CoRE Stack

Winter Internship

IIT Delhi

Abhinav Kumar Pandey

Documentation

Acknowledgment

I am immensely grateful to Prof. Aaditeshwar Seth of the Department of Computer Science and Engineering, IIT Delhi, under whom I have done this Internship and learned so much. I would also like to express my heartfelt thanks to Shivani Ma'am and the entire ICTD lab who have kept my spirits up and have always been ready to help me. Last but not in anyway the least, I am eternally grateful to my parents who have supported me throughout thick and thin in this project and beyond.

Contents

Abstract

The Commoning for Resilience and Equality (CoRE) stack is a collaborative effort with several academic research institutions, civil society organizations working on ecological sustainability and rural livelihood, innovation catalysts, and technology-led social enterprises. Being designed as a Digital Public Good, the CoRE stack enables an open-access co-creation network to innovate and scale digital technology solutions for ecosystem sustainability. Within the overall realm of CoRE Stack, my work in the ICTD lab focuses mainly on leveraging LLMs to analyze data in different forms, be it HTML, Excel, GeoJSON, Vector and Raster images. Eventually based on the abilities, we hope to one day be able to develop a chatbot which people in rural areas can access to get information about their area. Be it deforestation amounts, Land Use Land Cover(LULC) data, or soil profile for the area, the user should should be able to access them in a matter of minutes.

Chapter 1

Introduction

A watershed, also known as a drainage basin or catchment area, is an area of land where all the water that falls or drains into it ultimately drains into a common outlet, such as a river, lake, or ocean. This includes surface water runoff, precipitation, and even underground water flow. Watersheds play a crucial role in the hydrological cycle, influencing the distribution and movement of water across the landscape.

Based on the size, the hydrological units are termed as basin, sub-basin, catchment, sub-catchment, watershed, sub-watershed and micro-watershed. India's Water Resources Information System (WRIS) divides the Indian subcontinent into 25 major river basins with major river basin of Ganga-Brahmaputra-Meghna followed by Indus, Mahanadi, Godavari and Krishna.

These sub divisions are represented in two forms - Vector and Raster Images. Raster and vector files are the two most popular formats used for visual content. They represent images in very different ways, so there's a lot to consider when deciding which one to use.

Raster files are images built from pixels — tiny colour squares that, in great quantity, can form highly detailed images such as photographs. The more pixels an image has, the higher quality it will be, and vice versa. The number of pixels in an image depends on the file type (for example, JPEG, GIF or PNG).

Vector files use mathematical equations, lines and curves with fixed points on a grid to produce an image. There are no pixels in a vector file. A vector file's mathematical formulae capture shape, border and fill colour to build an image. Because the mathematical formula recalibrates to any size, you can scale a vector image up or down without affecting its quality.

All the files analyzed in this document can be found at the GramVaani Geoserver .

The colab files and images used in this project are located in a public repository hosted by Github. For the analysis present in this document, I have used GPT-4o-mini through API key.

The use of Large Language Models (LLMs) in geospatial analysis is a recent development, spurred by the advancements in LLMs' ability to reason and understand semantic information. The development of leveraging LLMs for geospatial analysis has been rapid, with increasing sophistication in approaches. The field is still in its nascent stage, with ongoing research addressing challenges like the uncertainty of LLM outputs, the need for specialized training data in GIS, and the development of more robust and adaptable frameworks. Nonetheless, the progress made so far points towards a promising future for LLMs as powerful tools in geospatial research and applications.

Chapter 2

Analysis of Tabuar Data by LLMs

1 Description of the Excel file

For my first analysis, I provided GPT-4o-mini with an Excel file labeled as "Masalia_data". This file had several sheets with different type of data on each one. The way I proceeded was to first define four functions, one which loads the Excel file, one which uses the Excel file to provide context to the LLM, the query to the LLM, and finally the main function which executes the LLM call. A representation of the first sheet is shown in Fig.2.1 and the function pipeline is shown as a flow chart in Fig.2.2.

UID	area_in_hec	terrainCluster_ID	Terrain_Description	% of area hill_slope	% of area plain_area	% of area ridge_area	% of area slopy_area	% of area valley_area
12_305795	771.7283439	3	Broad Plains and Slopes	11.13133132	51.01650844	7.685622448	17.24519018	12.92134761
12_307609	2116.443393	3	Broad Plains and Slopes	4.597655926	60.34905751	7.144276834	24.73331744	3.175692283
12_308492	1410.86473	3	Broad Plains and Slopes	3.435826316	60.82968506	5.580878356	26.63604269	3.517567582
12_308948	1618.368478	1	Mostly Plains	1.036581458	81.10297411	2.609962542	10.80531189	4.445169997
12_309695	1554.517784	1	Mostly Plains	0.102937308	91.21269304	5.630214303	0.81528046	2.238874889
12_312558	1850.151769	1	Mostly Plains	1.139574374	78.5543311	2.510694645	16.21628195	1.579117937
12_313014	1961.218626	1	Mostly Plains	0.1	80.58001458	4.564092016	12.27097908	2.446518999
12_313277	760.2056969	1	Mostly Plains	0.329558616	78.74155972	1.140672985	19.30254793	0.584160751
12_314843	1620.686474	3	Broad Plains and Slopes	4.211006838	66.03146704	5.762059814	19.50627148	4.489194823
12_315140	2457.589854	3	Broad Plains and Slopes	4.3135216	58.51832292	7.265600805	26.10417438	3.798380299
12_317063	1782.290065	1	Mostly Plains	2.06408291	75.32795853	2.262093402	18.52487842	1.820986729
12_318533	2342.956564	1	Mostly Plains	0.804574826	76.15909258	8.009838118	12.27945527	2.747039204
12_318789	707.307987	3	Broad Plains and Slopes	6.344315948	60.52906758	7.611164907	21.49531005	4.020141518
12_319549	2289.498236	1	Mostly Plains	0.186632809	78.5696694	6.377646714	11.4909067	3.375144381
12_319785	632.0660833	3	Broad Plains and Slopes	9.593165773	45.41874127	11.60675921	25.0773509	8.303982846
12_320259	649.7575988	1	Mostly Plains	0.07592989	82.47054401	3.390070934	13.51256624	0.550888923
12_320690	728.8831499	1	Mostly Plains	0.013539865	84.16234144	2.063920237	13.51031238	0.249886095
12_320805	2286.799688	1	Mostly Plains	0.190609192	77.8526656	6.540824076	13.47771444	3.938186696
12_321768	788.1564991	1	Mostly Plains	0	76.30800418	3.62534616	18.585984	1.480575656
12_322030	1635.946028	1	Mostly Plains	0.066137631	75.74790587	4.190259891	16.00456143	3.99113518
12_322313	1073.765148	1	Mostly Plains	1.266858431	79.88045309	2.374405725	15.41209644	1.066186308

Figure 2.1: Structure of the Excel file

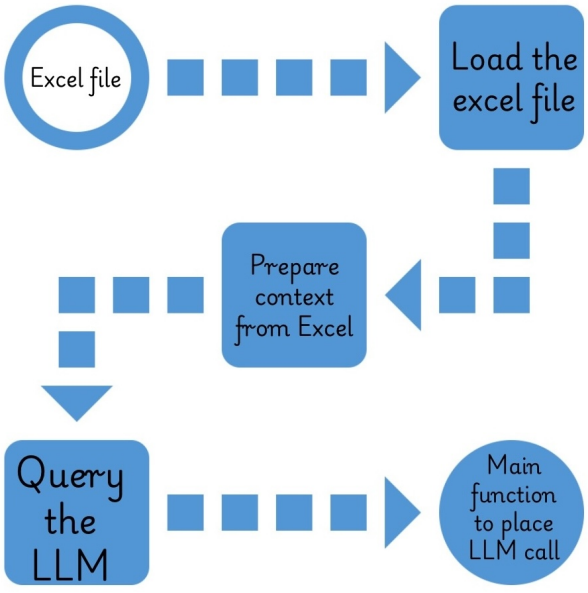


Figure 2.2: function pipeline for Excel analysis

2 Defining the functions

The Code snippets displayed in Fig.2.3, 2.4, and 2.5 show how I have defined the functions.

```python
def load_excel_file(file_path):
    """
    Loads an Excel file and returns all sheet names and data from each sheet.
    """
    xls = pd.ExcelFile(file_path)
    sheet_names = xls.sheet_names

    sheets_data = {}
    for sheet in sheet_names:
        df = pd.read_excel(file_path, sheet_name=sheet)
        sheets_data[sheet] = df

    return sheet_names, sheets_data
```

Figure 2.3: function to load the Excel file

```python
def prepare_context_from_excel(sheet_names, sheets_data):
    """
    Prepares a text-based context from the Excel file's data.
    """
    context = "Excel File Content:\n\n"
    for sheet in sheet_names:
        context += f"Sheet Name: {sheet}\n"
        context += f"Columns: {', '.join(sheets_data[sheet].columns)}\n"
        context += f"First 5 Rows:\n{sheets_data[sheet].head().to_string(index=False)}\n\n"
    return context

def ask_question_to_llm(context, question):
    """
    Sends the context and user question to the LLM and retrieves the answer.
    """
    messages = [
        {"role": "system", "content": "You are an expert at analyzing and summarizing data from Excel files."},
        {"role": "user", "content": f"Context:\n{context}\n\nQuestion: {question}"}
    ]
    response = openai.ChatCompletion.create(
        model="gpt-4o-mini",
        messages=messages,
        max_tokens=1000,
        temperature=0.7
    )
    return response.choices[0].message["content"]
```

Figure 2.4: functions to generate context and ask question to the LLM

```python
# Main function
def main(file_path, question):
    sheet_names, sheets_data = load_excel_file(file_path)
    context = prepare_context_from_excel(sheet_names, sheets_data)

    if len(context) > 100000:
        context = context[:100000]

    answer = ask_question_to_llm(context, question)
    return answer
```

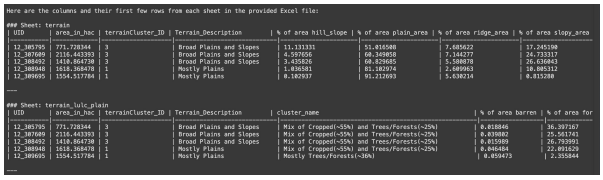
Figure 2.5: Main function

3 Different Excel functionalities through LLMs

3.A Displaying columns and rows from the Excel

Now what we want to explore is whether the LLM can display the Excel columns which it has analyzed.

```
file_path = '/content/Masalia_data.xlsx'
question = "Can you display the columns and
↪  their first few rows from the Excel file?"
answer = main(file_path, question)
print(answer)
```



Here are the columns and their first few rows from each sheet in the provided Excel file:

Sheet: terrain

UID	area_in_hac	terrainCluster_ID	Terrain_Description	% of area hill_slope	% of area plain_area	% of area ridge_area	% of area slopy_area
12_305795	771.728344	3	Broad Plains and Slopes	11.131331	51.016508	7.685622	17.245198
12_307609	2116.463393	3	Broad Plains and Slopes	4.597656	60.349058	7.144277	24.733317
12_308402	1410.864738	3	Broad Plains and Slopes	3.435826	60.829685	5.608878	26.630043
12_308948	1618.366478	1	Mostly Plains	1.836581	81.102974	2.608963	10.885312
12_309695	1554.517784	1	Mostly Plains	0.102937	91.271093	5.630214	0.815208

Sheet: terrain_luic_plain

UID	area_in_hac	terrainCluster_ID	Terrain_Description	cluster_name	% of area barren	% of area for
12_305795	771.728344	3	Broad Plains and Slopes	Mix of Cropped(~55%) and Trees/Forests(~25%)	0.818846	36.397167
12_307609	2116.463393	3	Broad Plains and Slopes	Mix of Cropped(~55%) and Trees/Forests(~25%)	0.839882	25.561741
12_308402	1410.864738	3	Broad Plains and Slopes	Mix of Cropped(~55%) and Trees/Forests(~25%)	0.815989	26.793991
12_308948	1618.366478	1	Mostly Plains	Mix of Cropped(~55%) and Trees/Forests(~25%)	0.846484	22.891629
12_309695	1554.517784	1	Mostly Plains	Mostly Trees/Forests(~36%)	0.059473	2.355844

Figure 2.6: Output which displays the Excel file

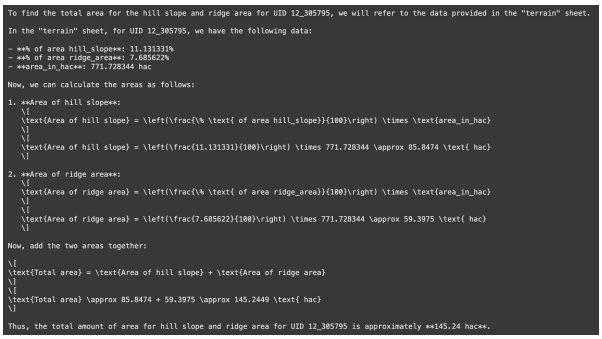
As we can see in Fig.2.6, the LLM is able to display the columns of Excel files with excellent accuracy.

3.B Complex functions

In this subsection, we explore if the LLM can perform more complex functions like addition and multiplication albiet simultaneously. What I am going to ask the LLM to compute is the total area which comes under hill slope and ridge area for a particular UID from the Excel file. For this, it would have to access the percentages of hill slope and ridge area, multiply them independently to the total area and then add them.

```
file_path = '/content/Masalia_data.xlsx'
question = "Can you tell me the total amount
↪  of area hill slope and ridge area for UID
↪  12_305795?"
answer = main(file_path, question)
print(answer)
```

As we can see in Fig.2.7, the LLM gives us the whole calculation for the process and is able to process the total area.



To find the total area for the hill slope and ridge area for UID 12_305795, we will refer to the data provided in the "terrain" sheet.

In the "terrain" sheet, for UID 12_305795, we have the following data:

- **% of area hill_slope**: 11.131331%
- **% of area ridge_area**: 7.685622%
- **area_in_hac**: 771.728344 hac

Now, we can calculate the areas as follows:

1. **Area of hill slope**:
   \[
   \text{Area of hill slope} = \left(\frac{\% \text{ of area hill_slope}}{100}\right) \times \text{area_in_hac}
   \]
   \[
   \text{Area of hill slope} = \left(\frac{11.131331}{100}\right) \times 771.728344 \approx 85.8474 \text{ hac}
   \]

2. **Area of ridge area**:
   \[
   \text{Area of ridge area} = \left(\frac{\% \text{ of area ridge_area}}{100}\right) \times \text{area_in_hac}
   \]
   \[
   \text{Area of ridge area} = \left(\frac{7.685622}{100}\right) \times 771.728344 \approx 59.3975 \text{ hac}
   \]

Now, add the two areas together:

\[
\text{Total area} = \text{Area of hill slope} + \text{Area of ridge area}
\]
\[
\text{Total area} \approx 85.8474 + 59.3975 \approx 145.2449 \text{ hac}
\]

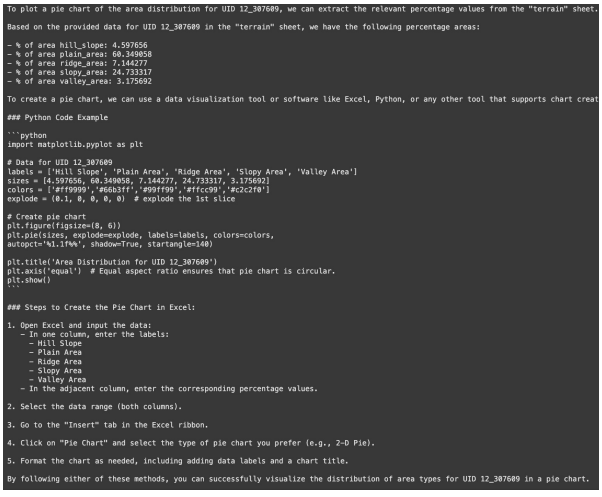
Thus, the total amount of area for hill slope and ridge area for UID 12_305795 is approximately **145.24 hac**.

Figure 2.7: LLM ouput for total area

3.C Plotting a pie chart

Now we are going to explore whether the LLM can produce visual outputs like a Pie chart based on the Excel data that it has.

```
file_path = '/content/Masalia_data.xlsx'
question = "Can you plot a pie chart of hill
↪  slope, plane area, ridge area, slopy area
↪  and valley area for UID 12_307609"
answer = main(file_path, question)
print(answer)
```



To plot a pie chart of the area distribution for UID 12_307609, we can extract the relevant percentage values from the "terrain" sheet.

Based on the provided data for UID 12_307609 in the "terrain" sheet, we have the following percentage areas:

- % of area hill_slope: 4.597656
- % of area plain_area: 60.349058
- % of area ridge_area: 7.144277
- % of area slopy_area: 24.733317
- % of area valley_area: 3.175692

To create a pie chart, we can use a data visualization tool or software like Excel, Python, or any other tool that supports chart creat

Python Code Example

```python
import matplotlib.pyplot as plt

# Data for UID 12_307609
labels = ['Hill Slope', 'Plain Area', 'Ridge Area', 'Slopy Area', 'Valley Area']
sizes = [4.597656, 60.349058, 7.144277, 24.733317, 3.175692]
colors = ['#ff9999','#66b3ff','#99ff99','#ffcc99','#c2c2f0']
explode = (0.1, 0, 0, 0, 0)  # explode the 1st slice

# Create pie chart
plt.figure(figsize=(8, 6))
plt.pie(sizes, explode=explode, labels=labels, colors=colors,
autopct='%1.1f%%', shadow=True, startangle=140)

plt.title('Area Distribution for UID 12_307609')
plt.axis('equal')  # Equal aspect ratio ensures that pie chart is circular.
plt.show()
```

Steps to Create the Pie Chart in Excel:

1. Open Excel and input the data:
   - In one column, enter the labels:
     - Hill Slope
     - Plain Area
     - Ridge Area
     - Slopy Area
     - Valley Area
   - In the adjacent column, enter the corresponding percentage values.

2. Select the data range (both columns).

3. Go to the "Insert" tab in the Excel ribbon.

4. Click on "Pie Chart" and select the type of pie chart you prefer (e.g., 2-D Pie).

5. Format the chart as needed, including adding data labels and a chart title.

By following either of these methods, you can successfully visualize the distribution of area types for UID 12_307609 in a pie chart.

Figure 2.8: output for pie chart creation

Contrary to our expectations, the LLM is not able to plot the pie chart. Instead it gives us the complete Python code which we can execute to plot the pie chart and the instructions on how to execute it as well. This is the limitation that we will be exploring in this document: whether LLMs are able to generate and analyze images for answers.

Chapter 3

Analysis of data from HTML files

In this chapter, we explore the analysis of data from a .html file. The key challenge in applying LLMs to HTML data analysis lies in effectively bridging the gap between the unstructured nature of HTML and the structured data formats that LLMs are typically trained on.

1 Defining the functions

The HTML file I use over here for my analysis is a Micro-watershed report of Pindwara, Rajasthan. What we want to explore is whether the LLM can analyze and retrieve information from inside the HTML file We use libraries like Pytesseract, BeautifulSoup, BytesIO and Image to analyze the data from the HTML file.

Figure 3.1: functions flow

Here as well, we use five functions. One for extracting content from the HTML, one for processing images, one for preparing context to the LLM, the question pipeline and lastly the main function where we place the LLM call.

2 Function executions

The code snippets provided in Figs. 3.2, 3.3, 3.4, and 3.5 display the code I used to write the functions for the HTML analysis.

```python
def extract_html_content(file_path):
    """
    Extracts text, images, and other data from an HTML file.
    """
    with open(file_path, "r", encoding="utf-8") as file:
        soup = BeautifulSoup(file, "html.parser")


    text_content = soup.get_text(separator="\n", strip=True)


    images = []
    for img_tag in soup.find_all("img"):
        img_src = img_tag.get("src")
        if img_src.startswith("data:image"):
            img_data = base64.b64decode(img_src.split(",")[1])
            images.append(Image.open(BytesIO(img_data)))
        elif img_src.startswith("http"):
            response = requests.get(img_src)
            images.append(Image.open(BytesIO(response.content)))
        else:

            img_path = os.path.join(os.path.dirname(file_path), img_src)
            if os.path.exists(img_path):
                images.append(Image.open(img_path))


    graphs = [img for img in images if "graph" in img.format.lower()]

    return text_content, images, graphs
```

Figure 3.2: function to extract content

```python
def process_images(images):
    """
    Extracts text from images using OCR.
    """
    image_texts = []
    for img in images:
        try:
            text = pytesseract.image_to_string(img)
            image_texts.append(text.strip())
        except Exception as e:
            image_texts.append(f"Error processing image: {str(e)}")
    return image_texts


def prepare_context(text, image_texts):
    """
    Combines textual data and extracted image text into a single context for the LLM.
    """
    context = f"Text Content:\n{text}\n\n"
    for i, img_text in enumerate(image_texts):
        context += f"Image {i+1} Text:\n{img_text}\n\n"
    return context
```

Figure 3.3: functions combining image and text for common context

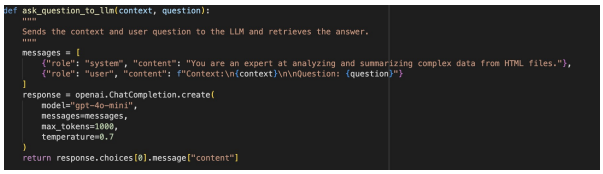


Figure 3.4: Question pipeline

```
# Main function
def main(html_file_path, question):
    text, images = extract_html_content(html_file_path)
    image_texts = process_images(images)
    context = prepare_context(text, images, image_texts)

    if len(context) > 100000:
        context = context[:100000]

    answer = ask_question_to_llm(context, question)
    return answer
```

Figure 3.5: Main function

3 Text based queries

First We will explore whether the LLM is able to answer questions from the text present in the HTML file.

```
html_file_path =
"/content/micro_watershed_report_pindwara.html"
question = "Can you display the distribution
↪  of land use types in microwatersheds?"
answer = main(html_file_path, question)
print(answer)
```

Fig 3.6 showcases that the LLM is able to answer queries based on text present in the LLM.

Figure 3.6: Output

4 Image based queries

Now what we want to explore is whether the LLM can answer queries base on images in the LLM or whether it can display images from the HTML file.

4.A Query for analysis based on image in HTML file.

First, we are placing a query to the LLM based on a figure of the graph of rainfall in mm vs time in years.

```
html_file_path =
"/content/micro_watershed_report_pindwara.html"
question = "What is the Rainfall trends for
↪  for UID 14_57832"
answer = main(html_file_path, question)
print(answer)
```

As displayed in Fig.3.7 the LLM gives us a hallucinated answer comprising of the text analysed from above the graph but is not able to retrieve any information from the actual graph itself.

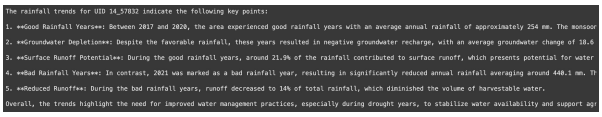


Figure 3.7: Failure to do image analysis

4.B Query to display image present in the HTML file

Since we have seen that the LLM is not able to analyse images present in the HTML file, we will now explore whether the LLM is able to display the same graphical image present in the HTML file.

```
html_file_path =
"/content/micro_watershed_report_pindwara.html"
question = "Can you display the rainfall
↪  trends for UID 14_57832?"
answer = main(html_file_path, question)
print(answer)
```

Unfortunately, the LLM is not able to this either which leads us to conclude that the LLM is not able to perform image analysis or process images from an HTML file.

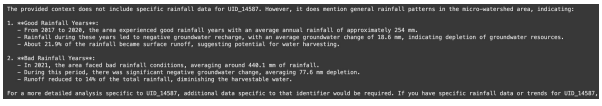


Figure 3.8: Failure to display image from HTML file.

Chapter 4

Geo-spatial analysis by LLMs

In this final chapter, we will explore whether the LLM can analyze and display geo-spatial data.

1 Plotting and analyzing the intersection of shapefiles

I will be using the angul_angul and the angul_drought shapefiles for the first part of my analysis. I will first plot their intersection in QGIS, then display it by Python, then query the LLM on whether it is able to plot the intersection. I have downloaded these files in .zip format from the geoserver which I had mentioned in the Introduction chapter.

1.A Defining the functions

Figure 4.1: Function pathway

As we can see in Fig4.1 we first unzip the files, store the shapefiles in a separate folder and access them, read them, prepare context based on these files, and lastly, place the LLM call for analysis of these shapefiles.

```python
# Function to unzip files
def unzip_files(zip_path, extract_to):
    with zipfile.ZipFile(zip_path, 'r') as zip_ref:
        zip_ref.extractall(extract_to)

# Paths to ZIP files (replace with your file paths)
zip_file_1 = "/content/angul_drought.zip"
zip_file_2 = "/content/anugul_anugul.zip"

# Directories to extract the shapefiles
extract_dir_1 = "extracted_shapefiles_1"
extract_dir_2 = "extracted_shapefiles_2"

# Unzip the files
unzip_files(zip_file_1, extract_dir_1)
unzip_files(zip_file_2, extract_dir_2)

# Locate shapefiles in the directories
shapefile_1 = [f for f in os.listdir(extract_dir_1) if f.endswith('.shp')][0]
shapefile_2 = [f for f in os.listdir(extract_dir_2) if f.endswith('.shp')][0]

# Read the shapefiles
gdf1 = gpd.read_file(os.path.join(extract_dir_1, shapefile_1))
gdf2 = gpd.read_file(os.path.join(extract_dir_2, shapefile_2))
```

Figure 4.2: Unzip files and read shapefiles

```python
# Prepare data for LLM
def prepare_geodata_context(gdf1, gdf2):
    """
    Prepare a text-based context from geospatial data for the LLM.
    """
    context = (
        f"Dataset 1: {len(gdf1)} geometries\n"
        f"Dataset 2: {len(gdf2)} geometries\n"
    )
    return context

# LLM analysis and plotting request
context = prepare_geodata_context(gdf1, gdf2)
prompt = (
    f"The context describes two geographical datasets. Use the following data to compute and plot their intersection:\n\n"
    f"{context}\n\n"
    "Please compute the intersection of these two datasets and plot the results, ensuring proper visualization of all geometries."
)

response = openai.ChatCompletion.create(
    model="gpt-4o-mini",
    messages=[
        {"role": "system", "content": "You are a geospatial analyst capable of computing and plotting shapefile intersections."},
        {"role": "user", "content": prompt}
    ]
)
```

Figure 4.3: Context preparation and LLM call

Fig.4.2 and 4.3 contain the code snippets which I used to access the shapefiles and process the intersection.

1.B Plotting the intersection with QGIS

Firstly I will plot the intersection in QGIS. Plotting the intersection of two shapefiles involves a simple library call. Fig.4.4 displays the angul_angul file in orange, the angul_drought in gray and the intersection in purple.

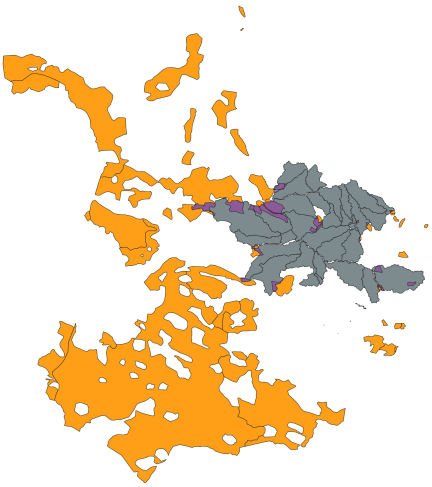


Figure 4.4: Intersection in QGIS

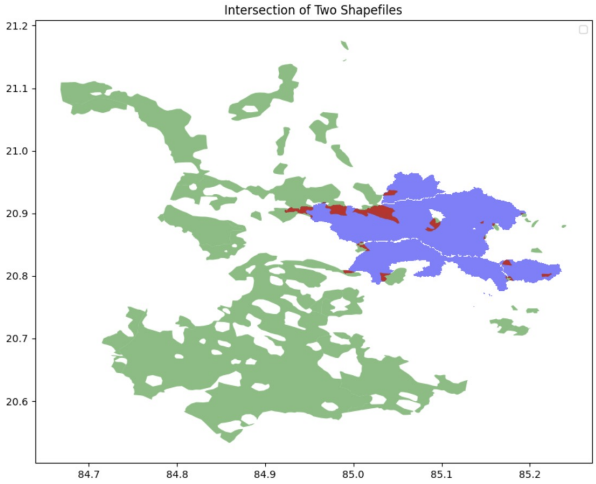


Figure 4.5: Intersection in Python

1.C Plotting the intersection in Python

To verify our diagram constructed in Python, I will plot the intersection in Python as well.

```
# Compute the intersection
intersection = gpd.overlay(gdf1, gdf2,
↪  how='intersection')

# Plot the result
plt.figure(figsize=(10, 8))
ax = plt.gca()
gdf1.plot(ax=ax, color='blue', alpha=0.5,
↪  label='Shapefile 1')
gdf2.plot(ax=ax, color='green', alpha=0.5,
↪  label='Shapefile 2')
if not intersection.empty:
    intersection.plot(ax=ax, color='red',
    ↪  alpha=0.7, label='Intersection')
else:
    print("No intersection found.")
ax.set_aspect('auto')
plt.legend()
plt.title('Intersection of Two Shapefiles')
plt.show()
```

As we can observe in Fig.4.5, the intersection has been plotted successfully with angul_angul in green, angul_drought in purple and the intersection in red.

1.D Plotting the intersection with the LLM

Now that we know how the final plot should look like, we will query the LLM to plot the intersection

```
# Output the LLM's response
print("Response from LLM:")
print(response['choices'][0]['message']
['content'])
```

Unlike what we expect, the LLM is not able to plot the intersection. Instead it gives us the Python code to plot the intersection.

Figure 4.6: Failure of the LLM to plot the intersection.

2 Plotting and analyzing the intersection of GeoJSON files

For our last analysis, we will try to plot the intersection of GEOJSON files of angul boundaries

and micro-watersheds(mws) of angul. These files can be found in the Github repository I mentioned in the Introduction chapter.

2.A Defining the functions

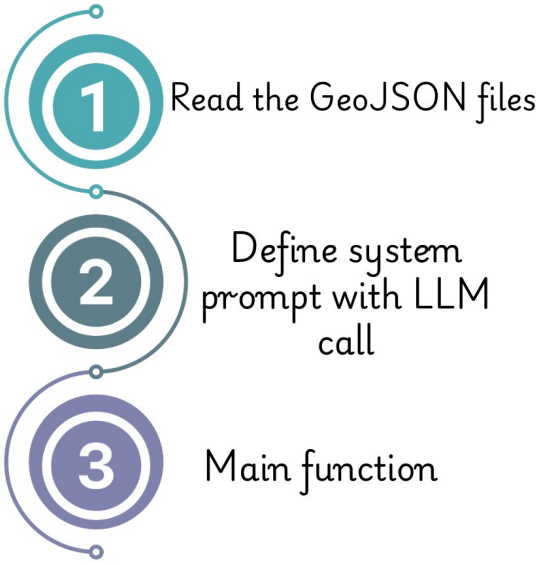


Figure 4.7: Function pathway

As we can see in Fig.4.7 we will define functions in such a manner that they read the GeoJSON files, defined the system propmts with the LLM call and then execute the code in the main function. Figs.4.8 and 4.9 contain the code snippets for the same.

```python
# Function to read GeoJSON from file paths
def read_geojson(file_path):
    with open(file_path, "r") as f:
        return json.load(f)

# Function to call the LLM with GeoJSON files
def call_llm_with_geojson_files(file_path1, file_path2):
    # Read the GeoJSON files
    geojson1 = read_geojson(file_path1)
    geojson2 = read_geojson(file_path2)

    # Prepare the GeoJSON data as strings
    geojson1_str = json.dumps(geojson1)
    geojson2_str = json.dumps(geojson2)
```

Figure 4.8: function to read the GeoJSON files

```python
    # Define the system prompt
    prompt = f"""
You are an assistant that can execute Python code. The user has provided two GeoJSON files.
Your task is:
1. Parse the GeoJSON files.
2. Compute their intersection using GeoPandas.
3. Plot GeoJSON 1 in blue, GeoJSON 2 in green, and their intersection in red using Matplotlib.
4. Return the plot as output.

GeoJSON 1:
{geojson1_str}

GeoJSON 2:
{geojson2_str}

Perform the task and show the resulting plot.
"""

    # Call the LLM with the prompt
    response = openai.ChatCompletion.create(
        model="gpt-4o-mini",
        messages=[
            {"role": "system", "content": "You can execute Python code to answer questions."},
            {"role": "user", "content": prompt},
        ]
    )

    # Return the LLM response
    return response
```

Figure 4.9: System prompt with LLM call

2.B Plotting the intersection

Now we will try to plot the intersection by leveraging the LLM.

```python
if __name__ == "__main__":
    # Paths to the GeoJSON files
    file_path1 =
    ↪  "/content/anugul_purunakot.geojson"
    file_path2 =
    ↪  "/content/mws_anugul_purunakot.geojson"

    # Call the function
    response =
    ↪  call_llm_with_geojson_files(file_path1,
    ↪  file_path2)

    print(response["choices"][0]["message"]
    ["content"])
```

As we can see in Fig.4.10 the LLM throws an error that our request exceeds the token limits for GPT-4o-mini.

```
RateLimitError                            Traceback (most recent call last)
<ipython-input-1-fdcabb22030b> in <cell line: 37>()
     35 )
     36
---> 37 response = openai.ChatCompletion.create(
     38     model="gpt-4o-mini",
     39     messages=[

                              4 frames
/usr/local/lib/python3.10/dist-packages/openai/api_requestor.py in _interpret_response_line(self, rbody, rcode, rheaders,
stream)
    763         stream_error = stream and "error" in resp.data
    764         if stream_error or not 200 <= rcode < 300:
--> 765             raise self.handle_error_response(
    766                 rbody, rcode, resp.data, rheaders, stream_error=stream_error
    767             )

RateLimitError: Request too large for gpt-4o-mini in organization org-wfisfhwq0zCxv5d2YCzcV8c9 on tokens per min (TPM):
Limit 200000, Requested 429854. The input or output tokens must be reduced in order to run successfully. Visit
https://platform.openai.com/account/rate-limits to learn more.
```

Figure 4.10: Failure to plot intersection

```
# Limit the size of the data sent to the LLM
summary_gdf1 =
↪  gdf1[['geometry']].head(5).to_json()
summary_gdf2 =
↪  gdf2[['geometry']].head(5).to_json()
summary_intersection =
↪  intersection[['geometry']].head(5).to_json()
↪  if not intersection.empty else "{}"
```

Instead, let us try to summarize the GeoJSON file using the LLM and then plot the intersection using that data. This way, we will be reducing the tokens and may fall into the safe limit.

````
Response from LLM:
To visualize the provided GeoJSON datasets, we can create a plot using appropriate libraries.

```python
import geopandas as gpd
import matplotlib.pyplot as plt

# Load the GeoJSON datasets
dataset_1 = {
    "type": "FeatureCollection",
    "features": [
        # Add Dataset 1 features here as per the provided summary
    ]
}

dataset_2 = {
    "type": "FeatureCollection",
    "features": [
        # Add Dataset 2 features here as per the provided summary
    ]
}

intersection_data = {
    "type": "FeatureCollection",
    "features": [
        # Add Intersection features here as per the provided summary
    ]
}

# Create GeoDataFrames
gdf1 = gpd.GeoDataFrame.from_features(dataset_1['features'])
gdf2 = gpd.GeoDataFrame.from_features(dataset_2['features'])
gdf_intersection = gpd.GeoDataFrame.from_features(intersection_data['features'])

# Plotting
fig, ax = plt.subplots(figsize=(15, 10))

# Plot Dataset 1 with blue color
gdf1.plot(ax=ax, color='blue', edgecolor='k', alpha=0.5, label='Dataset 1')

# Plot Dataset 2 with green color
gdf2.plot(ax=ax, color='green', edgecolor='k', alpha=0.5, label='Dataset 2')

# Plot Intersection with red color
gdf_intersection.plot(ax=ax, color='red', edgecolor='k', alpha=0.8, label='Intersection')

# Customize the plot
plt.title('GeoJSON Data Visualization')
plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.legend()
plt.grid()

# Show the plot
plt.show()
```
````

Figure 4.11: Failure to plot intersection

As in the case of the shapefiles, the LLM again returns the Python code to plot the intersection between these two files. This is visible in Fig.4.11.

Chapter 5

Conclusions

The findings reveal that while LLMs excel at understanding and generating textual data, their utility in handling visual and geo-spatial information remains constrained. For instance, tasks such as generating pie charts from Excel data or plotting intersections of geo-spatial files required supplementary Python code provided by the LLM, rather than direct execution. Furthermore, the inability of LLMs to analyze or extract meaningful insights from images embedded within HTML files or geo-spatial datasets underscores a critical gap in their functionality. These limitations highlight the need for integrating domain-specific tools and methodologies, such as QGIS or Python libraries, to complement LLM capabilities.

Despite these challenges, the internship successfully laid the groundwork for leveraging LLMs in building a chatbot aimed at rural accessibility to geo-spatial information. By enabling text-based queries to retrieve information on land use, soil profiles, and deforestation patterns, the envisioned chatbot could provide a temporary solution for rural communities, fostering ecological awareness and data-driven decision-making.

The iterative experimentation with datasets also emphasized the importance of data preparation and efficient token usage, especially when working with constrained resources like GPT-4o-mini. This learning experience accentuates the importance of developing lightweight yet powerful models tailored to specific applications, ensuring scalability and cost-effectiveness.

Looking forward, the path to fully realizing the potential of LLMs in geo-spatial analysis lies in addressing their current limitations. This entails advancements in multi-modal LLMs capable of processing text, images, and geo-spatial data holistically. Additionally, fostering collaborations between AI researchers and domain experts will be pivotal in creating robust, real world solutions.

In conclusion, this internship has not only enhanced proficiency in pioneering AI methodologies but also cultivated a deeper understanding of their societal impact, setting the stage for future innovations in geo-spatial analysis and technology-enabled community empowerment.