RAG知识库大模型应用开发

监控显存命令

使用jtop查看显存占用

```
jtop
```

若无jtop命令，则需先安装

```
sudo pip3 install -U pip
sudo pip3 install jetson-stats
```

安装后需要重启设备

```
sudo reboot
```

Vim操作

实验过程中需要使用vim命令在终端对文件进行读写操作。
若无vim命令，先安装：

```
sudo apt-get update
sudo apt-get install vim -y
```

vim 基本操作：

```
i 写入
需要在写入模式按Esc后使用命令：
:wq 保存并退出
:q 退出
:w 保存
```

环境部署

1.控制台输入

```
git clone https://github.com/dusty-nv/jetson-containers
bash jetson-containers/install.sh
```

2.安装docker
安装docker，添加docker官方GPG密钥（docker官方文档https://docs.docker.com/engine/install/ubuntu/）
在终端输入

```
sudo apt-get update
sudo apt-get install ca-certificates curl
sudo install -m 0755 -d /etc/apt/keyrings
sudo curl -fsSL https://mirrors.aliyun.com/docker-ce/linux/ubuntu/gpg -o
/etc/apt/keyrings/docker.asc
sudo chmod a+r /etc/apt/keyrings/docker.asc
```

添加docker仓库到apt源列表（echo这段命令粘贴为单行）

```
echo
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc]
https://download.docker.com/linux/ubuntu
  $(. /etc/os-release && echo "${UBUNTU_CODENAME:-$VERSION_CODENAME}") stable" |
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

sudo apt-get update
```

安装docker包（出现443 Error表示网络问题，重启终端重试几次）

```
sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin
docker-compose-plugin
```

安装docker后，需要更换docker拉取容器镜像的镜像源

```
sudo vim /etc/docker/daemon.json
```

将daemon.json中内容替换为以下内容

```
{
  "registry-mirrors": [
    "https://docker.1ms.run",
    "https://docker-0.unsee.tech",
    "https://docker.m.daocloud.io"
  ],
  "live-restore": true,
  "features": { "buildkit": true }
}
```

重启docker服务

```
sudo systemctl daemon-reload
sudo systemctl restart docker
```

测试拉取镜像

```
sudo docker run hello-world
```

当输出"Hello from Docker! "等内容时，说明docker配置成功

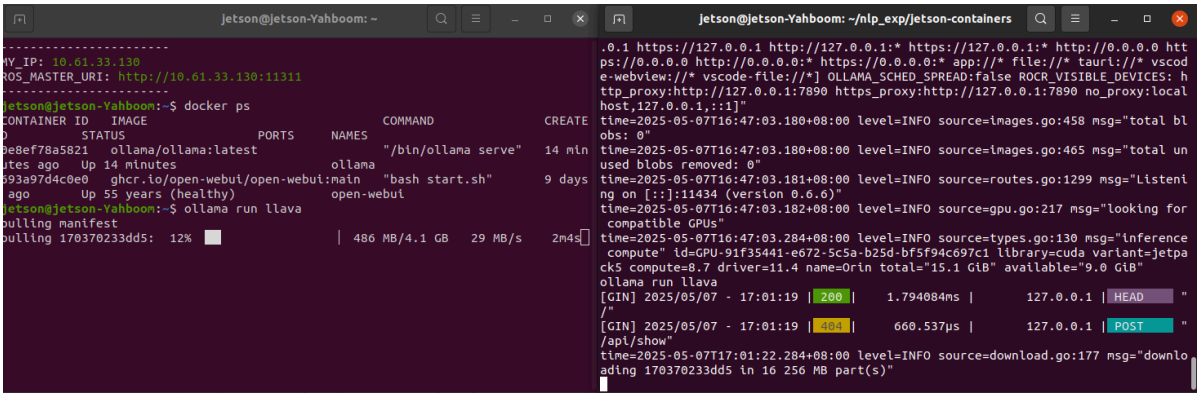
为docker配置NVIDIA Container Toolkit，
终端输入（distribution这行命令粘贴为单行）

```
distribution=$(. /etc/os-release;echo $ID$VERSION_ID)
curl -s -L https://nvidia.github.io/nvidia-docker/gpgkey | sudo apt-key add -
curl -s -L https://nvidia.github.io/nvidia-docker/$distribution/nvidia-
docker.list | sudo tee /etc/apt/sources.list.d/nvidia-docker.list

# 安装nvidia-container-toolkit
sudo apt-get update
sudo apt-get install -y nvidia-container-toolkit
# 生成默认配置（会自动添加nvidia运行时）
sudo nvidia-ctk runtime configure --runtime=docker
# 重启Docker服务使配置生效
sudo systemctl restart docker
```

3.配置jetson-containers环境,输入:

```
# Ollam:llava will be our multimodel model
jetson-containers run --name ollama $(autotag ollama)
ollama run llava
```

安装本实验所需的llava模型。



若配置jetson-containers环境时，关于“tuple”报错，手动修改python版本报错信息

```
sudo vim /home/cg/jetson-containers/jetson_containers/l4t_version.py
```

在l4t_version.py中添加导入

```
from typing import Tuple
```

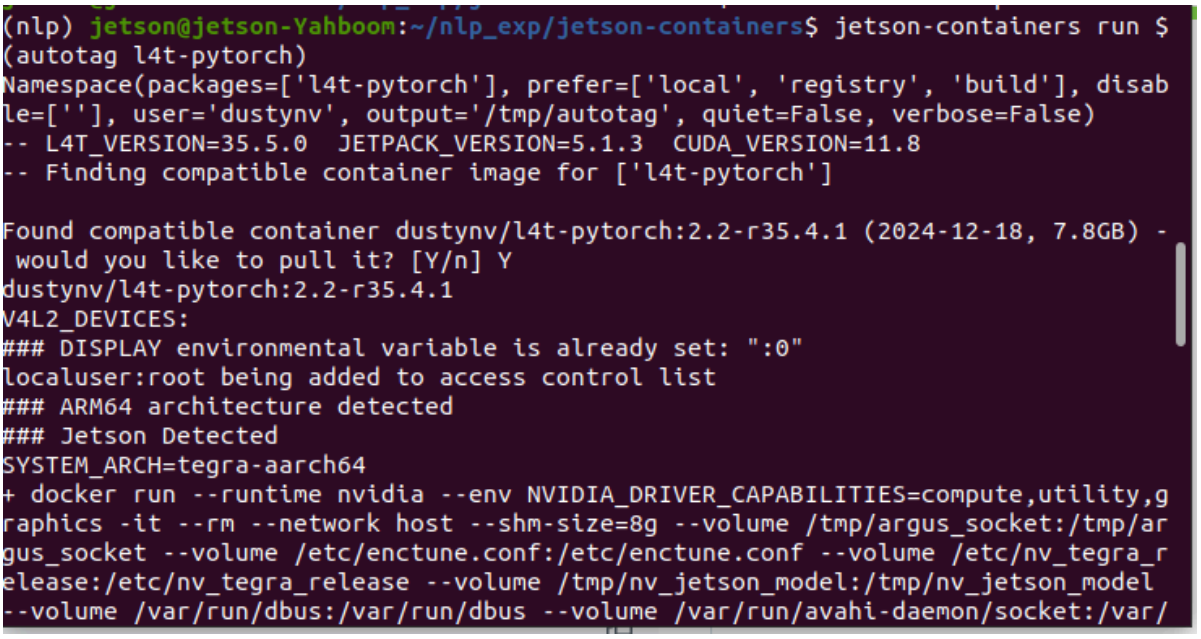
同时将函数_parse_python_ver_and_nogil的返回值类型从tuple[Version, bool]修改为Tuple[Version, bool]:

```
def _parse_python_ver_and_nogil(s) -> Tuple[Version, bool]:
```

4.打开新的终端，输入

```
cd jetson-containers
jetson-containers run $(autotag l4t-pytorch)
```

安装相关依赖torch



代码编写
控制台输入

```
cd data
mkdir Multimodal-RAG-on-Jetson
cd ./Multimodal-RAG-on-Jetson
```

进行代码编写:
创建 multiRAG.py 文件

```
import os
from moviepy.editor import VideoFileClip  # 用于处理视频文件，如提取音频和帧
import whisper  # 用于语音识别，将音频转为文本

from llama_index.core import SimpleDirectoryReader  # 读取目录中的文档
from llama_index.core import VectorStoreIndex  # 创建向量存储索引
from llama_index.embeddings.huggingface import HuggingFaceEmbedding  # 使用
HuggingFace 的嵌入模型
from llama_index.llms.ollama import Ollama  # 连接到 Ollama 大语言模型
from llama_index.core import Settings  # 设置全局的 LLM 和嵌入模型

from llama_index.core.schema import ImageDocument  # 表示图像文档
from llama_index.multi_modal_llms.ollama import OllamaMultiModal  # 使用支持多模态的
Ollama 模型

from pydub import AudioSegment  # 用于操作音频文件
from pydub.utils import make_chunks  # 将音频文件分割成小块

class VideoProcessor:
    """
```

```
    视频处理器类，用于从视频中提取音频、分段音频、提取文本和关键帧。
    """
    def __init__(self, video_path, output_audio_path, image_path, text_path):
        self.video_path = video_path  # 视频文件路径
        self.output_audio_path = output_audio_path  # 输出音频文件的路径
        self.image_path = image_path  # 提取图像的保存路径
        self.text_path = text_path  # 提取文本的保存路径

    def extract_audio(self):
        """
        从视频中提取音频并保存为 mp3 格式。
        """
        video = VideoFileClip(os.path.join(self.video_path, "video.mp4"))
        audio_part = video.audio
        audio_part.write_audiofile(os.path.join(self.output_audio_path,
"output_audio.mp3"))

    def segment_audio(self):
        """
        将音频文件分割成小块（2秒），便于后续处理。
        """
        audio = AudioSegment.from_mp3(os.path.join(self.output_audio_path,
"output_audio.mp3"))
        chunk_length_ms = 2000
        chunks = make_chunks(audio, chunk_length_ms)
        for i, chunk in enumerate(chunks):
            chunk_name = os.path.join(self.output_audio_path, f"{i}.mp3")
            chunk.export(chunk_name, format="mp3")
        os.remove(os.path.join(self.output_audio_path, "output_audio.mp3"))

    def extract_text(self):
        """
        使用 Whisper 模型将音频转为文本，并记录时间戳。
        """
        model = whisper.load_model("base.en")
        audio_text = ''
        for filename in os.listdir(self.output_audio_path):
                file_path = os.path.join(self.output_audio_path, filename)
                result = model.transcribe(file_path)
                time = int(filename[:-4]) * 2
                audio_text += str(f'At time {time}s:') + result['text'] + '\n'
        with open(os.path.join(self.text_path, "audio.md"), "w") as file:
            file.write(audio_text)
            file.close()

    def extract_frames(self):
        """
        从视频中提取关键帧，按照一定频率保存为图片。
        """
        clip = VideoFileClip(os.path.join(self.video_path, "video.mp4"))
        clip.write_images_sequence(os.path.join(self.image_path, "%04d.png"),
fps=0.5)

    def process_video(self):
        """
        执行视频处理流程：提取音频 -> 分割音频 -> 转换为文本 -> 提取关键帧。
```

```
        """
        self.extract_audio()
        self.segment_audio()
        self.extract_text()
        self.extract_frames()


class translate_image_to_text:
    """
    图像转文本类，用于对提取的图像进行描述并构建问答系统。
    """
    def __init__(self, image_path, text_path):
        self.image_path = image_path  # 图像文件路径
        self.text_path = text_path  # 文本输出路径
        self.response = ''  # 存储图像描述的响应

    def get_image_path(self):
        """
        获取所有图像文件的路径。
        """
        image_folder = self.image_path
        image_files = [os.path.join(image_folder, f) for f in
os.listdir(image_folder) if os.path.isfile(os.path.join(image_folder, f))]
        return image_files

    def image_to_text(self):
        """
        使用多模态模型对每张图像生成描述，并保存为文本文件。
        """
        mm_model = OllamaMultiModal(model='llava', temperature=0)
        image_file_names = self.get_image_path()
        for image in image_file_names:
            print(image)
            time = 2*int(image[8:-4])
            self.response += str(f'At time {time}s:')+
str(mm_model.complete(prompt='summarize the image and output as markdown format
with one line', image_documents=[ImageDocument(image_path=image)])) + '\n'
        with open(self.text_path+'image.md', 'w') as file:
                file.write(self.response)
                file.close()

    def reply(self):
        """
        构建查询引擎，允许用户提问并获取基于文本内容的回答。
        """
        # embed_model = TextEmbeddingsInference(model_name="BAAI/bge-large-en-
v1.5")
        embed_model = HuggingFaceEmbedding(model_name="BAAI/bge-small-en")
        llm = Ollama(model='llava', request_timeout=100, temperature=0)
        Settings.llm = llm
        Settings.embed_model = embed_model
        data = SimpleDirectoryReader(self.text_path).load_data()
        index = VectorStoreIndex.from_documents(data)
        query_engine = index.as_query_engine(similarity_top_k=3)
        while True:
            try:
```

```
                user_input = input('\033[94m' +"Prompt: " + '\033[0m')
                response = query_engine.query(user_input)
                print(response)
            except KeyboardInterrupt:
                break

if __name__ == '__main__':

    video_path = './video/'
    output_audio_path = './audio/'
    image_path = './image/'
    text_path = './text/'
    # output_folder= './output/'
# process video to images and text
    processor = VideoProcessor(video_path, output_audio_path, image_path,
text_path)
    processor.process_video()

    text = translate_image_to_text(image_path=image_path, text_path=text_path)
    text.image_to_text()
    text.reply()
```

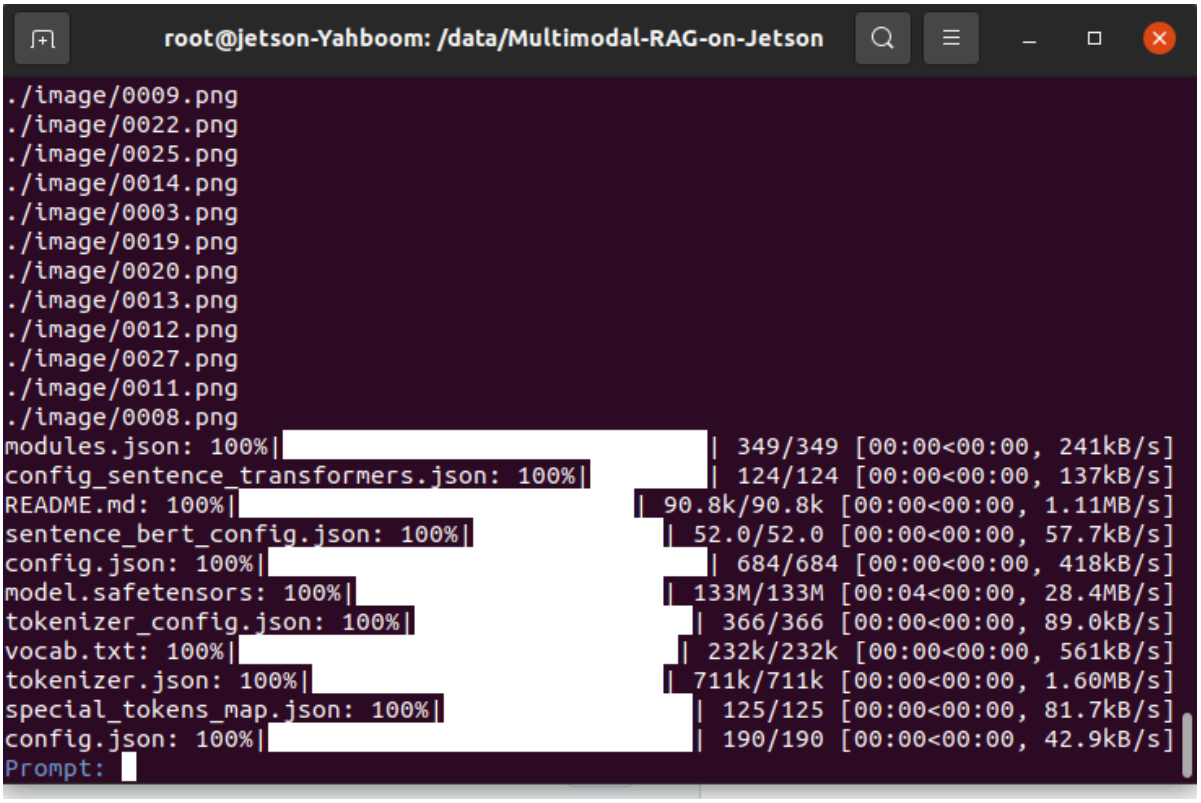
创建 run.sh 文件

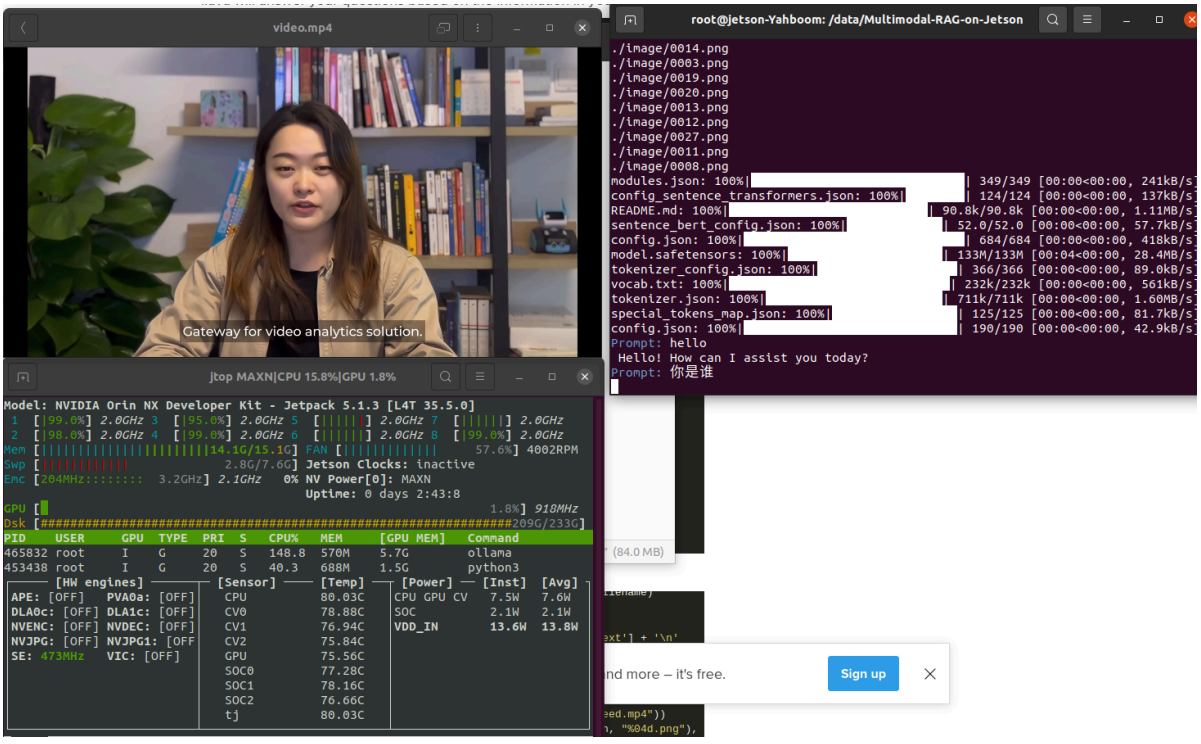
```
#!/bin/sh

mkdir ./audio
mkdir ./text
mkdir ./image
sudo apt update && sudo apt install ffmpeg -y

PIP_MIRROR="-i https://pypi.tuna.tsinghua.edu.cn/simple"
PIP_TRUST="--trusted-host pypi.tuna.tsinghua.edu.cn"
PIP_OPTS="$PIP_MIRROR $PIP_TRUST --default-timeout=180"

pip install $PIP_OPTS pydub
pip install $PIP_OPTS -U openai-whisper
pip install $PIP_OPTS llama-index
pip install $PIP_OPTS llama-index-multi-modal-llms-ollama
pip install $PIP_OPTS llama-index-llms-ollama
pip install $PIP_OPTS llama-index-llms-huggingface
pip install $PIP_OPTS moviepy
pip install $PIP_OPTS llama-index-embeddings-huggingface
pip install $PIP_OPTS llama-index-vector-stores-lancedb
pip install $PIP_OPTS llama-index-embeddings-clip
pip install git+https://github.com/openai/CLIP.git
export HF_ENDPOINT=https://hf-mirror.com
export LD_PRELOAD=/usr/local/lib/python3.8/dist-
packages/scikit_learn.libs/libgomp-d22c30c5.so.1.0.0
python3 ./multiRAG.py
```

实际测试
创建video文件夹，将video.mp4视频文件放入其中 完成代码编写后，控制台输入

```
bash run.sh
```

运行代码
可以看到，导入本地视频和图像后，可以对视频进行提问，测试成功。





若传入video.mp4时权限不足，解锁当前文件夹（username处输入当前用户名）

```
sudo chown -R username Multimodal-RAG-on-Jetson
```

如果运行run.sh运行报错：

```
AttributeError: 'GenerateResponse' object has no attribute 'items'
```

需手动修改报错信息，打开如下路径文件：

```
vim /usr/local/lib/python3.8/dist-
packages/llama_index/multi_modal_llms/ollama/base.py
```

将源代码

```
def get_additional_kwargs(response, exclude):
    return {k: v for k, v in response.items() if k not in exclude}
```

修改为

```
def get_additional_kwargs(response, exclude):
    # 将 Pydantic 模型转换为字典，再调用 items()
    return {k: v for k, v in response.dict().items() if k not in exclude}
```

如果进行视频prompt提问时报错：

```
ValueError: "ChatResponse" object has no field "usage"
```

需手动修改报错信息，打开如下路径文件：

```
vim /usr/local/lib/python3.8/dist-packages/llama_index/llms/ollama/base.py
```

将下列代码注释掉（共两处）

```
if token_counts:
    response["usage"] = token_counts
```

删除docker

实验结束后，删除docker相关文件

```
sudo apt purge docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-
compose-plugin
```