Assignment #3

Using agents to build complex programs

Dr. Daniel M. Yellin

DRAFT

Due date

Assignment is due July 18, 2024 at 17:00

18.07.24 @17:00

THIS IS A AN UPDATED VERSION OF THE ASSIGNMENT. 23/06/24

Additional small changes, 07/07/2024

Assignment

Build an agent that can answer queries. Each query may require multiple steps, including reading input files, and taking actions.

- This assignment specifies the list of actions you can take. You will need to implement each action and the agent control flow to invoke the actions.
- You do not know in advance what the query will be, which specific actions will be required, or how many steps (different actions) it will take. You only know that it is solvable using the tools and resources at your disposal.

Tools available

Your agent is equipped with the following tools (functions):

1. An LLM
2. A tool to extract entities from a text file
3. An Internet Search Tool
   - This tool will take very specific types of queries and return specific types of answers
4. A tool to write a Python program to solve a query on a .csv file
   - Similar to HW#2
5. A Python Repl tool for executing Python programs
6. A tool to write information to a file

The exact requirements of each tool 2-5 are given later in this presentation

The file input.json is the input to your program

```
{
  "query-name": "<query_name>.txt",
  "resources": [
      {"name":"<file_name>",
       "description":"<some_string>"
      },
      {"name":"<file_name>",
       "description":"<some_string>"
      }
      ...
  ]
}
```

```
{
   "query_name": "query1.txt",
   "file_resources": [
      {"file_name":”grades.csv",
       "description":"A csv file describing students and their grades.  the first line contains the columns (field names) of each following row.  They are:  First, Last, Year,  Class,  Name, Grade.  An example row is: Shalom,Levi,3,Calculus,93.5"
      },
      {"file_name":"students.txt",
       "description":"a text file containing information on some of the students, including his or her aspirations, favorite city, and hobbies."
      }]
}
```

input.json

Name of the query file.

Name of a file resource the solution can use.

```
{
  "query-name": "<query_name>.txt",
  "resources": [
      {"name":"<file_name>",
       "description":"<some_string>"
      },
      {"name":"<file_name>",
       "description":"<some_string>"
      }
      ...
   ]
}
```

```
{
  "query_name": "query1.txt",
  "file_resources": [
      {"file_name":"grades.csv",
       "description":"A csv file describing students and their grades.  the first line contains the columns (field names) of each following row.  They are:  First, Last, Year,  Class,  Name, Grade.  An example row is: Shalom,Levi,3,Calculus,93.5"
      },
      {"file_name":"students.txt",
       "description":"a text file containing information on some of the students, including his or her aspirations, favorite city, and hobbies."
      }]
}
```

Examples

For these examples, two resource files are defined, grades.csv and students.txt

grades.csv

First, Last, University-Year, Class, Grade

Shalom, Levi, 3, Calculus, 93

Nitzi, Matok, 1, Intro to programming, 89

Chava, Bina, 3, Algorithms, 77

Maor, Katz, 2, Automaton, 69

Hinon Chalek, 3, Calculus, 82

Rutie, Levana, 2, Algorithms, 95

...

students.txt

Shalom Levi wants to become a doctor. His favorite city is Tel Aviv and his hobby is playing chess.

Nitzi Matok loves Eilat. Her hobby is scuba diving.

Chava Bina hopes to become an entrepenuer. She would like to live in Haifa. For a hobby, she does crossword puzzles.

Example: query1.txt

Find the hobbies of all students. The result should be JSON array where each object in the array has one field, ‘hobby', whose value is the name of the hobby. Store the JSON in the file "query1.json".

Execution of query1

```
calling chat completions with messages =  [{'role': 'user', 'content': 'If there are no more tools to call, then just return "Done" '}, {'role': 'user', 'content': '\nYou
** Entering Agent extract_entities_from_file **
Parameters are: fine_name =  students.txt, entity_type = hobbies
** Leaving Agent extract_entities_from_file **
Result =  playing chess, scuba diving, crossword puzzles
calling chat completions with messages =  [{'role': 'user', 'content': 'If there are no more tools to call, then just return "Done" '}, {'role': 'user', 'content': '\nYou
** Entering Agent write_file **
Parameters are file_content = [{"hobby": "playing chess"}, {"hobby": "scuba diving"}, {"hobby": "crossword puzzles"}]..., fn = query1.json
** Leaving Agent write_file **
calling chat completions with messages =  [{'role': 'user', 'content': 'If there are no more tools to call, then just return "Done" '}, {'role': 'user', 'content': '\nYou
final response is =  The request was successfully executed. You can find the hobbies of all students in the JSON file named "query1.json".

Process finished with exit code 0
```

Output file query1.json:

[{"hobby": "playing chess"}, {"hobby": "scuba diving"}, {"hobby": "crossword puzzles"}]

I am including execution messages that I print to the screen just for you to see. You do not need to do this for your assignment, although you do need to produce a log file as explained later on. Note that most of the messages content is not visible in this screenshot.

Example: query2.txt

Find the current population of a city that is mentioned in the student.txt file that begins with the letter 'H’. Write the result as a JSON object with the fields 'city' and 'population', whose value is a number. The JSON should be stored in the file "query2.json".

Execution of query2

```
calling chat completions with messages =  [{'role': 'user', 'content': 'If there are no more tools to call, then just return "Done" '}, {'ro
** Entering Agent extract_entities_from_file **
Parameters are: fine_name =  students.txt, entity_type = city
** Leaving Agent extract_entities_from_file **
Result =  ['Tel Aviv', 'Eilat', 'Haifa']
calling chat completions with messages =  [{'role': 'user', 'content': 'If there are no more tools to call, then just return "Done" '}, {'ro
** Entering Agent Internet_search_attribute **
Parameters are a_entity = Haifa, a_attribute = current population
Internet Search results =  [{'title': 'Haifa, Israel Metro Area Population 1950-2024 | MacroTrends', 'href': 'https://www.macrotrends.net/gl
new content =  An Internet search asking for the current population of the entity Haifa returned the following results [{'title': 'Haifa, Is
** Leaving Agent Internet_search **
Result =  {
  "entity": "Haifa",
  "current population": 290306
}
calling chat completions with messages =  [{'role': 'user', 'content': 'If there are no more tools to call, then just return "Done" '}, {'ro
** Entering Agent write_file **
Parameters are file_content = { "city": "Haifa", "population": 290306 }..., fn = query2.json
** Leaving Agent write_file **
calling chat completions with messages =  [{'role': 'user', 'content': 'If there are no more tools to call, then just return "Done" '}, {'ro
final response is =  The request was executed successfully. You can find the answer in the file named "query2.json".

Process finished with exit code 0
```

query2.json

{ "city": "Haifa", "population": 290306 }

Example: query3.txt

Generate a Python program that finds the average grade of all the students. The result should be represented as a JSON object with one field, 'average_grade', whose value is a number, and written to the file query3.json. The Python program itself should be stored in the file "query3.py". Before finishing, execute the program stored in the file "query3.py".

Execution of query3

```
calling chat completions with messages =  [{'role': 'user', 'content': 'If there are no more tools to call, then just return "Done" '}, {'role': 'user', 'cor
** Entering Agent generate_analysis_program **
Parameters are analysis_request = find the average grade of all students, input_file = class.csv, columbs = First, Last, Year, Class, Name, Grade, row_examp
** Leaving Agent generate_analysis_program **
calling chat completions with messages =  [{'role': 'user', 'content': 'If there are no more tools to call, then just return "Done" '}, {'role': 'user', 'cor
** Entering Agent write_file **
Parameters are file_content = import pandas as pd

# Read the csv file
df = pd.read_csv('class.csv')

# Compute the average grade
..., fn = query3.py
** Leaving Agent write_file **
calling chat completions with messages =  [{'role': 'user', 'content': 'If there are no more tools to call, then just return "Done" '}, {'role': 'user', 'cor
** Entering Agent execute_Python_program **
Parameters are program_fn = query3.py, output_fn = query3.json
** Leaving Agent generate_analysis_program **
calling chat completions with messages =  [{'role': 'user', 'content': 'If there are no more tools to call, then just return "Done" '}, {'role': 'user', 'cor
final response is =  The Python program executed successfully. You can find the result in the file named "query3.json".

Process finished with exit code 0
```

query3.py and query3.json

query3.py

```
import pandas as pd

# Read the csv file
df = pd.read_csv('class.csv')

# Compute the average grade
average_grade = df['Grade'].mean()

# Write the output to json
with open('query3.json', 'w') as f:
    f.write('{"average_grade": ' +
str(round(average_grade, 2)) + '}')
```

query3.json

```
{"average_grade": 86.01}
```

Example: query4.txt

Generate a Python program that finds the average grade of each of the students mentioned in the file "students.txt". The result should be represented as a JSON array where each object in the array has two fields: 'student_name' (the name of the student) and 'average_grade' (a number that is the average grade of that student taking into consideration all of his courses). The program should write the JSON to the file query4.json. The Python program itself should be stored in the file "query4.py".
Before finishing, execute the program stored in the file "query4.py"

Execution of query4

```
calling chat completions with messages =  [{'role': 'user', 'content': 'If there are no more tools to call, then just return "Done" '}, {'role': 'user',
** Entering Agent extract_entities_from_file **
Parameters are: fine_name =  students.txt, entity_type = name
** Leaving Agent extract_entities_from_file **
Result =  ['Shalom Levi', 'Nitzi Matok', 'Chava Bina']
calling chat completions with messages =  [{'role': 'user', 'content': 'If there are no more tools to call, then just return "Done" '}, {'role': 'user',
** Entering Agent generate_analysis_program **
Parameters are analysis_request = find the average grade of each of the students, input_file = class.csv, columbs = First, Last, Year, Class, Name,Grade,
** Leaving Agent generate_analysis_program **
calling chat completions with messages =  [{'role': 'user', 'content': 'If there are no more tools to call, then just return "Done" '}, {'role': 'user',
** Entering Agent write_file **
Parameters are file_content = import csv
import json
from collections import defaultdict

def compute_average_grades():
    studen..., fn = query4.py
** Leaving Agent write_file **
calling chat completions with messages =  [{'role': 'user', 'content': 'If there are no more tools to call, then just return "Done" '}, {'role': 'user',
** Entering Agent execute_Python_program **
Parameters are program_fn = query4.py, output_fn = query4.json
** Leaving Agent generate_analysis_program **
calling chat completions with messages =  [{'role': 'user', 'content': 'If there are no more tools to call, then just return "Done" '}, {'role': 'user',
final response is =  The task was executed successfully. The resulting file, query4.json, contains the average grades for each student.
```

query4.py This is incorrect (my agent did not get this right)

```
import csv
import json
from collections import defaultdict

def compute_average_grades():
    student_grades = defaultdict(list)
    with open('class.csv', newline='') as csvfile:
        reader = csv.DictReader(csvfile)
        for row in reader:
            # Assuming each row represents a course taken by a student
            student_grades[row['First'] + ' ' + row['Last']].append(float(row['Grade']))

    for key in student_grades:
        student_grades[key] = sum(student_grades[key]) / len(student_grades[key])

    return student_grades

def write_to_json(data):
    with open('query4.json', 'w') as jsonfile:
        json.dump(data, jsonfile)

def main():
    student_average_grades = compute_average_grades()
    write_to_json(student_average_grades)

if __name__ == '__main__':
    main()
```

query4.json This is incorrect (my agent did not get this right)

{"Shalom Levi": 92.4, "Nitzi Matok": 89.0, "Chava Bina": 80.9, "Maor Katz": 76.75, "Eliezer Krep": 88.35, "Hinon Chalek": 82.0, "Rutie Levana": 92.1, "Boris Orlov": 86.83333333333333, "Carolyin Zeltzer": 96.2, "Gadi Renker": 94.2, "Marina Bergoff": 78.8, "Miri Boruch": 88.9, "Shlomit Chazak": 89.0, "Moe Machon": 66.3, "Haim Lesk": 74.3, "Pierre French": 89.0, "Shlomo Chacham

Another execution of query4

```
** Entering Agent execute_Python_program **
Parameters are program_fn = query4.py, output_fn = query4.json
Errors in program execution:
** Leaving Agent generate_analysis_program **
calling chat completions with messages =  [{'role': 'user', 'content': 'If there are
** Entering Agent generate_analysis_program **
Parameters are analysis_request = Average grade of each student, input_file = class.c
** Leaving Agent generate_analysis_program **
calling chat completions with messages =  [{'role': 'user', 'content': 'If there are
** Entering Agent write_file **
Parameters are file_content = import pandas as pd

# Reading data from csv file
df = pd.read_csv('class.csv')

# Computing average..., fn = query4.py
** Leaving Agent write_file **
calling chat completions with messages =  [{'role': 'user', 'content': 'If there are
** Entering Agent execute_Python_program **
Parameters are program_fn = query4.py, output_fn = query4.json
** Leaving Agent generate_analysis_program **
calling chat completions with messages =  [{'role': 'user', 'content': 'If there are
final response is =  The Python program was successfully executed and the result has
```

This execution also got an incorrect result. But it illustrates the fact that if the program generated has execution errors, the agent tries to generate another one.

Assignment guidelines

Functions (tools) available for your program to use

1. **extract_entities_from_file(file_name: str, entity_type:str) -> str**
   # The function is given a text file and an “entity type” (could be
   # “student”, “hobby”, “city” or any type of entity). It returns a string representing a list of entities of that type in the file (e.g.,
   # “[‘Haifa’, ‘Tel Aviv’]”).
2. **Internet_search_attribute(a_entity: str, a_attribute: str) -> str**
   # The function is given an entity (e.g, ‘Tel Aviv’) and an attribute
   # (e.g.,’current population’) and searches the Internet to find the
   # attribute of the entity. It then asks the LLM to review the
   # search results and return a JSON structure of the form
   # {"a_entity": {a_entity}, "{attribute}": Answer}. Example answer:
   # { "city": ”Tel Aviv", "population": 4,400,000 }

Actions (tools) available to your program

3. **generate_analysis_program(analysis_request: str, input_file: str, columns: str, row_example: str, output_file: str)-> str**
   # This function takes an analysis request, a csv input_file, the first
   # line of the file containing the column names, another line of the file
   # with example data, and an output_file. The analysis_request
   # describes analysis of the csv input_file to be performed. This|
   # function generates a Python program to perform the analysis. The
   # program writes the output, a JSON object, to the output_file.
   # **generate_analysis_program** returns the # generated Python code.

Actions (tools) available to your program

4. **execute_Python_program(program_fn: str, output_fn: str) -> str:**
   # This function executes the program in file program_fn. That
   # program should output its results into the file output_fn. It returns
   # either a message that the program executed successfully, or it
   # returns a message that the program did not execute successfully
   # and provides error messages.
5. **write_file(file_content:str, fn:str)**
   # This function writes the file contents to the file fn.

Agent rules

- Your agent can only use these functions with the same names. However, you can change the parameters to these functions if you desire. (But not completely change the function).
- All your program can do is to invoke the LLM or to invoke one of the functions.
- The program cannot invoke the LLM more than 10 times, and cannot invoke functions (all together) more than 10 times.

OpenAI tools

You are not required to use **OpenAI function calling, but it is recommended**. You can also use LangChain tools (which will use OpenAI functions/tools under the covers), or some other tookkit.

Some OpenAI function calling resources:

https://platform.openai.com/docs/guides/function-calling

https://www.analyticsvidhya.com/blog/2023/10/openai-function-calling/

Just search on-line. Also lots of videos.

Some LangChain tool references.

https://python.langchain.com/v0.1/docs/use_cases/tool_use/multiple_tools/

https://js.langchain.com/v0.1/docs/modules/agents/agent_types/chat_conversation_agent/

https://www.comet.com/site/blog/conversational-agents-in-langchain/

Agent loop

The following pseudocode is more or less how my program works, but you can do it differently. Of course, there are lots of details to fill in.

```
messages = [initial_system_message, initial_user_message]
while not done and not at limit:
        response = client.chat.completions.create(messages,...)
        if response is recommending tool T and parameters p1,...,pk
                tool_result = T(p1,...,pk)
                messages.append(tool_result_message)
summarize final status using LLM.
```

Getting good results

To get good results, you will need to formulate the the prompts to the LLM. Better prompts will get better results.

You should also choose an appropriate temperature.

Required outputs

What your program must output

1. The query will define what outputs are expected. This is usually a **query{i}.json** file giving the result in JSON.
2. If the solution requires generation of a Python program, the query will define what file to store it in. This is usually query{i}.py.
3. You need to create a log file named log_query{i}.txt (query{i}.txt is the name of the query file).
   - Every time you enter agent A, you need to log “**Entering agent A**”.
   - You should log, for each parameter p of A, “Parmeter p = <value of p>”. **If the parameter value is a string, just record the first 50 chars** of the string.
   - Every time you leave agent A, you need to log “**Leaving agent A**”.