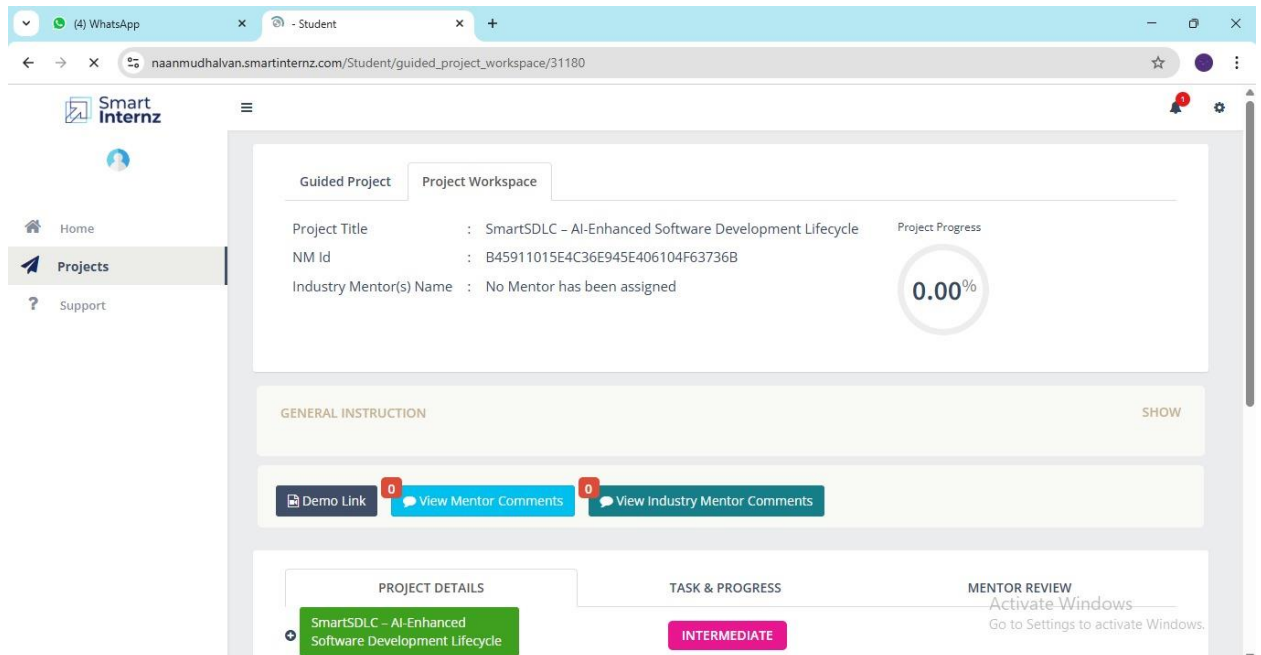
➤ Then click on “Access Resources” and go to the “Guided Project” Section



➤ Click on the “Go to workspace” section. Then you can find the detailed 3 explanation of Generative AI Project using IBM WatsonX API key



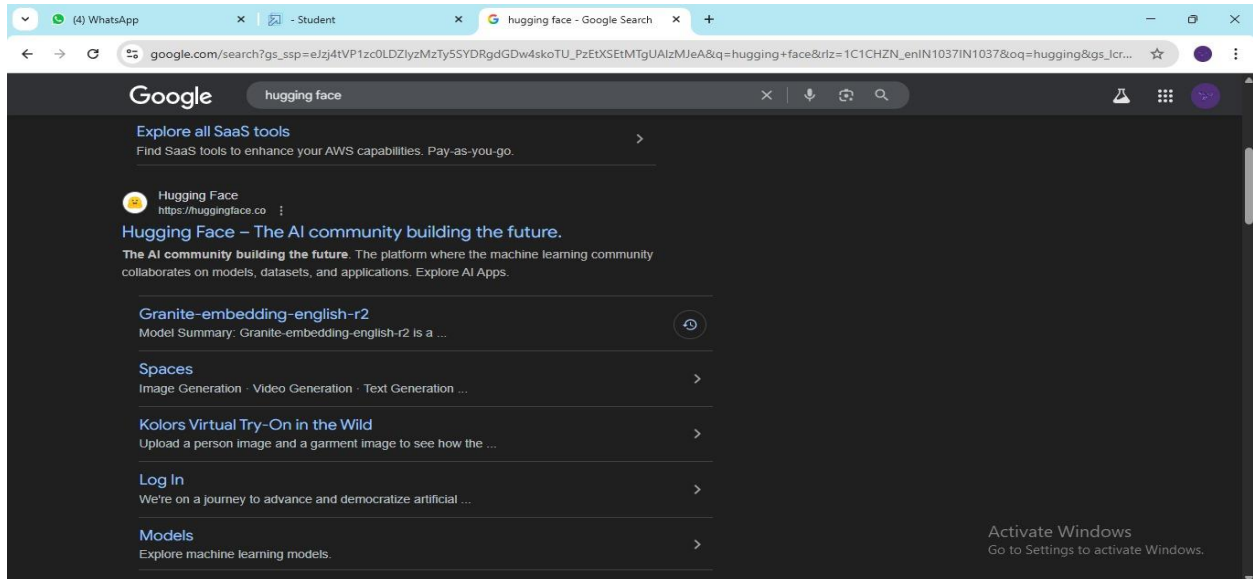
- Click on “Project Workspace”, there you can find your project progress and Place to upload “Demo link”.



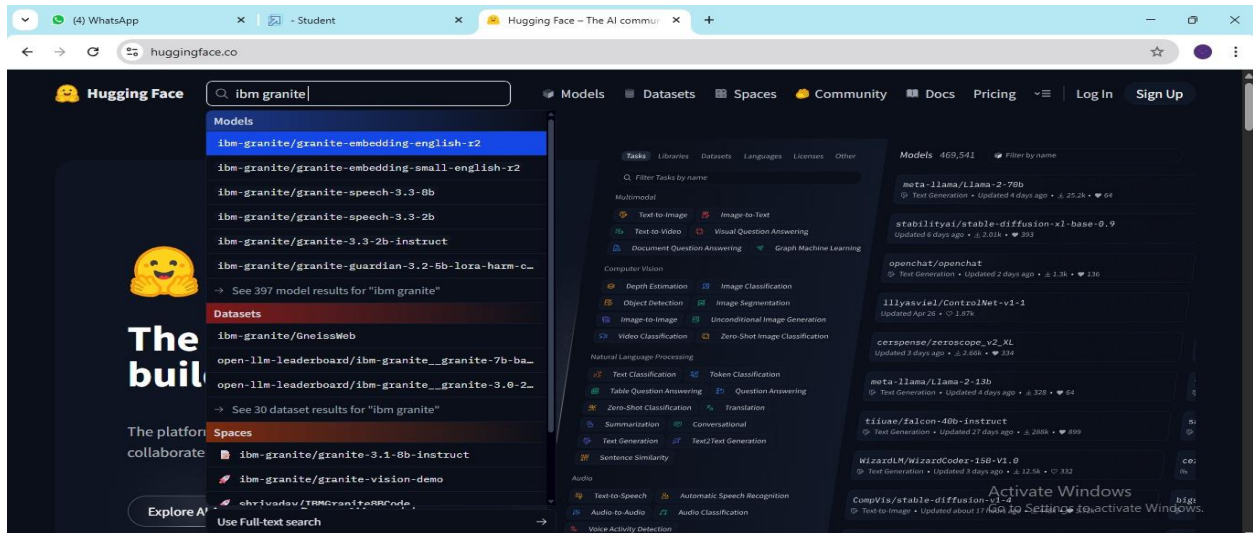
- Now we have gone through portal understanding, now lets find a IBM granite model from hugging face to integrate in our project

Activity-2: Choose a IBM Granite model From Hugging Face.

- Search for “Hugging face” in any browser.



- Then click on the first link (Hugging Face), then click on signup and create your own account in Hugging Face. Then search for “IBM-Granite models” and choose any model



- Here for this project we are using “granite-3.2-2b-instruct” which is compatible fast and light weight.

The screenshot shows the Hugging Face interface for the model `ibm-granite/granite-embedding-english-r2`. The browser tabs at the top include "(4) WhatsApp", "- Student", and "ibm-granite/granite-embedding-english-r2". The address bar shows the URL `huggingface.co/ibm-granite/granite-embedding-english-r2`. The Hugging Face logo and navigation links (Models, Datasets, Spaces, Community, Docs, Pricing, Log In, Sign Up) are at the top. The model card header shows the model name, a like count of 47, and a follow button for "IBM Granite" with 2.36k followers. Below the header, there are tags for "Sentence Similarity", "sentence-transformers", "PyTorch", "Safetensors", "Transformers", "English", "modernbert", "feature-extraction", "granite", "embeddings", "mteb", "text-embeddings-inference", "arxiv:2508.21085", and "License: apache-2.0". The "Model card" tab is selected. The main content area on the left is titled "Granite-Embedding-English-R2" and contains a "Model Summary" describing it as a 149M parameter dense biencoder embedding model. The right sidebar shows "Downloads last month" as 12,917 with a line graph, "Inference Providers" (Sentence Similarity), and a "Model tree" section.

ibm-granite/**granite-embedding-english-r2** like 47 Follow IBM Granite 2.36k

Sentence Similarity sentence-transformers PyTorch Safetensors Transformers English modernbert feature-extraction granite embeddings mteb

text-embeddings-inference arxiv:2508.21085 License: apache-2.0

Model card Files and versions xet Community 2

Granite-Embedding-English-R2

Model Summary: Granite-embedding-english-r2 is a 149M parameter dense biencoder embedding model from the Granite Embeddings collection that can be used to generate high quality text embeddings. This model produces embedding vectors of size 768 based on context length of upto 8192 tokens. Compared to most other open-source models, this model was only trained using open-source relevance-pair datasets with permissive, enterprise-friendly license, plus IBM collected and generated datasets.

The r2 models show strong performance across standard and IBM-built information

Downloads last month
12,917

Inference Providers NEW

Sentence Similarity

This model isn't deployed by any Inference Provider. Ask for provider support

Model tree for ibm-granite/granite-embedding-english-r2

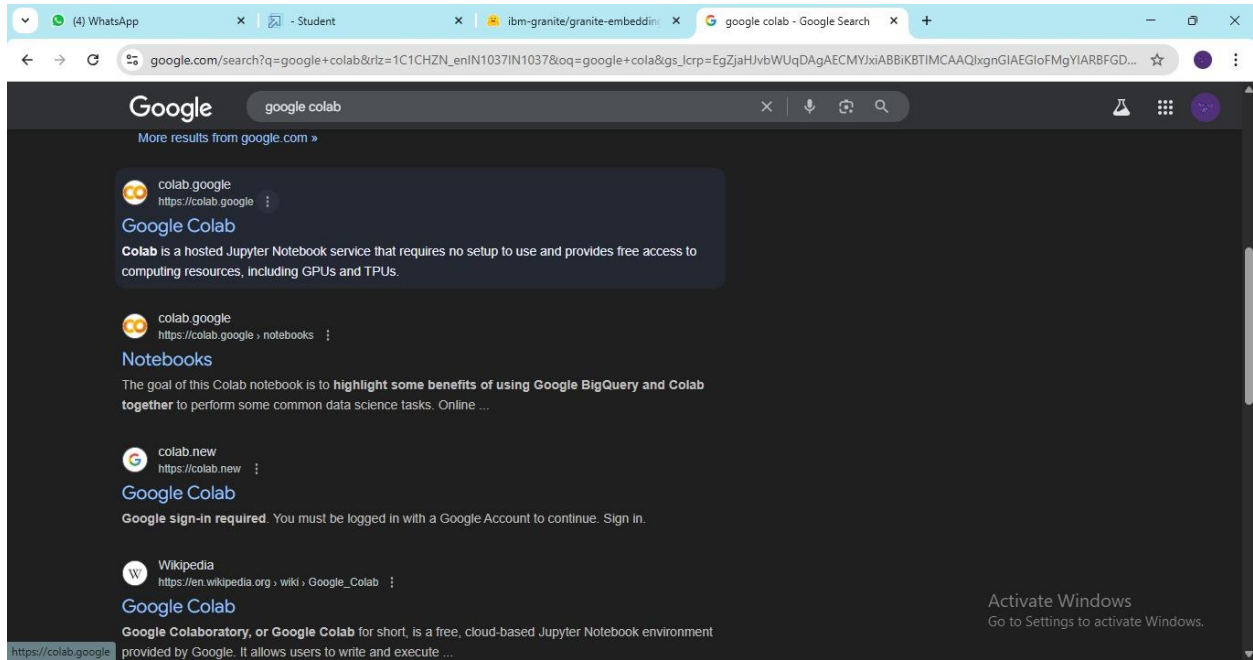
Finetunes

Go to Settings to activate Windows. 1 model

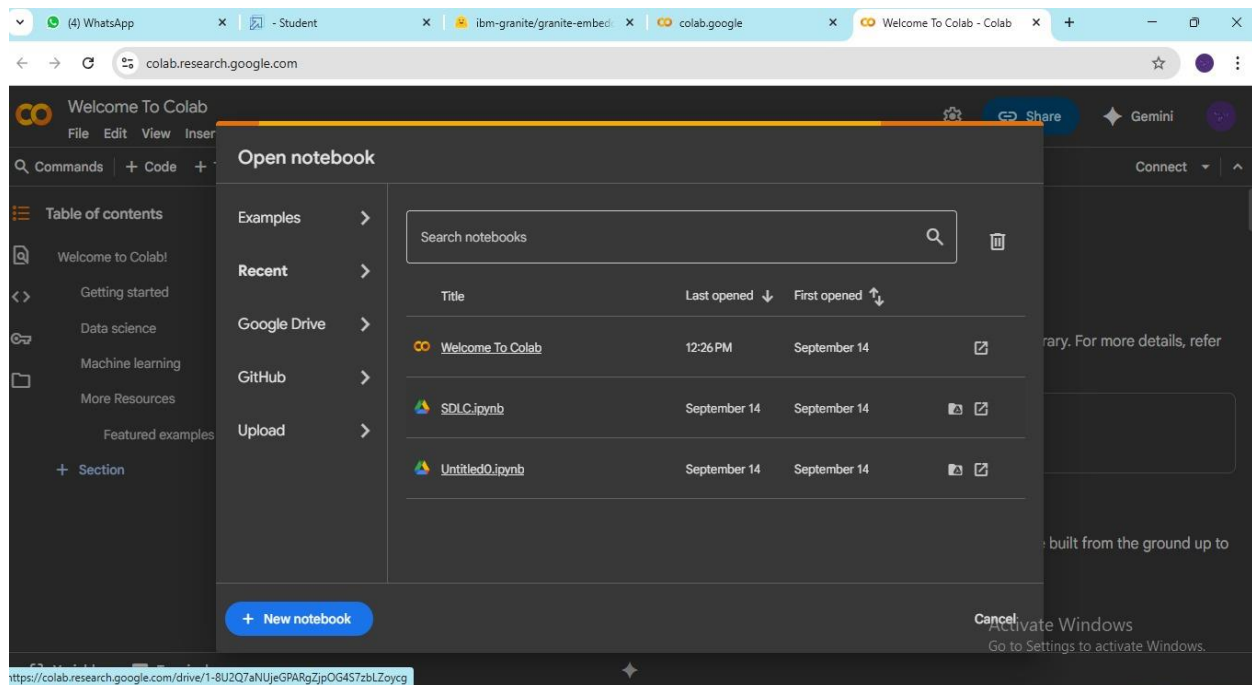
➤ Now we will start building our project in Google collab.

Activity-3: Running Application in Google Collab.

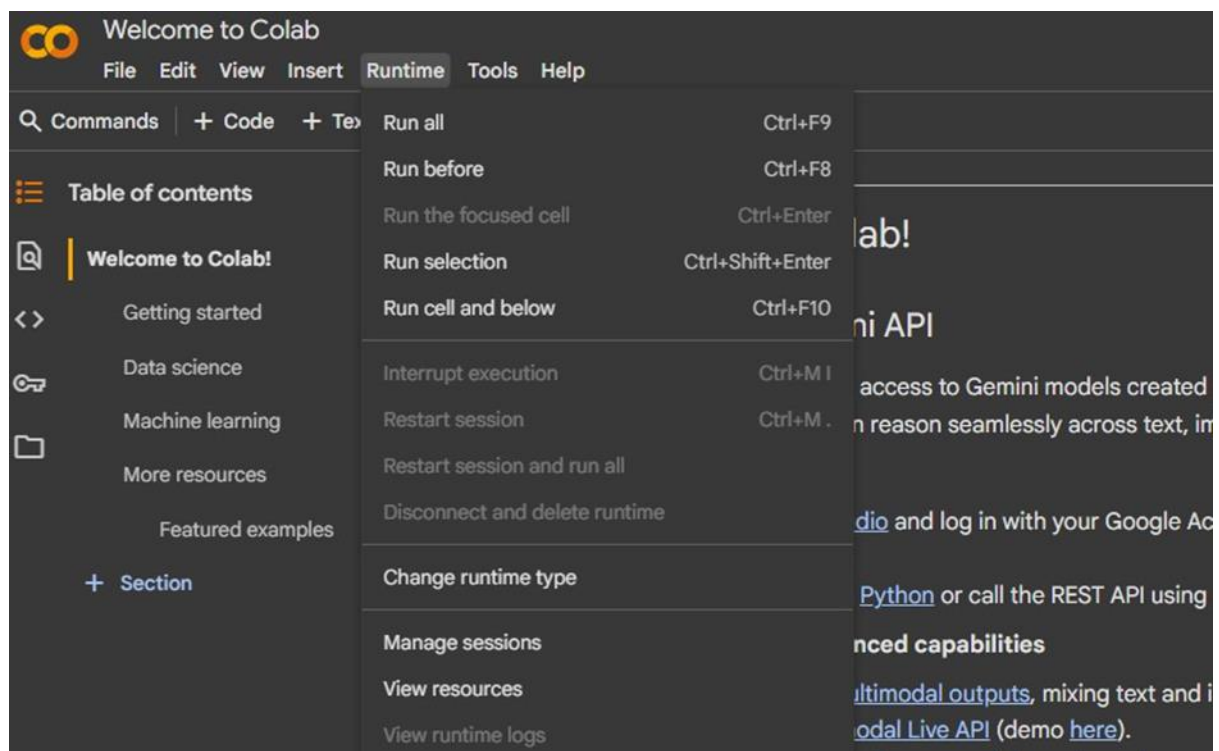
- Search for “Google collab” in any browser.



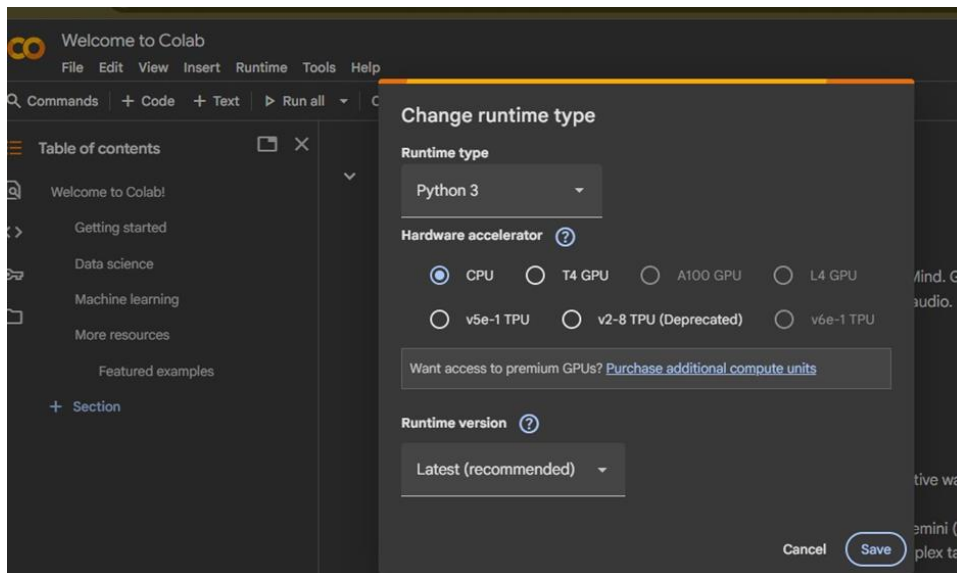
- Click on the first link (Google Colab), then click on “Files” and then “Open Notebook
- Click on “New Notebook”



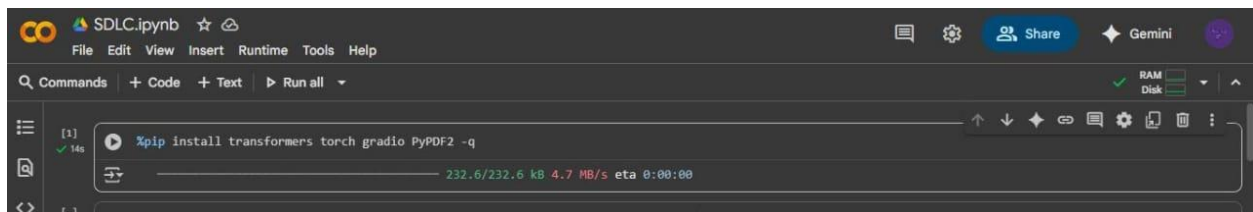
- Change the title of the notebook “Untitled” to “Health AI”. Then click on “Runtime”, then go to “Change Runtime Type”.



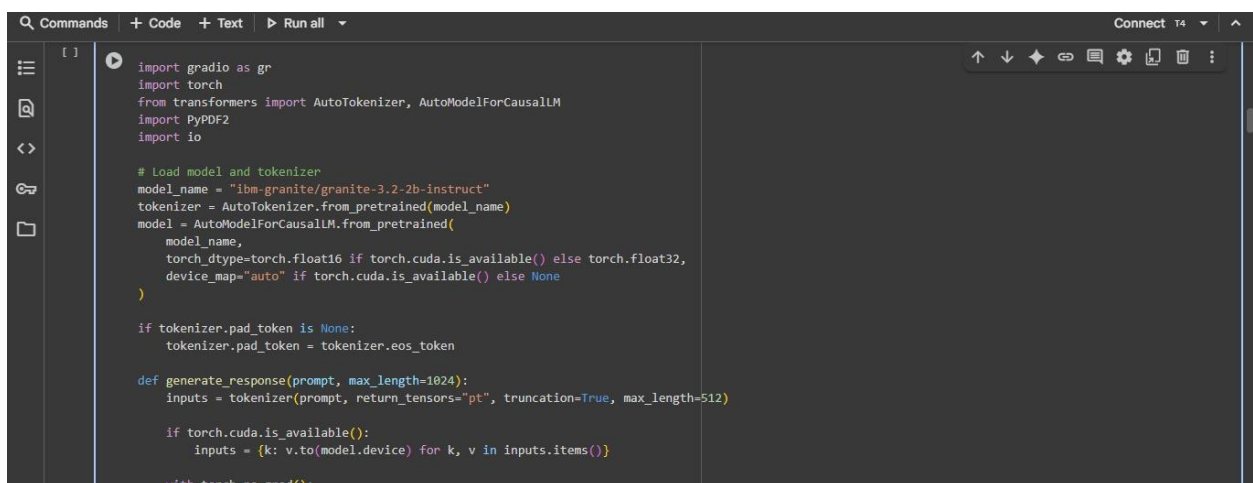
- Choose “T4 GPU” and click on “Save”.



- Then run this command in the first cell “!pip install transformers torch gradio PyPDF2 -q”. To install the required libraries to run our application.



- Then run the rest of the code in the next cell



```
Q Commands + Code + Text ▶ Run all Connect T4
[ ]
if torch.cuda.is_available():
    inputs = {k: v.to(model.device) for k, v in inputs.items()}

    with torch.no_grad():
        outputs = model.generate(
            **inputs,
            max_length=max_length,
            temperature=0.7,
            do_sample=True,
            pad_token_id=tokenizer.eos_token_id
        )

    response = tokenizer.decode(outputs[0], skip_special_tokens=True)
    response = response.replace(prompt, "").strip()
    return response

def extract_text_from_pdf(pdf_file):
    if pdf_file is None:
        return ""

    try:
        pdf_reader = PyPDF2.PdfReader(pdf_file)
        text = ""
        for page in pdf_reader.pages:
            text += page.extract_text() + "\n"
    except Exception as e:
        return f"Error reading PDF: {str(e)}"
```

```
Q Commands + Code + Text ▶ Run all Connect T4
[ ]
try:
    pdf_reader = PyPDF2.PdfReader(pdf_file)
    text = ""
    for page in pdf_reader.pages:
        text += page.extract_text() + "\n"
    return text
except Exception as e:
    return f"Error reading PDF: {str(e)}"
```

```
def requirement_analysis(pdf_file, prompt_text):
    # Get text from PDF or prompt
    if pdf_file is not None:
        content = extract_text_from_pdf(pdf_file)
        analysis_prompt = f"Analyze the following document and extract key software requirements. Organize them into functional requirements, non-functional requirements, and technical specifications."
    else:
        analysis_prompt = f"Analyze the following requirements and organize them into functional requirements, non-functional requirements, and technical specifications."

    return generate_response(analysis_prompt, max_length=1200)
```

```
def code_generation(prompt, language):
    code_prompt = f"Generate {language} code for the following requirement:\n\n{prompt}\n\nCode:"
    return generate_response(code_prompt, max_length=1200)
```

```
# Create Gradio interface
with gr.Blocks() as app:
    gr.Markdown("# AI Code Analysis & Generator")
```

{ } Variables Terminal

```
Q Commands + Code + Text ▶ Run all Connect T4
[ ]
with gr.Blocks():
    with gr.TabItem("Code Analysis"):
        with gr.Row():
            with gr.Column():
                pdf_upload = gr.File(label="Upload PDF", file_types=[".pdf"])
                prompt_input = gr.Textbox(
                    label="Or write requirements here",
                    placeholder="Describe your software requirements...",
                    lines=5
                )
                analyze_btn = gr.Button("Analyze")

            with gr.Column():
                analysis_output = gr.Textbox(label="Requirements Analysis", lines=20)

        analyze_btn.click(requirement_analysis, inputs=[pdf_upload, prompt_input], outputs=[analysis_output])

    with gr.TabItem("Code Generation"):
        with gr.Row():
            with gr.Column():
                code_prompt = gr.Textbox(
                    label="Code Requirements",
                    placeholder="Describe what code you want to generate...",
                    lines=5
                )
```



```
analyze_btn.click(requirement_analysis, inputs=[pdf_upload, prompt_input], outputs=analysis_output)

with gr.TabItem("Code Generation"):
    with gr.Row():
        with gr.Column():
            code_prompt = gr.Textbox(
                label="Code Requirements",
                placeholder="Describe what code you want to generate...",
                lines=5
            )
            language_dropdown = gr.Dropdown(
                choices=["Python", "JavaScript", "Java", "C++", "C#", "PHP", "Go", "Rust"],
                label="Programming Language",
                value="Python"
            )
            generate_btn = gr.Button("Generate Code")

        with gr.Column():
            code_output = gr.Textbox(label="Generated Code", lines=20)

    generate_btn.click(code_generation, inputs=[code_prompt, language_dropdown], outputs=code_output)

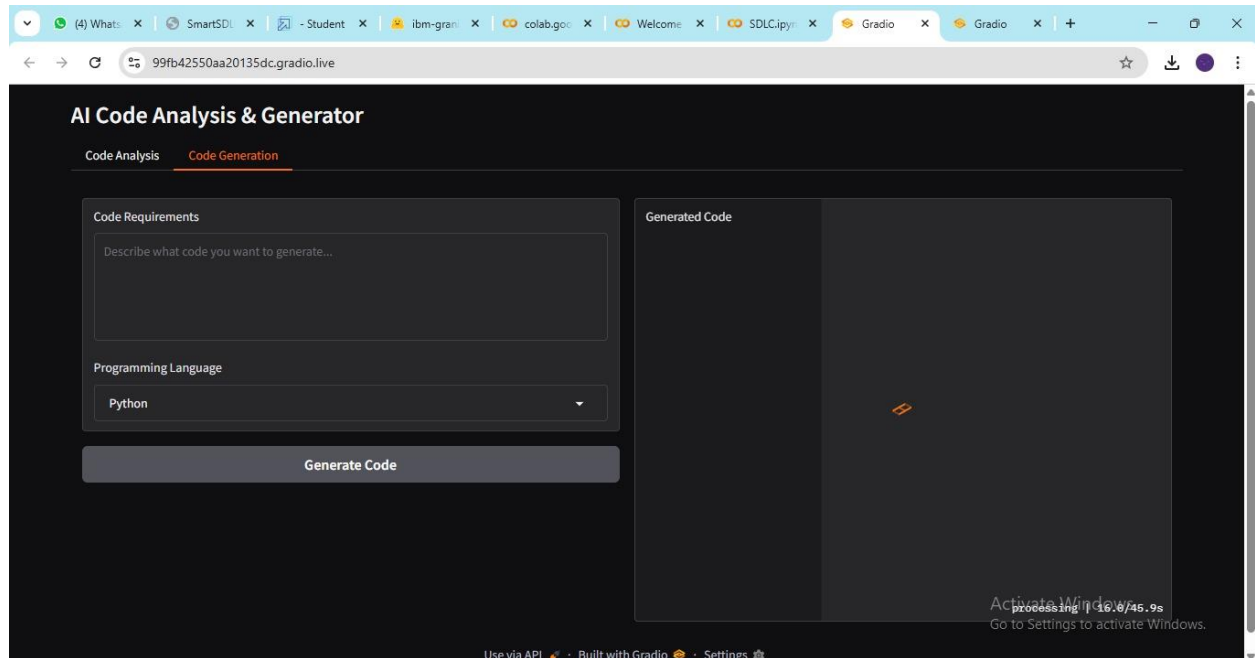
app.launch(share=True)
smartsdlc.py
Displaying smartsdlc.py.
```

➤ You can find the code here in this link: [SmartSDLC](#)

```
merges.txt: 442k? [00:00<00:00, 5.80MB/s]
tokenizer.json: 3.48M? [00:00<00:00, 27.7MB/s]
added_tokens.json: 100% 87.0/87.0 [00:00<00:00, 1.09kB/s]
special_tokens_map.json: 100% 701/701 [00:00<00:00, 19.5kB/s]
config.json: 100% 786/786 [00:00<00:00, 6.24kB/s]
`torch_dtype` is deprecated! Use `dtype` instead!
model.safetensors.index.json: 29.8k? [00:00<00:00, 1.78MB/s]
Fetching 2 files: 100% 2/2 [01:39<00:00, 99.00s/it]
model-00001-of-00002.safetensors: 100% 5.00G/5.00G [01:38<00:00, 107MB/s]
model-00002-of-00002.safetensors: 100% 67.1M/67.1M [00:01<00:00, 56.6MB/s]
Loading checkpoint shards: 100% 2/2 [00:26<00:00, 10.78s/it]
generation_config.json: 100% 137/137 [00:00<00:00, 10.2kB/s]
Colab notebook detected. To show errors in colab notebook, set debug=True in launch()
* Running on public URL: https://f4888bb452623b8fc4.gradio.live
This share link expires in 1 week. For free permanent hosting and GPU upgrades, run `gradio deploy` from the terminal in the working directory to deploy to
```

AI Code Analysis & Generator

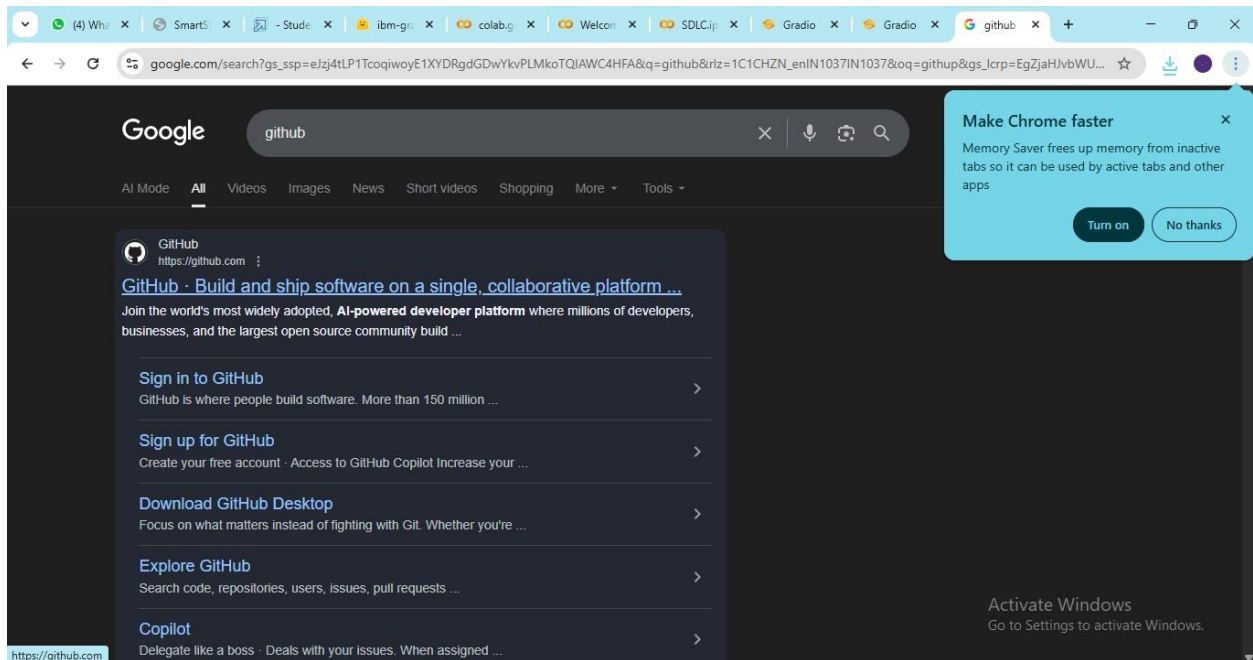
- Now you can see our model is being Downloaded and application is running
- Click on the URL to open the Gradio Application click on the link: <https://f4888bb452623b8fc4.gradio.live>



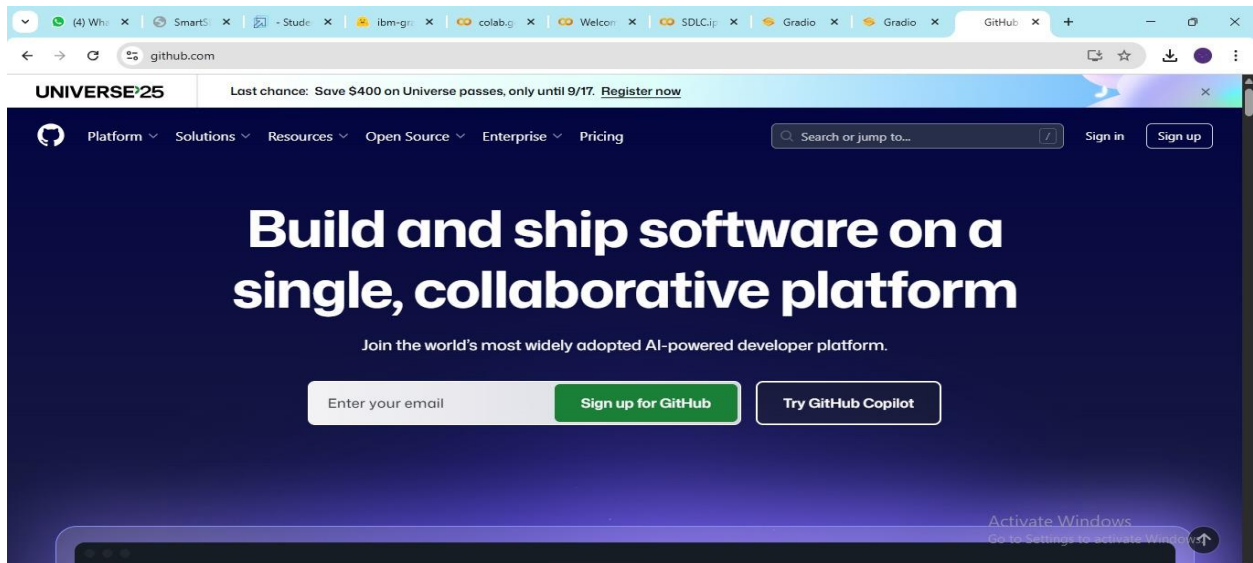
- You can View the Application is the running in the other tab

Activity-4: Upload Your Project in GitHub.

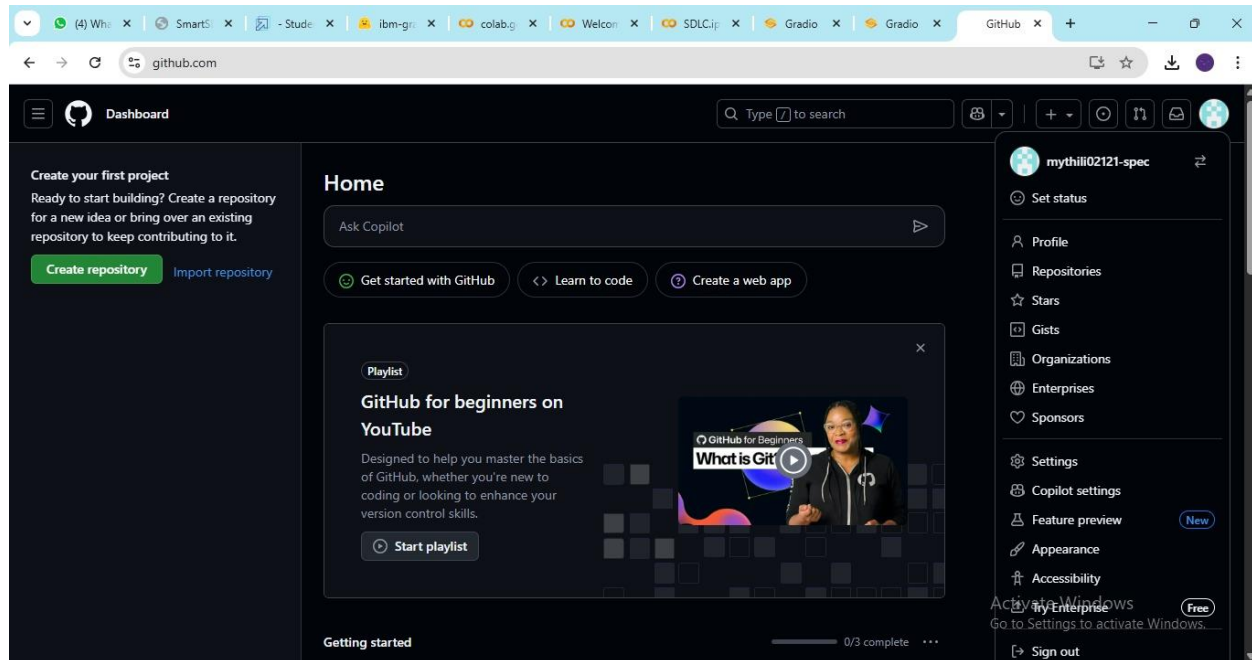
- Search for “GitHub” in any browser, then click on the first link (GitHub)



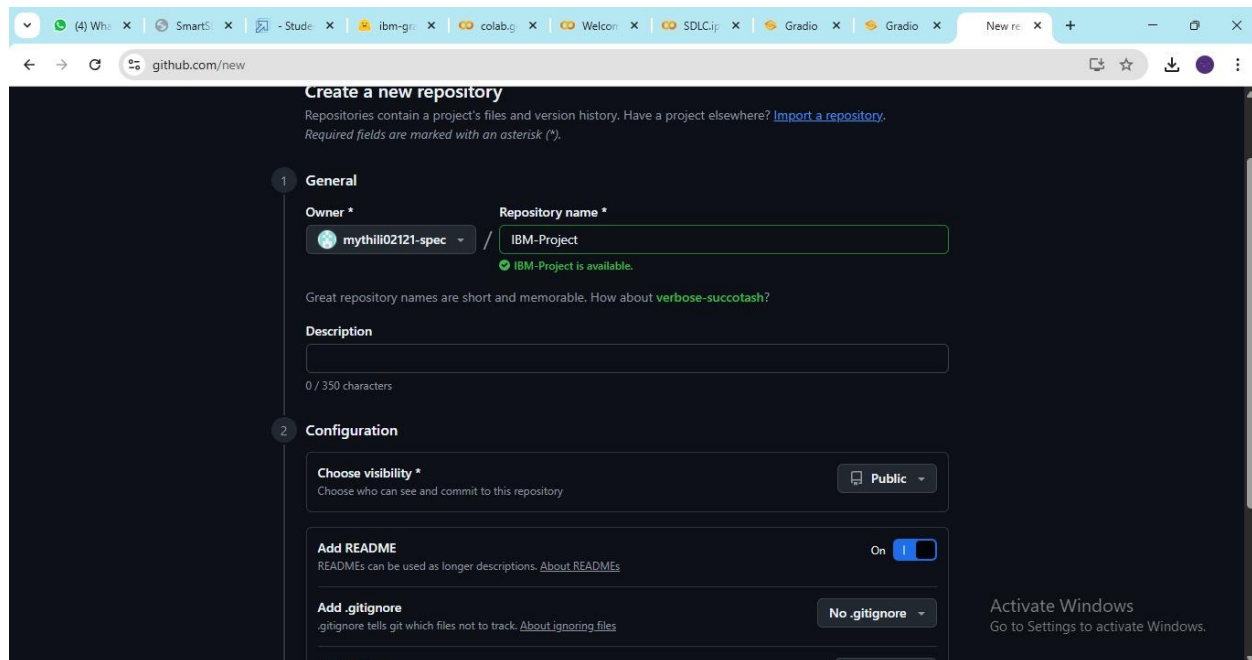
- Then click on “Signup” and create your own account in GitHub. If you already have an account click on “Sign in”.



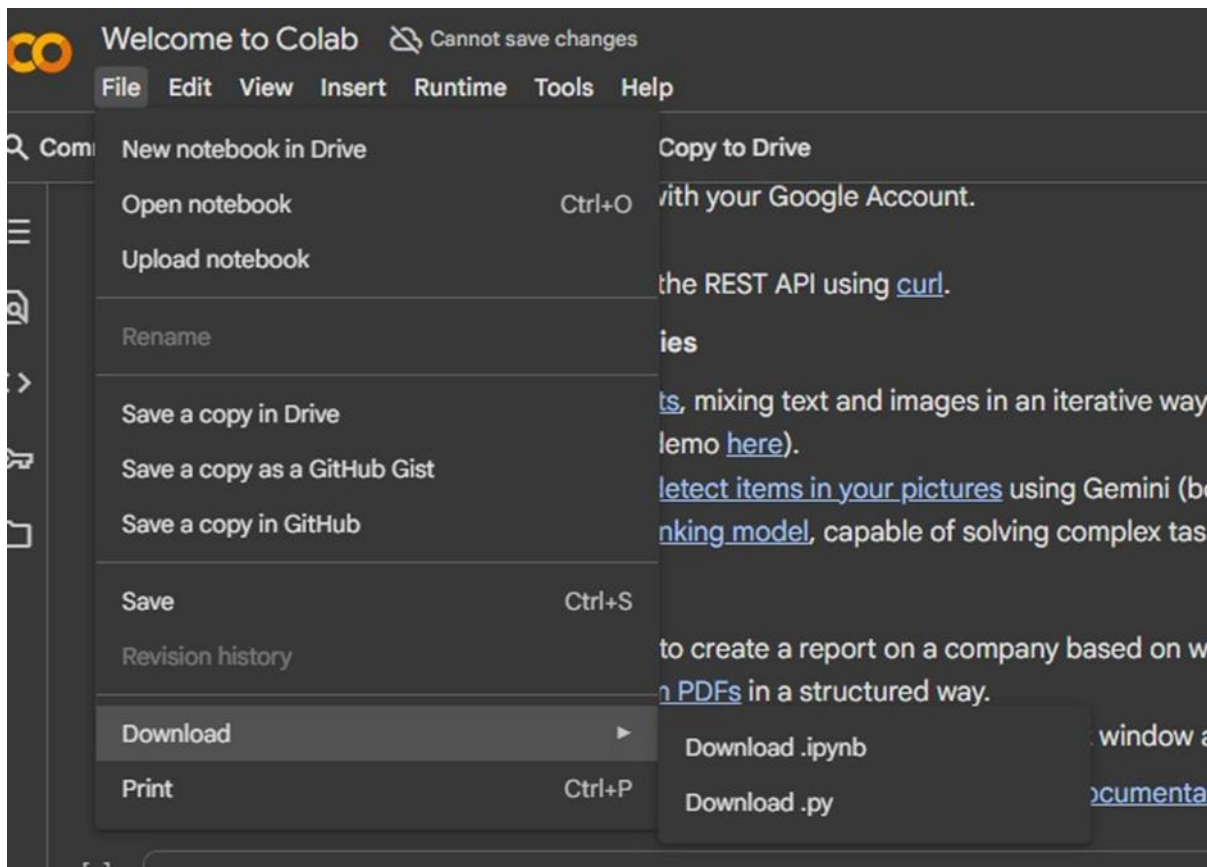
- Click on “Create repository”.



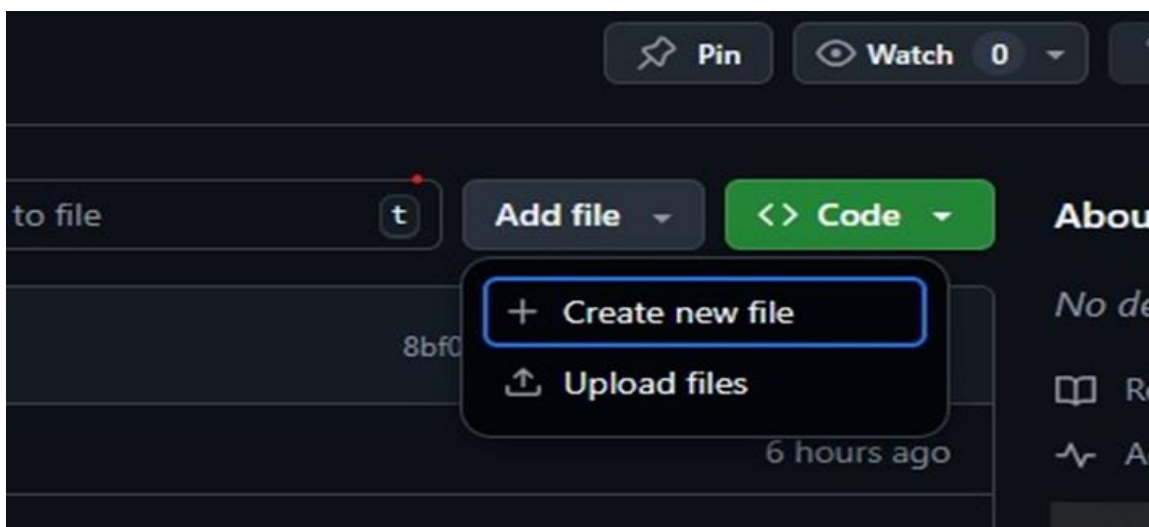
- In “General” Name your repo. (Here I have given “IBM-Project” as my repo name and it is available).
- In “Configurations” Turn On “Add readme” file Option.



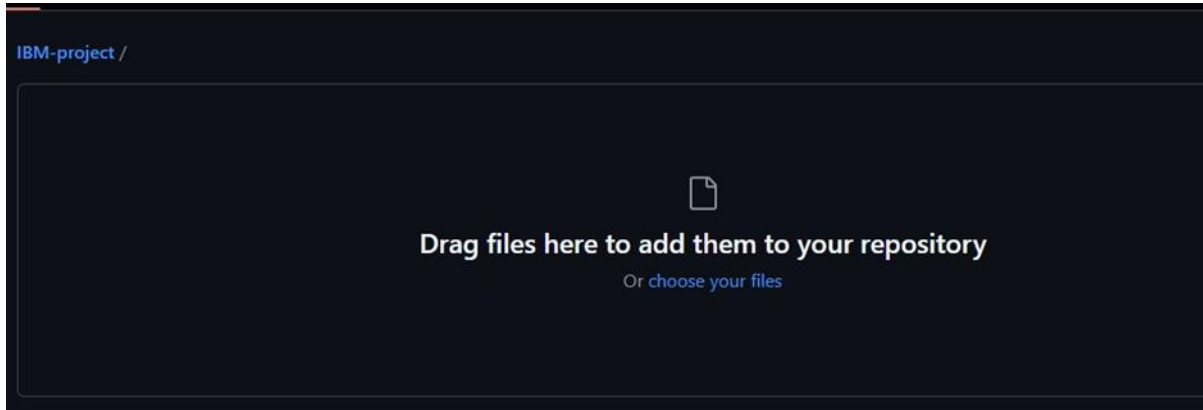
- Now Download your code from Google collab by Clicking on “File”, then Goto “Download” then download as “.py”.



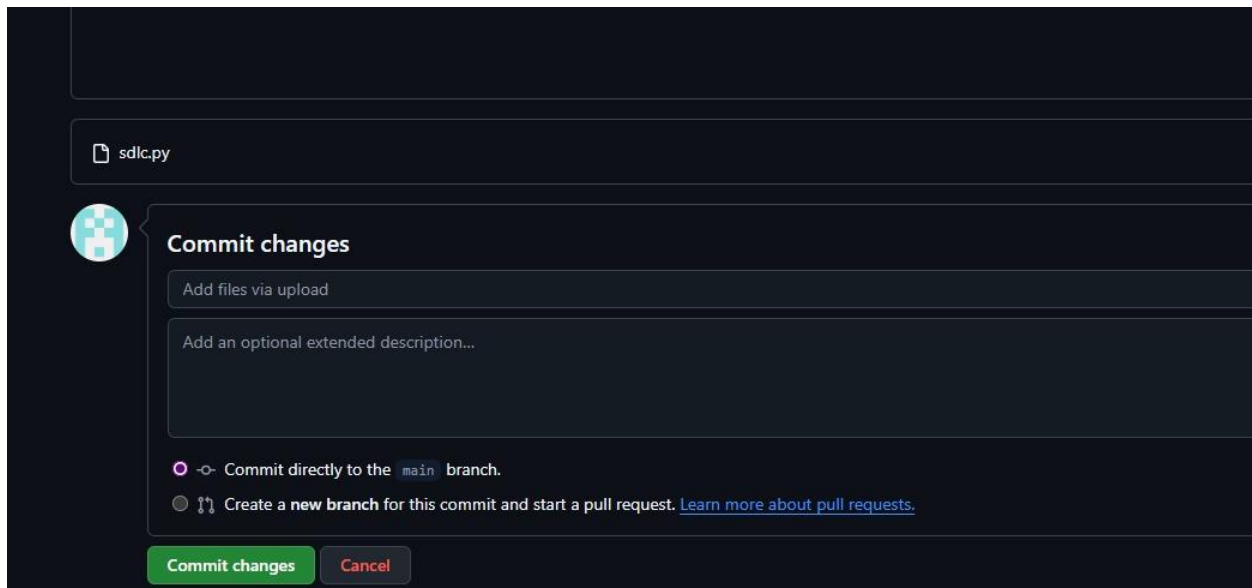
- Then your repository is created, then Click on “Add file” Option. Then Click 15 “Upload files” to upload your files.



- Click on “choose your files”.



- Choose your project file and click on “Open”.
- After your file has Uploaded Click on “Commit changes”.



- Successfully uploaded code into GitHub.