

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:

```
# ollama: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。

The screenshot shows two terminal windows. The left window displays the command 'docker ps' output, showing a container named 'ollama' running the image 'ollama/ollama:latest'. The right window shows logs from the 'ollama' container, specifically the 'llava' component. The logs include various HTTP requests and responses, such as 'GET / 200 OK', 'POST /api/show', and 'PUT /api/show'. The logs also mention GPU usage and inference times.

```
jetson@jetson-Yahboom:~
```

```
jetson@jetson-Yahboom:~/nlp_exp/jetson-containers
```

```
IV_IP: 10.61.33.130
ROS_MASTER_URI: http://10.61.33.130:11311

jetson@jetson-Yahboom:~$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS               NAMES
be8ef78a5821        ollama/ollama:latest   "/bin/ollama serve"   14 min ago         Up 14 minutes      ollama
593a97d4c0e0        ghr.io/open-webui/open-webui:main   "bash start.sh"    9 days ago        Up 55 years (healthy)   open-webui
jetson@jetson-Yahboom:~$ ollama run llava
pulling manifest
pulling 170370233dd5: 12% [■]          | 486 MB/4.1 GB  29 MB/s  2m4s[■] time=2025-05-07T16:47:03.180+08:00 level=INFO source=images.go:458 msg="total blobs: 0"
time=2025-05-07T16:47:03.180+08:00 level=INFO source=images.go:465 msg="total unused blobs removed: 0"
time=2025-05-07T16:47:03.181+08:00 level=INFO source=routes.go:1299 msg="Listening on [::]:11434 (version 0.6.6)"
time=2025-05-07T16:47:03.182+08:00 level=INFO source=gpu.go:217 msg="looking for compatible GPUs"
time=2025-05-07T16:47:03.284+08:00 level=INFO source=types.go:130 msg="Inference compute" id=GPU-91f35441-e672-5c5a-b25d-bfsf9ac697c1 library=cuda variant=jetpac5 compute=8.7 driver=11.4 name=Orin total="15.1 GiB" available="9.0 GiB"
ollama run llava
[GIN] 2025/05/07 - 17:01:19 | 200 | 1.794084ms | 127.0.0.1 | HEAD |
[GIN] 2025/05/07 - 17:01:19 | 200 | 660.537µs | 127.0.0.1 | POST |
/api/show
time=2025-05-07T16:47:03.284+08:00 level=INFO source=download.go:177 msg="downloading 170370233dd5 in 16 256 MB part(s)"
```

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

```
(nlp) jetson@jetson-Yahboom:~/nlp_exp/jetson-containers$ jetson-containers run $(autotag l4t-pytorch)
Namespace(packages=['l4t-pytorch'], prefer=['local', 'registry', 'build'], disable=[''], user='dustynv', output='/tmp/autotag', quiet=False, verbose=False)
-- L4T_VERSION=35.5.0 JETPACK_VERSION=5.1.3 CUDA_VERSION=11.8
-- Finding compatible container image for ['l4t-pytorch']

Found compatible container dustynv/l4t-pytorch:2.2-r35.4.1 (2024-12-18, 7.8GB) -
would you like to pull it? [Y/n] Y
dustynv/l4t-pytorch:2.2-r35.4.1
V4L2_DEVICES:
### DISPLAY environmental variable is already set: ":0"
localuser:root being added to access control list
### ARM64 architecture detected
### Jetson Detected
SYSTEM_ARCH=tegra-aarch64
+ docker run --runtime nvidia --env NVIDIA_DRIVER_CAPABILITIES=compute,utility,graphics -it --rm --network host --shm-size=8g --volume /tmp/argus_socket:/tmp/argus_socket --volume /etc/enctune.conf:/etc/enctune.conf --volume /etc/nv_tegra_release:/etc/nv_tegra_release --volume /tmp/nv_jetson_model:/tmp/nv_jetson_model --volume /var/run/dbus:/var/run/dbus --volume /var/run/avahi-daemon/socket:/var/
```

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:\n')+
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-llama
pip install $PIP_OPTS llama-index-llms-llama
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。

./image/0009.png
./image/0022.png
./image/0025.png
./image/0014.png
./image/0003.png
./image/0019.png
./image/0020.png
./image/0013.png
./image/0012.png
./image/0027.png
./image/0011.png
./image/0008.png
modules.json: 100% | ██████████ | 349/349 [00:00<00:00, 241kB/s]
config_sentence_transformers.json: 100% | ██████████ | 124/124 [00:00<00:00, 137kB/s]
README.md: 100% | ██████████ | 90.8k/90.8k [00:00<00:00, 1.11MB/s]
sentence_bert_config.json: 100% | ██████████ | 52.0/52.0 [00:00<00:00, 57.7kB/s]
config.json: 100% | ██████████ | 684/684 [00:00<00:00, 418kB/s]
model.safetensors: 100% | ██████████ | 133M/133M [00:04<00:00, 28.4MB/s]
tokenizer_config.json: 100% | ██████████ | 366/366 [00:00<00:00, 89.0kB/s]
vocab.txt: 100% | ██████████ | 232k/232k [00:00<00:00, 561kB/s]
tokenizer.json: 100% | ██████████ | 711k/711k [00:00<00:00, 1.60MB/s]
special_tokens_map.json: 100% | ██████████ | 125/125 [00:00<00:00, 81.7kB/s]
config.json: 100% | ██████████ | 190/190 [00:00<00:00, 42.9kB/s]
Prompt: █

video.mp4

jtop MAXN/CPU 15.8%GPU 1.8%

Model:	NVIDIA Orin NX Developer Kit - Jetpack 5.1.3 [L4T 35.5.0]
1	[99.0%] 2.0GHz 3 [99.0%] 2.0GHz 5 [██████] 2.0GHz 7 [██████] 2.0GHz
2	[99.0%] 2.0GHz 4 [99.0%] 2.0GHz 6 [██████] 2.0GHz 8 [99.0%] 2.0GHz
Mem	[████████████████] 14.1G/15.1G FAN [████████████████] 57.6% 4002RPM
swp	[████████████████] 2.8G/7.6G Jetson Clocks: inactive
EMC	[████████████████] 3.2GHz] 2.1GHz 0% NV Power[0]: MAXN Uptime: 0 days 2:43:8

GPU [1.8%] 910MHz

Disk #####/##### 209G/23G

PID	USER	GPU	Type	PRI	S	CPU%	MEM	[GPU MEM]	Command
465832	root	I	G	20	S	148.8	570M	5.7G	oillama
45343B	root	I	G	20	S	40.3	688M	1.5G	python3

[HW engines] [Sensor] [Temp] [Power] [Inst] [Avg]

APE: [OFF]	PVA0a: [OFF]	CPU	80.03C	CPU GPU CV	7.5W	7.6W
DLA0c: [OFF]	DLA1c: [OFF]	CV0	78.88C	SOC	2.1W	2.1W
NVENC: [OFF]	NVDEC: [OFF]	CV1	76.94C	VDD_IN	13.6W	13.8W
NVJPG: [OFF]	NVJPG1: [OFF]	CV2	75.84C			
SE: 473MHz	VIC: [OFF]	GPU	75.56C			
		SOC0	77.28C			
		SOC1	78.16C			
		SOC2	76.66C			
		tj	80.03C			

sed.mp4")
mksd4d.mp4")

root@jetson-Yahboom: /data/Multimodal-RAG-on-Jetson

Hello! How can I assist you today?
Prompt: 你是谁

Sign up X

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
```