Supplementary Information

This file includes prompts of precise relation extraction, prompts of comprehensive relation extraction, codes of precise relation extraction, and codes of comprehensive relation extraction.

1. Prompts of Precise Relation Extraction

（1）Entity extraction

system_content = "You're an expert in the biomedical field, well versed in named entity recognition tasks."

extractre_prompt = ('Extract all chemical and Disease entities in the above text. The output is in json format only.The keys are Chemical and Disease. For example :{"Chemical": [],"Disease": []},The final output is only JSON format in the code box, without much explanation.')

（2）Relation extraction

system_content = '''You are an expert in the field of biomedicine and know a lot about the relationship between chemicals and disease.'''

extractre_prompt = ("According to the above text as well as the list of chemical substances and the list of diseases.To extract relation triples, simply output the json format with relation_id, Chemical, Disease, and relation(induced or treated) as keys. For example :[{}, {}, {}, {}]. In the process of output results, it is necessary to judge the relationship between each entity in the list of chemical substances and each entity in the list of diseases, which is very serious and wants to be as complete and accurate as possible.")

（3）Follow-up inquiry

emotion_prompt = f'Take pride in your work and give it your best. Your commitment to excellence sets you apart.'

judge_prompt = (f'{text}\n Only answer yes or no, understand the text carefully, and determine whether there is a "causative effect" relationship between {chemical} and {disease}.')

re_judge_prompt = (Only answer yes or no, understand the text carefully, and determine whether there is a "therapeutic effect" relationship between {chemical} and {disease}.')

judge_prompt = (f'{text}\n Only answer yes or no, understand the text carefully, and determine whether there is a "therapeutic effect" relationship between {chemical} and {disease}.')

re_judge_prompt = ('Only answer yes or no, understand the text carefully, and determine whether there is a "causative effect" relationship between {chemical} and {disease}.')

（4）Semantic disambiguation

chemical_messages1 = [{'role': 'system', 'content': 'You are an expert in the biomedical field and need to do semantic disambiguation of entities.'},

{'role':'user','content': '"serotonin" and which of the list ["antidepressants","lithium", "serotonin reuptake inhibitors"] is most similar, returning only one value from the list.'},

{'role': 'assistant', 'content': 'serotonin reuptake inhibitors'},
{'role': 'user','content': chemical_content1}]
chemical_content2 = f'Only answer yes or no to whether "{chemical}" and "{chemical_response1}" are semantically similar.'
chemical_messages2 = [
{'role': 'system', 'content': 'You are an expert in the biomedical field, and you know a lot about entities.'},
{'role': 'user','content': chemical_content2}]
disease_content1 = f'"{disease}" and which of the list "{disease_list}" is most similar, returning only one value from the list.'
disease_messages1 = [{'role': 'system', 'content': 'You are an expert in the biomedical field and need to do semantic disambiguation of entities.'},
{'role': 'user','content': '"serotonin" and which of the list ["antidepressants","lithium", "serotonin reuptake inhibitors"] is most similar, returning only one value from the list.'},
{'role': 'assistant', 'content': 'serotonin reuptake inhibitors'},
{'role': 'user', 'content': disease_content1}]
disease_content2 = f'Only answer yes or no to whether "{disease}" and "{disease_response1}" are semantically similar.'
disease_messages2 = [
{'role': 'system', 'content': 'You are an expert in the biomedical field, and you know a lot about entities.'},{'role': 'user', 'content': disease_content2}]

2. Prompts of Comparison for Comprehensive Relation Extraction

(1) Main relation extraction

system_content = '''You are an expert in the field of biomedicine and know a lot about the relationship between chemicals and disease.'''
extractre_prompt = ("Fully understand the title (first sentence) and the text content, combined with the provided list of Chemical and Disease entities, summarize the main research content of the text, and then just extract the most important 1-2 Chemical Disease relationships. The final answer only needs to return the most important relation extracted, simply output the json format, and use relation_id, Chemical, Disease, and relation(induced or treated) as keys. For example :[{}, {}]. Note: Entities are extracted from the list and no additional explanation is required.")

(2) Text structuring and Side effects and condition extraction

promp = ("Please read the above abstract in English and arrange the text in the format OBJECTIVES, METHODS (experimental part), RESULTS and CONCLUSION. Make the structure of the paragraph summary more clear, and finally output only OBJECTIVES, RESULTS and CONCLUSION, and delete the METHODS part.")
input_text = (f'Fully understanding the above structured text and combining the information the text is intended to express, Output the following information for relational triples:[{che},{relation},{dis}] occur,1."adverse_reactions": If the relationship is treated, what other adverse reactions can occur when {che},{relation},{dis}? If relationship as induced, fill in "none".

2."accelerate_factor":Based solely on the provided text, Extract factors that promote or exacerbate the occurrence of relationships.
3."PreCondition":Extraction what are the prerequisites for a relationship to occur? If precondition is only a use of {che}/takeing {che}, no output is required.
4."Mitigating_factors": Extracting the factors only from the provided text can prevent or mitigate the occurrence of {relation}.'
Note: Answer the questions strictly according to the above structured text provided, do not rely on existing knowledge.Finally only output in json key-value pair format, and summarize the corresponding value of the key with a few phrases,concise and easy to understand, not redundant and complex.'''{

```
                    "adverse_reactions": "",
                    "accelerate_factor": "",
                    "PreCondition": "",
                    "Mitigating_factors": ""
                         }''')
```

(3) Follow-up inquiry
inquiry_prompt = 'According to your answer, please make sure again whether your answer is correct or not, which is very important to me. Just answer Yes or No'
again_prmopt = ('Please give the correct answer again according to the text information provided above. This is important to me.')

3. Codes of Precise Relation Extraction

（1）Entity extraction

```
import openai
import os
import json

openai.api_key = '**********'
file_path = r"text path"
output_file = r"entity output path"


def prompt(Q):
    response = openai.ChatCompletion.create(
        # model="gpt-4-0125-preview",
        model="claude-3-opus-20240229",
        messages=Q,
        temperature=0,
        # max_tokens=500,
        frequency_penalty=0,
        presence_penalty=0
    )
    return response['choices'][0]['message']['content']
```

```
system_content = "You're an expert in the biomedical field, well versed in named entity recognition tasks."
extractre_prompt = ('Extract all chemical and Disease entities in the above text. The output is in json format only. '
                    'The keys are Chemical and Disease. For example :{"Chemical": [],"Disease": []}'
                    'The final output is only JSON format in the code box, without much explanation.')
json_data = []
with open(file_path, 'r', encoding='utf-8') as file:

    i = 0
    for line in file:
        promp = line.strip() + extractre_prompt
        sss = [{"role": "system", "content": system_content}, {"role": "user", "content": promp}]
        ans = prompt(sss)
        try:
            ans = json.loads(ans.replace("```json", "").replace("```", ""))
            data = {
                "question": line.strip(),
                "answer": ans
            }
            i += 1
            json_data.append(data)
        except json.decoder.JSONDecodeError as e:
            data = {
                "question": line.strip(),
                "answer": ans
            }
            print("error {}".format(i))
            print(ans)
            i += 1
            json_data.append(data)
            continue
        with open(output_file, 'w', encoding='utf-8') as json_file:
            json.dump(json_data, json_file, ensure_ascii=False, indent=4)
```

（2）Relation extraction

```
import openai
import json

openai.api_key = '****'
```

```
file_path = r'*****'
output_file = r'****'

def prompt(Q):
        response = openai.ChatCompletion.create(
                # model="gpt-4-0125-preview",
                model="gpt-3.5-turbo-0125",
                messages=Q,
                temperature=0,
                # max_tokens=500,
                frequency_penalty=0,
                presence_penalty=0
        )
        return response['choices'][0]['message']['content']


system_content = '''You are an expert in the field of biomedicine and know a lot about the
relationship between
chemicals and disease.'''

extractre_prompt = ("According to the above text as well as the list of chemical substances
and the list of "
                                        "diseases.To extract relation triples, simply output the json format
with relation_id, Chemical, "
                                        "Disease, and relation(induced or treated) as keys. For example :[{},
{}, {}, {}]. In the process "
                                        "of output results, it is necessary to judge the relationship between
each entity in the list of "
                                        "chemical substances and each entity in the list of diseases, which is
very serious and wants to "
                                        "be as complete and accurate as possible.")
json_data = []
with open(file_path, 'r', encoding='utf-8') as file:
        i = 1
        lines = file.readlines()
        for line in lines:
                promp = line.strip() + extractre_prompt
                sss = [{"role": "system", "content": system_content}]
                sss.append({"role": "user", "content": promp})
                ans = prompt(sss)
                try:
                        ans = json.loads(ans.replace("```json", "").replace("```", ""))
                        data = {
                                "question": line.strip(),
```

```
                    "answer": ans
                }
                json_data.append(data)
                i += 1
        except json.decoder.JSONDecodeError as e:
                print("JSON parsing error:", e)
                print("Error{}".format(i))
                i += 1
                continue
        with open(output_file, 'w', encoding='utf-8') as json_file:
                json.dump(json_data, json_file, ensure_ascii=False, indent=4)
```

（3）Follow-up inquiry

```
import openai
import json

openai.api_key = '***'
def get_completion(prompt, model="gpt-3.5-turbo-0125"):
        messages = [{"role": "user", "content": prompt}]
        response = openai.ChatCompletion.create(
                model=model,
                messages=messages,
                temperature=0,
        )
        return response.choices[0].message["content"]


def get_completion_from_messages(messages, model="gpt-3.5-turbo-0125", temperature=0):
        response = openai.ChatCompletion.create(
                model=model,
                messages=messages,
                temperature=temperature,
        )
        return response.choices[0].message["content"]


system_content = ("You are an expert in the field of biomedicine and are very familiar with the relationship between "
                                    "chemicals and diseases")

re_file_path = r"Relational triplet file path"
text_file_path = r"Text path"

with open(text_file_path, 'r', encoding='utf-8') as file:
    lines = file.readlines()
with open(re_file_path, 'r') as f:
    datas = json.load(f)

output_file = r"Define the output path after the question is asked"
existing_data = []
i = 1

for data, text in zip(datas, lines):
    relations = data["answer"]
    change_num = 0
    change_re = []
    delete_num = 0
    delete_re = []
    for relation in relations:
        emotion_prompt = f'Take pride in your work and give it your best. Your
commitment to excellence sets you apart.'
        if relation['relation'] == 'induced':
            chemical = relation['Chemical']
            disease = relation['Disease']
            judge_prompt = (f'{text}\n'
                                f'Only answer yes or no, understand the text carefully, and
determine whether there '
                                f'is a "causative effect" relationship between {chemical}
and {disease}.')
            message = [
                {'role': 'system', 'content': system_content},
                {'role': 'user', 'content': judge_prompt}
            ]
            ans = get_completion_from_messages(message)
            if 'yes' in ans.lower():
                continue
            elif 'no' in ans.lower():
                re_judge_prompt = (
                    f'Only answer yes or no, understand the text carefully, and determine
whether there '
                    f'is a "therapeutic effect" relationship between {chemical} and
{disease}.')
                message.append({"role": "assistant", "content": ans})
                message.append({"role": "user", "content": re_judge_prompt})
                ans = get_completion_from_messages(message)
                if 'yes' in ans.lower():
                    relation['relation'] = 'treated'
                    change_num += 1
                    change_re.append(relation['relation_id'])
                elif 'no' in ans.lower():
                    relations.remove(relation)
                    delete_num += 1
                    delete_re.append(relation['relation_id'])
                else:
                    print("number{}:".format(relation['relation_id']), ans)

            else:
                print("number{}:".format(relation['relation_id']), ans)
        elif relation['relation'] == 'treated':
            chemical = relation['Chemical']
            disease = relation['Disease']
            judge_prompt = (f'{text}\n'
                            f'Only answer yes or no, understand the text carefully, and
determine whether there '
                            f'is a "therapeutic effect" relationship between {chemical}
and {disease}.')
            message = [
                {'role': 'system', 'content': system_content},
                {'role': 'user', 'content': judge_prompt}
            ]
            ans = get_completion_from_messages(message)
            if 'yes' in ans.lower():
                continue
            elif 'no' in ans.lower():
                re_judge_prompt = (
                    f'Only answer yes or no, understand the text carefully, and determine
whether there '
                    f'is a "causative effect" relationship between {chemical} and
{disease}.')
                message.append({"role": "assistant", "content": ans})
                message.append({"role": "user", "content": re_judge_prompt})
                ans = get_completion_from_messages(message)
                if 'yes' in ans.lower():
                    relation['relation'] = 'induced'
                    change_num += 1
                    change_re.append(relation['relation_id'])
                elif 'no' in ans.lower():
                    relations.remove(relation)
                    delete_num += 1
                    delete_re.append(relation['relation_id'])
                else:
                    print("number{}:".format(relation['relation_id']), ans)
            else:
                print("number{}:".format(relation['relation_id']), ans)
        else:
            continue
    i += 1
    existing_data.append(data)
    with open(output_file, 'w') as out_f:
        json.dump(existing_data, out_f, indent=4)
```

（4）Semantic disambiguation

```
# -*- coding: utf-8 -*-

import openai
import json
openai.api_key = '***'


def get_completion(prompt, model="gpt-3.5-turbo-0125"):
    messages = [{"role": "user", "content": prompt}]
    response = openai.ChatCompletion.create(
        model=model,
        messages=messages,
        temperature=0,
    )
    return response.choices[0].message["content"]

def get_completion_from_messages(messages, model="gpt-3.5-turbo-0125", temperature=0):
    response = openai.ChatCompletion.create(
        model=model,
        messages=messages,
        temperature=temperature,
    )
    return response.choices[0].message["content"]


def evaluate_relations(predicted_relations, true_relations):
    chemical_num = 0
    disease_num = 0
    true_positives = 0
    false_positives = 0
    true_relation_mesh = true_relations["relation"]
    yes_re = []
    re_mesh = []
    re = []
    global chemical_list, disease_list
    for predicted_relation in predicted_relations:
        chemical = predicted_relation["Chemical"].strip().replace(" ", "").lower()
        disease = predicted_relation["Disease"].strip().replace(" ", "").lower()
        relation_type = predicted_relation["relation"]
        # Check if chemical and disease entities have corresponding MESH values
        chemical_mesh = None
        disease_mesh = None

        chemical_list = []
        for mesh, entities in true_relations["Chemical"].items():
            chemical_list.extend(entities)
            cleaned_entities = [entity.strip().replace(" ", "").lower() for entity in entities]
            if chemical in cleaned_entities:
                chemical_mesh = mesh
        if chemical_mesh is None:
            chemical = predicted_relation["Chemical"]
            chemical_list = list(set(chemical_list))
            chemical_content1 = f'"{chemical}" and which of the list "{chemical_list}" is
most similar, returning only one value from the list.'
            chemical_messages1 = [
                {'role': 'system', 'content': 'You are an expert in the biomedical field and
need to do semantic disambiguation of entities.'},
                {'role': 'user',
                 'content': '"serotonin" and which of the list ["antidepressants","lithium",
"serotonin reuptake inhibitors"] is most similar, returning only one value from the list.'},
                {'role': 'assistant', 'content': 'serotonin reuptake inhibitors'},
                {'role': 'user','content': chemical_content1}]
            chemical_response1 = get_completion_from_messages(chemical_messages1,
temperature=0)
            chemical_content2 = f'Only answer yes or no to whether "{chemical}" and
"{chemical_response1}" are semantically similar.'
            chemical_messages2 = [
                {'role': 'system', 'content': 'You are an expert in the biomedical field, and
you know a lot about entities.'},
                {'role': 'user',
                 'content': chemical_content2}
            ]
            chemical_response2 = get_completion_from_messages(chemical_messages2,
temperature=0)
            # print(chemical_response2)
            if 'yes' in chemical_response2.lower():
                for mesh, entities in true_relations["Chemical"].items():
                    cleaned_entities = [entity.strip().replace(" ", "").lower() for entity in
entities]
                    chemical_resp1 = chemical_response1.strip().replace(" ", "").lower()
                    if chemical_resp1 in cleaned_entities:
                        print("Chemical:Succeeded in replacing {} with
{}".format(predicted_relation["Chemical"], chemical_response1))
                        predicted_relation["Chemical"] =
true_relations["Chemical"][mesh][0]

                        chemical_mesh = mesh
                        chemical_num += 1



        disease_list = []
        for mesh, entities in true_relations["Disease"].items():
            disease_list.extend(entities)
            cleaned_entities = [entity.strip().replace(" ", "").lower() for entity in entities]
            if disease in cleaned_entities:
                disease_mesh = mesh
        if disease_mesh is None:
            disease = predicted_relation["Disease"]
            disease_list = list(set(disease_list))
            disease_content1 = f'"{disease}" and which of the list "{disease_list}" is most
similar, returning only one value from the list.'
            disease_messages1 = [
                {'role': 'system', 'content': 'You are an expert in the biomedical field and
need to do semantic disambiguation of entities.'},
                {'role': 'user',
                 'content': '"serotonin" and which of the list ["antidepressants","lithium",
"serotonin reuptake inhibitors"] is most similar, returning only one value from the list.'},
                {'role': 'assistant', 'content': 'serotonin reuptake inhibitors'},
                {'role': 'user', 'content': disease_content1}]
            disease_response1 = get_completion_from_messages(disease_messages1,
temperature=0)
            disease_content2 = f'Only answer yes or no to whether "{disease}" and
"{disease_response1}" are semantically similar.'
            disease_messages2 = [
                {'role': 'system', 'content': 'You are an expert in the biomedical field, and
you know a lot about entities.'},
                {'role': 'user', 'content': disease_content2}
            ]
            disease_response2 = get_completion_from_messages(disease_messages2,
temperature=0)
            if 'yes' in str(disease_response2.lower()):
                for mesh, entities in true_relations["Disease"].items():
                    cleaned_entities = [entity.strip().replace(" ", "").lower() for entity in
entities]
                    disease_resp1 = disease_response1.strip().replace(" ", "").lower()
                    if disease_resp1 in cleaned_entities:
                        disease_mesh = mesh
                        print("Disease:Succeeded in replacing {} with
{}".format(predicted_relation["Disease"], disease_response1))
                        predicted_relation["Disease"] =
true_relations["Disease"][mesh][0]

                        disease_num += 1
        if chemical_mesh is not None and disease_mesh is not None:
            if [chemical_mesh, disease_mesh] not in re_mesh:
                re_mesh.append([chemical_mesh, disease_mesh])
                re.append(predicted_relation)
            for relation_typ,relation_list in true_relation_mesh.items():
                if [chemical_mesh, disease_mesh] in relation_list.values():
                    # Check if the predicted relation type matches the true relation type
                    if relation_type == "induced" and "Positive_Correlation" ==
relation_typ:
                        true_positives += 1
                        yes_re.append([chemical_mesh, disease_mesh])
                        break
                    elif relation_type == "treated" and "Negative_Correlation" ==
relation_typ:
                        true_positives += 1
                        yes_re.append([chemical_mesh, disease_mesh])
                        break
            else:
                false_positives += 1
        else:
            false_positives += 1
            re.append(predicted_relation)
    yes_re = [list(x) for x in set(tuple(x) for x in yes_re)]
    recall_true_positives = len(yes_re)
    false_negatives =
len(true_relation_mesh['Positive_Correlation'])+len(true_relation_mesh['Negative_Correlatio
n']) - true_positives
    precision = true_positives / (true_positives + false_positives) if (true_positives +
false_positives) > 0 else 0
    recall = recall_true_positives / (true_positives + false_negatives) if (true_positives +
false_negatives) > 0 else 0
    f1_score = 2 * (precision * recall) / (precision + recall) if (precision + recall) > 0 else 0
    return precision, recall, f1_score, chemical_num, disease_num,re

with open(r'\200_gold_re.json', 'r') as f:
    true_relations_data = json.load(f)

with open(r'predict_re.json', 'r') as f:
    predicted_relations_data = json.load(f)

all_precision = 0
all_recall = 0
all_f1_score = 0
chemical_num = 0
disease_num = 0
i = 0
output_file = r'Define the output path'
re_json = []
for predicted_relations, true_relations in zip(predicted_relations_data, true_relations_data):
    precision, recall, f1_score, chemicalnum, diseasenum, re = evaluate_relations(predicted_relations["answer"], true_relations)
    all_precision += precision
    all_recall += recall
    all_f1_score += f1_score
    i += 1
    chemical_num += chemicalnum
    disease_num += diseasenum
    predicted_relations["answer"] = re
    re_json.append(predicted_relations)
with open(output_file, 'w') as out_f:
    json.dump(re_json, out_f, indent=4)
F1 = 2*(all_precision/200 * all_recall/200)/(all_precision/200 + all_recall/200)
print("Precision:{},Recall:{},F1 Score:{}".format(all_precision/200, all_recall/200, F1))
```

4. Code of Comparison for Comprehensive Relation Extraction

(1) Main relation extraction

```
import openai
import os
import json

openai.api_key = '***'
file_path = r"text path"
output_file = r"mainre output path"
```

```
def prompt(Q):
    response = openai.ChatCompletion.create(
        # model="gpt-4-0125-preview",
        # model = "gpt-3.5-turbo-0125",
        model="gpt-3.5-turbo-0125",
        messages=Q,
        temperature=0,
        # max_tokens=500,
        frequency_penalty=0,
        presence_penalty=0
    )
    return response['choices'][0]['message']['content']


system_content = '''You are an expert in the field of biomedicine and know a lot about the
relationship between
chemicals and disease.'''

extractre_prompt = ("Fully understand the title (first sentence) and the text content, combined
with the provided list "
                    "of Chemical and Disease entities, summarize the main research
content of the text, and then just "
                    "extract the most important 1-2 Chemical Disease relationships. The
final answer only needs to "
                    "return the most important relation extracted, simply output the json
format, and use relation_id, "
                    "Chemical, Disease, and relation(induced or treated) as keys. For
example :[{}, {}]. Note: "
                    "Entities are extracted from the list and no additional explanation is
required.")
json_data = []

with open(file_path, 'r', encoding='utf-8') as file:
    lines = file.readlines()

i = 1
for line in zip(lines):
    promp = line.strip() + '\n' + extractre_prompt
    sss = [{"role": "system", "content": system_content}]
    sss.append({"role": "user", "content": promp})
    ans = prompt(sss)
    try:
        ans = json.loads(ans.replace("```json", "").replace("```", ""))
```

```
            data = {
                "question": line.strip(),
                "answer": ans
            }
            json_data.append(data)
            i += 1
        except json.decoder.JSONDecodeError as e:
            print("error{}".format(i))
            print(ans)
            i += 1
            continue
        with open(output_file, 'w', encoding='utf-8') as json_file:
            json.dump(json_data, json_file, ensure_ascii=False, indent=4)
```

(2) Text structuring and Side effects and condition extraction

```
import openai
import json

openai.api_key = '****'

file_path = r"text path"
re_file = r"mainre path"

def prompt(Q):
    response = openai.ChatCompletion.create(
        model="gpt-3.5-turbo-0125",
        messages=Q,
        temperature=0,
        frequency_penalty=0,
        presence_penalty=0
    )
    return response['choices'][0]['message']['content']


with open(re_file, 'r') as f:
    datas = json.load(f)

with open(file_path, 'r', encoding='utf-8') as file:
    lines = file.readlines()

num = 1
json_data = []
for ids, line in zip(range(0, len(datas)), lines):
    re = datas[ids]
    re_answer_str = json.dumps(re["answer"], ensure_ascii=False)
    #Structuring is optional
    promp = (
        f"{line} + '\n'"
        "Please read the above abstract in English and arrange the text in the format OBJECTIVES, METHODS ("
        "experimental part), RESULTS and CONCLUSION. Make the structure of the paragraph summary more clear, "
        "and finally output only OBJECTIVES, RESULTS and CONCLUSION, and delete the METHODS part.")
    sss = [{"role": "user", "content": promp}]
    ans = prompt(sss)
    j = 1
    mainre = {}
    for i in range(len(re["answer"])):
        re1 = re["answer"][i]
        che = re1["Chemical"]
        dis = re1["Disease"]
        relation = re1["relation"]
        input_text = (
            f'Fully understanding the above structured text and combining the information the text is intended to express, Output the following information for relational triples:[{che},{relation},{dis}] occur,'
            f'1."adverse_reactions": If the relationship is treated, what other adverse reactions can occur when {che},{relation},{dis}? If relationship as induced, fill in "none".'
            f'2."accelerate_factor":Based solely on the provided text, Extract factors that promote or exacerbate the occurrence of relationships.'
            f'3."PreCondition":Extraction what are the prerequisites for a relationship to occur? If precondition is only a use of {che}/takeing {che}, no output is required.'
            f'4."Mitigating_factors": Extracting the factors only from the provided text can prevent or mitigate the occurrence of {relation}.'
            f'Note: Answer the questions strictly according to the above structured text provided, do not rely on existing '
            f'knowledge.Finally only output in json key-value pair format, and summarize the corresponding value of the key with a few phrases,'
            f'concise and easy to understand, not redundant and complex.'
            '''{
                "adverse_reactions": "",
                "accelerate_factor": "",
                "PreCondition": "",
                "Mitigating_factors": ""
                    }''')
        sss.append({"role": 'assistant', "content": ans})
        sss.append({"role": 'user', "content": input_text})
```

```
            ans1 = prompt(sss)
            try:
                ans1 = json.loads(ans1.replace("```json", "").replace("```", ""))
                id = i+1
                mainre.update({
                    f'{id}': {
                        "chemical": f"{che}",
                        "disease": f"{dis}",
                        "relation": f"{relation}",
                        "condition": ans1
                    }
                })
            except json.decoder.JSONDecodeError as e:
                print("Error{}".format(num))
                print(ans1)
                continue
        data = {
            "abstract": line.strip(),
            "mainre": mainre
        }
        json_data.append(data)
        num += 1
        with open(r'output path', 'w', encoding='utf-8') as json_file:
            json.dump(json_data, json_file, ensure_ascii=False, indent=4)
```

(3) Follow-up inquiry

```
# -*- coding: utf-8 -*-


import openai
import os
import json

openai.api_key = '*****'
file_path = r"text path"
re_file = r"mianre path"

def prompt(Q):
    response = openai.ChatCompletion.create(
        # model="gpt-3.5-turbo-0125",
        model="gpt-4-turbo-2024-04-09",
        messages=Q,
        temperature=0,
        frequency_penalty=0,
```

```
        presence_penalty=0
    )
    return response['choices'][0]['message']['content']


with open(re_file, 'r') as f:
    datas = json.load(f)

with open(file_path, 'r', encoding='utf-8') as file:
    lines = file.readlines()

num = 1
json_data = []
for ids, line in zip(range(0, len(datas)), lines):
    re = datas[ids]
    re_answer_str = json.dumps(re["answer"], ensure_ascii=False)
    promp = (
        f"{line} + '\n'"
        "Please read the above abstract in English and arrange the text in the format OBJECTIVES, METHODS ("
        "experimental part), RESULTS and CONCLUSION. Make the structure of the paragraph summary more clear, "
        "and finally output only OBJECTIVES, RESULTS and CONCLUSION, and delete the METHODS part.")
    sss = [{"role": "user", "content": promp}]
    ans = prompt(sss)
    j = 1
    mainre = {}
    for i in range(len(re["answer"])):
        re1 = re["answer"][i]
        che = re1["Chemical"]
        dis = re1["Disease"]
        relation = re1["relation"]
        input_text = (
            f'Fully understanding the above text and combining the information the text is intended to express, Output the following information for relational triples:[{che},{relation},{dis}] occur,'
            f'1."adverse_reactions": If the relationship is treated, what other adverse reactions can occur when {che},{relation},{dis}? If relationship as induced, fill in "none".'
            f'2."accelerate_factor":Based solely on the provided text, Extract factors that promote or exacerbate the occurrence of relationships.'
            f'3."PreCondition":Extraction what are the prerequisites for a relationship to occur? If precondition is only a use of {che}/takeing {che}, no output is required.'
            f'4."Mitigating_factors": Extracting the factors only from the provided text can
```

```
prevent or mitigate the occurrence of {relation}.'
            f'Note: Answer the questions strictly according to the above text provided, do
not rely on existing '
            f'knowledge.Finally only output in json key-value pair format, and summarize
the corresponding value of the key with a few phrases,'
            f"concise and easy to understand, not redundant and complex.Let's think step
by step"
            '''{
                "adverse_reactions": "",
                "accelerate_factor": "",
                "PreCondition": "",
                "Mitigating_factors": ""
                    }''')
        sss.append({"role": 'assistant', "content": ans})
        sss.append({"role": 'user', "content": input_text})
        ans1 = prompt(sss)
        sss.append({"role": 'assistant', "content": ans1})
        inquiry_prompt = 'According to your answer, please make sure again whether your
answer is correct or not, which is very important to me. Just answer Yes or No'
        sss.append({"role": 'user', "content": inquiry_prompt})
        ans2 = prompt(sss)
        if 'yes' in ans2.lower():
            try:
                ans1 = json.loads(ans1.replace("```json", "").replace("```", ""))
                id = i+1
                mainre.update({
                    f'{id}': {
                        "chemical": f"{che}",
                        "disease": f"{dis}",
                        "relation": f"{relation}",
                        "condition": ans1
                    }
                })
            except json.decoder.JSONDecodeError as e:
                print("error{}".format(num))
                print(ans1)
                continue
        elif 'no' in ans2.lower():
            again_prmopt = ('Please give the correct answer again according to the text
information provided above. '
                            'This is important to me.')
            sss.append({"role": 'assistant', "content": ans2})
            sss.append({"role": 'user', "content": again_prmopt})
            ans3 = prompt(sss)
```

````
        try:
            ans3 = json.loads(ans3.replace("```json", "").replace("```", ""))
            id = i+1
            mainre.update({
                f'{id}': {
                    "chemical": f"{che}",
                    "disease": f"{dis}",
                    "relation": f"{relation}",
                    "condition": ans3
                }
            })
        except json.decoder.JSONDecodeError as e:
            print("error{}".format(num))
            print(ans1)
            continue
data = {
    "abstract": line.strip(),
    "mainre": mainre
}
json_data.append(data)
num += 1
with open(r'output file', 'w', encoding='utf-8') as json_file:
    json.dump(json_data, json_file, ensure_ascii=False, indent=4)
````