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

- config.ini
- helper.py
- LLM_Geo_Constants.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 are 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_opeara...
  - 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)
```

LLM_Geo_Constants.py

```python
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.