gyeonghoonyi fine tuning, few shot, zero shot, follow up questions 391089a · last year

598 lines (504 loc) · 35.5 KB

Code Blame Raw

```python
import json
import os
import openai
import pandas as pd

openai.api_key = "your api key"

class FollowQ:
    def __init__(self, json_path, representation_path, save_path):
        self.json_path = json_path
        self.representation_path = representation_path
        self.save_path = save_path

        # Questions related to catalysts
        self.questions_for_catalyst = {
            "1_list": ["representation", "Question 1. Please tell me the names of catalysts in the contents within the <table> tags of input representation into a Python list. Give me the names of catalysts ONLY. Only output the Python list(like string).\n\n"],
            "2_list": ["json", "Question 2. Please output the Python list of catalysts from the input json provided by me into a Python list. Only output the Python list(like string).\n\n"],
            "3_list": ["None", "Question 3. Based on the answer to Question 1, modify or remove any catalysts from the answer to Question 2 and provide the updated list in Python. Give me the names of catalysts ONLY. Only output the Python list.(like string)\n\n"]
        # Questions related to performance
        self.questions_for_performance = {
            "1_list": ["json", "Question 1. Inform me about what performance type does {catalyst_name} have in the input json I provided? Only output the Python list.(like string)\n\n"]

        # Questions related to properties
        self.questions_for_property = {
            "1_both": ["json", "Question 1. Provide detailed information about all sublayers of the {element} of {catalyst_name} in input json. Remove keys from the dictionary that do not have a value. Present it in either Python list or JSON format. If the {element} is not [Heading', 'Question 2. If there is a Python list or JSON format (like string) or `python content` in your response, Respond with yes or no. You must answer strictly with "yes" or "no". Question 3. In the answer to question 1, remove any
```

```
answer strictly with yes/none. \Question 3. In the answer to question 1, remove any
parts corresponding to 'NA', 'na', 'unknown', or similar contents. Show the modified
JSON. Only display the JSON (like string not ```json)"], "4_str": ["representation", "Question 4. Based on the input representation,
provide values of the {element} of the {catalyst_name} as a Python list. If there is a
unit, please provide strictly the value including the unit. The elements of a Python
list must be composed of value plus unit. Only output the Python list (like string not
python)\n\n"], "5_str": ["json", "Question 5. Based on the input representation, provide the '''value''' key of the sublayers of the {element} as a Python list. Only
output the Python list (like string not python)\n\n"], "6_str": ["json", "Question 6. Based on only numerical values, provide a
list of elements that exist in the answer to Question 5 but are not present in the
answer to Question 4. Only output the Python list (like string not python)\n\n"], "7_str": ["json", "Question 7. If there are elements in the list that is the answer to question 6, remove the sub-dictionary
containing those elements from the json I provided. If the answer to 6 is a list
containing elements, be sure to delete from the json. Please show the modified json of
{catalyst_name}. Only display the JSON (like string not ```json)\n\n"], "8_str": ["json", "Question 8. the json I provided. Only
output the JSON of {catalyst_name}. the catalyst_name is {catalyst_name}. The first key
of the dictionary should be {catalyst_name}. Remove keys from the dictionary that do not
have a value. (like string not ```json)"]
        }
        # Questions related to electrolyte, reaction_type, substrate
        self.questions_for_representation = {
            "1_str": ["representation_title", "Question 1. If there is any occurrence of
'OER', 'HER', 'oxygen evolution reaction', 'hydrogen evolution reaction' or similar
contents in input representation, respond with yes or no. Please answer with either yes
or no\n\n"], "2_str": ["representation_table_caption", "Question 2. If there is any
occurrence of 'OER', 'HER', 'oxygen evolution reaction', 'hydrogen evolution reaction'
or similar contents in input representation, respond with yes or no. Please answer with
either yes or no\n\n"], "3_str": ["representation_title", "Question 3. Does the input representation contain information corresponding to substrate? Please answer with either yes or
no\n\n"],    "4_str": ["representation_table_caption", "Question 4. Does the input
representation contain information corresponding to substrate? Please answer with either
yes or no\n\n"], "5_str": ["representation_title", "Question 5. Does the input representation
contain information corresponding to electrolyte? Please answer with either yes or
no\n\n"],    "6_str": ["representation_table_caption", "Question 6. Does the input
representation contain information corresponding to electrolyte? Please answer with
either yes or no\n\n"]
        }

        # system prompt
        self.system_prompt = {"role": "system", "content": "You need to modify the JSON
representing the table presenter.\n\n JSON templete : {'catalyst_name' :
{'performance_name' : \{property templete\}}}\n property templete : {'electrolyte': '',
'reaction_type': '', 'value': '', 'current_density': '', 'overpotential': '',
'potential': '', 'substrate': '', 'versus': '', 'condition': ''}\n performance_list =
[overpotential, tafel_slope, Rct, stability, Cdl, onset_potential, potential, TOF, ECSA,
water_splitting_potential, mass_activity, exchange_current_density, Rs,
specific_activity, onset_overpotential, BET, surface_area, loading,
apparent_activation_energy]\n In the JSON template, 'catalyst_name' and
'performance_name' should be replaced with the actual names present in the input
representation."}

    def input_prompt(self, representation, json, want_type):
        """
        Generates a formatted input string based on the type of content requested.
        Parameters:
        representation (str): The input representation string containing HTML content.
        json (str): The JSON string to be included in the input.
        want_type (str): The type of content required ('both', 'representation', 'json',
'representation_title', 'representation_table_caption').
        Returns:
        str: The formatted input string.
        """
```

```
63
64              # Split the representation string at each occurrence of '</table>'
65              splitted_strings = representation.split('</table>')
66              # Determine the format based on the requested type
67              if want_type == 'both':
68                  format_for_input = "<input representation>\n" + str(representation) +
69      "\n<input json>\n" + str(json) + "\n\n"
70              elif want_type == 'representation':
71                  format_for_input = "<input representation>\n" + str(representation) + "\n\n"
72              elif want_type == 'json':
73                  format_for_input = "<input json>\n" + str(json) + "\n\n"
74              elif want_type == 'representation_title':
75                  title_string = splitted_strings[0] + '</table>'
76                  format_for_input = "<input representation>\n" + str(title_string) + "\n\n"
77              elif want_type == 'representation_table_caption':
78                  table_caption_string = splitted_strings[1]
79                  format_for_input = "<input representation>\n" + str(table_caption_string) +
80      "\n\n"
81              return format_for_input
82
83          def load_file(self, file_type, file_path, file_name):
84              """
85              Loads the content of a file based on the specified type and path.
86
87              Parameters:
88              file_type (str): The type of the file (json, html, txt, etc.)
89              file_path (str): The path to the file
90              file_name (str): The name of the file
91
92              Returns:
93              output: The content of the file. The type of the content depends on the
94      file_type.
95              Raises:
96              ValueError: If the file type is unsupported.
97              """
98              with open(file_path + file_name, 'r', encoding='utf-8-sig') as f:
99                  # Load JSON files
100                 if file_type == 'json':
101                     output = json.load(f)
102                 # Read text or HTML files
103                 elif file_type in ['html', 'txt']:
104                     output = f.read()
105                 # Raise an error for unsupported file formats
106                 else:
107                     raise ValueError("Unsupported file format.")
108
109             return output
110
111         def formatting_type(self, key, answer):
112             """
113             Formats the answer based on the specified key type.
114
115             Parameters:
```

```
        key (str): The key indicating the type of format required (e.g., '1_list',
'2_dict').answer (str): The answer to be formatted.

        Returns:
        The formatted answer, which can be a list or a dictionary based on the key type.
        """
        # Determine the desired format type from the key
        want_type = key.split('_')[1]

        # Handle formatting if the desired type is a list
        if want_type == "list":
            if answer[0] == '"' and answer[-1] == '"':
                answer = answer.strip('"')  # Remove surrounding quotes if present
            answer = eval(answer)  # Evaluate the string as a Python expression (e.g.,
convert to list)
        # Handle formatting if the desired type is a dictionary
        elif want_type == "dict":
            if "```" in answer:
                answer = answer.replace("```json", "").replace("```", "")  # Remove
markdown codeblock=fjsonloads(answer)  # Parse the string as JSON

        return answer

    def check_type(self, key, answer):
        """
        Checks if the type of the answer matches the expected type based on the key.

        Parameters:
        key (str): The key indicating the expected type (e.g., '1_list', '2_dict').
        answer (any): The answer whose type needs to be checked.

        Returns:
        tuple: A tuple containing the expected type (str) and a boolean indicating
whether the types match.
        # Extract the question number and expected type from the key
        question_number = key.split('_')[0]
        want_type = key.split('_')[1]

        # Determine the actual type of the answer
        answer_type = type(answer).__name__

        # Check if the expected type matches the actual type
        type_bool = want_type == answer_type

        return want_type, type_bool

    def prompt(self, Q):
        """
        Sends a list of messages to the OpenAI ChatCompletion API and returns the
response.
        Parameters:
        Q (list): A list of message dictionaries to be sent to the API.

```

```
168             Returns:
169             tuple: A tuple containing the original list of messages (Q) and the response
170     content."""
171             while True:
172                 try:
173                     # Send a request to the OpenAI ChatCompletion API
174                     response = openai.ChatCompletion.create(
175                         model="gpt-4-0125-preview",
176                         messages=Q,
177                         temperature=0,
178                         frequency_penalty=0,
179                         presence_penalty=0
180                     )
181                     break  # Exit the loop if the request is successful
182                 except Exception as e:
183                     # Print the error message and retry
184                     print("An error occurred:", e)
185 
186             # Return the original messages and the content of the response
187             return Q, response['choices'][0]['message']['content']
188 
189         def run(self, input_type, split_mode):
190             """
191             Executes the main process for modifying JSON files based on given questions.
192 
193             Parameters:
194             input_type (str): The type of input files (e.g., 'json', 'txt').
195             split_mode (str): The mode for handling splits (e.g., 'split', 'no-split').
196 
197             Returns:
198             None
199             """
200             json_name_list = os.listdir(self.json_path)
201             error_file_name = []
202             transpose_bool = False
203 
204             for json_name in json_name_list:
205                 # try:
206                     txt_name = json_name.split('.')[0]
207                     # Load input files
208                     input_json = self.load_file('json', self.json_path, json_name)
209                     input_representation = self.load_file(input_type,
210     self.representation_path, txt_name + '.txt')
211                     # Initialize messages with system prompt
212                     messages = [self.system_prompt]
213 
214                     # Catalyst check
215                     catalyst_result = {}
216                     question_list = []
217                     answer_list = []
218                     question_token_list = []
219                     answer_token_list = []
```

```
                final_result = []
                final_json = {"catalysts": []}

                # Iterate through catalyst questions
                for key, value in self.questions_for_catalyst.items():
                    if value[0] != "None":
                        messages.append({"role": "user", "content":
self.input_prompt(input_json, value[0])})
                        messages.append({"role": "user", "content": value[1]})
                        messages, answer = self.prompt(messages)
                        question_token_list.append(messages)
                        answer_token_list.append(answer)

                        try:
                            mod_answer = self.formatting_type(key, answer)
                        except Exception as e:
                            try:
                                messages, answer = self.prompt(messages)
                                question_token_list.append(messages)
                                answer_token_list.append(answer)
                                question_list.append("Throwing the same message once again")
                                answer_list.append("Throwing the same message once again")
                            except Exception as e:
                                error_file_name.append(json_name)
                                print(json_name)
                                break

                        catalyst_result[key[0]] = mod_answer
                        question_list.append(value[1])
                        answer_list.append(mod_answer)

                        print('--------------------------')
                        print(self.questions_for_catalyst[key])
                        print(mod_answer)
                        print(catalyst_result)
                        print('--------------------------')

                        messages.append({"role": "assistant", "content": answer})
                        if key[0] == '2' and catalyst_result["1"] == catalyst_result["2"]:
                            print("GO TO THE PERFORMANCE STAGE !!")
                            break

                # Performance check
                performance_result = {}
                mod_catalyst_list = answer_list[-1]

                if len(mod_catalyst_list) > 1:
                    transpose_bool = True

                for i in range(len(mod_catalyst_list)):
                    messages = [self.system_prompt]
                    for key, value in self.questions_for_performance.items():
                        if value[0] != "None":
                            messages.append({"role": "user", "content":
```

```
                        messages.append({"role": "user", "content":
self.input_prompt(input_question_value[1]_json(value[0]))name = '"""'+
mod_catalyst_list[i] +'"""messages.append({"role": "user", "content": question})
                        messages, answer = self.prompt(messages)
                        question_token_list.append(messages)
                        answer_token_list.append(answer)

                        try:
                            mod_answer = self.formatting_type(key, answer)
                        except:
                            try:
                                messages, answer = self.prompt(messages)
                                question_token_list.append(messages)
                                answer_token_list.append(answer)
                            except:
                                error_file_name.append(json_name)

                                break

                        performance_result[f'{mod_catalyst_list[i]}_{key[0]}'] =
mod_answer              question_list.append(question)
                        answer_list.append(mod_answer)

                        messages.append({"role": "assistant", "content": answer})
                        if key[0] == '3' and
performance_result[f'{mod_catalyst_list[i]}_{key[0]}'] != "GO TO THE 3PROPERTY:STAGE !!")
                            break

                    if isinstance(answer_list[-1], list):
                        mod_performance_list =
performance_result[f'{mod_catalyst_list[i]}_1']
                        while not isinstance(answer_list[-1], list):
                            messages.pop()
                            messages, answer = self.prompt(messages)
                            question_token_list.append(messages)
                            answer_token_list.append(answer)
                            mod_answer = self.formatting_type("1_list", answer)
                            question_list.append(question)
                            answer_list.append(mod_answer)
                            mod_performance_list = answer

                    # Property check
                    property_result = {}
                    skip_questions = False
                    print("#####################")
                    print(mod_performance_list)

                    if mod_performance_list == []:
                        mod_answer = {str(mod_catalyst_list[i]): {}}
                    else:
                        for j in range(len(mod_performance_list)):
                            messages = []
                            messages.append(self.system_prompt)
```

```
325                             print("@@@@@@@@@@@@")
326                             print(mod_performance_list[j])
327                             # system prompt 넣어주고, input 표현자, json 넣어주는 코드
328                             # messages = []
329                             # messages.append(self.system_prompt)
330
331                             skip_questions = False
332                             anwer3_no = False
333                             for key, value in self.questions_for_property.items():
334                                 # 질문 2의 답변이 "no"일 때 질문 3번과 4번 skip하기 위한 조
335     건문                            if key[0] in ['3','7'] and skip_questions:
336                                     print("SKIP THE NEXT QUESTIONS")
337                                     question_list.append("SKIP THE NEXT QUESTIONS")
338                                     answer_list.append("SKIP THE NEXT QUESTIONS")
339                                     if key[0] in ['3','7']:
340                                         skip_questions = False
341                                     continue  # 질문 3번과 4번 건너뛰기
342
343                                 if value[0] != "None":
344                                     messages.append({"role": "user", "content":
345     self.input_prompt(input_representation, input_json, value[0])})
346                                 question = value[1].format(catalyst_name = '"""'+
347     mod_catalyst_list[i] +'"""', element = '"""'+ mod_performance_list[j] +'"""')
348                                 messages, answer = self.prompt(messages)
349                                 question_token_list.append(messages)
350                                 answer_token_list.append(answer)
351                                 try:
352                                     mod_answer = self.formatting_type(key, answer)
353                                 except:
354                                     try:
355                                         messages, answer = self.prompt(messages)
356                                         question_token_list.append(messages)
357                                         answer_token_list.append(answer)
358                                     except:
359                                         print(json_name)
360                                         error_file_name.append(json_name)
361                                         break
362
363                                 property_result[mod_catalyst_list[i] + '_' +
364     mod_performance_list[j] + '_' + key[0]] = answer
365                                 question_list.append(question)
366                                 answer_list.append(mod_answer)
367
368                                 print('--------------------------')
369                                 print(question)
370                                 print(mod_answer)
371                                 print('--------------------------')
372
373                                 messages.append({"role": "assistant", "content":
374     answer})
375                                 if key[0] == '2' and mod_answer.lower() == "no":
                                        question_list.append("Question 3. Based on the
        answer to question 2, remove any parts corresponding to 'NA', 'na', 'unknown', or
```

```
similar content from the answer to question 1. Show the modified JSON. Only display the JSON. If the answer to question 2 is not present in the JSON, respond with 'NA'. Do not provide any explanations or additional text.")
                    messages.append({"role": "user", "content": "Question 3. Based on the answer to question 2, remove any parts corresponding to 'NA', 'na', 'unknown', or similar content from the answer to question 1. Show the modified JSON. Only display the JSON. If the answer to question 2 is not present in the JSON, respond with 'NA'. Do not provide any explanations or additional text."})
                    messages, answer = self.prompt(messages)
                    question_token_list.append(messages)
                    answer_token_list.append(answer)
                    try:
                        mod_answer = self.formatting_type(key, answer)
                    except:
                        try:
                            messages, answer = self.prompt(messages)
                            question_token_list.append(messages)
                            answer_token_list.append(answer)
                        except:
                            print('#$@%@#%@#%#@%@#%@#')
                            print(json_name)
                            print('#$@%@#%@#%#@%@#%@#')
                            error_file_name.append(json_name)
                            break

                    count += 1
```

```
421
422                                     property_result[mod_catalyst_list[i] + '_' +
423     mod_performance_list[j] + '_' + key[0]] = answer
424                                     question_list.append(question)
425                                     answer_list.append(mod_answer)
426
427                                     print('--------------------------')
428                                     print(question)
429                                     print(mod_answer)
430                                     print('--------------------------')
431
432                                     messages.append({"role": "assistant", "content":
433     answer})
434                         if len(mod_catalyst_list) == 1 and split_mode == 'split':
435                             final_result.append(mod_answer)
436                         else:
437                             input_json = self.load_file(input_type, self.json_path,
438     json_name)                  final_json["catalysts"].append(mod_answer)
439
440                     if transpose_bool:
441                         final_result.append(final_json)
442
443                     # Final JSON after property modifications
444                     if final_result[0] == []:
445                         input_json = self.load_file('json', self.json_path, json_name)
446                         new_json = input_json
447                     else:
448                         new_json = final_result[0]
449
450                     # Handle representation questions
451                     remove_elements = []
452                     representation_result = {}
453                     for key, value in self.questions_for_representation.items():
454                         messages = []
455                         if value[0] == "None":
456                             pass
457                         else:
458                             messages.append({"role": "user", "content":
459     self.input_prompt(input_representation, new_json, value[0])})
460                         question = value[1]
461                         messages.append({"role": "user", "content": question})
462                         messages, answer = self.prompt(messages)
463                         question_token_list.append(messages)
464                         answer_token_list.append(answer)
465
466                         mod_answer = self.formatting_type(key, answer)
467
468                         question_list.append(question)
469                         answer_list.append(mod_answer)
470
471                         print('--------------------------')
472                         print(question)
473                         print(mod_answer)
```

```
                    print('--------------------------')

                    messages.append({"role": "assistant", "content": answer})
                    representation_result[key[0]] = mod_answer.replace(".", "").lower()

                if representation_result['1'] == 'no' and representation_result['2'] ==
'no':                 remove_elements.append('reaction_type')
                if representation_result['3'] == 'no' and representation_result['4'] ==
'no':                 remove_elements.append('substrate')
                if representation_result['5'] == 'no' and representation_result['6'] ==
'no':                 remove_elements.append('electrolyte')

                remove_elements = list(set(remove_elements))

                if remove_elements != []:
                    for delete_element in remove_elements:
                        messages = []
                        messages.append({"role": "user", "content":
self.input_prompt(input_question=f"Remove all elements with the key name
{delete_element} from the input JSON and display it in only JSON format. Other
explanation is not allowed. Show only JSON results, display the JSON like
string not ```json)".format(delete_element=delete_element)})
                        question_token_list.append(messages)
                        answer_token_list.append(answer)
                        try:
                            mod_answer = self.formatting_type('1_dict', answer)
                        except:
                            try:
                                messages, answer = self.prompt(messages)
                                question_token_list.append(messages)
                                answer_token_list.append(answer)
                            except:
                                print('#$@%@#%@#%#@%@#%@#')
                                print(json_name)
                                print('#$@%@#%@#%#@%@#%@#')
                                error_file_name.append(json_name)
                                break

                        question_list.append(question)
                        answer_list.append(mod_answer)
                        print('--------------------------')
                        print(question)
                        print('answer')
                        print(answer)
                        print('mod answer')
                        print(mod_answer)
                        print('--------------------------')

                        if not isinstance(mod_answer, dict):
                            count = 0
                            while not isinstance(mod_answer, dict) and count < 3:
                                # 원래 했던 질문 제거하고 다시
                                messages, answer = self.prompt(messages)
```

```
                                    messages, answer = self.prompt(messages)
                                    question_token_list.append(messages)
                                    answer_token_list.append(answer)
                                    try:
                                        mod_answer = self.formatting_type('1_dict', answer)
                                    except:
                                        try:
                                            messages, answer = self.prompt(messages)
                                            question_token_list.append(messages)
                                            answer_token_list.append(answer)
                                        except:
                                            print('#$@%@#%@#%#@%@#%@#')
                                            print(json_name)
                                            print('#$@%@#%@#%#@%@#%@#')
                                            error_file_name.append(json_name)
                                            break

                                    count += 1
                                    question_list.append(question)
                                    answer_list.append(mod_answer)

                                    print('--------------------------')
                                    print(question)
                                    print(mod_answer)
                                    print('--------------------------')

                            if not isinstance(mod_answer, dict):
                                new_json = new_json
                            else:
                                new_json = mod_answer
                        else:
                            mod_answer = new_json

                        # Ensure the necessary directories exist
                        os.makedirs(os.path.join(self.save_path, 'log'), exist_ok=True)
                        os.makedirs(os.path.join(self.save_path, 'token'), exist_ok=True)
                        os.makedirs(os.path.join(self.save_path, 'json'), exist_ok=True)

                        # Save the modified JSON and log
                        if json_name not in error_file_name:
                            log_path = self.save_path + 'log/'+ txt_name
                            df = pd.DataFrame({'Question': question_list, 'GPT answer':
    answer_list})       df.to_csv(log_path+'.csv', index=False)

                            token_path = self.save_path + 'token/'+ txt_name
                            token_df = pd.DataFrame({'Question': question_token_list, 'GPT
    answer': answer_token_list})    token_df.to_csv(token_path+'.csv', index=False)

                            new_json_path = self.save_path + 'json/'+ json_name
                            if mod_answer == [] :
                                input_json = self.load_file('json', self.json_path, json_name)
                                with open(new_json_path, "w") as json_file:
                                    json.dump(input_json, json_file, indent=4)
```

```
577                         else:
578                             if isinstance(mod_answer, list):
579                                 with open(new_json_path, "w") as json_file:
580                                     json.dump(new_json, json_file, indent=4)
581 
582                             elif isinstance(mod_answer, str):
583                                 with open(new_json_path, "w") as json_file:
584                                     json.dump(new_json, json_file, indent=4)
585                             else:
586                                 with open(new_json_path, "w") as json_file:
587                                     json.dump(mod_answer, json_file, indent=4)
588 
589 
590     if __name__ == "__main__":
591         representation_path = 'table representer path'
592         json_path = 'gpt prediction'
593         save_path = 'save path'
594 
595         assistant = FollowQ(json_path, representation_path, save_path)
596         assistant.run('txt', 'split')
597 
598 
```