PTLog 25/7

0. Reliance & Previous Settings

```
import pyterrier as pt
import pyterrier_rag.readers
from pyterrier_rag.backend import HuggingFaceBackend
from pyterrier_rag.backend import Seq2SeqLMBackend
from pyterrier_rag.backend import OpenAIBackend
from pyterrier_rag.prompt import Concatenator
from pyterrier_rag.readers import Reader
from pyterrier_rag.prompt import PromptTransformer, prompt
from fastchat.model import get_conversation_template
from transformers import AutoTokenizer
```

```
#dataset
sparse_index = pt.Artifact.from_hf('pyterrier/ragwiki-terrier')
#retriever
bm25_ret = pt.rewrite.tokenise() >> sparse_index.bm25(include_fields=
['docno', 'text', 'title'], threads=5) >> pt.rewrite.reset()
```

1. Llama-OpenAIBackend

According to the instance of using llama model in GitHub repo.

```
system_message = r"""You are an expert Q&A system that is trusted around
the world.
        Always answer the query using the provided context information,
        and not prior knowledge.
        Some rules to follow:
        1. Never directly reference the given context in your answer
        2. Avoid statements like 'Based on the context, ...' or
        'The context information ...' or anything along those lines."""

prompt_text = """Context information is below.
            ---------------------
            {{ qcontext }}
            ---------------------
            Given the context information and not prior knowledge, answer
the query.
            Query: {{ query }}
```

```
            "Answer: """

template = get_conversation_template("meta-llama-3.1-sp")
prompt = PromptTransformer(
    conversation_template=template,
    system_message=system_message,
    instruction=prompt_text,
    api_type="openai"
)
```

```
generation_args={
    "temperature": 0.1,
    "max_tokens": 128,
}

llama = OpenAIBackend("llama-3-8b-instruct",

 api_key="ida_k0ont6O9V7pSTqi1MJUgeAfvVKb8yPHQzmL5l1b8",
                      # api="completions",
                      generation_args=generation_args,
                      base_url="http://api.llm.apps.os.dcs.gla.ac.uk/v1")
llama.device = "cuda"
llama._model = model_name
llama.text_max_length = 1024
```

input:

```
llama.generate(["What are chemical reations?"])[0].text
```

output:

```
'Chemical reactions are processes in which one or more substances, called
reactants, are transformed into new substances, called products. These
reactions involve the breaking and forming of chemical bonds between
atoms or molecules, resulting in a change in the composition and
properties of the substances involved.\n\nChemical reactions can be
classified into several types, including:\n\n1. **Synthesis reactions**:
Two or more substances combine to form a new compound. Example: 2H2 + O2
→ 2H2O (water formation)\n2. **Decomposition reactions**: A single
substance breaks down into two or more simpler substances. Example: 2H'
```

Summary

At first I tried to use 'llama-3-8b-instruct' as SearchO1 class's generator, and it turns out SearchO1 only supports HuggingFaceBackend, so this attempt failed due to not having the sense of reading source code. But to have a try of llama model and the `generate` function of backend is good for me though. :)

---

2. HuggingFaceBackend(Deepseek-R1) + SearchO1

> *After failing to combine HuggingFaceBackend with Search R1 and R1 Searcher, I turned back to SearchO1, tried to gain some idea. It has a search move at one point. So I guess the point is not with the searcher class. Here I record the related code and out put.*

Basic Settings

```
import search_o1
from search_o1 import SearchO1

generation_args = {
    "do_sample" : True,
    "max_tokens" : 128,
    "max_length": 100,
}

hf_backend = HuggingFaceBackend(
    model_id = "deepseek-ai/DeepSeek-R1-Distill-Qwen-1.5B",
    model_args = {
        "do_sample": True,
        "temperature": 0.7,
        "top_k": 500,
    },
    generation_args = generation_args,
    max_input_length = 512,
    max_new_tokens=64,
    logprobs_topk=10,
    verbose=True,
    device="cuda"
)
hf_backend.text_max_length = hf_backend.max_input_length

search_o1 = SearchO1(
    retriever = bm25_ret,
    generator = hf_backend,
    max_turn = 5,
    max_retrieval_step = 5
)
```

form a single query:

```
res = search_o1.search("Who is the singer of Lemon?")
print(res)
```

output:

qid	question	prompt	output	finished	history	search_count	search_queries	retrieval_results	qanswer	
0	1	Who is the singer of Lemon?	You are a reasoning assistant with the ability...	<|begin_search_query|>Lemon song<|end_search_q...	False	[<|begin_search_query|>Lemon song<|end_search_...	3	{Lemon song, Lemon song by whom, Lemon singer}	{'Lemon song': [{'qid': '1', 'docid': 14364855...	<|begin_search_query|>Lemon song<|end_search_q...

```
print(res.iloc[0].history[0])
```

output:

```
The maximum number of turns 5 is exceeded, stopping...
<|begin_search_query|>Lemon song<|end_search_query|>
```

Next part will be about the functions and analysis of 3 different searcher files. The aim is to find the difference that matters and to consider how to extract a super-class.

3. About SearchO1

This chapter will discuss the functions in Searchers in detail. Try to find the source of each parameter and ablate the differences. This file is just about SearchO1, the other two will be analysis in latter md files.

3.1 SearchO1.py

3.1.1 Reliance

```
# Standard library for regular expressions, used for string pattern
matching and extraction
import re

import torch

# Type annotations for better code readability and static checking
from typing import List, Dict, Iterable

import pyterrier as pt

# Experimental extension of PyTerrier, provides flexible pipeline and
decorators
import pyterrier_alpha as pta

# Backend for integrating HuggingFace generative models and stop word
control in PyTerrier RAG
from pyterrier_rag.backend import HuggingFaceBackend, StopWordCriteria
```

3.1.2 Key Functions

`def get_*qa_search_o1_instruction(MAX_SEARCH_LIMIT)`

Differences and Similarities

- Differences:
  - `singleqa` is for straightforward, single-step questions; `multiqa` is for questions that require breaking down into multiple steps and searches.
  - The instructions and examples reflect the complexity and stepwise nature of the task.
- Similarities:
  - Both functions generate structured prompts that standardize how the model interacts with the retrieval system.
  - Both use special tokens to mark search queries and results, making the process machine-readable and automatable.

`def get_webpage_to_reasonchain_instruction(prev_reasoning, search_query, document)`

generates a detailed prompt (instruction) for a language model. Its purpose is to guide the model to:

- Analyze the retrieved web pages (`document`) in the context of the previous reasoning steps (`prev_reasoning`) and the current search query (`search_query`).
- Extract only the information from the web pages that is relevant and helpful for the current query.
- Seamlessly integrate this new information into the ongoing reasoning process to continue solving the original question.
- Output the helpful information in a specific format, starting with **`Final Information`**. If no useful information is found, it should explicitly state so.

This function standardizes how the model should process, filter, and incorporate external knowledge into its reasoning chain during multi-step, retrieval-augmented tasks.

`def replace_recent_steps(origin_str, replace_str)`

How it works:

- It parses both the original reasoning steps (`origin_str`) and the replacement instructions (`replace_str`), which are formatted as numbered steps (e.g., `Step 1: ...`).
- For each step in the replacement instructions:
  - If the content contains `"DELETE THIS STEP"`, the corresponding step is removed from the original reasoning chain.
  - Otherwise, the step is replaced or added with the new content.
- Finally, it reconstructs the reasoning steps in order and returns the updated chain as a single string.

Purpose:
This function allows dynamic editing of multi-step reasoning chains, enabling the model to correct, update, or remove specific steps as new information becomes available.

```
class SearchO1(pt.Transformer)
```

init

`__init__` 方法参数解析（SearchO1类）

- **retriever : pt.Transformer**
  检索器对象，用于根据模型生成的检索query，从外部知识库中检索相关文档或段落。
  The retriever object, used to fetch relevant documents or passages from an external knowledge base based on the model's generated queries.
- **generator : HuggingFaceBackend**
  生成器对象，通常是一个大语言模型（如T5、Llama等），负责根据prompt生成推理内容和答案。
  The generator object, usually a large language model, responsible for generating reasoning content and answers based on the prompt.
- **max_turn : int = 10**
  最大推理轮数。每个问题最多允许模型推理和检索多少轮，防止死循环。
  Maximum number of reasoning turns allowed per question, preventing infinite loops.
- **max_retrieval_step : int = 5**
  每个问题最多允许检索的次数。限制模型在一次推理过程中最多能发起多少次检索请求。
  Maximum number of retrieval steps allowed per question.
- **topk : int = 10**
  每次检索时返回的文档数量上限（top-k检索）。
  The maximum number of documents returned per retrieval (top-k).
- **temperature : float = 0.7**
  生成时的“温度”参数，控制输出的多样性。值越高，生成越随机；值低则更确定。
  Temperature parameter for generation, controlling output diversity (higher is more random, lower is more deterministic).
- **top_p : float = 0.8**
  nucleus sampling参数，控制生成时的概率截断，影响生成内容的多样性。
  Nucleus sampling parameter, controlling probability cutoff for generation diversity.

- **top_k : int = 20**
  生成时采样的top-k参数，控制每步生成时可选的最大词汇数。
  Top-k sampling parameter for generation, limiting the number of candidate tokens per step.
- **multihop_qa : bool = True**
  是否启用多跳问答模式。为True时，prompt和推理流程会适配多步推理场景。
  Whether to enable multi-hop QA mode; if True, the prompt and reasoning process are adapted for multi-step reasoning.
- **kwargs**
  其他可选参数，便于扩展和兼容更多自定义配置。
  Other optional parameters for extensibility and custom configuration.

这些参数共同决定了检索-生成流程的检索策略、生成策略、推理轮数、检索次数、生成多样性等关键行为，确保 `SearchO1` 能灵活适配不同的实验需求和实际应用场景。

transform_iter(*self*, *inp*: Iterable[dict]) -> Iterable[dict]

1. 初始化序列列表
2. 主循环
3. 生成模型输出
4. 提取检索请求
5. 判断是否需要检索
6. 执行检索
7. 答案抽取
8. 返回结果

初始化序列列表，每个输入样本包装成推理序列字典
Initialize the sequence list, wrapping each input sample as a reasoning sequence dictionary

```
sequences = [
    {
        "qid" : row['qid'],
        "question": row['query'],
        "prompt": self.get_init_prompt(row['query'], self.multihop_qa),
        "output": "",
        "finished": False,
        "history": [],
        "search_count": 0,
        "search_queries": set(),
        "retrieval_results": {}
    }
    for row in inp
]
```

主循环，处理所有未完成的推理序列
Main loop, processing all unfinished reasoning sequences

```
turn = 0
while True:
    sequences_not_finished = [seq for seq in sequences if not
seq["finished"]]
    if sequences_not_finished:
        turn += 1
        # ...
```

选出未完成的序列，准备生成新一轮推理
Select unfinished sequences and prepare to generate a new round of reasoning

```
sequences_not_finished = [seq for seq in sequences if not
seq["finished"]]
```

生成模型输出，追加到prompt和output
Generate model outputs and append them to prompt and output

```
outputs: List[str] = self.generate(sequences_not_finished)
```

提取检索请求，判断是否需要检索
Extract search queries and determine whether retrieval is needed

```
queries = []
prev_reasonings = []
sequences_require_retrieval = []
for seq, output in zip(sequences_not_finished, outputs):
    seq["history"].append(output)
    seq["prompt"] += output
    seq["output"] += output
    query = self.extract_search_query(output)
    # ...
```

整理推理步骤，收集需要检索的query和上下文
Organize reasoning steps and collect queries and context that require retrieval

```
if seq['search_count'] < self.max_retrieval_step and query not in
seq['search_queries']:
    all_reasoning_steps = seq["output"]
    all_reasoning_steps = all_reasoning_steps.replace('\n\n',
'\n').split("\n")
    truncated_prev_reasoning = ""
    for i, step in enumerate(all_reasoning_steps):
        truncated_prev_reasoning += f"Step {i + 1}: {step}\n\n"
    # ...（后续整理和收集）
    queries.append(query)
    prev_reasonings.append(truncated_prev_reasoning)
    sequences_require_retrieval.append(seq)
    seq["search_count"] += 1
    seq["search_queries"].add(query)
```

执行检索与分析，整合检索结果到推理链
Perform retrieval and analysis, integrate retrieval results into the reasoning chain

```
if sequences_require_retrieval:
    retrieval_results: List[List[dict]] = self.retrieve_docs(queries)
    doc_analyses_outputs = self.analyze_docs(
        sequences=sequences_require_retrieval,
        prev_reasonings=prev_reasonings,
        queries=queries,
        retrieval_results=retrieval_results
    )
    doc_analyses = [item["extracted_info"] for item in
doc_analyses_outputs]
    for seq, query, retrieval_result, analysis in
zip(sequences_require_retrieval, queries, retrieval_results,
doc_analyses):
        if isinstance(analysis, str):
            text = f"\n\n{BEGIN_SEARCH_RESULT}{analysis}
{END_SEARCH_RESULT}\n\n"
        else:
            text = replace_recent_steps(seq['output'], analysis)
        seq["prompt"] += text
        seq["output"] += text
        seq["history"].append(text)
        seq["retrieval_results"][query] = retrieval_result
```

判断终止条件，所有序列完成或超出最大轮数
Check termination conditions: all sequences are finished or the maximum number of turns is exceeded

```
unfinished = [seq for seq in sequences if not seq["finished"]]
if not unfinished:
    break
else:
    if turn >= self.max_turn:
        print(f"The maximum number of turns {self.max_turn} is exceeded,
stopping...")
        break
```

抽取最终答案，存入qanswer字段
Extract the final answer and store it in the qanswer field

```
for seq in sequences:
    seq["qanswer"] = self.get_answer(seq["output"])
return sequences
```

4. Summary&Current Work

This note is about the succussed part of the code work I'd been working on these days, leading to a understand of searchers class structure and give inspiration to super class construction to me. Now I did some attempt to figure out what makes the combination of searchers classes and LLM not that useful in this case(Indeed, sometimes it has a search, sometimes doesn't). My current conclusion is, unideal result might be caused by not proper model that 'd been select to process experiment.

- I tried to force the model to perform a search by doing some change on prompt. Turns out useless.
- Then I thought it might be the llms are not really proper to the O1 class. So I tried different models and no one was good.
- I got a basic plan to make a super class.