

## Front end (e.g., Jupyter lab)

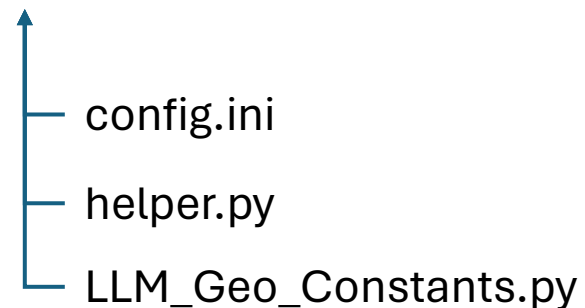
LLM\_Geo\_API4.ipynb

1. The core functions of LLM-Geo are implemented in LLM\_Geo\_kernel.py; LLM\_Geo\_API4.ipynb is merely a showcase of LLM-Geo.
2. Developers can use LLM\_Geo\_kernel.py like APIs to create their own implementation of LLM-Geo, e.g., Python version or webservice version.

## Back end (Python)



LLM\_Geo\_kernel.py



1. The core functions of LLM-Geo are implemented in LLM\_Geo\_kernel.py.
2. Config.ini, helper.py, and LLM-Geo\_Constants.py are the supporting files of LLM\_Geo\_kernel.py.
3. Config.ini store the APIs of the adopted large language models (e.g., GPT-4 in our study).
4. Helper.py contains some supporting functions.
5. LLM\_Geo\_Constants.py includes the components of prompts, such as roles, tasks, and technical requirements.

Note: helper.py, LLM-Geo\_Constants.py, and LLM\_Geo\_kernel.py are not perfect from the viewpoint of soft engineering, and there are still being developed. Feel free to contact us if you are confused with LLM-Geo.

# LLM\_Geo\_kernel.py

▼ class Solution

- func \_\_init\_\_
- func get\_LLM\_reply
- func get\_LLM\_response\_for\_graph
- func load\_graph\_file
- func operation\_node\_names
- func get\_ancestor\_operations
- func get\_descendant\_operations
- func get\_descendant\_operations...
- func get\_prompt\_for\_an\_opera...
- func initial\_operations
- func get\_LLM\_responses\_for\_op...
- func prompt\_for\_assembly\_prog...
- func get\_LLM\_assembly\_response
- func save\_solution
- func get\_solution\_at\_one\_time
- func direct\_request\_prompt
- func get\_direct\_request\_LLM\_res...
- func execute\_complete\_program
- func get\_debug\_prompt
- func ask\_LLM\_to\_review\_operati...
- func ask\_LLM\_to\_review\_assemb...
- func ask\_LLM\_to\_review\_direct\_c...
- func ask\_LLM\_to\_sample\_data

1. **LLM\_Geo\_kernel.py** contains a class Solution.
2. All core behaviors of LLM-Geo are implemented in the Solution class.
3. Please refer to **LLM\_Geo\_API4.ipynb** to learn how to use **LLM\_Geo\_kernel.py**.
4. Initial a solution:

```
solution = Solution(  
    task=TASK,  
    task_name=task_name,  
    save_dir=save_dir,  
    data_locations=DATA_LOCATIONS,  
    model=model,  
)
```

5. Generate the solution graph:

```
solution.get_LLM_response_for_graph()
```

6. Generate prompts and code for operations:

```
operations = solution.get_LLM_responses_for_operations(review=isReview)
```

7. Generate prompts and code for assembly program:

```
solution.get_LLM_assembly_response(review=isReview)
```

8. Execute the complete program:

```
all_operation_code_str = '\n'.join([operation['operation_code'] for operation in operations])  
all_code = all_operation_code_str + '\n' + solution.code_for_assembly  
all_code = solution.execute_complete_program(code=all_code)
```

```

class Solution():
    def __init__(self,
        self.data_locations_str = '\n'.join([f"{idx + 1}. {line}" for idx, line in enumerate(self.data_locations)])

        graph_requirement = constants.graph_requirement.copy()
        graph_requirement.append(f"Save the network into GraphML format, save it at: {self.graph_file}")
        graph_requirement_str = '\n'.join([f"{idx + 1}. {line}" for idx, line in enumerate(graph_requirement)])

        graph_prompt = f'Your role: {self.role} \n\n' + \
            f'Your task: {constants.graph_task_prefix} \n {self.task} \n\n' + \
            f'Your reply needs to meet these requirements: \n {graph_requirement_str} \n\n' + \
            f'Your reply example: {constants.graph_reply_exmaple} \n\n' + \
            f'Data locations (each data is a node): {self.data_locations_str} \n'
        self.graph_prompt = graph_prompt

```

const graph\_role  
const graph\_task\_prefix  
const graph\_reply\_exmaple  
const graph\_requirement  
const operation\_role  
const operation\_task\_prefix  
const operation\_reply\_exmaple  
const operation\_requirement  
const assembly\_role  
const assembly\_requirement  
const direct\_request\_role  
const direct\_request\_task\_prefix  
const direct\_request\_reply\_exma...  
const direct\_request\_requirement  
const debug\_role  
const debug\_task\_prefix  
const debug\_requirement  
const operation\_review\_role  
const operation\_review\_task\_prefix  
const operation\_review\_requirem...  
const assembly\_review\_role  
const assembly\_review\_task\_prefix  
const assembly\_review\_requirem...

## How to compose prompts in LLM-Geo?

1. A structured prompt in LLM-Geo may contain: AI roles, task, reply examples, and technical requirements. This figure shows to the compose the prompt for solution graph generation.
2. We found that the technical requirements need to be tested intensively; to facilitate this process, we organize them into individual rules stored in a Python list. You can easily add, remove, and update those rules by modifying the list elements.