Qwen-7B

项目地址：https://github.com/QwenLM/Qwen-7B/blob/main/README_CN.md

Qwen-7B-Chat

微调数据构成

微调数据由指令对话、安全防护以及服务响应三方面构成。指令对话数据涵盖写作、问答、头脑风暴、计划、内容理解、摘要等等基本任务；安全防护数据旨在防止模型输出不恰当的内容；而服务响应数据是Qwen较之其他Chat类模型更为特殊的一部分，目的在于让模型学习到如何在适当的时机调用合适的外部系统来推进相关工作。

此外，考虑到微调数据都是对话轮的形式，所以Qwen项目组用ChatML标准对数据进行格式化。

数据格式：ChatML

ChatML官方简述：https://github.com/openai/openai-python/blob/main/chatml.md

ChatML是OpenAI提出的一种结构化文本格式，其特点在于以相对结构化的形式标明了角色及相应内容，以官方文档的例子来说明：

```
# 经典版（Qwen使用的是这个）
<|im_start|>system
You are ChatGPT, a large language model trained by OpenAI. Answer as concisely as possible.
Knowledge cutoff: 2021-09-01
Current date: 2023-03-01<|im_end|>
<|im_start|>user
How are you<|im_end|>
<|im_start|>assistant
I am doing well!<|im_end|>
<|im_start|>user
How are you now?<|im_end|>

# 新版（list of dicts）
[
 {"token": "<|im_start|>"},
 "system\nYou are ChatGPT, a large language model trained by OpenAI. Answer as concisely as possible.\nKnowledge
cutoff: 2021-09-01\nCurrent date: 2023-03-01",
 {"token": "<|im_end|>"}, "\n", {"token": "<|im_start|>"},
 "user\nHow are you",
 {"token": "<|im_end|>"}, "\n", {"token": "<|im_start|>"},
 "assistant\nI am doing well!",
 {"token": "<|im_end|>"}, "\n", {"token": "<|im_start|>"},
 "user\nHow are you now?",
 {"token": "<|im_end|>"}, "\n"
]
```

特殊用法-ReAct Prompting

得益于Qwen-7B-Chat在微调阶段使用了服务响应数据进行训练，Qwen-7B-Chat具备了一定的工具调用能力，这使得直接使用Qwen-7B-Chat进行ReAct也能取得不错的效果。

ReAct

ReAct可以理解为“Reasoning and Acting”，即“推理与行动”，是一种在模型推理侧提升输出准确性的技术。关于Reasoning方面，COT对模型推理效果的提升有目共睹，但在相关研究中表明，其链路中涉及的知识也将对输出结果产生一定的影响，在模型本身性能不足够推理出正确结果、也无法调用外部世界知识的情况下，会产生事实幻觉和错误传播的问题。因此，ReAct在COT的基础之上，引入了“Action”操作，即通过Few-Shot Prompt来引导模型生成推理内容及与外界交互获取知识的动作，通过逐步推理与动作交互，将外部知识以动态的方式融入进COT链路中，以此来提高模型最终输出结果的准确性。

我们可以简单理解为，在对模型进行提问的时候，不要求模型直接就推断出结果，我们倾向于让模型使用COT的思路一点点推理，推理的过程中如果需要外部知识的介入，那么模型就会先调用外部工具来获取外部知识，然后将外部知识补充进上下文中继续推理，如此往复，直到推理出最终的结果。基于缜密且真实的上下文得到的结果，自然准确性会高不少。

整体的思想源自于这篇论文：Synergizing Reasoning and Acting in Language Models

https://arxiv.org/pdf/2210.03629.pdf

Qwen官方ReAct文档示例

https://github.com/QwenLM/Qwen-7B/blob/main/examples/react_prompt.md

ReAct实例演示

In [1]:
```
# 指定好Qwen-7B-Chat模型文件路径
model_file_path = "./Qwen-7B-Chat"
```

In [8]:
```
import json
import torch
import json5
```

In [14]:
```
# query = '准确计算3.5+1.5*3+3/2.55的结果'
# query = '职工张三的生日是在今年10月份吗？'
query = '职工张三的生日是在今年10月份吗？现在是2月份，距离张三的生日还有几个月？'
```

In [13]:
```
# （注册）定义工具
TOOLS = [
    {
        'name_for_human':
        '企业员工数据库',
        'name_for_model':
        'database',
        'description_for_model':
        '企业员工数据库储存有我司员工个人基本信息，通过输入员工名字可返回其个人信息。',
        'parameters': [{
            'name': 'database_query',
            'description': '员工名字，所需查阅信息的员工姓名',
            'required': True,
            'schema': {
                'type': 'string'
            },
        }],
    },
    {
        'name_for_human':
        '计算器',
        'name_for_model':
        'calculator',
        'description_for_model':
        '计算器是一个用于进行数值计算的工具，输入具体的数值计算公式文本，返回其运算结果',
        'parameters': [{
            'name': 'calculate',
            'description': '数值计算公式文本，包含了需要进行计算的数值和运算符',
            'required': True,
            'schema': {
                'type': 'string'
            },
        }],
    },
]
```

In [9]:
```
class database:
    def __init__(self):
        # 数据库存有员工基本信息
        self.database = {
            "李四": {
                "性别": "女",
                "生日": "10月3日",
                "职位": "画师"
            },
            "张三": {
                "性别": "男",
                "生日": "5月5日",
                "职位": "业务员"
            },
            "王五": {
                "性别": "男",
                "生日": "12月19日",
                "职位": "业务经理"
            }
        }
    def database_query(self, name):
        # 输入姓名返回员工信息
        return self.database.get(name, "查无此人")
```

In [10]:
```
db = database()
db.database_query("李四")
```

Out[10]:
```
{'性别': '女', '生日': '10月3日', '职位': '画师'}
```

In [11]:
```
class calculator:
    def __init__(self):
        pass
    def calculate(self, formula_str):
        return eval(formula_str)
```

In [12]:
```
cal = calculator()
cal.calculate("1+5")
```

Out[12]:
```
6
```

In [21]:
```
def parse_latest_plugin_call(text: str):
    """
    text: 模型的ReAct输出内容
    return: 工具方法名字, 工具方法入参
    """
    i = text.rfind('\nAction:')
    j = text.rfind('\nAction Input:')
    k = text.rfind('\nObservation:')
    if 0 <= i < j:  # If the text has `Action` and `Action input`,
        if k < j:  # but does not contain `Observation`,
            # then it is likely that `Observation` is ommited by the LLM,
            # because the output text may have discarded the stop word.
            text = text.rstrip() + '\nObservation:'  # Add it back.
            k = text.rfind('\nObservation:')
    if 0 <= i < j < k:
        plugin_name = text[i + len('\nAction:'):j].strip()
        plugin_args = text[j + len('\nAction Input:'):k].strip()
        return plugin_name, plugin_args
    return '', ''
```

In [15]:
```
# 定义react prompt模板

TOOL_DESC = """{name_for_model}: Call this tool to interact with the {name_for_human} API. What is the {name_for_human} API useful

REACT_PROMPT = """Answer the following questions as best you can. You have access to the following tools:

{tool_descs}

Use the following format:

Question: the input question you must answer
Thought: you should always think about what to do
Action: the action to take, should be one of [{tool_names}]
Action Input: the input to the action
Observation: the result of the action
```

```
... (this Thought/Action/Action Input/Observation can be repeated zero or more times)
Thought: I now know the final answer
Final Answer: the final answer to the original input question

Begin!

Question: {query}"""
```

In [16]:

```
# 定义真正传入模型的react prompt
tool_descs = []
tool_names = []
for info in TOOLS:
    tool_descs.append(
        TOOL_DESC.format(
            name_for_model=info['name_for_model'],
            name_for_human=info['name_for_human'],
            description_for_model=info['description_for_model'],
            parameters=json.dumps(
                info['parameters'], ensure_ascii=False),
        )
    )
    tool_names.append(info['name_for_model'])
tool_descs = '\n\n'.join(tool_descs)
tool_names = ','.join(tool_names)

prompt = REACT_PROMPT.format(tool_descs=tool_descs, tool_names=tool_names, query=query)
print(prompt)
```

```
Answer the following questions as best you can. You have access to the following tools:

database: Call this tool to interact with the 企业员工数据库 API. What is the 企业员工数据库 API useful for? 企业员工数据库储存有我司员工个人基本信息，通过输入员工名字可返回其个人信息。 Parameters: [{"name": "database_query", "description": "员工名字，所需查阅信息的员工姓名", "required": true, "schema": {"type": "string"}}] Format the arguments as a JSON object.

calculator: Call this tool to interact with the 计算器 API. What is the 计算器 API useful for? 计算器是一个用于进行数值计算的工具，输入具体的数值计算公式文本，返回其运算结果 Parameters: [{"name": "calculate", "description": "数值计算公式文本，包含了需要进行计算的数值和运算符", "required": true, "schema": {"type": "string"}}] Format the arguments as a JSON object.

Use the following format:

Question: the input question you must answer
Thought: you should always think about what to do
Action: the action to take, should be one of [database,calculator]
Action Input: the input to the action
Observation: the result of the action
... (this Thought/Action/Action Input/Observation can be repeated zero or more times)
Thought: I now know the final answer
Final Answer: the final answer to the original input question

Begin!

Question: 职工张三的生日是在今年10月份吗？现在是2月份，距离张三的生日还有几个月？
```

In [17]:

```python
from transformers import AutoModelForCausalLM, AutoTokenizer
from transformers.generation import GenerationConfig

# 请注意：分词器默认行为已更改为默认关闭特殊token攻击防护。
tokenizer = AutoTokenizer.from_pretrained(model_file_path, trust_remote_code=True)

# 打开bf16精度，A100、H100、RTX3060、RTX3070等显卡建议启用以节省显存
# model = AutoModelForCausalLM.from_pretrained("Qwen/Qwen-7B-Chat", device_map="auto", trust_remote_code=True, bf16=True).eval()
# 打开fp16精度，V100、P100、T4等显卡建议启用以节省显存
# model = AutoModelForCausalLM.from_pretrained("Qwen/Qwen-7B-Chat", device_map="auto", trust_remote_code=True, fp16=True).eval()
# 使用CPU进行推理，需要约32GB内存
# model = AutoModelForCausalLM.from_pretrained("Qwen/Qwen-7B-Chat", device_map="cpu", trust_remote_code=True).eval()
# 默认使用自动模式，根据设备自动选择精度
model = AutoModelForCausalLM.from_pretrained(model_file_path, device_map="auto", trust_remote_code=True).eval()

# 可指定不同的生成长度、top_p等相关超参
model.generation_config = GenerationConfig.from_pretrained(model_file_path, trust_remote_code=True)
# print(model.generation_config.top_p)
# print(model.generation_config.temperature)
# 设置stop_word
react_stop_words = [
    # tokenizer.encode('Observation'),  # [37763, 367]
    tokenizer.encode('Observation:'),  # [37763, 367, 25]
    tokenizer.encode('Observation:\n'),  # [37763, 367, 510]
]
```

```
The model is automatically converting to bf16 for faster inference. If you want to disable the automatic precision, please manually
add bf16/fp16/fp32=True to "AutoModelForCausalLM.from_pretrained".
Try importing flash-attention for faster inference...
Loading checkpoint shards:   0%|          | 0/8 [00:00<?, ?it/s]
```

In [19]:
```
model
```

Out[19]:
```
QWenLMHeadModel(
  (transformer): QWenModel(
    (wte): Embedding(151936, 4096)
    (drop): Dropout(p=0.0, inplace=False)
    (h): ModuleList(
      (0-31): 32 x QWenBlock(
        (ln_1): RMSNorm()
        (attn): QWenAttention(
          (c_attn): Linear(in_features=4096, out_features=12288, bias=True)
          (c_proj): Linear(in_features=4096, out_features=4096, bias=False)
          (core_attention_flash): FlashSelfAttention()
          (rotary_emb): RotaryEmbedding()
          (attn_dropout): Dropout(p=0.0, inplace=False)
        )
        (ln_2): RMSNorm()
        (mlp): QWenMLP(
          (w1): Linear(in_features=4096, out_features=11008, bias=False)
          (w2): Linear(in_features=4096, out_features=11008, bias=False)
          (c_proj): Linear(in_features=11008, out_features=4096, bias=False)
        )
      )
    )
    (ln_f): RMSNorm()
  )
  (lm_head): Linear(in_features=4096, out_features=151936, bias=False)
)
```

In [20]:
```
response, history = model.chat(
    tokenizer, prompt, history=None,
    stop_words_ids=react_stop_words  # 此接口用于增加 stop words
)
print(response)
```

```
Thought: 需要查询数据库中的员工信息，才能回答问题，应该使用database工具。
Action: database
Action Input: {"database_query": "张三"}
Observation:
```

In [22]:
```
# v2

# 从模型的react输出中提取“工具方法”及“工具方法入参”
tool_cfg = parse_latest_plugin_call(response)

# 根据“工具方法”及“工具方法入参”调用工具并取得返回结果
result_dict = {"status_code":200}
tools_msg = [TOOL for TOOL in TOOLS if tool_cfg[0] in TOOL['name_for_model']]
for tool_msg in tools_msg:
    for tool_cfg_each in tool_cfg[1:]:
        tool_dict_each = json5.loads(tool_cfg_each)
        for k, v in tool_dict_each.items():
            if k in [tool_p['name'] for tool_p in tool_msg['parameters']]:
                tool_inst = eval(tool_cfg[0])()
                tool_func = getattr(tool_inst, k)
                result = tool_func(v)
                result_dict["result"] = result
# 工具返回结果
print("工具返回: {}".format(result_dict))
print("---"*20)

# 将工具返回结果与前文进行拼接，将在后续传入模型继续进行生成
response += " " + str(result_dict)
print(response)
```

```
工具返回: {'status_code': 200, 'result': {'性别': '男', '生日': '5月5日', '职位': '业务员'}}
------------------------------------------------------------
Thought: 需要查询数据库中的员工信息，才能回答问题，应该使用database工具。
Action: database
Action Input: {"database_query": "张三"}
Observation: {'status_code': 200, 'result': {'性别': '男', '生日': '5月5日', '职位': '业务员'}}
```

In [23]:
```
# 将带有工具返回结果（事实知识）的内容传给模型进一步得到结果
response, history = model.chat(
    tokenizer, response, history=history,
    stop_words_ids=react_stop_words  # 此接口用于增加 stop words
)
print(response)
```

```
Thought: 张三的生日是在5月份，现在是2月份，距离张三的生日还有3个月。
Final Answer: 张三的生日是在5月份，现在是2月份，距离张三的生日还有3个月。
```