

Project-Title:

Azure-powered RAG (Retrieval Augmented generation): An Industrial Research Assistant (IRA) and document translation chatbot.

Input:

Research Papers

Output:

1. English Summary
2. Japanese Summary
3. Chat with your own pdf
4. English to Japanese Translation

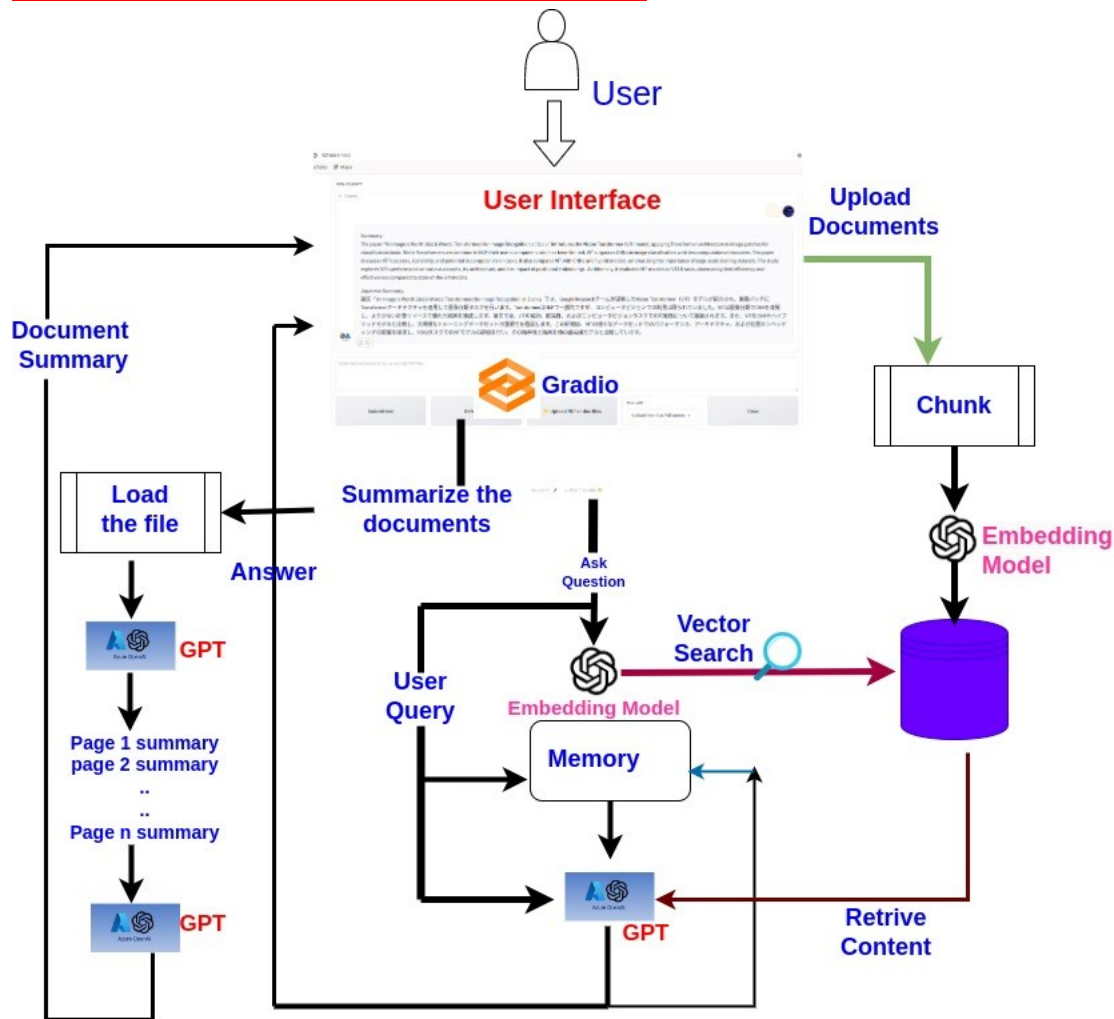
RAG-ChatGPT: Retrieval Augmented generation (RAG) chatbot using Azure OpenAI GPT Model, Langchain, ChromaDB, and Gradio.

RAG-GPT supports both PDFs and Docs.

The chatbot offers two convenient ways to use it:

1. Real-time Uploads: You can easily upload documents during your chat sessions, letting the chatbot process and respond to the content immediately.
2. Summarization Requests: You can ask the chatbot to provide a comprehensive summary of an entire PDF or document in a single interaction, making it easy and quick to retrieve information.

RAG Application Design Schema



Document Storage

Documents are stored in two separate folders within the data directory:

1. data/docs_2: For files that you want to upload.
2. data/docs: For files that should be processed in advance.

Database Creation

Vector databases, also known as vectorDBs, are created and stored in the data folder to enhance the functionality of the project.

Running the Project:

To get the project up and running, you'll need to set up your environment and install the necessary dependencies. You can do this by following

1. Go to the **RAG-ChatGPT** folder and locate the requirements.txt file. Install all dependencies using the following command for Ubuntu:

```
pip install -r requirements.txt
```

2. First, create the .env file in the parent directory. Refer to the env.example file for details.
3. Open **configs/app_config.yml**, and enter your Azure API embedding model and GPT engine name.
4. Run the Application from the terminal

```
python src\raggpt_app.py
```

5. Chat with your documents.

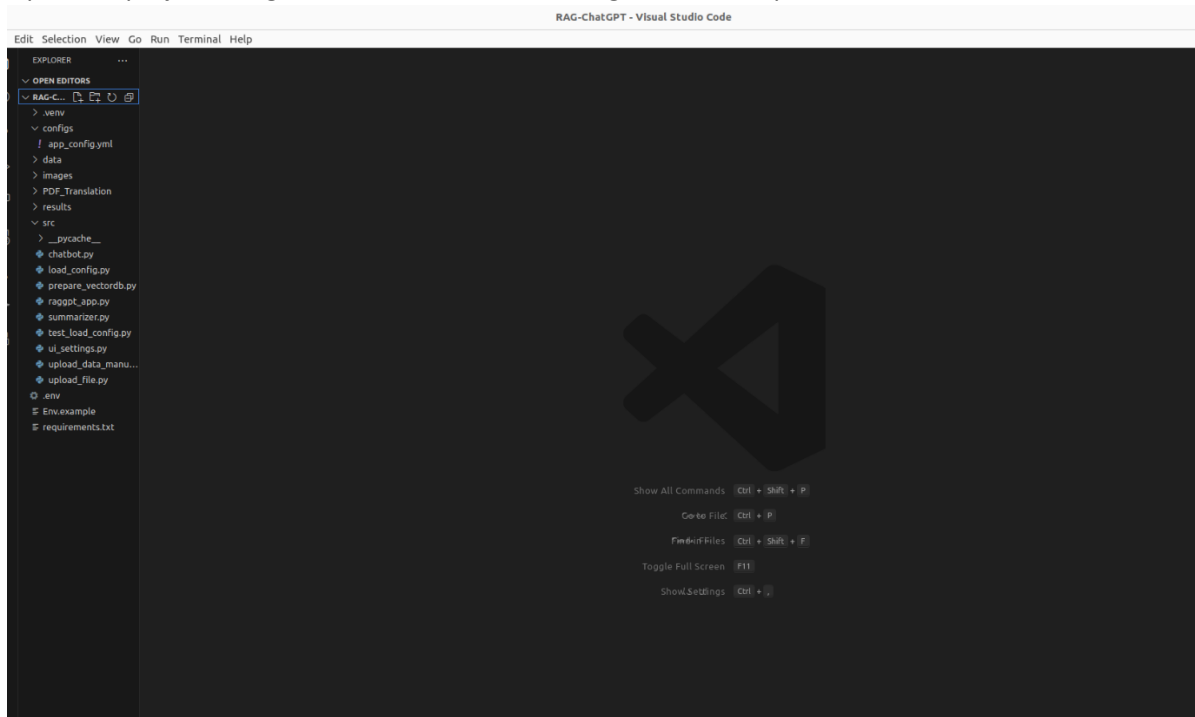
Please refer to the picture below and follow the project details information presented in a step-by-step manner. Kindly review each item carefully to ensure successful completion. Once done, kindly mark each item with a checkmark (✓).

To utilize the project, please follow the following instructions:

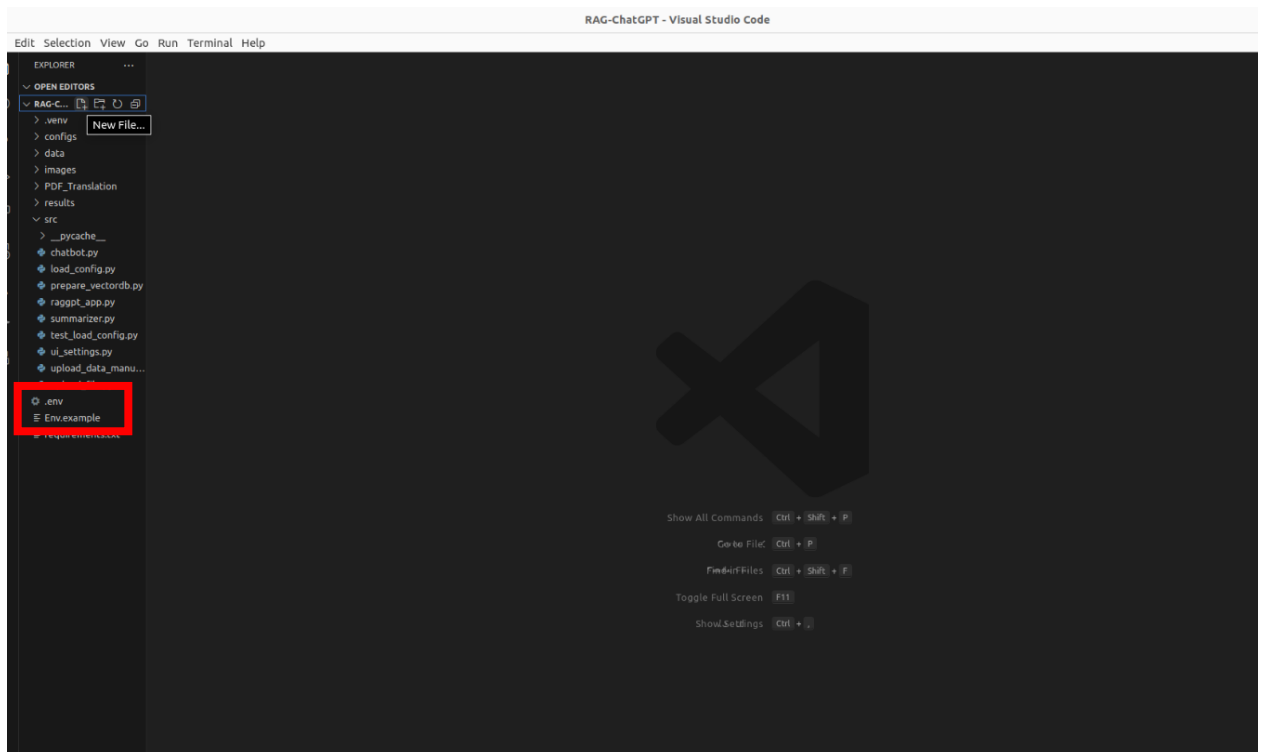
1. Download the project and extract it to your personal computer.



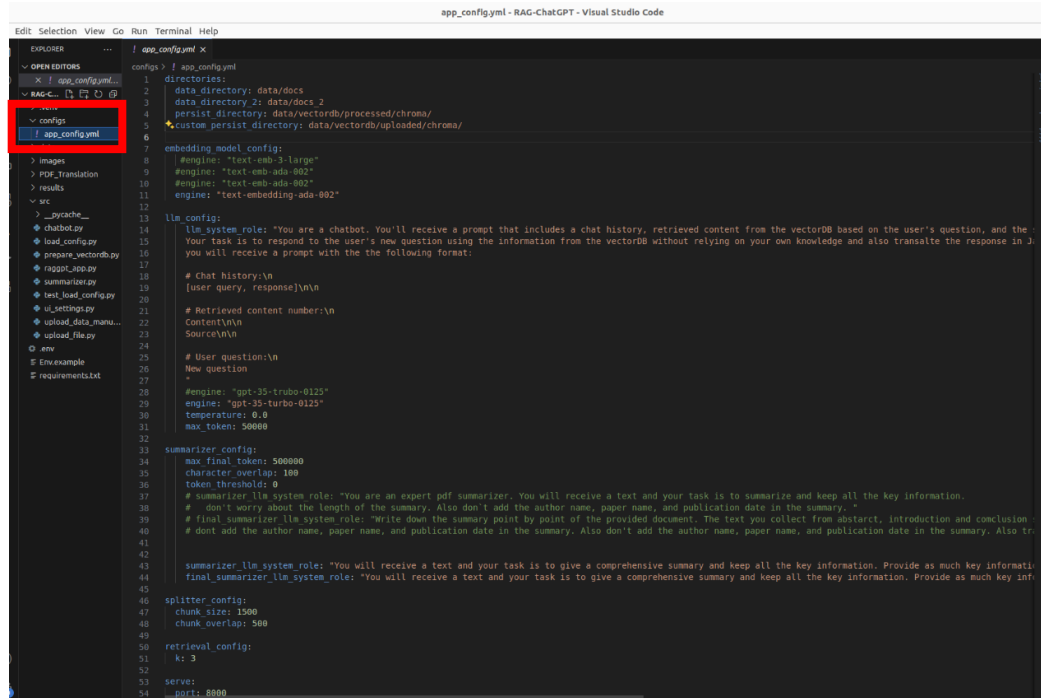
2. Open the project using the Visual Studio Code Integrated Development Environment.



3. Create a .env file in the parent directory and follow the instruction like env.example file.

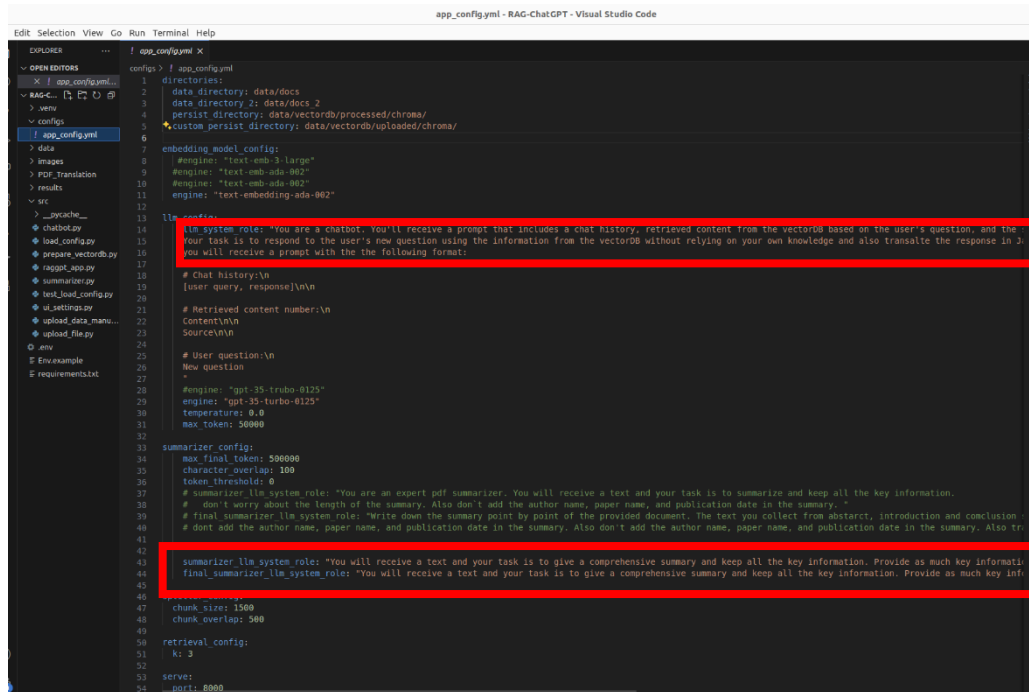


4. Access the `app_config.yml` file in the `configs` directory, and input your Azure API embedding model and GPT engine name.



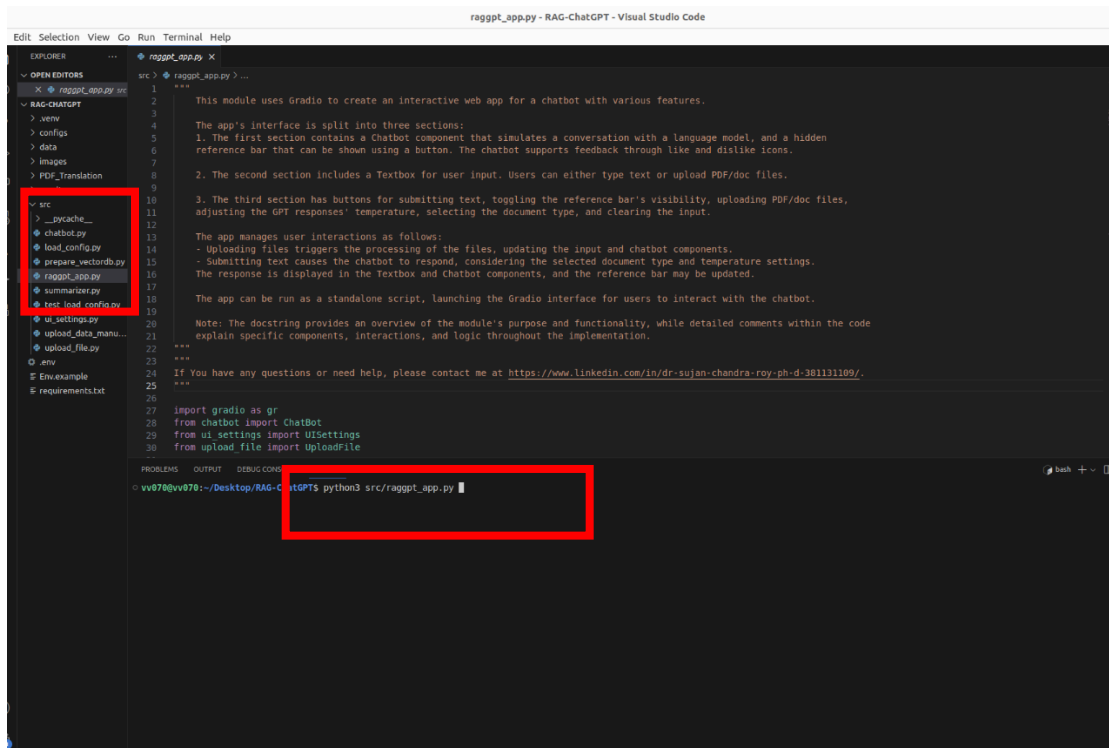
```
1 directories:
2   data_directory: data/docs
3   data_directory_2: data/docs_2
4   persist_directory: data/vectordb/processed/chroma/
5   custom_persist_directory: data/vectordb/uploaded/chroma/
6
7 embedding_model_config:
8   #engine: "text-emb-3-large"
9   #engine: "text-emb-ada-002"
10  #engine: "text-emb-ada-002"
11  engine: "text-embedding-ada-002"
12
13 llm_config:
14  llm_system_role: "You are a chatbot. You'll receive a prompt that includes a chat history, retrieved content from the vectorDB based on the user's question, and the
15  Your task is to respond to the user's new question using the information from the vectorDB without relying on your own knowledge and also translate the response in J
16  you will receive a prompt with the the following format:
17
18  # Chat history:\n\n
19  (user query, response)\n\n
20
21  # Retrieved content number:\n\n
22  Content\n\n
23  Source\n\n
24
25  # User question:\n\n
26  New question
27
28  #engine: "gpt-35-turbo-0125"
29  engine: "gpt-35-turbo-0125"
30  temperature: 0.0
31  max_token: 50000
32
33 summarizer_config:
34  max_final_token: 500000
35  character_overlap: 100
36  token_threshold: 0
37
38  # summarizer_llm_system_role: "You are an expert pdf summarizer. You will receive a text and your task is to summarize and keep all the key information.
39  # don't worry about the length of the summary. Also don't add the author name, paper name, and publication date in the summary. "
40  # final_summarizer_llm_system_role: "Write down the summary point by point of the provided document. The text you collect from abstract, introduction and conclusion
41  # don't add the author name, paper name, and publication date in the summary. Also don't add the author name, paper name, and publication date in the summary. Also tr
42
43  summarizer_llm_system_role: "You will receive a text and your task is to give a comprehensive summary and keep all the key information. Provide as much key informati
44  final_summarizer_llm_system_role: "You will receive a text and your task is to give a comprehensive summary and keep all the key information. Provide as much key inf
45
46 splitter_config:
47  chunk_size: 1500
48  chunk_overlap: 500
49
50 retrieval_config:
51  k: 3
52
53 serve:
54  port: 8000
```

5. If a more concise and enhanced output is desired from the chatbot, please modify the `llm_system_role`, `summarizer_llm_system_role`, and `final_summarizer_llm_system_role` prompts.

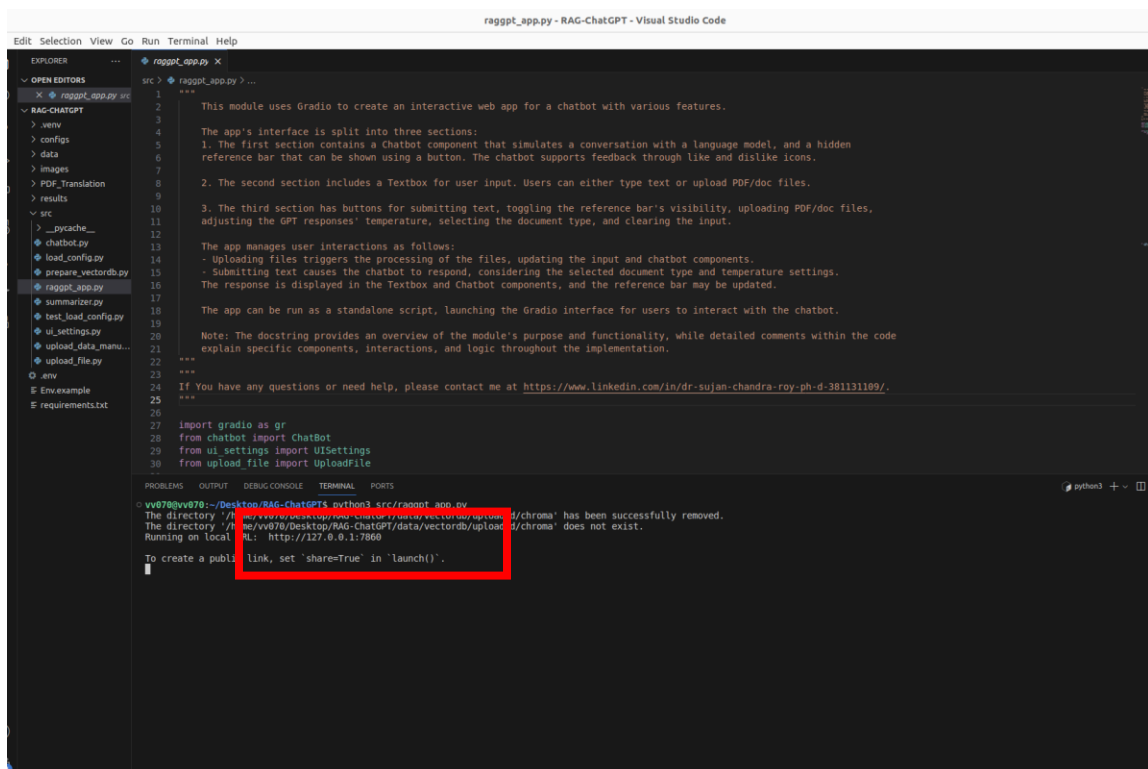


```
1 directories:
2   data_directory: data/docs
3   data_directory_2: data/docs_2
4   persist_directory: data/vectordb/processed/chroma/
5   custom_persist_directory: data/vectordb/uploaded/chroma/
6
7 embedding_model_config:
8   #engine: "text-emb-3-large"
9   #engine: "text-emb-ada-002"
10  #engine: "text-emb-ada-002"
11  engine: "text-embedding-ada-002"
12
13 llm_config:
14  llm_system_role: "You are a chatbot. You'll receive a prompt that includes a chat history, retrieved content from the vectorDB based on the user's question, and the
15  Your task is to respond to the user's new question using the information from the vectorDB without relying on your own knowledge and also translate the response in J
16  you will receive a prompt with the the following format:
17
18  # Chat history:\n\n
19  (user query, response)\n\n
20
21  # Retrieved content number:\n\n
22  Content\n\n
23  Source\n\n
24
25  # User question:\n\n
26  New question
27
28  #engine: "gpt-35-turbo-0125"
29  engine: "gpt-35-turbo-0125"
30  temperature: 0.0
31  max_token: 50000
32
33 summarizer_config:
34  max_final_token: 500000
35  character_overlap: 100
36  token_threshold: 0
37
38  # summarizer_llm_system_role: "You are an expert pdf summarizer. You will receive a text and your task is to summarize and keep all the key information.
39  # don't worry about the length of the summary. Also don't add the author name, paper name, and publication date in the summary. "
40  # final_summarizer_llm_system_role: "Write down the summary point by point of the provided document. The text you collect from abstract, introduction and conclusion
41  # don't add the author name, paper name, and publication date in the summary. Also don't add the author name, paper name, and publication date in the summary. Also tr
42
43  summarizer_llm_system_role: "You will receive a text and your task is to give a comprehensive summary and keep all the key information. Provide as much key informati
44  final_summarizer_llm_system_role: "You will receive a text and your task is to give a comprehensive summary and keep all the key information. Provide as much key inf
45
46 splitter_config:
47  chunk_size: 1500
48  chunk_overlap: 500
49
50 retrieval_config:
51  k: 3
52
53 serve:
54  port: 8000
```

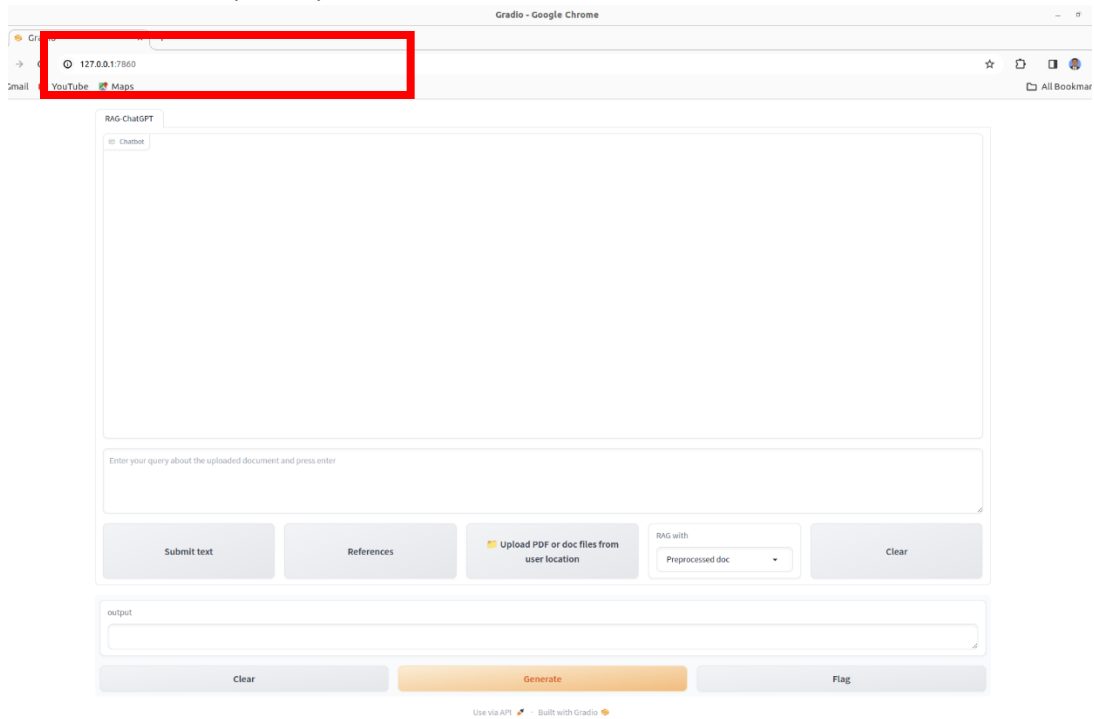
- Open the `raggpt_app.py` file in the `src` folder and run it using the command given below from the terminal.



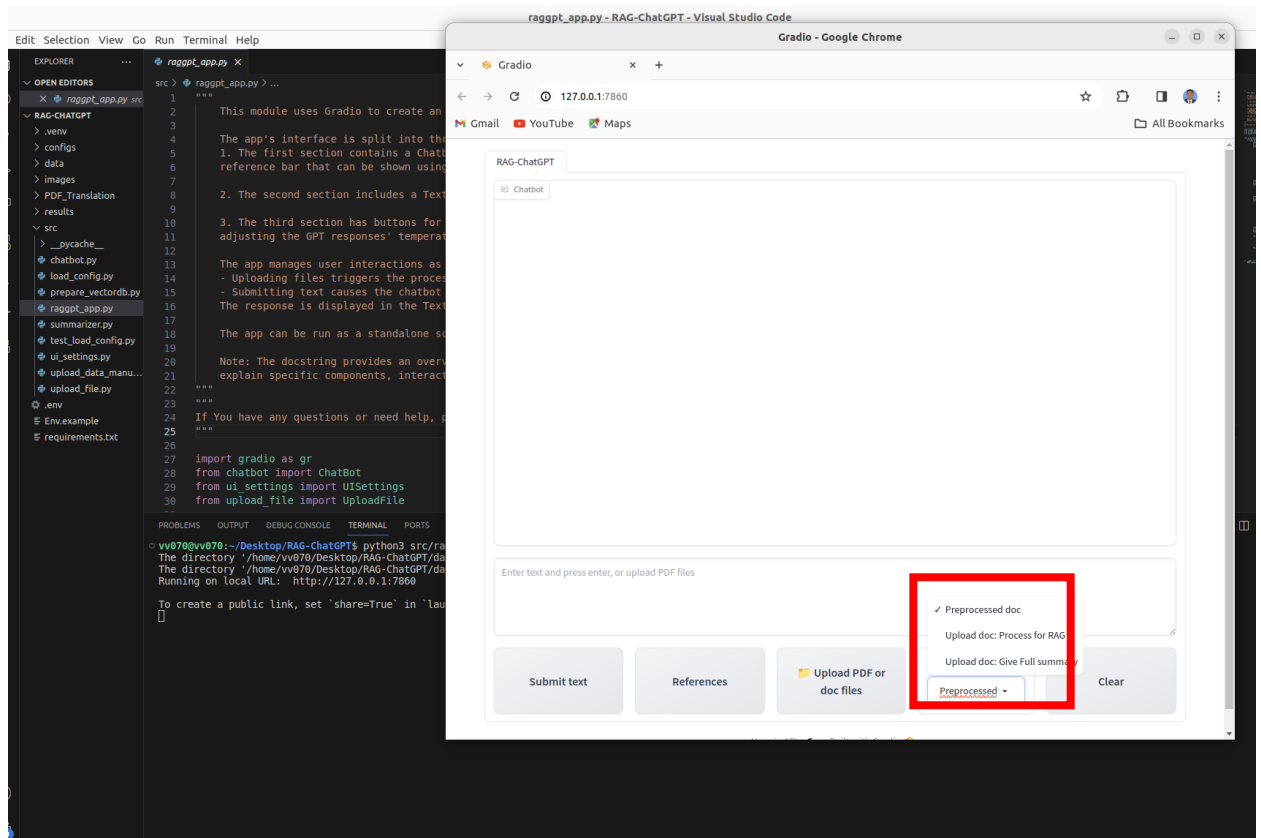
- Click on the URL by using the Ctrl+click command.



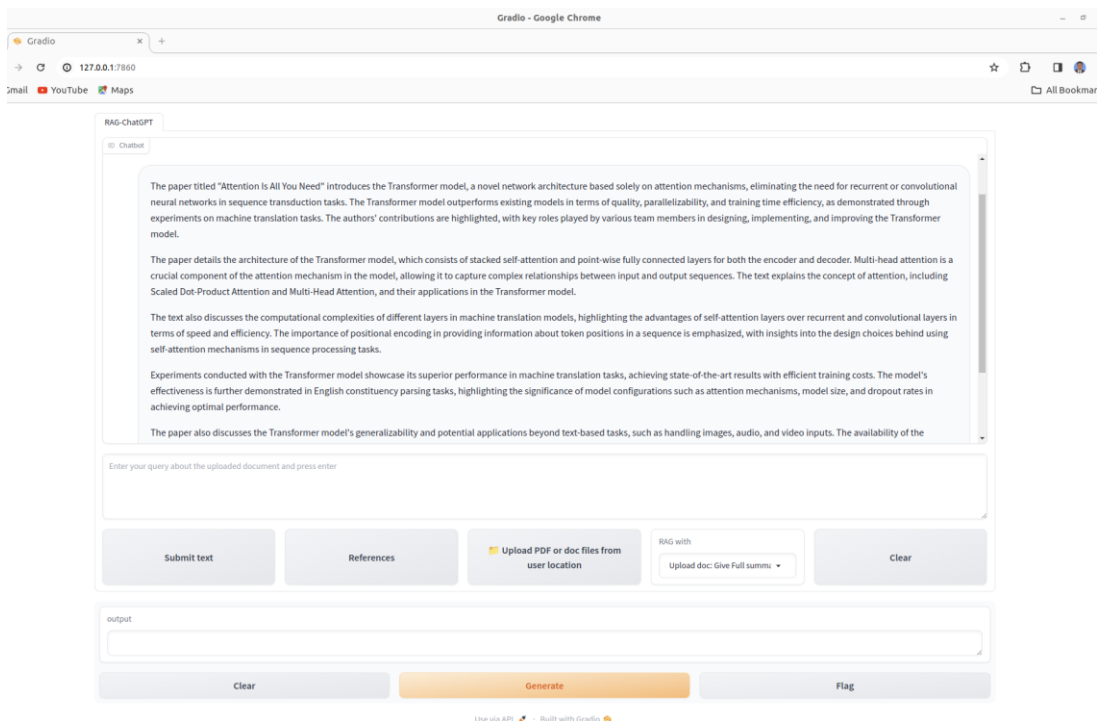
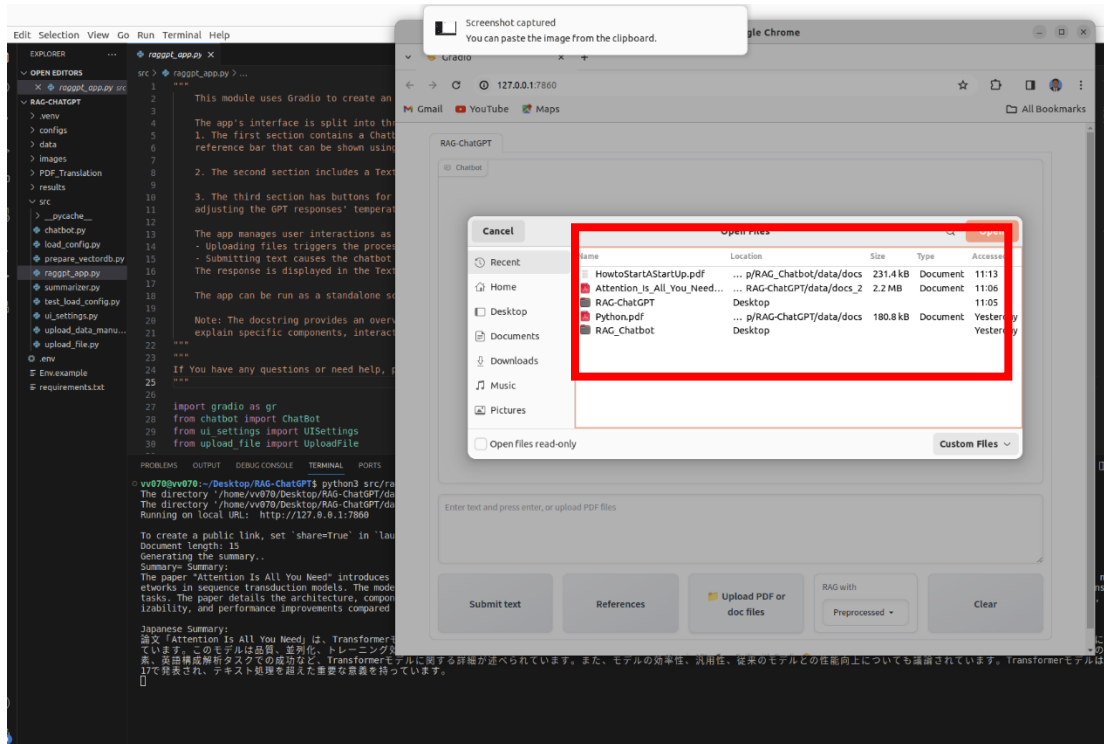
8. The link will then open in your browser.

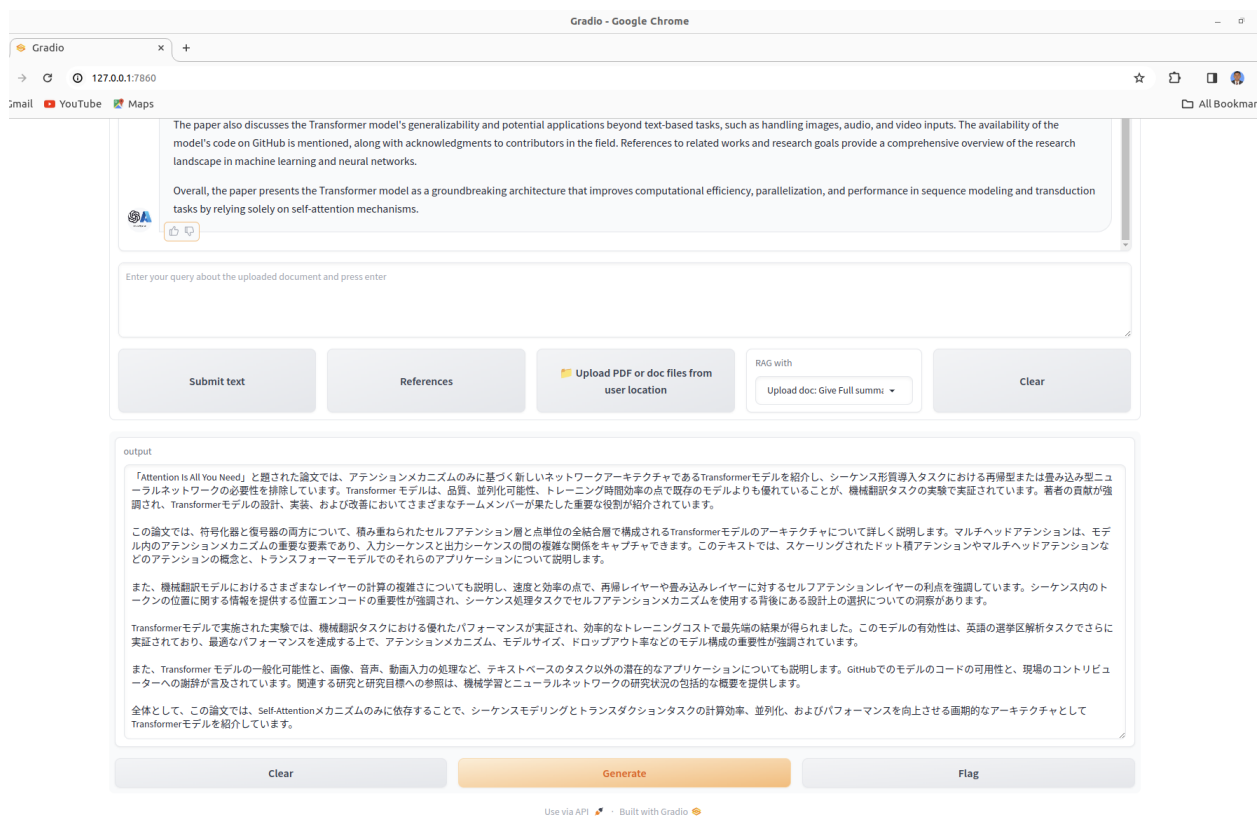
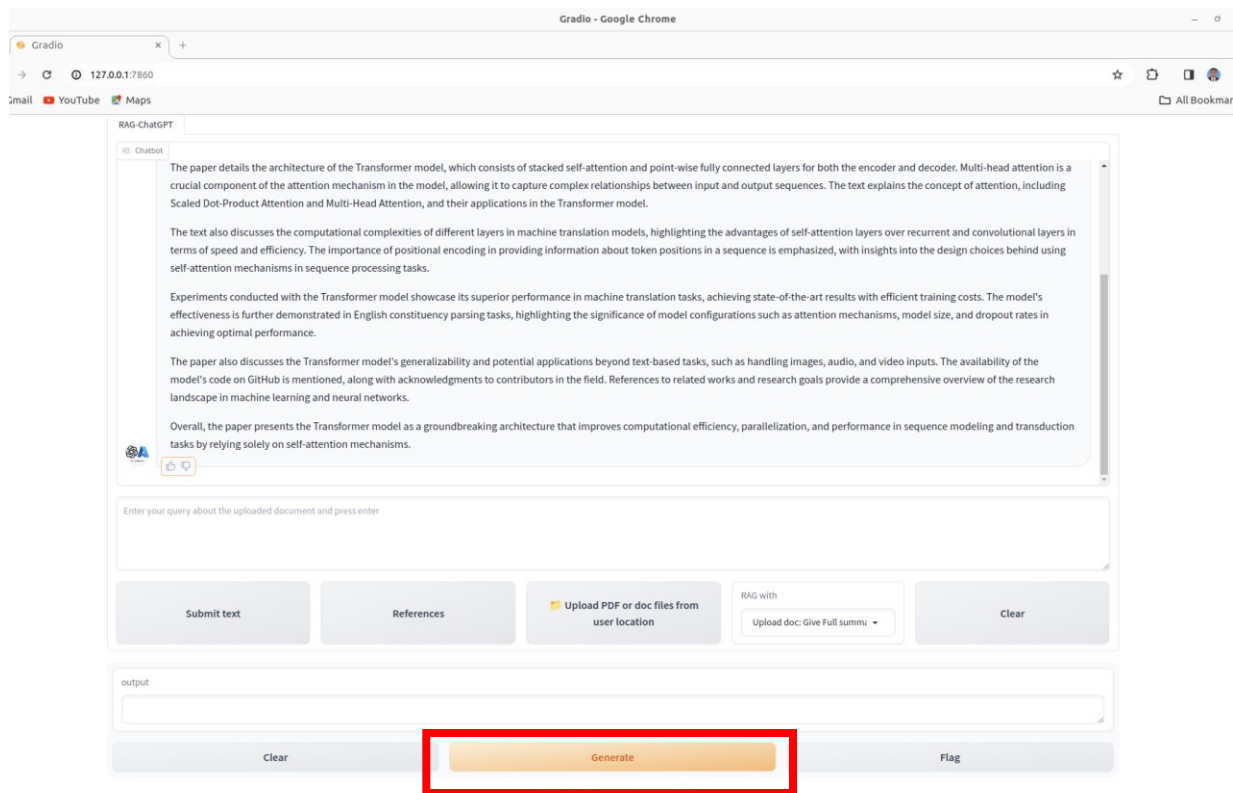


9. Proceed to the "Upload Document" section.

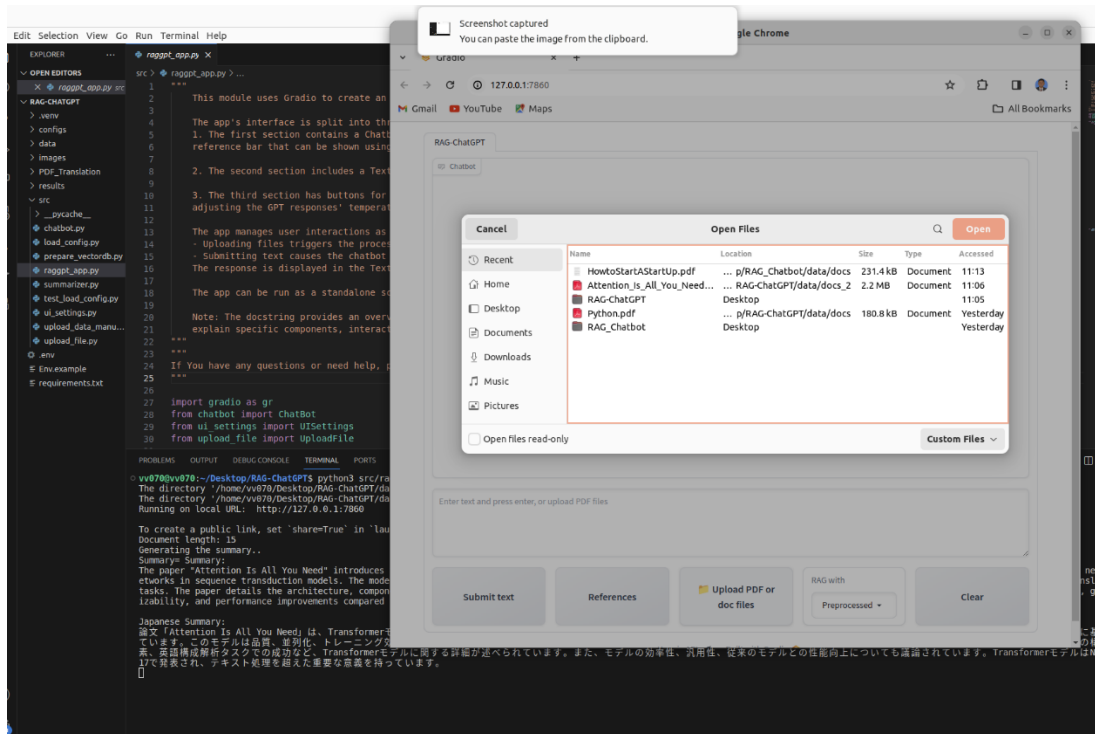
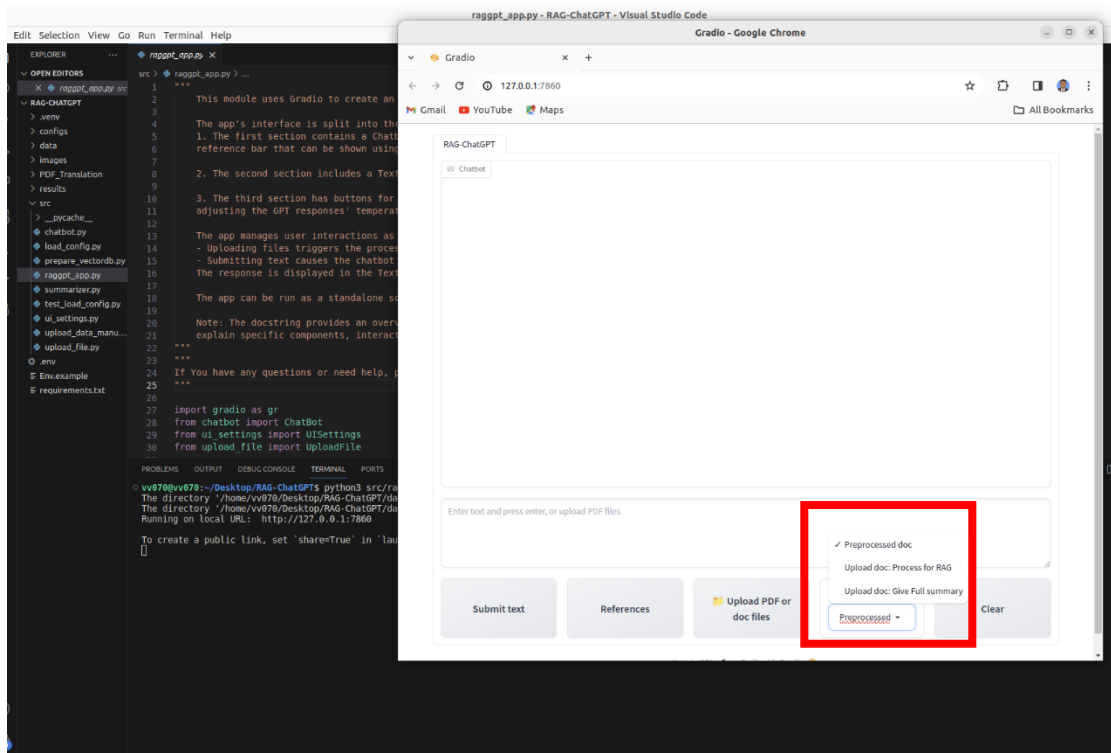


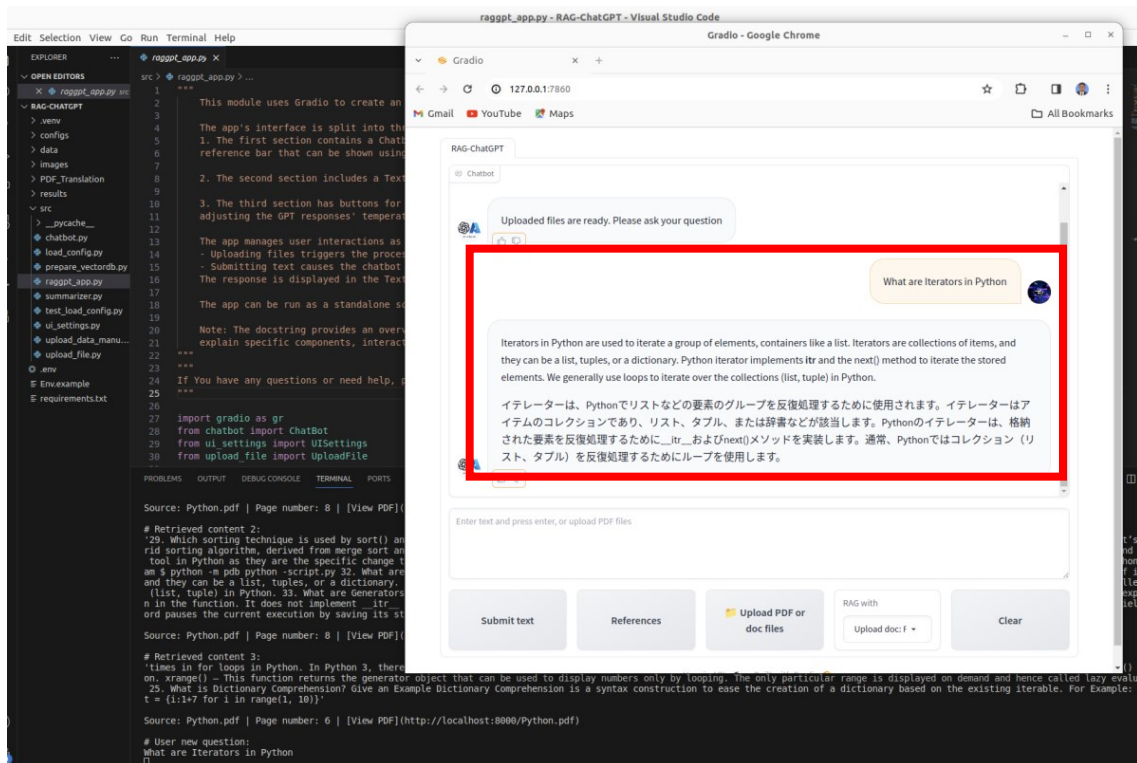
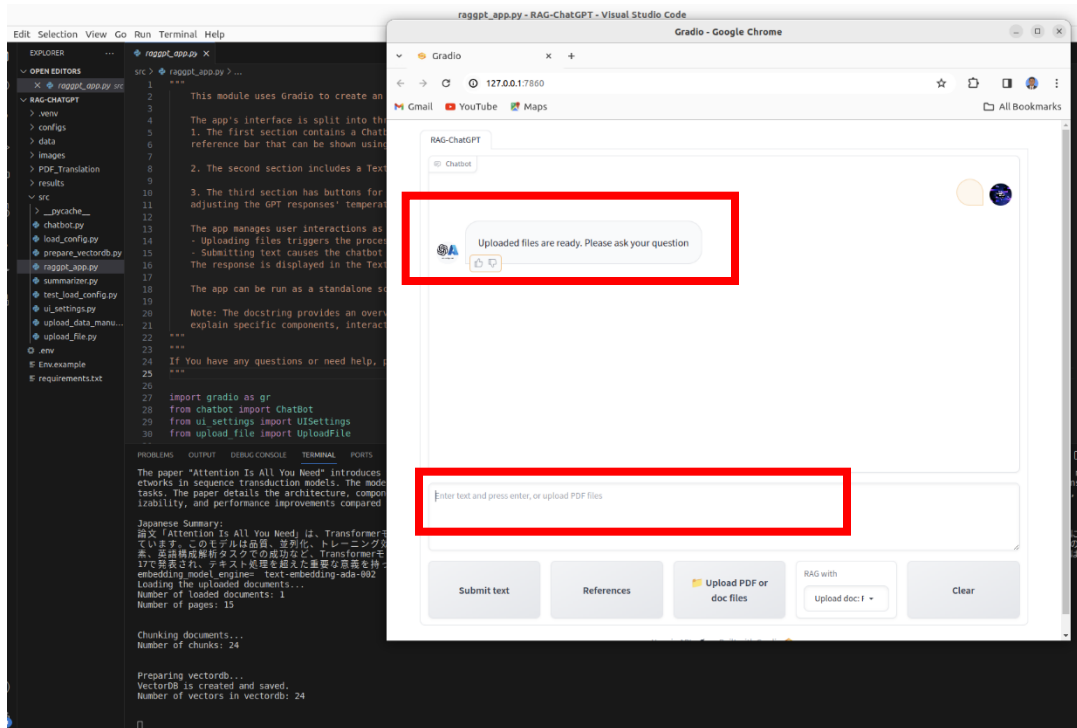
10. Upload the required document for a full summary, and allow some time for the chatbot to display the results automatically.

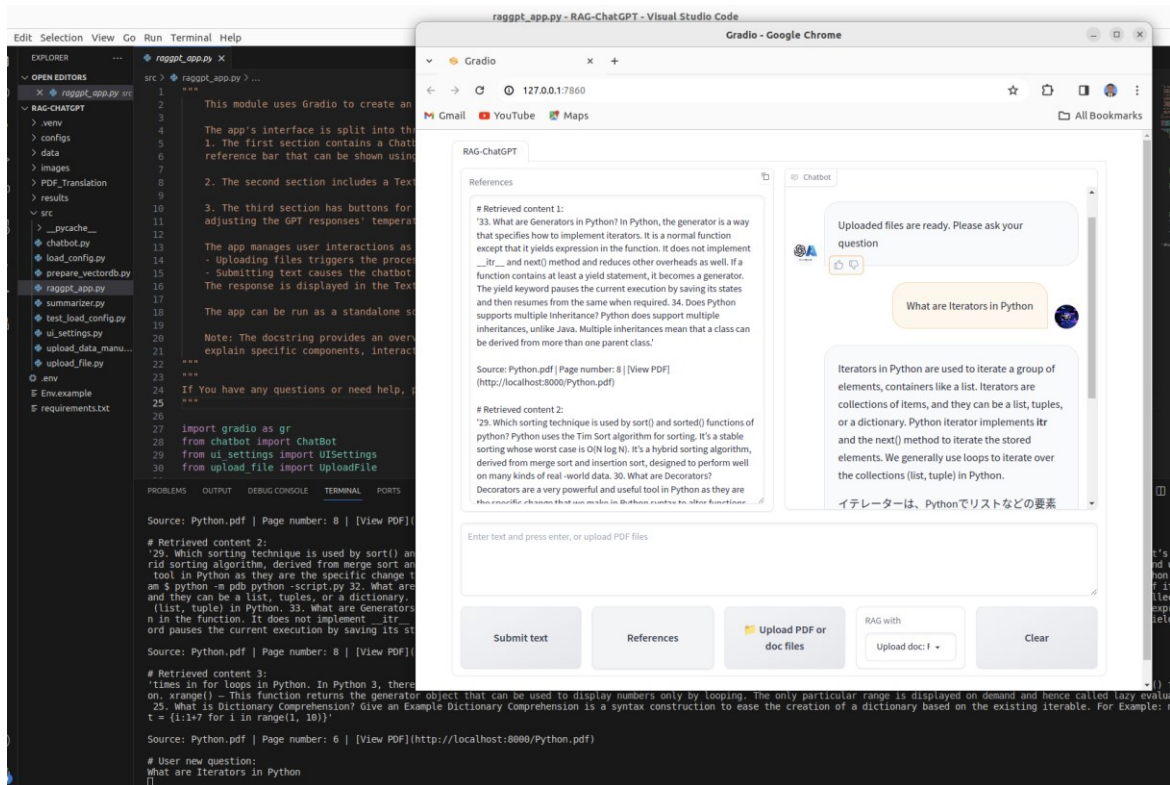
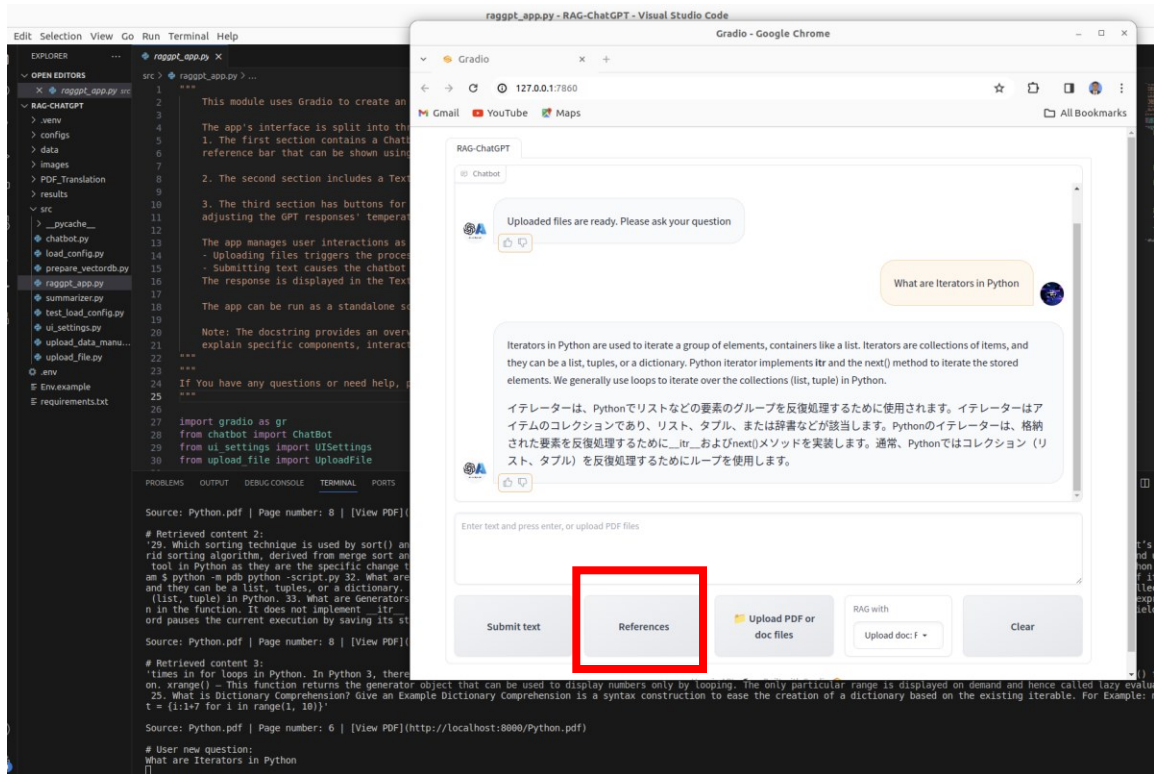




11. Use the "Upload doc: Process for RAG" feature to upload your own document and make inquiries regarding it. The chatbot will display its answer with reference.







12. Finally, complete the project.

✓ Document Translation Project

In the RAG-GPT folder, there is a folder named pdf_Translation. Inside this folder, you can find another folder named Translated_PDFs. This is where you should put all the documents that you want to have translated into Japanese. To run the program, open the terminal and use the following command:

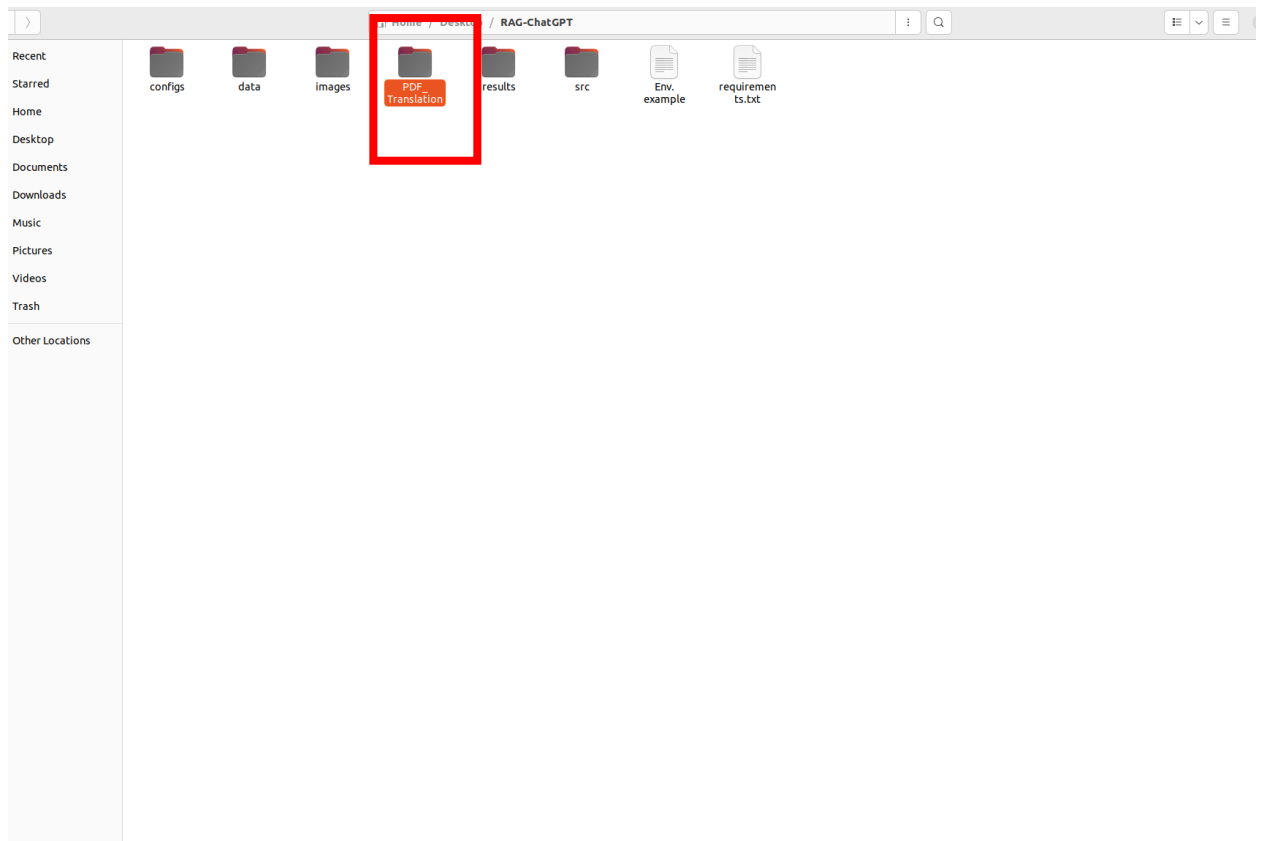
```
python PDF_Translation/Translation.py
```

Once the program has finished running, check the Translated_PDFs folder and you will find the translated document file.

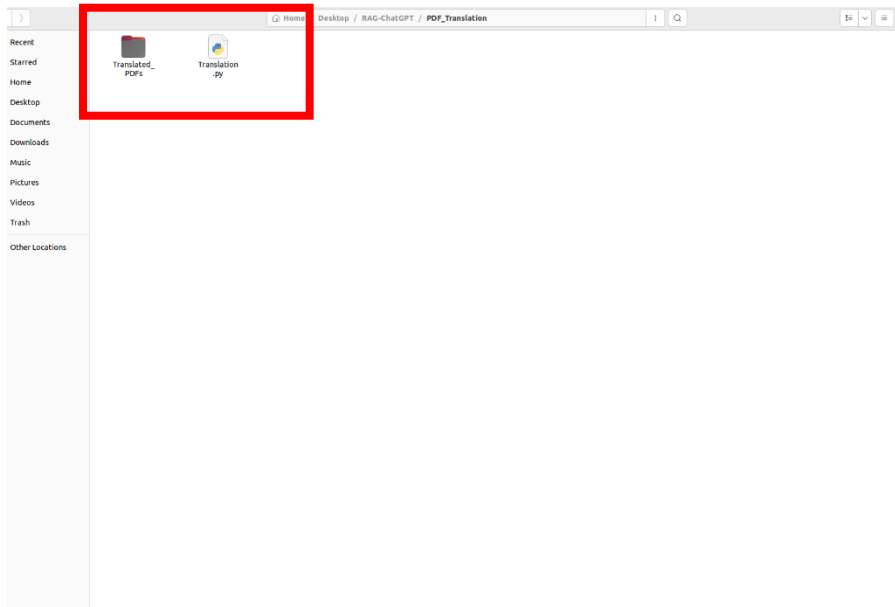
Running Instruction:

To translate your English documents into Japanese using RAG-chatGPT, please follow the steps below:

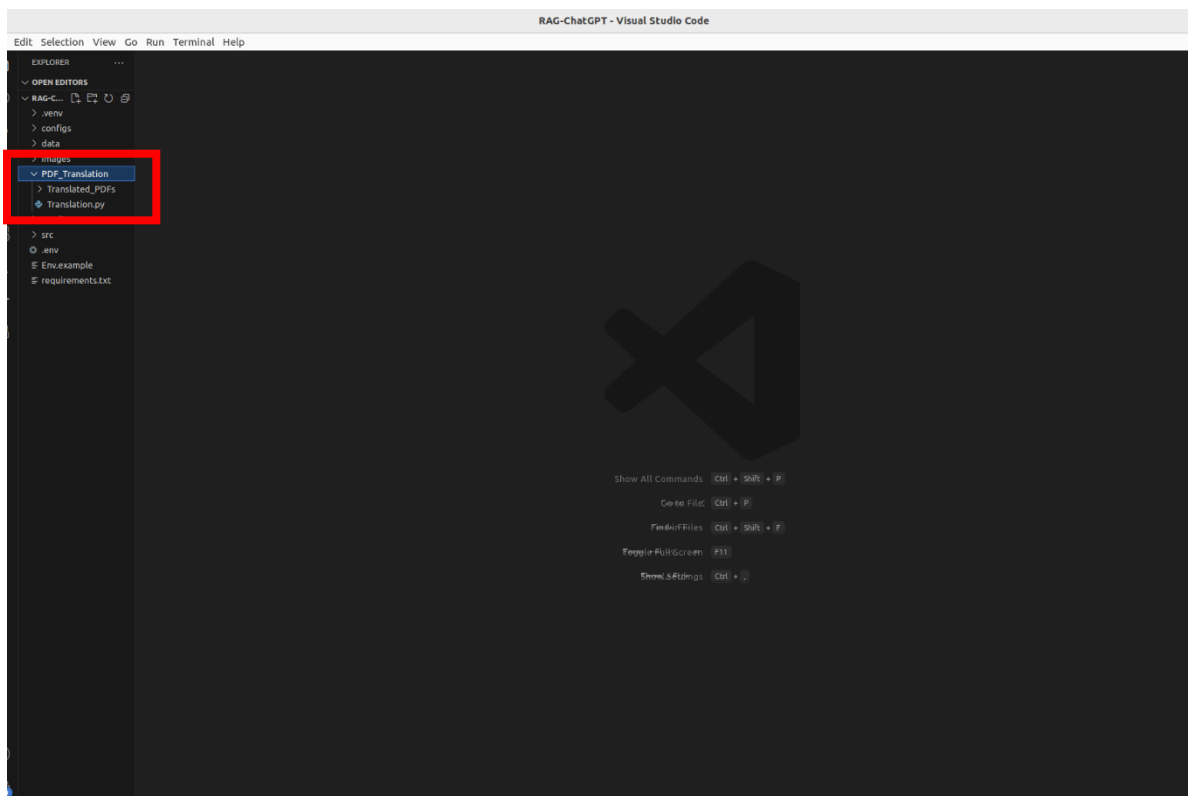
1. Access the RAG-chatGPT system and navigate to the PDF_Translation folder.



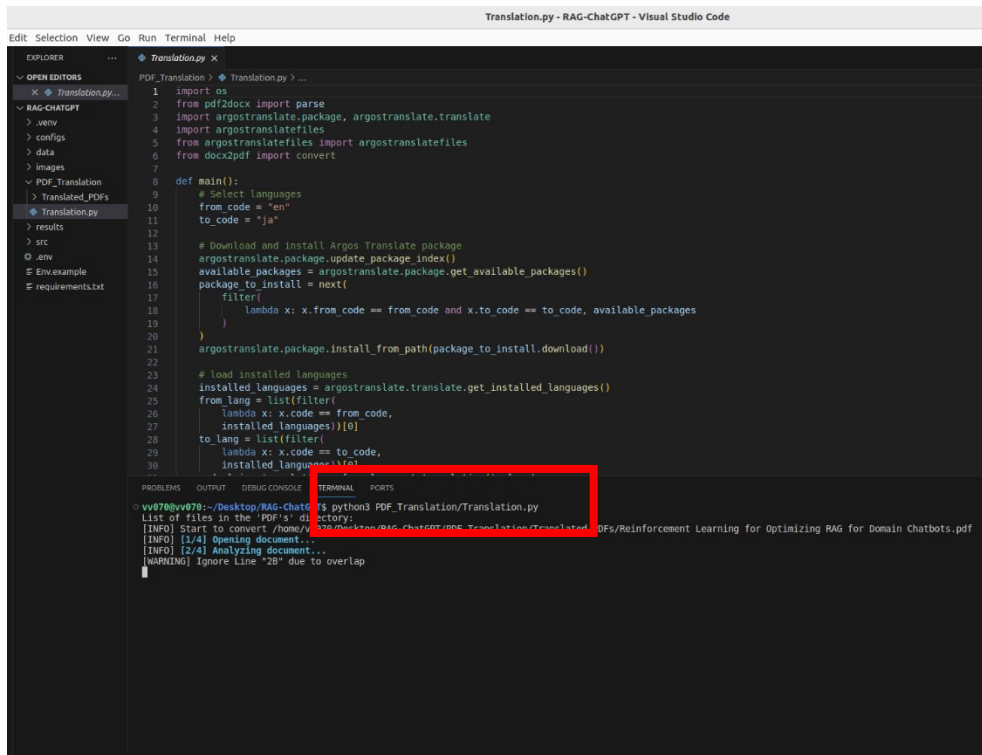
2. Within the PDF_Translation folder, locate the Translated_PDFs folder and place all documents that require translation into this folder.



3. Open the PDF_Translation folder in VS Code IDE.



4. Access the Translation.py program.



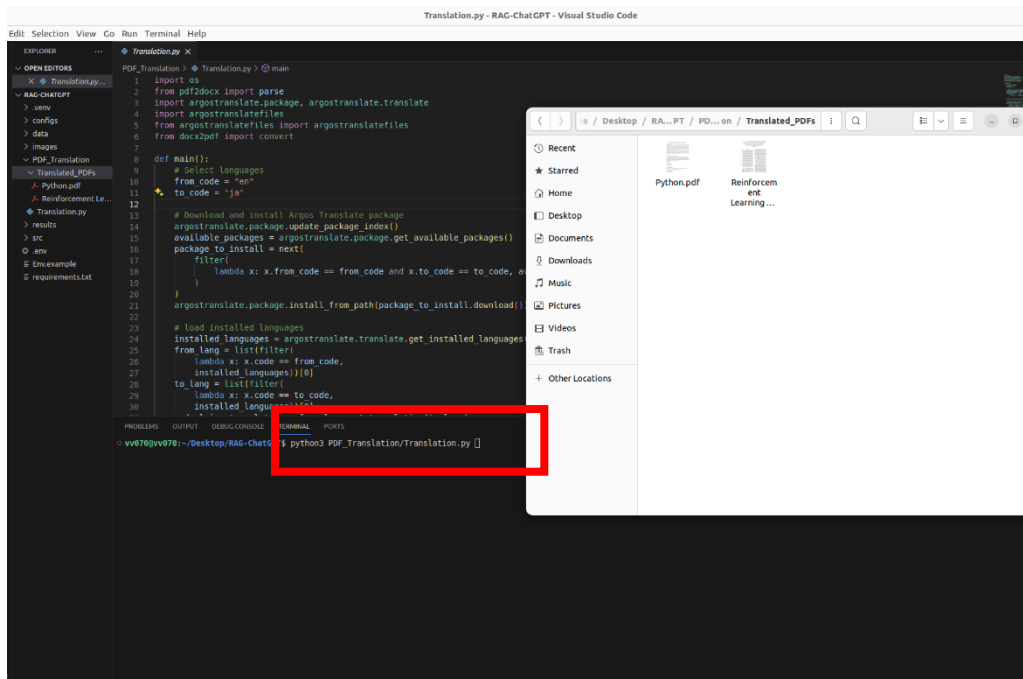
The screenshot shows the Visual Studio Code interface with the 'Translation.py' file open in the editor. The file contains Python code for translating PDFs using Argos Translate. The terminal at the bottom shows the command `python3 PDF_Translation/Translation.py` being executed, with output indicating the start of the conversion process and a warning about a line overlap.

```
1 import os
2 from pdf2docx import parse
3 import argostranslate.package, argostranslate.translate
4 import argostranslatefiles
5 from argostranslatefiles import argostranslatefiles
6 from docx2pdf import convert
7
8 def main():
9     # Select languages
10    from_code = "en"
11    to_code = "ja"
12
13    # Download and install Argos Translate package
14    argostranslate.package.update_package_index()
15    available_packages = argostranslate.package.get_available_packages()
16    package_to_install = next(
17        filter(
18            lambda x: x.from_code == from_code and x.to_code == to_code, available_packages
19        )
20    )
21    argostranslate.package.install_from_path(package_to_install.download())
22
23    # Load installed languages
24    installed_languages = argostranslate.translate.get_installed_languages()
25    from_lang = list(
26        filter(
27            lambda x: x.code == from_code,
28            installed_languages
29        ))[0]
30    to_lang = list(
31        filter(
32            lambda x: x.code == to_code,
33            installed_languages
34        ))[0]
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
vv070@vv070:~/Desktop/RAG-ChatGPT$ python3 PDF_Translation/Translation.py
List of files in the 'PDF's' directory:
[INFO] Start to convert /home/vv070/Desktop/RAG-ChatGPT/PDFs/Reinforcement Learning for Optimizing RAG for Domain Chatbots.pdf
[INFO] [1/4] opening document...
[INFO] [2/4] Analyzing document...
[WARNING] Ignore Line "2B" due to overlap
```

5. Execute the program from the terminal.



The screenshot shows the Visual Studio Code interface with the 'Translation.py' file open in the editor. The file contains Python code for translating PDFs using Argos Translate. The terminal at the bottom shows the command `python3 PDF_Translation/Translation.py` being executed. A file explorer window is overlaid on the right side of the terminal, showing the contents of the 'Translated_PDFs' directory, which includes files named 'Python.pdf' and 'Reinforcement Learning.pdf'.

```
1 import os
2 from pdf2docx import parse
3 import argostranslate.package, argostranslate.translate
4 import argostranslatefiles
5 from argostranslatefiles import argostranslatefiles
6 from docx2pdf import convert
7
8 def main():
9     # Select languages
10    from_code = "en"
11    to_code = "ja"
12
13    # Download and install Argos Translate package
14    argostranslate.package.update_package_index()
15    available_packages = argostranslate.package.get_available_packages()
16    package_to_install = next(
17        filter(
18            lambda x: x.from_code == from_code and x.to_code == to_code, a
19        )
20    )
21    argostranslate.package.install_from_path(package_to_install.download())
22
23    # Load installed languages
24    installed_languages = argostranslate.translate.get_installed_languages()
25    from_lang = list(
26        filter(
27            lambda x: x.code == from_code,
28            installed_languages
29        ))[0]
30    to_lang = list(
31        filter(
32            lambda x: x.code == to_code,
33            installed_languages
34        ))[0]
```

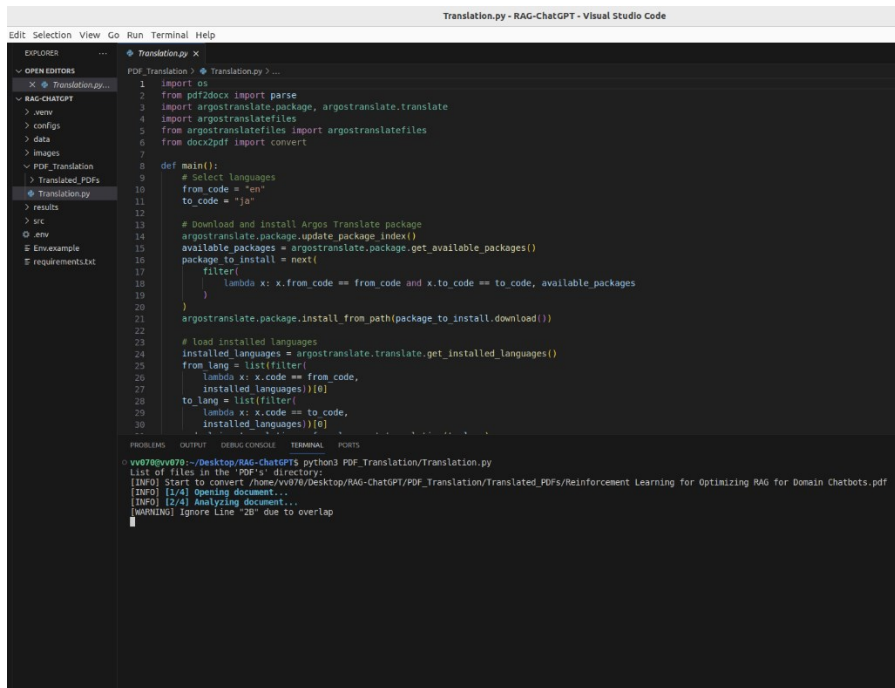
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
vv070@vv070:~/Desktop/RAG-ChatGPT$ python3 PDF_Translation/Translation.py
```

File Explorer: / Desktop / RA...PT / PD...on / Translated_PDFs

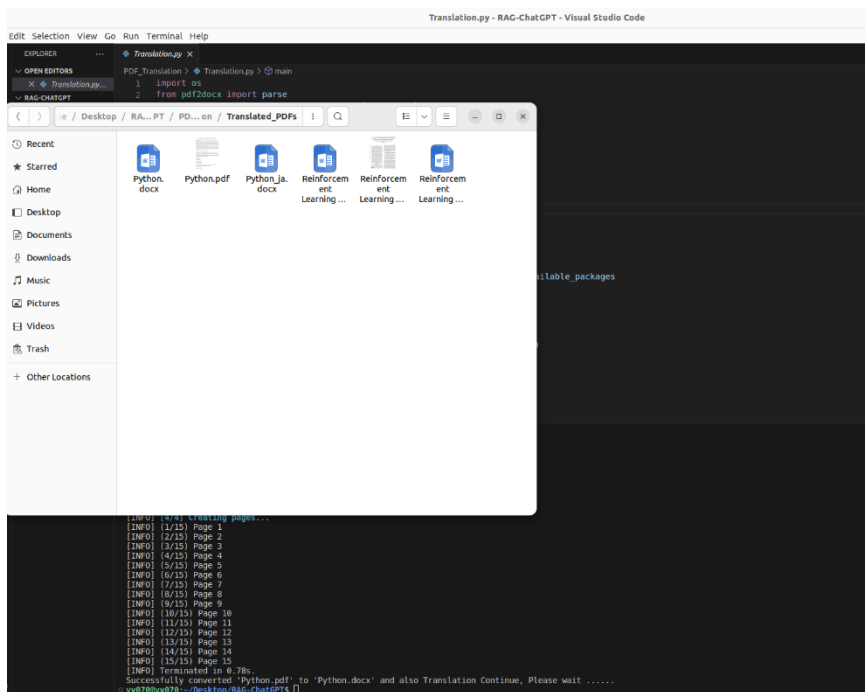
- Python.pdf
- Reinforcement Learning.pdf

6. Please wait while the program automatically translates your English documents into Japanese.



The screenshot shows the Visual Studio Code interface with the file 'Translation.py' open in the editor. The file contains Python code for translating PDFs using the Argos Translate API. The code includes imports for 'os', 'pdf2docx', 'parse', 'argostranslate.package', 'argostranslate.translate', 'argostranslatefiles', and 'docx2pdf'. The 'main' function selects languages (from 'en' to 'ja'), updates the package index, filters available packages, and installs the necessary language package. It then loads the installed languages and filters them to match the target language. The terminal output shows the execution of the script, including the command 'python3 PDF_Translation/Translation.py' and the resulting log messages: 'List of files in the "/>

7. Once the program has completed, open the Translated_PDFs folder to find your translated documents.



The screenshot shows a Windows File Explorer window titled 'Translated_PDFs'. The address bar shows the path 'C:\Desktop\RAAG-PT\PD...on\Translated_PDFs'. The left sidebar shows the 'Recent' tab. The main area displays several files: 'python.docx', 'Python.pdf', 'python.ja.docx', 'Reinforcem...ent Learning...', 'Reinforcem...ent Learning...', and 'Reinforcem...ent Learning...'. Below the file explorer, the terminal output from the previous step is visible, showing the completion of the translation process and the successful conversion of 'Python.pdf' to 'Python.docx'.

By following these steps, you will be able to easily and accurately translate your English documents into Japanese using the RAG-chatGPT system.