import os

import sys

import logging

import argparse

import hashlib

import numpy as np

import requests

from typing import Any, Dict, List, Tuple, Optional, Union

from dataclasses import dataclass, asdict

from datetime import datetime

# 配置日志

logging.basicConfig(

level=logging.INFO,

format='%(asctime)s - %(name)s - %(levelname)s - %(message)s',

handlers=[

logging.FileHandler("translation\_qa.log"),

logging.StreamHandler(sys.stdout)

]

)

logger = logging.getLogger("TranslationQA")

# 语言代码映射表（适配百度API）

LANGUAGE\_MAP = {

"中文": "zh",

"英文": "en",

"日文": "jp",

"韩文": "kor",

"法文": "fra",

"德文": "de",

"西班牙文": "spa",

"俄文": "ru",

"阿拉伯文": "ara",

"葡萄牙文": "pt",

"auto": "auto" # 自动检测语言

}

@dataclass

class TranslationResult:

"""翻译结果数据类"""

source\_text: str

source\_language: str

target\_language: str

translated\_text: str

confidence: float

translation\_time: float

model\_used: str

additional\_info: Dict = None

@dataclass

class QAResult:

"""问答结果数据类"""

question: str

answer: str

confidence: float

source\_context: str

model\_used: str = "rule" # rule或qianfan

class TranslationModel:

"""翻译模型实现（使用百度翻译API）"""

def \_\_init\_\_(self, appid: str = None, secret\_key: str = None):

self.appid = appid or os.environ.get("BAIDU\_FANYI\_APPID")

self.secret\_key = secret\_key or os.environ.get("BAIDU\_FANYI\_SECRET\_KEY")

self.api\_url = "http://api.fanyi.baidu.com/api/trans/vip/translate"

self.language\_codes = LANGUAGE\_MAP

self.proxies = self.\_get\_proxies() # 获取代理配置

if not self.appid or not self.secret\_key:

logger.warning("未提供百度翻译API凭证，云翻译功能可能受限")

logger.info("请通过环境变量或参数传入APP ID和Secret Key")

def \_get\_proxies(self) -> Optional[Dict]:

"""获取代理配置（如需通过代理访问网络）"""

http\_proxy = os.environ.get("HTTP\_PROXY", os.environ.get("http\_proxy"))

https\_proxy = os.environ.get("HTTPS\_PROXY", os.environ.get("https\_proxy"))

if http\_proxy or https\_proxy:

return {

"http": http\_proxy,

"https": https\_proxy

}

return None

def translate(self, text: str, source\_lang: str, target\_lang: str) -> TranslationResult:

"""使用百度API翻译文本"""

if not self.appid or not self.secret\_key:

return TranslationResult(

source\_text=text,

source\_language=source\_lang,

target\_language=target\_lang,

translated\_text="未配置百度翻译API凭证，无法进行云翻译",

confidence=0.0,

translation\_time=0.0,

model\_used="baidu-translate"

)

start\_time = datetime.now()

# 处理语言代码映射

name\_to\_code = {name: code for name, code in self.language\_codes.items()}

code\_to\_name = {code: name for name, code in self.language\_codes.items()}

from\_lang = source\_lang.lower()

to\_lang = target\_lang.lower()

if from\_lang in name\_to\_code:

from\_lang = name\_to\_code[from\_lang]

if to\_lang in name\_to\_code:

to\_lang = name\_to\_code[to\_lang]

if from\_lang not in self.language\_codes.values() or to\_lang not in self.language\_codes.values():

error\_msg = f"不支持的语言: 源语言={from\_lang}, 目标语言={to\_lang}"

logger.error(error\_msg)

return TranslationResult(

source\_text=text,

source\_language=source\_lang,

target\_language=target\_lang,

translated\_text=error\_msg,

confidence=0.0,

translation\_time=0.0,

model\_used="baidu-translate"

)

# 生成签名

salt = str(np.random.randint(32768, 65536))

sign = self.appid + text + salt + self.secret\_key

sign = sign.encode("utf-8")

sign = hashlib.md5(sign).hexdigest()

# 发送请求（支持代理）

try:

response = requests.post(

self.api\_url,

data={

"q": text,

"from": from\_lang,

"to": to\_lang,

"appid": self.appid,

"salt": salt,

"sign": sign

},

proxies=self.proxies, # 应用代理

timeout=10

)

response.raise\_for\_status()

result = response.json()

if "error\_code" in result:

error\_msg = f"百度翻译API错误: {result.get('error\_msg', '未知错误')}"

logger.error(error\_msg)

translated\_text = error\_msg

confidence = 0.0

else:

translated\_text = result["trans\_result"][0]["dst"]

confidence = min(95.0, max(60.0, 100 - len(text) \* 0.1))

end\_time = datetime.now()

translation\_time = (end\_time - start\_time).total\_seconds()

return TranslationResult(

source\_text=text,

source\_language=code\_to\_name.get(from\_lang, from\_lang),

target\_language=code\_to\_name.get(to\_lang, to\_lang),

translated\_text=translated\_text,

confidence=confidence,

translation\_time=translation\_time,

model\_used="baidu-translate",

additional\_info=result

)

except Exception as e:

logger.error(f"云翻译请求失败: {str(e)}")

return TranslationResult(

source\_text=text,

source\_language=source\_lang,

target\_language=target\_lang,

translated\_text=f"翻译请求失败: {str(e)}",

confidence=0.0,

translation\_time=0.0,

model\_used="baidu-translate"

)

def supported\_languages(self) -> List[str]:

"""获取百度API支持的语言列表"""

return list(self.language\_codes.keys())

# === 百度千帆对话模型API ===

class QianfanChatModel:

"""百度千帆对话模型API接口"""

def \_\_init\_\_(self, api\_key: str = None, model\_name: str = "ernie-3.5-8k"):

self.api\_key = api\_key or os.environ.get("QIANFAN\_API\_KEY")

self.model\_name = model\_name

self.api\_endpoint = "https://qianfan.baidubce.com/v2/chat/completions" # 修正为平台正确地址

self.headers = {

"Content-Type": "application/json",

"Authorization": f"Bearer {self.api\_key}"

}

self.proxies = self.\_get\_proxies() # 获取代理配置

if not self.api\_key:

logger.warning("未提供百度千帆API凭证，大模型功能可能受限")

def \_get\_proxies(self) -> Optional[Dict]:

"""获取代理配置（如需通过代理访问网络）"""

http\_proxy = os.environ.get("HTTP\_PROXY", os.environ.get("http\_proxy"))

https\_proxy = os.environ.get("HTTPS\_PROXY", os.environ.get("https\_proxy"))

if http\_proxy or https\_proxy:

return {

"http": http\_proxy,

"https": https\_proxy

}

return None

def generate(self, prompt: str, history: List[Dict] = None) -> str:

"""调用百度千帆对话模型生成回答"""

if not self.api\_key:

return "未配置百度千帆API凭证，无法使用大模型功能"

try:

messages = []

if history:

messages.extend(history)

messages.append({"role": "user", "content": prompt})

response = requests.post(

self.api\_endpoint,

json={

"model": self.model\_name,

"messages": messages,

"temperature": 0.7,

"top\_p": 0.8,

"max\_gen\_len": 2048

},

headers=self.headers,

proxies=self.proxies, # 应用代理

timeout=30

)

response.raise\_for\_status()

result = response.json()

# 解析模型返回结果

if "choices" in result and len(result["choices"]) > 0:

return result["choices"][0]["message"]["content"]

else:

logger.error(f"模型返回异常: {result}")

return "模型返回结果解析失败"

except Exception as e:

logger.error(f"调用百度千帆模型失败: {e}")

return f"大模型调用失败: {e}"

# === 混合调用的问答系统 ===

class HybridQASystem:

"""混合调用百度翻译API和智能千帆大模型的问答系统"""

def \_\_init\_\_(self, fanyi\_appid: str = None, fanyi\_secret: str = None,

qianfan\_api\_key: str = None):

self.translation\_model = TranslationModel(fanyi\_appid, fanyi\_secret)

self.qianfan\_model = QianfanChatModel(qianfan\_api\_key)

self.history = []

self.max\_context\_length = 1000

logger.info("混合问答系统初始化完成（翻译+理解一体化）")

def extract\_translation\_history(self) -> str:

"""提取翻译历史作为上下文"""

history\_text = ""

for item in self.history[-5:]: # 取最近5条记录

if item["action"] == "translate":

src = item["source\_text"]

tgt = item["result"]["translated\_text"]

src\_lang = item["result"]["source\_language"]

tgt\_lang = item["result"]["target\_language"]

history\_text += f"[{src\_lang}→{tgt\_lang}] {src} → {tgt}\n"

return history\_text

def translate\_and\_understand(self, text: str, source\_lang: str = "auto", target\_lang: str = "en") -> Dict:

"""翻译并理解文本，返回翻译结果和理解摘要"""

# 1. 先进行翻译

translation\_result = self.translation\_model.translate(text, source\_lang, target\_lang)

self.history.append({

"timestamp": datetime.now().isoformat(),

"action": "translate",

"source\_text": text,

"source\_lang": source\_lang,

"target\_lang": target\_lang,

"result": asdict(translation\_result)

})

# 2. 构建理解提示词

context = self.extract\_translation\_history()

prompt = f"""

你是一个专业的翻译助手，擅长语言翻译和语义理解。

以下是最近的翻译历史：

{context}

请对以下翻译结果进行理解和分析：

原文 ({translation\_result.source\_language}): {translation\_result.source\_text}

译文 ({translation\_result.target\_language}): {translation\_result.translated\_text}

请提供：

1. 译文的语义解释

2. 相关领域的背景知识（如果有）

3. 可能的应用场景

4. 翻译技巧说明（可选）

请简明扼要地回答。

"""

# 3. 调用大模型理解

understanding = self.qianfan\_model.generate(prompt)

return {

"translation": translation\_result,

"understanding": understanding

}

def answer\_question(self, question: str) -> QAResult:

"""回答与翻译或理解相关的问题"""

context = self.extract\_translation\_history()

# 1. 判断是否为简单翻译问题（规则匹配）

question\_lower = question.lower()

if "为什么" in question\_lower or "原因" in question\_lower:

for item in self.history:

if item["action"] == "translate":

src = item["source\_text"]

tgt = item["result"]["translated\_text"]

if src.lower() in question\_lower or tgt.lower() in question\_lower:

return QAResult(

question=question,

answer=f"'{src}'翻译为'{tgt}'是百度翻译API的标准译法",

confidence=70.0,

source\_context=context,

model\_used="rule"

)

return QAResult(

question=question,

answer="翻译结果基于百度翻译API的词典和算法",

confidence=60.0,

source\_context=context,

model\_used="rule"

)

# 2. 其他类型问题调用大模型

else:

prompt = f"""

你是一个专业的翻译助手，擅长语言翻译和语义理解。

以下是最近的翻译历史：

{context}

用户问题：{question}

请基于翻译历史用\*\*中文\*\*回答问题，如果问题与翻译无关，请提供相关领域的专业中文回答。

"""

answer = self.qianfan\_model.generate(prompt)

return QAResult(

question=question,

answer=answer,

confidence=80.0,

source\_context=context,

model\_used="qianfan"

)

# === 一体化翻译助手 ===

class IntegratedTranslationAssistant:

"""整合翻译和理解功能的一体化助手"""

def \_\_init\_\_(self, fanyi\_appid: str = None, fanyi\_secret: str = None,

qianfan\_api\_key: str = None):

self.qa\_system = HybridQASystem(fanyi\_appid, fanyi\_secret, qianfan\_api\_key)

logger.info("一体化翻译助手初始化完成（支持翻译+理解+问答）")

def translate(self, text: str, source\_lang: str = "auto", target\_lang: str = "en") -> Dict:

"""翻译并理解文本"""

return self.qa\_system.translate\_and\_understand(text, source\_lang, target\_lang)

def ask\_question(self, question: str) -> QAResult:

"""回答问题"""

return self.qa\_system.answer\_question(question)

def get\_history(self, limit: int = 10) -> List[Dict]:

"""获取历史记录"""

return self.qa\_system.history[-limit:]

# === 交互式命令行界面 ===

def run\_interactive\_mode(fanyi\_appid: str = None, fanyi\_secret: str = None,

qianfan\_api\_key: str = None):

"""交互式命令行界面（支持翻译+理解+问答）"""

assistant = IntegratedTranslationAssistant(fanyi\_appid, fanyi\_secret, qianfan\_api\_key)

print("=== 智能翻译助手（翻译+理解一体化）===")

print("功能:")

print(" 1. 输入文本直接翻译并获取语义理解")

print(" 2. 输入'?'开头的问题进行问答（可关于翻译或其他领域）")

print(" 3. 输入'h'查看历史记录")

print(" 4. 输入's'修改源语言")

print(" 5. 输入't'修改目标语言")

print(" 6. 输入'q'退出")

source\_lang = "auto"

target\_lang = "en"

while True:

try:

print(f"\n当前: 源语言={source\_lang} → 目标语言={target\_lang}")

user\_input = input("请输入文本或问题: ").strip()

if user\_input.lower() == 'q':

break

elif user\_input.lower() == 'h':

history = assistant.get\_history(5)

if not history:

print("暂无历史记录")

continue

print("\n最近操作:")

for i, item in enumerate(history, 1):

if item["action"] == "translate":

src = item["source\_text"]

tgt = item["result"]["translated\_text"]

print(

f"{i}. [{item['source\_lang']}→{item['target\_lang']}] {src[:30]}{'...' if len(src) > 30 else ''} → {tgt[:30]}{'...' if len(tgt) > 30 else ''}")

elif user\_input.lower() == 's':

source\_lang = input("请输入新的源语言 (默认自动): ").strip() or "auto"

# 验证语言是否支持

supported\_langs = assistant.qa\_system.translation\_model.supported\_languages()

if source\_lang not in supported\_langs and source\_lang != "auto":

print(f"警告: 源语言 '{source\_lang}' 暂不支持，已恢复为自动检测")

source\_lang = "auto"

continue

elif user\_input.lower() == 't':

target\_lang = input("请输入新的目标语言 (默认英文): ").strip() or "en"

# 验证语言是否支持

supported\_langs = assistant.qa\_system.translation\_model.supported\_languages()

if target\_lang not in supported\_langs:

print(f"警告: 目标语言 '{target\_lang}' 暂不支持，已恢复为英文")

target\_lang = "en"

continue

elif not user\_input:

continue

elif user\_input.startswith('?'):

# 处理问答

question = user\_input[1:].strip()

if not question:

print("请输入具体问题内容")

continue

print("\n思考中...")

qa\_result = assistant.ask\_question(question)

model\_tag = "(规则)" if qa\_result.model\_used == "rule" else "(文心一言)"

print(f"\n问题: {question}")

print(f"回答 {model\_tag}: {qa\_result.answer}")

print(f"置信度: {qa\_result.confidence:.1f}%")

else:

# 处理翻译+理解

if not user\_input:

print("请输入要翻译的文本")

continue

print("\n处理中...")

result = assistant.translate(user\_input, source\_lang, target\_lang)

print(f"\n翻译结果 ({result['translation'].target\_language}): {result['translation'].translated\_text}")

print(f"置信度: {result['translation'].confidence:.1f}%")

print(f"耗时: {result['translation'].translation\_time:.2f}秒")

print("\n语义理解:")

# 处理大模型返回的过长内容（截断显示前500字）

understanding = result['understanding']

if len(understanding) > 500:

print(understanding[:500] + "...")

else:

print(understanding)

except KeyboardInterrupt:

print("\n操作已取消，按'q'可退出程序")

except requests.exceptions.ProxyError:

print("\n错误: 代理连接失败，请检查代理配置是否正确")

except requests.exceptions.ConnectionError:

print("\n错误: 网络连接失败，请检查网络是否正常")

except requests.exceptions.Timeout:

print("\n错误: 请求超时，请重试或检查网络稳定性")

except Exception as e:

print(f"\n发生错误: {str(e)}")

logger.error(f"交互模式错误: {str(e)}")

print("\n=== 翻译助手已退出 ===")

# === 主函数 ===

def main():

"""命令行入口"""

parser = argparse.ArgumentParser(description="智能翻译助手（翻译+理解一体化）")

parser.add\_argument("--text", type=str, help="要翻译的文本")

parser.add\_argument("--question", type=str, help="要提问的问题")

parser.add\_argument("--source", type=str, default="auto", help="源语言")

parser.add\_argument("--target", type=str, default="en", help="目标语言")

parser.add\_argument("--fanyi-appid", type=str, help="百度翻译APP ID")

parser.add\_argument("--fanyi-secret", type=str, help="百度翻译Secret Key")

parser.add\_argument("--qianfan-api-key", type=str, help="百度智能千帆API Key")

args = parser.parse\_args()

assistant = IntegratedTranslationAssistant(

fanyi\_appid=args.fanyi\_appid,

fanyi\_secret=args.fanyi\_secret,

qianfan\_api\_key=args.qianfan\_api\_key

)

if args.question:

# 直接提问模式

print(f"\n问题: {args.question}")

qa\_result = assistant.ask\_question(args.question)

print(f"回答: {qa\_result.answer}")

print(f"置信度: {qa\_result.confidence:.1f}%")

elif args.text:

# 翻译并理解模式

result = assistant.translate(args.text, args.source, args.target)

print(f"\n翻译结果 ({result['translation'].target\_language}): {result['translation'].translated\_text}")

print(f"置信度: {result['translation'].confidence:.1f}%")

print("\n语义理解:")

print(result['understanding'])

else:

# 显示帮助并进入交互模式

parser.print\_help()

run\_interactive\_mode(

args.fanyi\_appid, args.fanyi\_secret,

args.qianfan\_api\_key

)

if \_\_name\_\_ == "\_\_main\_\_":

main()