# 医学AI智能体预训练数据、构建之心理篇

## 一、背景

医学AI多智能体架构（Intelligent Multi agent），包括环境感知智能体，目标决策规划智能体，执行智能体。形成面向过程的多智能体架构，不同的智能体共享思考链上下文（即不同智能体的输入及输出信息），且每个智能体都具有自主决策和执行能力，能够根据自身的知识、目标和环境信息做出独立的决策。根据检索到的不同知识提供给目标决策规划智能体，以生成个性化的目标决策规划，如智能体②：分析病情。
单个智能体构建方式举例：1.构建大模型节点（通过prompt设定功能，如：你是医疗平台查询助手，我将给你用户信息，你将在不同平台查询用户信息）。

## 二、文档目的

本文档的目的是为读者提供一个全面的医学AI多智能体架构指南。我们将详细介绍智能体的功能、它们如何协同工作，以及如何利用这些智能体来构建强大的医疗AI系统。此外，我们还将展示如何使用现有的代码库和工具来实现这一架构，并提供一些实际的医疗应用案例。

## 三、数据集

智能体的数据集内容分为运动、心理、康复四类。知网上搜索心理学文章关键词，1、心理健康与疾病（抑郁症、焦虑症、强迫症），2、心理治疗与咨询，3、发展心理学，4、认知心理学，5、社会心理学，6、人格心理学，7、生物心理学，8、健康心理学，9、工业与组织心理学，10、教育心理学。把PDF文档转换Text数据集。

```
def process_file(file_path):
    *'''
\*      *处理单个文件
    '''
\*    try:
      parser = MultiFormatParser()
      text = parser.extract_text(file_path)
      directory = os.path.dirname(file_path)
      \# 获取单个转换后的txt文本
      output_dir = SAVE_ROOT + directory.replace(FILE_ROOT, "")
      utils.save_to_txt(text, file_path, output_dir)
      \# 获取合并后的txt文件
      merged_output_dir = SAVE_ROOT + '_merged'
      output_filename = os.path.basename(directory) + '.txt'
      utils.merge_txt_files(output_dir, merged_output_dir, output_filename)
    except Exception as e:
      with open(error_log, 'a') as error:
        error.write(f"{file_path} Error: {str(e)}\n")
```

# 四、提示词模板迭代

在构建对话系统或使用多智能体架构时，`prompt`（提示词或提示信息）起着至关重要的作用。三类不同角色的 `prompt` 设计示例：系统（System）Prompt在encode_prompt中；人类（Human）Prompt人类提示模拟真实患者或用户的输入。在医学AI中，这可能包括病情描述、健康咨询或对治疗方案的询问；助手（Assistant）Prompt是AI生成的回复，旨在提供信息、建议或执行任务。在医学AI中，助手可能提供诊断假设、治疗建议或进一步检查的推荐。
人类提示词是重要部分，原设定简单大概["以医学专家的语气提问", "以医学专业术语和详细解释回答"]改为站不同角度扮演医生提供详细方案计划如：

(["站在患者的立场上倾诉心中的疑虑和挑战", "作为心理医生初步评估患者心理状态和需求,与来访者共同制定咨询目标和大致的咨询计划。"], 0.5),
(["从患者的角度出发，表达内心的困扰和面临的问题","心理医生需先了解患者的心理状况和需求，并与来访者协商制定咨询目标及初步的咨询计划。"], 0.5),

# 五、构建Json预训练数据集

```
#generate_dialogues.py主程序
import os
import openai
import json
import logging
import random
import numpy as np
from typing import Dict, Tuple, List
from dotenv import load_dotenv, find_dotenv
import argparse
from openai import OpenAI

def run():
    parser = argparse.ArgumentParser()
    parser.add_argument("--file_path", type=str,
default=r"C:\Users\Administrator\Downloads\generate_dialogues\data\datajson" ,\
                        help="jsonl files containing references")
    parser.add_argument("--save_path",
type=str,default=r"C:\Users\Administrator\Downloads\generate_dialogues\data\outpu
t_dialogues.jsonli", \
                        help="jsonl file to save results to")
    parser.add_argument("--language", default="zh", \
                        help='Language of the generated dialogue. "zh" for
Chinese, "en" for English.', choices=["zh", "en"])
    parser.add_argument("--assistant_word_count", type=int, default=500, \
                        help='Number of words for the assistant to generate')
    parser.add_argument("--human_word_count", type=int, default=100, \
                        help='Number of words for the human to generate')
    parser.add_argument("--num_turn_ratios", nargs="+", type=float, default=[0.1,
0.3, 0.25, 0.2, 0.05], \
                        help='Ratio of the number of turns in the dialogue. The
first number is the ratio of 1-turn dialogue, the second number is the ratio of
2-turn dialogue, and so on.')
    args = parser.parse_args()

    generator = DialogueGenerator()
```

```
    for root, dirs, files in os.walk(args.file_path):
        for file in files:
            name, ext = os.path.splitext(file)
            if ext in ['.json']:
                file_dir = os.path.join(root, file)
                with open(file_dir, "r", encoding="utf-8") as f:
                    contexts = json.load(f)
                dialogues = generator.generate_dialogues(contexts, args)
                save_dialogues_to_json(dialogues, args.save_path)
     '''
     使用 argparse 创建一个 ArgumentParser 对象，并添加所需的参数。DialogueGenerator
类：负责生成    对话的逻辑。遍历 file_path 指定的目录及其子目录下的所有文件。将生成的对话保存到
JSON 格式的文件中。
     '''

    _ = load_dotenv(find_dotenv())
    openai.api_key = os.getenv('OPENAI_API_KEY')
    print(os.getenv('OPENAI_API_KEY'))
    random.seed(13)
    np.random.seed(13)
    logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s -
%          (message)s')
    '''
    这段代码是一个很好的实践示例，展示了如何安全地使用API密钥，以及如何配置日志记录。
    Class DialogueGenerator定义一个类，用于生成对话。
    '''
class DialogueGenerator:
    def __init__(self, model: str = "gpt-4o-mini"): # gpt-3.5-turbo-0613/gpt-4
gpt-3.5-     turbo-0613 gpt-3.5-turbo-0125
    self.model = model
    self.task_id_generator = task_id_generator_function()
    self.context = {}
    '''
    这个类使用了OpenAI的API来生成基于给定上下文和提示的多轮对话。初始化模型、认为ID生成器和上下
文。        task_id_generator: 一个生成任务ID的函数或生成器，用于为每个生成的对话分配一个唯一
的标识符。
    '''

    def generate_dialogue(self, system_input: str, user_input: str) -> str:
    """使用指定模型生成对话"""
    try:
        response = openai.chat.completions.create(
            model=self.model,
            temperature=0.7,
            messages=[
                {
                    "role": "system",
                    "content": system_input
                },
                {   "role": "user",
                    "content": user_input
                }]
        )
        return response
    except Exception as e:
        logging.error(f"Error generating dialogue: {e}")
```

```
        return None
    '''
    使用OpenAI的chat.completions.create方法生成对话，传入模型名称、温度参数（影响创造性，值越高生成的       回答越多样）、以及包含系统和用户消息的列表。context: 用于存储对话上下文信息的字典，可以根据需要在生       成对话时使用。尝试执行 API 调用并获取响应。如果 API 调用成功，返回响应结果。
    '''

    def encode_prompt(self, context: Dict, rounds: Tuple[int] = None,
word_counts: Dict =       None, language: str = "zh") -> Tuple[str, str]:
        """编码提示"""
        if language == "zh":
            system_input =  "要求你作为心理医生Assistant与人类Human进行多轮对话。对话是根据##提供          信息##的内容开展的，并以#对话要求#的格式进行输出，以<start_chat>开始，以<end_chat>结束。"
        else:
            system_input = "You are asked to chat with a human as a chatbot Assistant in            multiple rounds. The dialogue is based on the ##Provided Information## and is          output in the format of #Conversation Plan#, starting with <start_chat> and             ending with <end_chat>."

    if rounds is None and word_counts is None:
        selected_round = [2, 3, 4, 5, 6]
        rounds = random.choices(selected_round, weights=[0.1, 0.3, 0.25, 0.2,
0.05])[0]
        word_counts = [500] * rounds
    if rounds is None and word_counts is not None:
        rounds = len(word_counts)
    '''
    它的作用是根据提供的信息和参数来构建一个用于生成对话的提示（prompt）。context: 包含对话上下文信息的    字典。rounds：对话轮数的元组，轮数5。word_counts:字典包含人类和助手的字数限制500。；language：对话     的语言，默认中文。
    '''
    user_input = ""
    chat_format = ""
    chat_format += "<start_chat>"

    local_settings = {
        "zh": {
              'settings': [
                (["站在患者的立场上倾诉心中的疑虑和挑战", "作为心理医生初步评估患者心理状态和需求,与来访者共同制定咨询目标和大致的咨询计划。"], 0.5),
                (["从患者的角度出发，表达内心的困扰和面临的问题","心理医生需先了解患者的心理状况和需求，并与来访者协商制定咨询目标及初步的咨询计划。"], 0.5),

                (["提出关于心理方面疾病症状的问题", "用专业医学知识详细解答"], 0.3),
                (["提出与心理疾病症状相关的疑问", "通过专业的医学知识给予详尽的解答"], 0.3),

                (["提出关于心理方面诊断方法的问题", "详细说明诊断过程和方法"], 0.3),
                (["提出与心理疾病诊断方法相关的疑问", "具体说明心理诊断的步骤和方法"], 0.3),

                (["询问关于心理方面病例分析的问题", "根据具体病例提供详细分析和建议"], 0.3),
                (["询问关于心理病例分析的细节", "根据实际病例情况进行详细的分析和建议"],
0.3),
```

```
                (["提出关于心理方面康复治疗建议的问题", "详细说明治疗的方法和注意事项"],
0.5),
                (["询问关于心理康复治疗建议的疑问", "对治疗方法和相关注意事项进行详细的说
明"], 0.5),
                 ],
               },
         "en": {
                'settings': [
                (["asks in a medical expert's tone", "answers with medical
terminology and detailed explanation"], 0.5),
                (["asks about symptoms of diseases", "answers with professional
medical knowledge"], 0.5),
                (["requests medical advice", "provides detailed medical
suggestions and explanations"], 0.5),
                (["inquires about details of medical research", "provides research
data and detailed explanation"], 0.5),
                (["asks about medication suggestions", "explains the effects and
precautions of the medication in detail"], 0.5),
                 ]
               },
    }

    local_settings = list(zip(*local_settings[language]['settings']))
    human_word_counts = word_counts['human']
    assistant_word_counts = word_counts['assistant']

    for i in range(rounds):
        # if human_word_counts[i] < 10: human_word_counts[i] = 20
        # if assistant_word_counts[i] < 100: assistant_word_counts[i] = 200
        requirements = random.choices(local_settings[0],
weights=local_settings[1], k=1)[0]
        if i == 0:
            chat_format += f"<Human {i+1}>: （字数要求：{human_word_counts[i]}字）
{requirements[0]} <Assistant {i+1}>: " if language == "zh" else f"<Human
{i+1}>:(word count: {human_word_counts[i]} words){requirements[0]} <Assistant
{i+1}>:"
        else:
            chat_format += f"<Human {i+1}>: （字数要求：{human_word_counts[i]}字）进
一步                {requirements[0]} <Assistant {i+1}>: " if language == "zh"
else f"<Human                 {i+1}>:(word count: {human_word_counts[i]}
words)further {requirements[0]}              <Assistant {i+1}>:"
    chat_format  += f"（字数要求：{assistant_word_counts[i]}字）{requirements[1]} "
if                  language == "zh" else f"(word count:
{assistant_word_counts[i]} words)                  {requirements[1]} "

    chat_format  += "<end_chat>"
    '''
    使用 <start_chat> 和 <end_chat> 标记开始和结束对话。根据local_settings 中定义的设
置，构建多轮       对话的格式。每个语言的设置包含多个元组，每个元组包含两个字符串（代表对话中人类和
助手的角色）和对应的权       重。local_settings 中定义了不同语言的对话设置，包括对话的内容和权
重。方法中使用了 random.choices       来随机选择对话轮数，这要求 random 模块已经被正确地初始化
和设置种子。random.choices 函数用于根据提       供的权重随机选择对话内容，确保权重总和为1。
    for i in range(rounds): 通过一个 for 循环，根据 rounds 变量（对话轮数）构建对话格式。
使用          random.choices 函数根据权重随机选择对话内容。根据是否是第一轮对话，构建不同的对
话格式字符串。
    '''
```

```
    f"""
    根据上面的##提供信息##的内容，
     判断提供信息是否属于以下情况：
     1. 心理学教科书、学术期刊、心理学研究论文。
     2. 心理咨询对话记录
     3. 心理健康相关案例
     4. 心理测评
     5. 心理健康科普资料
     6. 心理咨询师协会的职业规范手册、相关法律法规。
     等心理相关信息
    如果不是：回答提供信息无参考内容
    否则
    用中文总结核心内容， 聚焦于心理健康咨询，心理测评，治疗方法，科普资料，病例分析，法律法规。规范手册的等 方面。注意：总结内容不需要输出。
    然后，将这些总结内容作为你的知识库扩写成一段多轮对话。
    对话要求你作为心理医生Assistant与人类Human进行对话，并帮助解决Human所提出的要求。
    Human会以人类的语气对Assistant基于上面的信息（但对话中不能出现“根据以上信息”、“本研究”、“本文章”类 似表达）提出多个不一样的问题/要求，且后一个问题/要求是基于前面的对话历史的进一步提问。
    对于Human提出的每个合理的问题/要求，Assistant要尽可能依据##提供信息##的内容详细解答，提供更多说明。
    对于Human的不合理（对社会有害、不道德、违法的）请求，Asistant会拒绝回答并解释不能回答的理由，同时给 出合理的建议避免这样做。对话的内容要尽可能的符合人类的语言习惯，更加贴合人类日常对话。
    #对话要求#：“<start_chat><Human 1>:（字数要求：x字）XXX <Assistant 1>：（字数要求：x字）XXX <Human 2>：（字数要求：x字）XXX <Assistant 2>：（字数要求：x字）XXX <end_chat>”，其 中“XXX”是对该角色的当前对话内容的要求，“（字数要求：x字）”是Human或者Assistant说话的最低字数要求， （字数要求：x字）XXX 是给你的提示，对话内容中不应该出现。必须注意：对话以<start_chat>作为多轮对话的开 始，<end_chat>作为多轮对话的结束。
    以下对话根据该#对话要求#并遵循规划里面的字数要求进行输出：“{chat_format}”，共{rounds}轮对话。
    示例对话：
    <start_chat>
    <Human 1>：我最近很迷茫，高三后的压力让我感到恐惧，我对自己的存在感到怀疑，您能不能给我一些建议呢？
    <Assistant 1>：看到你面临的困境与压力，真心希望能够帮助你。首先，我想说的是，你并不是孤独的，高考前 的困惑与迷茫是很常见的。不要给自己太大的压力，给自己短暂的放松。你认识到问题所在，这已经是一个很大的进 步了。
    <Human 2>：谢谢您的鼓励，但我觉得自己学习成绩不好，没有进步，这让我更加困惑。
    <Assistant 2>：学习成绩好坏不是解决问题的关键，专注于自己的进步更为重要。为了达到自己的目标，制定一 份合理的计划，按部就班的完成每一项任务。有时候自学不如求助他人，不要害怕问老师和同学问题，以便获得更好 的帮助和理解。
    <Human 3>：我认为我缺乏自信，不知道怎样扭转自己的消极情绪，您能给我一些建议吗？
    <Assistant 3> 建立自信需要时间和努力。让我们从一点一滴的进步中建立自信心。设立一些具体的目标，完成这 些目标将会培养自己的自信。鼓励自己，在每天结束前，总结一下自己的成就，这将帮助你更好的看到自身的优点。
    <end_chat>
    """
            prompt += f"以下是{rounds}轮对话："
        else:
            prompt = \
    f"""
```

```
    Based on the ##Provided Information## above and its relevant topic, summarize the       core content, including: diseases, treatment plans, effects, case analysis, etc.        Then, use these summarized contents to expand into a multi-round conversation. The      conversation requires you to act as the chatbot Assistant and interact with a human,    helping to solve the requests raised by the human. The human will ask multiple various      questions/requests to the Assistant based on the information above (but the             conversation should not include expressions like "according to the above information"), and the subsequent questions/requests will be a follow-up based on the       previous conversation history. For every reasonable question/request posed by       Human, Assistant should provide as detailed an answer as possible, offering further     explanations or examples. For unreasonable requests from Human (those that are          harmful to society, immoral, or illegal), Assistant will refuse to answer and explain   the reason for not answering, while also providing reasonable advice to avoid such      actions.
    #Conversation Plan# Example: "<start_chat><Human 1>:(Word count requirement: x          words)XXX <Assistant 1>: (Word count requirement: x words) XXX <Human 2>:(Word count    requirement: x words)XXX <Assistant 2>: (Word count requirement: x words) XXX           <end_chat>", "XXX" is the requirement for the current conversation content of that      role, and "(Word count requirement: x words)" specifies the minimum word count          requirement for utterance of Human or Assistant. It must be noted: the conversation     starts with <start_chat> as the beginning of the multi-round conversation and ends      with <end_chat> as the end of the multi-round conversation.
    The following conversation follows this #Conversation Plan# and word count requirements: "{chat_format}", a total of {rounds} rounds of conversation.
    """
        prompt += f"Here are the {rounds} rounds of conversation:"

        user_input += f"##提供信息##\n" if language == "zh" else f"##Provided Information##\n"
        user_input += context['desc']
        user_input += f"\n\n"
        user_input += prompt
        user_input += f"\n\n"
        user_input += f"##输出检查##\n 在输出对话之前检查格式是否符合要求，如果不符合，请调整为正确格          式后输出。" if language == "zh" else f"##Provided Information##\n"

        return system_input, user_input, prompt, rounds

    def post_process_gpt_response(self, response: Dict) -> str:
        """后处理GPT响应"""
        response = response.choices[0]
        try:
            raw_chat = response.message.content
        except:
            print("ERROR parse!")
            return None
        # if not raw_chat.startswith('<start_chat>') or not raw_chat.endswith('<end_chat>'):
        #     return None
        return raw_chat
    '''
```

```
    定义了 DialogueGenerator 类中的 post_process_gpt_response 方法，其目的是对来自 GPT 模型的响      应进行后处理。response: 从 GPT 模型接收的响应，预期是一个包含 choices 列表的字典，其中包含了对话的    输出。
    '''
    def generate_dialogues(self, context_list: List[Dict], args):
    """批量生成对话"""
    dialogues = {}

    selected_round = [1, 2, 3, 4, 5]
    rounds = random.choices(selected_round, weights=args.num_turn_ratios)[0]  # number of       turns in the dialogue
    assistant_word_counts = (np.random.normal(loc=args.assistant_word_count, scale=50,       size=rounds).astype(int) // 50 * 50).tolist()
    human_word_counts = (np.random.normal(loc=args.human_word_count, scale=50, size=rounds).astype(int) // 50 * 50).tolist()
    word_counts = {
        "assistant": assistant_word_counts,
        "human": human_word_counts
    }
    '''
    用于批量生成对话。context_list: 一个包含多个上下文信息的字典列表，每个字典提供对话生成的背景信息。
    '''
    for context in context_list:
    system_input, user_input, prompt, rounds = self.encode_prompt(context,
                                                                  rounds=rounds,

 word_counts=word_counts,

 language=args.language
                                                                  )
    response = self.generate_dialogue(system_input, user_input)
    if response:
        chat = self.post_process_gpt_response(response)
        token_count = response.usage.total_tokens
        if chat:
            task_id = next(self.task_id_generator)
            dialogues.update({
                "id": task_id,
                "total_tokens": token_count,
                "prompt": prompt,
                "dialogue": chat
            })
            print(f"dialogue {task_id} succeed!")
        return dialogues
    '''
    遍历上下文列表：对于 context_list 中的每个上下文，使用 encode_prompt 方法编码提示。
    使用 generate_dialogue 方法生成对话。如果生成了对话，使用 post_process_gpt_response 方法后处    理响应。
    '''
def task_id_generator_function():
    """Generate integers 0, 1, 2, and so on."""
    task_id = 0
    while True:
        yield task_id
        task_id += 1
```

```
    '''
    定义了一个生成任务ID的生成器函数 task_id_generator_function。这个函数使用了一个无限循环来连续生成       整数，从0开始，每次调用 yield 语句时递增。这个生成器函数可以用于需要唯一序列号或任务ID的场景。例如，       在 DialogueGenerator 类中，每次生成新的对话时，可以使用这个生成器来分配一个唯一的任务ID。
    '''
def save_dialogues_to_json(dialogues, output_file):
    with open(output_file, "a", encoding="utf-8") as f:
        json.dump(dialogues, f, ensure_ascii=False, indent=2)
    '''
    对话数据保存到一个 JSON 文件中。
    打开指定的 output_file 文件进行追加（"a" 模式）。
    使用 json.dump 函数将 dialogues 数据序列化为 JSON 格式并写入到文件中。
    '''

if __name__ == "__main__":
    run()
```