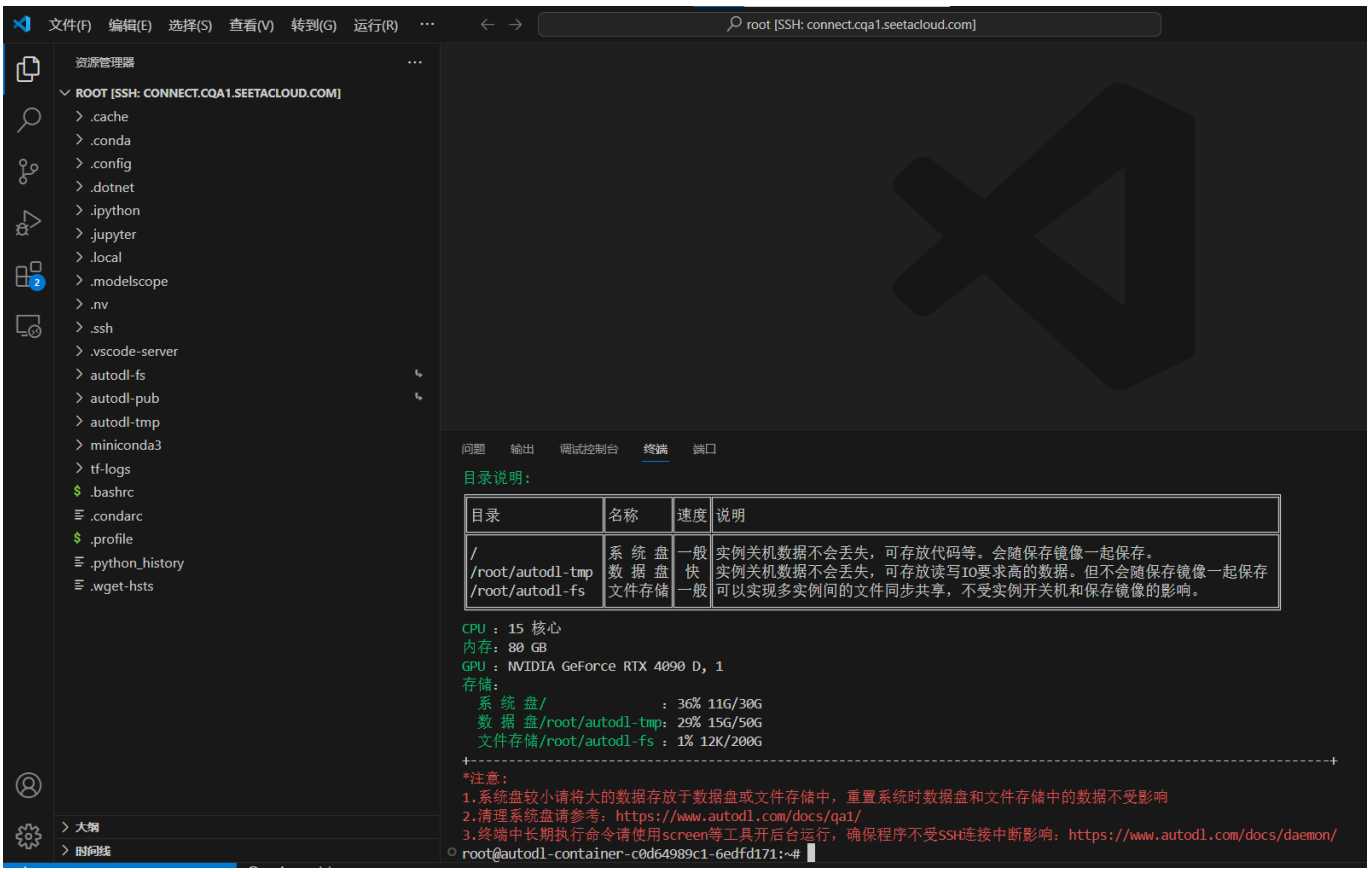
VsCode连接AutoDL远程环境

先说一个技巧,如果想直接用autoDL上的环境来编辑调测代码,可以用本机的VSCODE通过SSH工具连接到 autoDL服务器。

详细攻略见: `https://blog.csdn.net/ABV09876543210/article/details/134811442`

最后的效果如下:



早知道可以这样就不用在autoDL的jupyter上这么辛苦来回切了。

核心知识点

所谓AI模拟面试,就是由AI充当面试官,和面试者进行声音文字的实时交流,并且AI能够对面试者的回答做出评价以及提出下一个问题,或者终结面试。

做一个结合AI能力的模拟面试web应用,有几个核心功能要做好设计:

1. 如何录音
2. 如何播放声音
3. 如何声音文件转化成文字

4. 如何将文字转化成声音

除此之外,采用两个核心框架:

1. gradio 用于快速构建web应用的前端演示界面
2. fastApi 用于快速构建web后端服务

本次实验我们依然采用autoDL租赁服务器,但是由于它只能对外开放一个6006端口,所以我如果想在我本机对gradio代码进行调测,我就只能将所有的后端fastApi服务都放在6006端口上。本机启动gradio界面,用python的request访问这个远程服务。

本次我采用的python版本是 3.11,和老师课程里面用到的3.8有所不同。所以以下代码都是参考老师代码自己手写,解决版本兼容问题。

核心功能攻略

如何录音

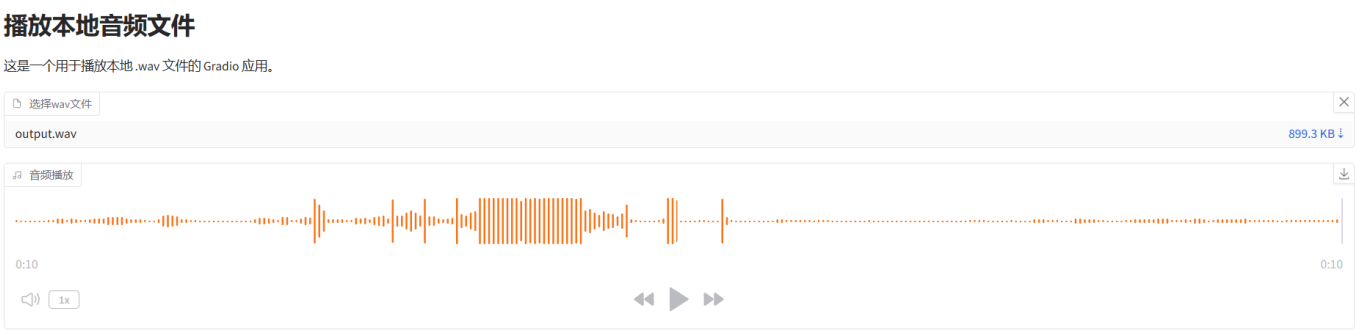
gradio本身就提供了录音组件`gr.Audio(...)`,下面的例子演示了如何创建一个录音组件,并且录音之后保存在本地.wav文件:

```python
import gradio as gr
import scipy


def save_audio(audio):
    try:
        print("==========audioType==========", type(audio))
        print("==========audio==========", audio)
        sample_rate, audio_data = audio
        savePath = "output.wav"
        scipy.io.wavfile.write(savePath, sample_rate, audio_data)
        return f"已经保存在{savePath}"
    except Exception as e:
        return f"保存失败{e}"

input_audio = gr.Audio(sources=["microphone"])

with gr.Blocks() as demo:
    # 录音组件
    audio_input = gr.Audio(sources=["microphone"], format="mp3")
    # 保存音频按钮
    save_button = gr.Button("保存录音")
    # 输出组件
    output = gr.Label()

    # 按钮点击事件处理
    save_button.click(save_audio, inputs=audio_input, outputs=output)

# 启动界面
demo.launch()
```

如何播放声音

如果有一个本地的.wav文件,我如何用gr.Audio组件将它播放出来呢？看下面的例子:

```python
import gradio as gr


def play_audio(file):
    # 返回文件路径以供播放
    return file.name


# 创建 Gradio Blocks
with gr.Blocks() as demo:
    # 添加标题和描述
    gr.Markdown("# 播放本地音频文件")
    gr.Markdown("这是一个用于播放本地 .wav 文件的 Gradio 应用。")

    file_input = gr.File(
        label="选择wav文件",
        file_types=[".wav"],
    )

    # 添加音频组件来播放音频
    audio_output = gr.Audio(
        label="音频播放",
        type="filepath",
    )

    # 当页面加载时,自动加载并播放音频
    file_input.change(
        play_audio,
        inputs=file_input,
        outputs=audio_output,
    )

# 启动 Gradio 应用
demo.launch()
```



如何声音文件转化成文字

要将声音文件转化成文字,我们需要借助STT模型.

STT,即（Sound transform to Text），它可以将声音文件转化成文字。

由于启动大模型需要用到GPU,所以接下来的试验我转移到autoDL上去做。

老师采用的STT模型是huggingface上的 `whisper`, 这里涉及到两个知识点:

1. 如何安装模型的使用环境
2. 如何用模型将声音转化成文字

试试老师提供的huggingface的whisper

进入autoDL的jupyter页面,然后

- 安装环境:

```
conda create --name stt python=3.8
conda activate stt

conda install pytorch==1.10.0 torchaudio cudatoolkit=11.3 -c pytorch
pip install transformers accelerate
pip install -U huggingface_hub
pip install "fastapi[standard]"

pip install -U openai-whisper
pip install git+https://github.com/openai/whisper.git
sudo apt update && sudo apt install ffmpeg
pip install setuptools-rust

conda install ipykernel
python -m ipykernel install --user --name stt --display-name "Python 3.8 (stt)"

pip install datasets
pip install librosa soundfile
```

注意,这里如果出现 `conda activate stt` 失败,导致无法进入 stt虚拟环境,尝试用 `source activate stt`.

测试代码

```python
import whisper

model = whisper.load_model("tiny")

result = model.transcribe("output.wav")

print(result["text"])
```

```
(stt) root@autodl-container-8c094bba4f-910937cd:~/sttTest# python stt_test01.py
100%|████████████████████████████████████| 72.1M/72.1M [00:05<00:00, 12.8MiB/s]
不~~~你好
```

运行这段代码的时候,会很明显看到有一个模型的加载进度条。大概花费了5秒的时间。然后才是识别语音文件 output.wav的过程,最后输出识别的结果。

试试modelscope上的国产STT大模型

https://modelscope.cn/models/iic/SenseVoiceSmall

> SenseVoice专注于高精度多语言语音识别、情感辨识和音频事件检测
>
> - 多语言识别: 采用超过40万小时数据训练,支持超过50种语言,识别效果上优于Whisper模型。
> - 富文本识别:具备优秀的情感识别,能够在测试数据上达到和超过目前最佳情感识别模型的效果。支持声音事件检测能力,支持音乐、掌声、笑声、哭声、咳嗽、喷嚏等多种常见人机交互事件进行检测。
> - 高效推理: SenseVoice-Small模型采用非自回归端到端框架,推理延迟极低,10s音频推理仅耗时70ms,15倍优于Whisper-Large。
> - 微调定制: 具备便捷的微调脚本与策略,方便用户根据业务场景修复长尾样本问题。
> - 服务部署: 具有完整的服务部署链路,支持多并发请求,支持客户端语言有,python、c++html、java与c#等

据说性能是比whisper更强,试试看。

重新开一个conda环境

```
conda create --name funsar python=3.8
conda activate funsar
```

安装需要的依赖

```
pip install -r requirements.txt
```

requirements.txt文件的内容是:

```
torch<=2.3
torchaudio
modelscope
huggingface
huggingface_hub
funasr>=1.1.3
numpy<=1.26.4
gradio
fastapi>=0.111.1
```

安装依赖的时候会自动下载模型,不需要我们手动去下。但是,如果非要去手动下载然后手动加载的话,也不是不行。

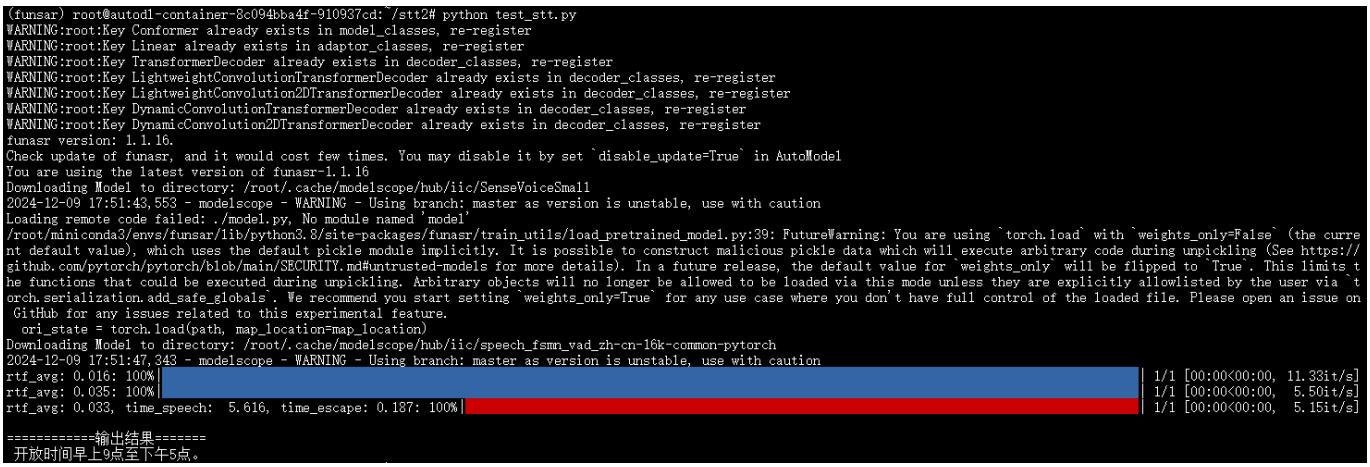
注意,某些机器可能会缺少 `ffmpeg` 和 `onnxconverter-common` ,如下方式可以解决:

```
# 更新包列表（Ubuntu）
sudo apt update

# 安装 ffmpeg（Ubuntu）
sudo apt install ffmpeg

# 安装 onnxconverter-common
pip install onnxconverter-common
```

测试代码:

```python
from funasr import AutoModel
from funasr.utils.postprocess_utils import rich_transcription_postprocess

model_dir = "iic/SenseVoiceSmall"


model = AutoModel(
    model=model_dir,
    trust_remote_code=True,
    remote_code="./model.py",
    vad_model="fsmn-vad",
    vad_kwargs={"max_single_segment_time": 30000},
    device="cuda:0",
)

res = model.generate(
    input="https://isv-data.oss-cn-hangzhou.aliyuncs.com/ics/MaaS/ASR/test_audio/asr_example_zh.wav",
    cache={},
    language="auto",  # "zn", "en", "yue", "ja", "ko", "nospeech"
    use_itn=True,
    batch_size_s=60,
    merge_vad=True,  #
    merge_length_s=15,
)
text = rich_transcription_postprocess(res[0]["text"])
print(text)
```

执行结果为:



这里的input参数,还能换成 本地文件:

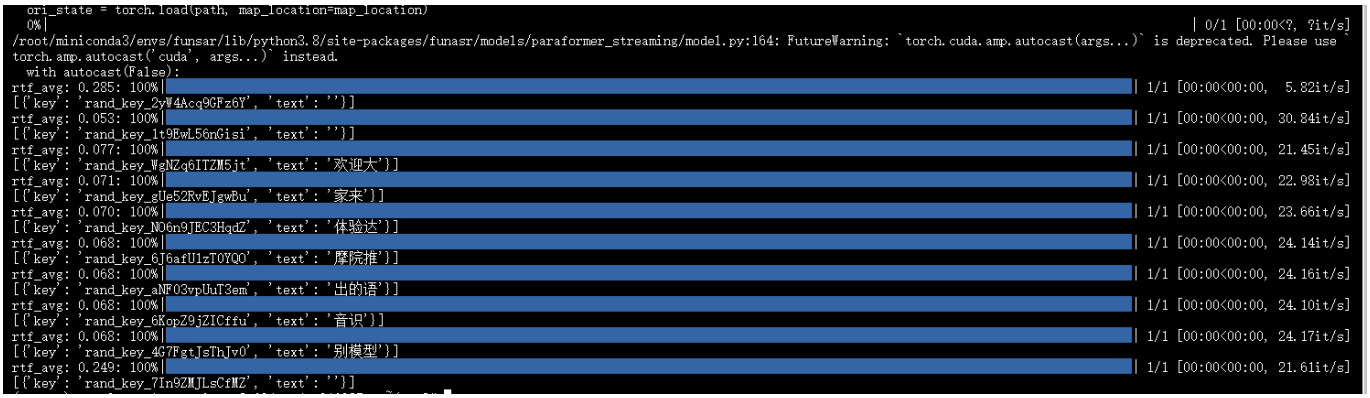
```
res = model.generate(
    input=f"{model.model_path}/example/zh.mp3",
    ......
)
```

此案例的相关官方文档都在:https://github.com/modelscope/FunASR/blob/main/examples/industrial_data_pretraining/paraformer_streaming/README_zh.md

另外,还支持实时语音识别,也就是按照流式输入的语音进行流式的文字输出。

```
from funasr import AutoModel

chunk_size = [0, 10, 5] #[0, 10, 5] 600ms, [0, 8, 4] 480ms
encoder_chunk_look_back = 4 #number of chunks to lookback for encoder self-attention
decoder_chunk_look_back = 1 #number of encoder chunks to lookback for decoder cross-attention

model = AutoModel(model="paraformer-zh-streaming")

import soundfile
import os

wav_file = os.path.join(model.model_path, "example/asr_example.wav")
speech, sample_rate = soundfile.read(wav_file)
chunk_stride = chunk_size[1] * 960 # 600ms

cache = {}
total_chunk_num = int(len((speech)-1)/chunk_stride+1)
for i in range(total_chunk_num):
    speech_chunk = speech[i*chunk_stride:(i+1)*chunk_stride]
    is_final = i == total_chunk_num - 1
    res = model.generate(input=speech_chunk, cache=cache, is_final=is_final, chunk_size=chunk_size, encoder_chunk_look_back=encoder_chunk_look_back, decoder_chunk_look_back=decoder_chunk_look_back)
    print(res)
```

输出结果:



如何将文字转化成声音

上面的STT,实际上也可以称之为语音识别,而反过来,就是语音生成,将文本转化成语音。

试一下老师提供的TTS模型

安装环境

```
conda create --name tts2 python=3.8
conda activate tts2
conda install pytorch==1.10.0 torchvision torchaudio cudatoolkit=11.3 -c pytorch

pip install scipy
pip install transformers accelerate
pip install -U huggingface_hub
pip install "fastapi[standard]"
pip install soundfile

conda install ipykernel
python -m ipykernel install --user --name tts2 --display-name "Python 3.8 (tts2)"

cd /root/autodl-tmp/
```

STT模型不需要手动下载,而且根据运行日志来看,STT模型都比较小,可以在启动任务时同步下载。
与上面STT有所不同,TTS模型需要提前下载,本次使用huggingface下载。

```
pip install -U huggingface_hub

# 设置huggingface网站镜像
export HF_HOME=/root/autodl-tmp/huggingface-cache/
export HF_ENDPOINT=https://hf-mirror.com

cd /root/autodl-tmp
# 下载模型
huggingface-cli download --resume-download --local-dir-use-symlinks False
facebook/mms-tts-eng --local-dir mms-tts-eng
```

运行测试

```
from pydantic import BaseModel
from transformers import AutoTokenizer, VitsModel
import torch


# 加载预训练模型和分词器
model = VitsModel.from_pretrained("mms-tts-eng")
tokenizer = AutoTokenizer.from_pretrained("mms-tts-eng")

text = '''
Sure, here's a 100-word English monologue:

"Standing here, I feel the weight of the world on my shoulders. Every decision, every
action, seems to hold the potential for consequence, for change. I'm aware of my own
power, and yet I'm humbled by the vastness of what lies ahead. I'm not afraid, though.
I'm ready to face the challenges, to embrace the unknown. After all, it's in these
moments of uncertainty that we grow the most."
'''
inputs = tokenizer(text, return_tensors="pt")

with torch.no_grad():
    output = model(**inputs).waveform

from IPython.display import Audio
Audio(output.numpy(), rate=model.config.sampling_rate)

import scipy

scipy.io.wavfile.write("techno.wav", rate=model.config.sampling_rate,
data=output.float().numpy().T)
```

我这里运行报错:

```
Traceback (most recent call last):
  File "tts.py", line 14, in <module>
    output = model(**inputs).waveform
  File "/root/miniconda3/envs/tts/lib/python3.8/site-
packages/torch/nn/modules/module.py", line 1102, in _call_impl
    return forward_call(*input, **kwargs)
  File "/root/miniconda3/envs/tts/lib/python3.8/site-
packages/transformers/models/vits/modeling_vits.py", line 1423, in forward
    text_encoder_output = self.text_encoder(
  File "/root/miniconda3/envs/tts/lib/python3.8/site-
packages/torch/nn/modules/module.py", line 1102, in _call_impl
    return forward_call(*input, **kwargs)
  File "/root/miniconda3/envs/tts/lib/python3.8/site-
packages/transformers/models/vits/modeling_vits.py", line 1222, in forward
    encoder_outputs = self.encoder(
  File "/root/miniconda3/envs/tts/lib/python3.8/site-
packages/torch/nn/modules/module.py", line 1102, in _call_impl
    return forward_call(*input, **kwargs)
  File "/root/miniconda3/envs/tts/lib/python3.8/site-
packages/transformers/models/vits/modeling_vits.py", line 1143, in forward
    synced_gpus = is_deepspeed_zero3_enabled() or is_fsdp_managed_module(self)
  File "/root/miniconda3/envs/tts/lib/python3.8/site-
packages/transformers/integrations/fsdp.py", line 29, in is_fsdp_managed_module
    import torch.distributed.fsdp
ModuleNotFoundError: No module named 'torch.distributed.fsdp'
```

缺少模块,一般我们通过更新或者降低 torch版本就能解决。我本次的解决方式是:`pip install --upgrade torch`

最后的运行效果是,生成了一个名叫:`techno.wav` 的音频文件。我们将它down下来播放看看。OK,听过了之后,发现这个音频文件确实是读的我上面代码里面提供的英文段落。

魔塔上找一个TTS模型

想找一个支持中文的,使用步骤比较清楚的,居然找不到! 继续找。

我好像明白了为什么老师要分开几个虚拟环境来安装各种模型,可能是因为每个模型之间的依赖库可能有冲突,而且冲突还不好解决。

`iic/CosyVoice-300M-SFT` 这个好像还可以。地址:`https://modelscope.cn/models/iic/CosyVoice-300M-SFT`

github官方:`https://github.com/FunAudioLLM/CosyVoice`

安装方式:

```
git clone --recursive https://github.com/FunAudioLLM/CosyVoice.git
# If you failed to clone submodule due to network failures, please run following
command until success
cd CosyVoice
git submodule update --init --recursive
```

然后

```
conda create -n cosyvoice python=3.8
conda activate cosyvoice
# pynini is required by WeTextProcessing, use conda to install it as it can be
executed on all platform.
conda install -y -c conda-forge pynini==2.1.5
pip install -r requirements.txt -i https://mirrors.aliyun.com/pypi/simple/ --trusted-
host=mirrors.aliyun.com

# If you encounter sox compatibility issues
# ubuntu
sudo apt-get install sox libsox-dev
# centos
sudo yum install sox sox-devel
```

下一步,下载模型:

```
# SDK模型下载
from modelscope import snapshot_download
snapshot_download('iic/CosyVoice-300M', local_dir='pretrained_models/CosyVoice-300M')
snapshot_download('iic/CosyVoice-300M-25Hz', local_dir='pretrained_models/CosyVoice-
300M-25Hz')
snapshot_download('iic/CosyVoice-300M-SFT', local_dir='pretrained_models/CosyVoice-
300M-SFT')
snapshot_download('iic/CosyVoice-300M-Instruct',
local_dir='pretrained_models/CosyVoice-300M-Instruct')
snapshot_download('iic/CosyVoice-ttsfrd', local_dir='pretrained_models/CosyVoice-
ttsfrd')
```

在使用模型之前:

上面下载了5个模型,我们选择其中第一个 `CosyVoice-300M` 来做实验, 测试代码 `test.py` 内容如下:

```
from cosyvoice.cli.cosyvoice import CosyVoice
from cosyvoice.utils.file_utils import load_wav
import torchaudio

cosyvoice = CosyVoice("third_party/Matcha-TTS/pretrained_models/CosyVoice-300M")
# zero_shot usage, <|zh|><|en|><|jp|><|yue|><|ko|> for
Chinese/English/Japanese/Cantonese/Korean
prompt_speech_16k = load_wav("zero_shot_prompt.wav", 16000)
for i, j in enumerate(
    cosyvoice.inference_zero_shot(
        "收到好友从远方寄来的生日礼物,那份意外的惊喜与深深的祝福让我心中充满了甜蜜的快乐,笑
容如花儿般绽放。",
        "希望你以后能够做的比我还好呦。",
        prompt_speech_16k,
        stream=False,
    )
):
    torchaudio.save("zero_shot_{}.wav".format(i), j["tts_speech"], 22050)
```

执行命令:

```
export PYTHONPATH=third_party/Matcha-TTS
python test.py
```

`cosyVoice` 的思路和 老师提供的 `facebook/mms-tts-eng` 有差别, `cosyVoice` 在生成语音的时候甚至考虑了长文本的分段生成语音的策略。

执行成功,确实可以将语音生成文本,而且它可以传入一个包含声音的音频文件作为提示词,让生成的音频文件模拟它的声音。

如何让AI给应聘者出题,以及根据应聘者的回答做出回复

这一环节实际上就是我们最熟悉的 文本生成LLM了,通过输入的文本生成文本输出。

我们可以通过 部署大模型到ollama,然后通过 openai的标准接口来生成,也可以不用ollama,直接通过动态加载模型的方式来生成文本,本次实验我们将会采用后者。

FastApi的基本使用

上一章节讲的是 语音文本相互转化的大模型的使用,以及 文生文大模型的使用。

本机将会基于FastApi这么一个快速构建Web服务的框架来将这些模型启动起来,并且对外开放服务。开放之后,外部可以通过调用http接口的方式来进行 将语音转化成文本,将文本转化成语音,以及 将根据文本生成文本。

必要前提

要使用fastApi,必须先安装如下依赖:

```
pip install "fastapi[standard]"
pip install pytest
```

基本使用

创建服务

创建一个名叫 `service_001.py` 的文件,内容如下:

```
from typing import Union
from fastapi import FastAPI

app = FastAPI()


@app.get("/")
def read_root():
    return {"Hello": "World"}


@app.get("/items/{item_id}")
def read_item(item_id: int, q: Union[str, None] = None):
    return {"item_id": item_id, "q": q}
```

这是一个简单的后端服务代码。

启动服务

启动这个后端服务的方式为:

```
uvicorn service_001:app --reload --port 8030
```

其中 service_001 是上面 `service_001.py` 的文件名,后面的8030是服务启动的端口 , `--reload` 是指此服务启动后支持热重载。

访问服务

如果我们使用本文开头的VsCode连接autoDL的方式的话,在启动上面的命令之后,我们本地vscode会自动将这个服务转发到我们本地, 所以当我在

- 我本机浏览器中输入 `http://127.0.0.1:8030/` 时,应该能看到上面 `service_001.py` 中定义的 `{"Hello":"World"}`
- 当我本机浏览器输入 `http://127.0.0.1:8030/items/1?q=1` 回车,应该能看到 `{"item_id":1,"q":"1"}`

此外,还有一种方式,那就是用python文件构建 请求访问这个服务,这是一个单元测试的python代码:

```python
import requests

# 假设 FastAPI 应用在本地的8000端口运行
BASE_URL = "http://127.0.0.1:8030"

def test_read_root():
    response = requests.get(f"{BASE_URL}/")
    assert response.status_code == 200
    assert response.json() == {"Hello": "World"}

def test_read_item():
    # 测试没有查询参数的情况
    response = requests.get(f"{BASE_URL}/items/1")
    assert response.status_code == 200
    assert response.json() == {"item_id": 1, "q": None}

    # 测试带有查询参数的情况
    response = requests.get(f"{BASE_URL}/items/1?q=somequery")
    assert response.status_code == 200
    assert response.json() == {"item_id": 1, "q": "somequery"}

if __name__ == "__main__":
    test_read_root()
    test_read_item()
    print("All tests passed!")
```

如果执行后输出 `All tests passed!` ,说明服务一切正常。

参数传递

简单参数

以下例子就是一个简单参数的传递,我们可以传递两个简单参数,一个是item_id,直接放在路径中,一个是q,要放在 问号的后面。

```
from typing import Union
from fastapi import FastAPI

app = FastAPI()


@app.get("/")
def read_root():
    return {"Hello": "World"}


@app.get("/items/{item_id}")
def read_item(item_id: int, q: Union[str, None] = None):
    return {"item_id": item_id, "q": q}
```

传参方式为:

```
http://127.0.0.1:8030/items/1?q=1
```

复杂参数

如果是复合型的参数,则需要借助BaseModel,如下面的例子,创建一个名叫Item的类,继承BaseModel,Item中定义多个字段:

创建一个 `service_002.py` 文件,内容如下:

```python
from pydantic import BaseModel
from typing import Union
from fastapi import FastAPI

app = FastAPI()

class Item(BaseModel):
    name: str
    description: str = None
    price: float
    tax: float = None

@app.post("/items/")
def create_item(item: Item):
    return item
```

启动它:

```
uvicorn service_002:app --reload --port 8030
```

这个服务我们可以用python代码来访问,创建一个test_002.py文件,运行它:

```python
import requests

BASE_URL = "http://127.0.0.1:8030"

def test_create_item():
    url = f"{BASE_URL}/items/"
    payload = {
        "name": "Test Item",
        "description": "This is a test item",
        "price": 10.5,
        "tax": 0.5
    }

    response = requests.post(url, json=payload)

    assert response.status_code == 200
    response_json = response.json()
    assert response_json["name"] == "Test Item"
    assert response_json["description"] == "This is a test item"
    assert response_json["price"] == 10.5
    assert response_json["tax"] == 0.5

if __name__ == "__main__":
    test_create_item()
    print("Test passed!")
```

混合使用

简单和复杂参数传递可以混合使用,比如再创建一个 service_003.py :

```python
from pydantic import BaseModel
from typing import Union
from fastapi import FastAPI

app = FastAPI()

class Item(BaseModel):
    name: str
    description: str = None
    price: float
    tax: float = None

@app.post("/items/")
def create_item(item: Item):
    return item

@app.put("/items/{item_id}")
def update_item(item_id: int, item: Item, q: str = None):
    return {"item_id": item_id, "item": item, "q": q}
```

启动它:

```
uvicorn service_003:app --reload --port 8030
```

测试代码:

```python
import requests


def test_create_item():
    url = f"http://127.0.0.1:8030/items/123"
    headers = {
        "Content-Type": "application/json"
    }
    payload = {
            "name": "Test Item",
            "description": "This is a test item",
            "price": 10.5,
            "tax": 0.5
    }

    params = {
        "q": "exampleQuery"  # 添加 q 参数
    }

    response = requests.put(url, headers=headers, json=payload, params=params)

    print("status_code",response.status_code)
    print("response_json",response.json())

if __name__ == "__main__":
    test_create_item()
    print("Test passed!")
```

运行结果是:

```
status_code 200
response_json {'item_id': 123, 'item': {'name': 'Test Item', 'description': 'This is a
test item', 'price': 10.5, 'tax': 0.5}, 'q': 'exampleQuery'}
Test passed!
```

更多功能

fastApi还支持表单提交,以及文件上传,等到了后面的业务场景再详细写。

使用FastApi创建TTS服务

TTS服务 tts_service.py

接下来将演示 使用FastApi构造一个 文字转语音的TTS服务:

主要思路为:

1. 针对输入的文本内容,调用 cosyVoice TTS大模型,生成多段 wav文件
2. 将多段wav文件,按照序号依次组装,最终拼接成一整个wav
3. 将最终的wav,通过 torchaudio.load转化成数据结构,并返回给外界
4. 用 python代码的方式启动服务（请看最后一行）,这种方式,不支持reload,只有从命令行直接用 uvicorn指令的方式支持热重载

完整代码 `tts_service.py` 如下:

```python
import os
from pydantic import BaseModel
from typing import Union
from fastapi import FastAPI, HTTPException

from cosyvoice.cli.cosyvoice import CosyVoice
from cosyvoice.utils.file_utils import load_wav
import torchaudio

import glob
import torch
import uvicorn

"""
这是一个用fastApi构建的 文字转语音的服务
"""
app = FastAPI()


def combine_audio_files(
    file_pattern="save_zero_shot_*.wav",
    output_filename="combined_audio.wav",
):
    # 获取所有匹配的文件
    files = sorted(glob.glob(file_pattern))

    if not files:
        raise FileNotFoundError("没有找到匹配的文件")

    # 初始化一个空的列表来存储音频数据
    audio_segments = []
    sample_rate = None

    for file in files:
        # 加载音频文件
        waveform, current_sample_rate = torchaudio.load(file)

        # 确保所有音频文件的采样率一致
        if sample_rate is None:
            sample_rate = current_sample_rate
        elif sample_rate != current_sample_rate:
            raise ValueError("所有音频文件的采样率必须一致")

        # 将音频数据添加到列表中
        audio_segments.append(waveform)

    # 拼接所有音频片段
    combined_waveform = torch.cat(audio_segments, dim=1)

    # 保存拼接后的音频数据到指定文件
```

```
    torchaudio.save(output_filename, combined_waveform, sample_rate)

    # 删除原来的多个文件
    delete_files(file_pattern)

    # 返回合并后的文件路径
    return output_filename


def delete_files(file_pattern="save_zero_shot_*.wav"):
    # 获取所有匹配的文件
    files = glob.glob(file_pattern)

    if not files:
        print("没有找到匹配的文件,无需删除。")
        return

    # 遍历并删除每个文件
    for file in files:
        try:
            os.remove(file)
            print(f"已删除文件: {file}")
        except Exception as e:
            print(f"删除文件 {file} 时出错: {e}")

        # 这里我采用国产的CosyVoice模型来进行文本转语音的操作
        # 一定要记得在启动TTS服务器之前,设置路径 export PYTHONPATH=third_party/Matcha-
TTS

import os
import torchaudio

def check_audio_file_validity(audio_file_path):
    """
    检查音频文件是否正常。

    :param audio_file_path: 音频文件的路径
    :return: 如果文件正常返回 True,否则返回 False 并打印错误信息
    """
    # 检查文件是否存在
    if not os.path.exists(audio_file_path):
        print(f"错误: 文件不存在: {audio_file_path}")
        return False

    # 检查文件大小是否为零
    file_size = os.path.getsize(audio_file_path)
    if file_size == 0:
        print(f"错误: 文件大小为零: {audio_file_path}")
        return False

    # 尝试获取文件信息
```

```
    try:
        info = torchaudio.info(audio_file_path)
        print(f"文件信息: {info}")
    except RuntimeError as e:
        print(f"错误: 无法读取文件信息: {e}")
        return False

    # 尝试加载文件
    try:
        waveform, sample_rate = torchaudio.load(audio_file_path)
    except Exception as e:
        print(f"错误: 无法加载音频文件: {e}")
        return False

    # 检查波形数据是否为空
    if waveform.numel() == 0:  # numel() 返回张量中的元素总数
        print(f"错误: 波形数据为空: {audio_file_path}")
        return False

    # 检查采样率是否合理
    if sample_rate <= 0:
        print(f"错误: 采样率不合法: {sample_rate}")
        return False

    # 如果所有检查都通过,返回 True
    print(f"文件正常: {audio_file_path}")
    return True

cosyvoice = CosyVoice("third_party/Matcha-TTS/pretrained_models/CosyVoice-300M")
prompt_speech_16k = load_wav(
    "zero_shot_prompt.wav",
    16000,
)  # 定义一个声纹文件,让生成的语音模拟这个声纹


class TextInput(BaseModel):
    text: str


@app.post("/generate_audio/")
def generate_audio(text_input: TextInput):
    try:
        print("输入的文本是", text_input)

        # 执行生成音频的具体过程

        for i, j in enumerate(
            # 参数2 和 参数3的作用是,为生成语音提供一个示例,两者是映射关系,参数2是文本,参数3是语音,提高生成语音的准确性
            cosyvoice.inference_zero_shot(
                text_input.text,  # 即将转化的语音
```

```
                "希望你以后能够做的比我还好哟。",  # 语音文件提示词,这是为最终生成的语音提供一个参照
                prompt_speech_16k,  # 语音文件
                stream=False,
            )
        ):
            torchaudio.save("save_zero_shot_{}.wav".format(i), j["tts_speech"], 22050)

        # 将所有的声音文件组合成一个
        final_result_file = combine_audio_files()

        # 检查合成之后的文件是否正常
        is_valid = check_audio_file_validity(final_result_file)
        print(f"文件是否正常: {is_valid}")
        waveform, current_sample_rate = torchaudio.load(final_result_file)

        print("waveform.numel()=",waveform.numel())
        print("waveform.前10个采样=",waveform[0,:10])

        return {"waveform": waveform}

    except Exception as e:
        raise HTTPException(status_code=500, detail=str(e))


if __name__ == "__main__":
    uvicorn.run(app, host="0.0.0.0", port=8030, reload=False)
```

启动这个服务的方式为(注意设置PYTHONPATH是必须的,不然会报错):

```
export PYTHONPATH=third_party/Matcha-TTS
python tts_service.py
```

TTS服务测试代码 tts_service_test.py

测试代码 `tts_service_test.py` 如下,启动的方式是: `python tts_service_test.py` :

```python
import requests

BASE_URL = "http://127.0.0.1:8030"


def test_generate_audio(text):
    url = f"{BASE_URL}/generate_audio/"
    payload = {"text": text}

    response = requests.post(url, json=payload)

    assert response.status_code == 200
    response_json = response.json()
    print("生成的语音结果成功")


if __name__ == "__main__":
    speech = """
尊敬的各位来宾,大家好!

今天我们聚集在这里,是为了庆祝一个非常特别的时刻。我们的公司终于成功研发出了世界上第一款可以自动写新闻稿的人工智能。是的,您没有听错,这台神奇的机器不仅会写新闻稿,还会唱歌、跳舞,甚至还会讲笑话!

我们公司的工程师们为了这个项目,付出了无数个不眠之夜。他们在实验室里熬了整整一个月,每天只靠咖啡和薯片维持生命。哦,对了,我们的首席工程师还发明了一种新口味的薯片,叫做“焦虑辣味”,非常受欢迎。

当然,我们这台人工智能也有一些小缺点。比如说,它有时候会把“新闻”写成“新婚”,把“科技”写成“科技鸡”(不知道这是什么鸡),但这都不是问题,毕竟它还是一只雏鸡嘛!

最后,我要感谢所有支持我们的朋友们。没有你们的支持,我们是不可能完成这个伟大项目的。是的,我们做到了,我们真的做到了!谢谢大家!

(鼓掌)

(记者提问:这台人工智能有没有什么特别的功能?)

(发言人回答:它会自动回复您“稍等,我正在忙着吃薯片”。)

谢谢大家!

(再次鼓掌)
"""
    test_generate_audio(speech)
```

就上面这种长文本的测试,我们会发现,TTS模型会给生成多段 wav文件,然后组装成一个最终的wav,上面的TTS服务,会将这个最终的wav加载成数据结构,返回给调用者,调用者可以将这段数据放到

`gr.Audio`中加以播放。

使用FastApi创建STT服务

STT服务,即将 声音转化成文本,也需要调用 STT模型。

依然采用国产STT。

服务端

`stt_service.py`

```python
from fastapi import FastAPI, UploadFile
from fastapi.responses import JSONResponse
from funasr import AutoModel
from funasr.utils.postprocess_utils import rich_transcription_postprocess
import os

app = FastAPI()

model_dir = "iic/SenseVoiceSmall"

model = AutoModel(
    model=model_dir,
    trust_remote_code=True,
    remote_code="./model.py",
    vad_model="fsmn-vad",
    vad_kwargs={"max_single_segment_time": 30000},
    device="cuda:0",
)

@app.post("/stt/transcribe/")
async def transcribe(file: UploadFile):
    # 读取上传的文件内容
    audio_bytes = await file.read()

    temp_file_name = "temp_audio.wav"

    # 将二进制数据保存为临时文件
    with open(temp_file_name, "wb") as f:
        f.write(audio_bytes)

    # 使用 funasr 进行语音识别
    res = model.generate(
        input=temp_file_name,
        cache={},
        language="auto",  # "zn", "en", "yue", "ja", "ko", "nospeech"
        use_itn=True,
        batch_size_s=60,
        merge_vad=True,
        merge_length_s=15,
    )

    # 进行后处理
    text = rich_transcription_postprocess(res[0]["text"])

    # 删除临时文件
    os.remove("temp_audio.wav")

    return JSONResponse(content={"text": text})

if __name__ == "__main__":
```

```
    import uvicorn
    uvicorn.run(app, host="0.0.0.0", port=8040, reload=False)
```

测试代码

上面介绍国产STT模型的时候用了 单元测试形式,本次改用 gradio构建的UI形式:

```
import gradio as gr
import requests

def transcribe_audio(audio_path):
    # 读取音频文件并发送到 FastAPI
    with open(audio_path, 'rb') as f:
        audio_bytes = f.read()

    response = requests.post(
        "http://127.0.0.1:8040/stt/transcribe/",
        files={"file": ("audio.wav", audio_bytes, "audio/wav")}
    )
    return response.json().get("text", "无法获取转写结果")

iface = gr.Interface(
    fn=transcribe_audio,
    inputs=[
        gr.Audio(type="filepath", label="录音或上传音频"),  # 移除 source 参数
    ],
    outputs="text",
    title="语音转文字演示",
    description="上传音频文件或使用麦克风录音进行语音转文字转换。"
)

iface.launch()
```

界面显示如下:

左侧是录入音频,或者手动选择音频,在选择之后,点submit,可以调用大模型识别语音,并且在右侧显示识别内容。

使用FastApi创建LLM服务

之前使用LLM服务,都会借助一些大模型部署推理的工具,其实也能不借助这些工具,直接用下载模型到本地,直接运行。

环境安装

这里由于我磁盘空间不够,就不重复创建新环境了（每个环境的相同依赖会并存,从而浪费空间）直接复用的 funsar环境,并且我使用比较小的 qwen2.5-0.5b：

环境配置:

```
conda activate funsar
conda install pytorch==1.10.0 cudatoolkit=11.3 -c pytorch

pip install transformers accelerate
pip install -U huggingface_hub
pip install "fastapi[standard]"

conda install ipykernel
python -m ipykernel install --user --name llm --display-name "Python 3.8 (llm)"

# 下载模型
pip install modelscope
modelscope download --model Qwen/Qwen2.5-0.5B-Instruct --local_dir Qwen/Qwen2.5-0.5B-
Instruct
```

后端服务

```python
from fastapi import FastAPI, Request
from fastapi.responses import JSONResponse
from modelscope import AutoModelForCausalLM, AutoTokenizer
import torch

app = FastAPI()

# 加载模型和分词器
model_name = "Qwen/Qwen2.5-0.5B-Instruct"
model = AutoModelForCausalLM.from_pretrained(
    model_name,
    torch_dtype="auto",
    device_map="auto"
)
tokenizer = AutoTokenizer.from_pretrained(model_name)

@app.post("/llm/generate_text/")
async def generate_text(request: Request):
    data = await request.json()
    prompt = data.get("prompt", "")

    if not prompt:
        return JSONResponse(content={"error": "Prompt is required"}, status_code=400)

    messages = [
        {"role": "system", "content": "You are Qwen, created by Alibaba Cloud. You are
a helpful assistant."},
        {"role": "user", "content": prompt}
    ]

    text = tokenizer.apply_chat_template(
        messages,
        tokenize=False,
        add_generation_prompt=True
    )

    model_inputs = tokenizer([text], return_tensors="pt").to(model.device)

    generated_ids = model.generate(
        **model_inputs,
        max_new_tokens=512
    )

    generated_ids = [
        output_ids[len(input_ids):] for input_ids, output_ids in
zip(model_inputs.input_ids, generated_ids)
    ]

    response = tokenizer.batch_decode(generated_ids, skip_special_tokens=True)[0]
```

```
    return JSONResponse(content={"response": response})

if __name__ == "__main__":
    import uvicorn
    uvicorn.run(app, host="0.0.0.0", port=8050)
```

单元测试

```
from modelscope import AutoModelForCausalLM, AutoTokenizer

model_name = "Qwen/Qwen2.5-0.5B-Instruct"

model = AutoModelForCausalLM.from_pretrained(
    model_name,
    torch_dtype="auto",
    device_map="auto"
)
tokenizer = AutoTokenizer.from_pretrained(model_name)

prompt = "给我科普一下什么叫大模型"

print("你的提问是")
print(prompt)

messages = [
    {"role": "system", "content": "You are Qwen, created by Alibaba Cloud. You are a
helpful assistant."},
    {"role": "user", "content": prompt}
]
text = tokenizer.apply_chat_template(
    messages,
    tokenize=False,
    add_generation_prompt=True
)
model_inputs = tokenizer([text], return_tensors="pt").to(model.device)

generated_ids = model.generate(
    **model_inputs,
    max_new_tokens=512
)
generated_ids = [
    output_ids[len(input_ids):] for input_ids, output_ids in
zip(model_inputs.input_ids, generated_ids)
]

response = tokenizer.batch_decode(generated_ids, skip_special_tokens=True)[0]

print("================Ai的回答是============")
print(response)
```

gradio UI

将上面单元测试结合到 gradio 中，

```python
import gradio as gr
import requests

# FastAPI 服务的地址
FASTAPI_URL = "http://127.0.0.1:8050/llm/generate_text"  # 替换为你的 FastAPI 服务地址


# 定义发送请求的函数
def send_request(prompt):
    """
    通过 HTTP 请求调用 FastAPI 服务生成文本。
    """
    try:
        # 发送 POST 请求
        response = requests.post(
            FASTAPI_URL,
            json={"prompt": prompt},  # 传入 prompt
            timeout=60,  # 设置超时时间为 60 秒
        )

        # 检查响应状态码
        if response.status_code == 200:
            # 获取生成的文本
            result = response.json()["response"]
            return result
        else:
            return (
                f"请求失败，状态码：{response.status_code}\n错误信息：{response.text}"
            )
    except Exception as e:
        return f"请求出错：{str(e)}"


# 构建 Gradio 界面
def create_gradio_interface():
    # 定义 Gradio 输入和输出
    inputs = gr.Textbox(lines=5, placeholder="请输入你的提问...", label="输入 Prompt")
    outputs = gr.Markdown(label="AI 的回答")  # 设置输出为 Markdown 格式

    # 创建 Gradio 界面
    interface = gr.Interface(
        fn=send_request,  # 调用发送请求的函数
        inputs=inputs,  # 输入框
        outputs=outputs,  # 输出框
        title="文生文服务测试",
        description="输入你的问题，AI 将通过 FastAPI 服务为你生成回答。",
        examples=[
            ["给我科普一下什么叫大模型"],
            ["介绍一下人工智能的基本概念"],
            ["深度学习和大模型有什么关系？"],
```

```
        ],
        allow_flagging="never",  # 禁止用户标记
    )

    return interface


# 启动 Gradio 界面
if __name__ == "__main__":
    interface = create_gradio_interface()
    interface.launch(share=True)  # share=True 允许生成公共链接
```

测试界面如下：

实战案例

我没有那么多时间去扣业务细节，所谓模拟面试，最重要的就是 人能够和AI进行嘴对嘴，语音对语音的交流。

所以只要能成功将 语音输入->AI语音回复 这个流程打通，就算成功。

下面列出关键代码：

UI界面 `gradio_模拟面试.py`

```python
import gradio as gr
import requests
import numpy as np
from requests.exceptions import RequestException
from gradio import processing_utils


def transcribe_audio(audio_path):
    # 读取音频文件并发送到 FastAPI
    with open(audio_path, "rb") as f:
        audio_bytes = f.read()

    response = requests.post(
        "http://127.0.0.1:8040/stt/transcribe/",
        files={"file": ("audio.wav", audio_bytes, "audio/wav")},
    )
    return response.json().get("text", "无法获取转写结果")


# 用户说话
def user_speaks(audio_path, history):
    print("执行 user_speaks")
    if not audio_path:
        return None, history

    print("audio_path", audio_path)
    # 我拿到了录音数据，现在我要将她传递给 STT服务，帮我翻译成文字
    user_message = transcribe_audio(audio_path)
    print("语音转化的结果是:", user_message)

    """
    发送按钮的点击事件处理函数
    :param user_message: 用户消息
    :param history: 聊天记录
    :return: 更新后的聊天记录和清空的文本框
    """
    if user_message.strip():
        print("history:", history)
        print("user_message:", user_message)
        """
        将用户的消息添加到历史记录中，显示用户头像
        :param history: 聊天记录
        :param user_message: 用户消息
        :return: 更新后的聊天记录
        """
        formatted_user_message = f"{user_message}"
        formatted_ai_response = f"面试官回复中..."
        history.append((formatted_user_message, formatted_ai_response))
        return None, history
    else:
```

```
        return None, history


# 定义发送请求的函数
def send_llm_request(prompt):
    """
    通过 HTTP 请求调用 FastAPI 服务生成文本。
    """
    try:
        # 发送 POST 请求
        response = requests.post(
            "http://127.0.0.1:8050/llm/generate_text",
            json={"prompt": prompt},  # 传入 prompt
            timeout=60,  # 设置超时时间为 60 秒
        )

        # 检查响应状态码
        if response.status_code == 200:
            # 获取生成的文本
            result = response.json()["response"]
            return result
        else:
            return (
                f"请求失败，状态码：{response.status_code}\n错误信息：{response.text}"
            )
    except Exception as e:
        return f"请求出错：{str(e)}"


def llm_answer(chat_history):
    question = chat_history[-1][0]  # 倒数第一对记录，取0位置，也就是用户发送的内容
    chat_history[-1][1] = ""  # 先把AI的回复清空
    llm_result = send_llm_request(question)
    print("面试官的回复是->", llm_result)
    chat_history[-1][1] = llm_result
    return chat_history


def play_audio(chat_history):
    text = chat_history[-1][1]
    url = "http://127.0.0.1:8030/tts/generate_audio/"
    payload = {"text": text}
    print("payload->",payload)
    response = requests.post(url, json=payload,timeout=60,)  # 设置超时时间
    print("response.status_code->",response.status_code)
    if response.status_code == 200:
        if response.content:
            return response.content  # 直接返回音频数据
        else:
            return None
    else:
```

```
        return None


with gr.Blocks() as demo:

    # 聊天对话框
    chatbot = gr.Chatbot(
        elem_id="chatbot",
        show_copy_all_button=False,
        show_copy_button=True,
        label="聊天机器人BOT",
        show_share_button=False,
    )

    with gr.Row():
        # 语音录入
        microphone = gr.Audio(type="filepath", sources="microphone", label="面试候选
人")
        # 语音录入
        interviewer_microphone = gr.Audio(label="面试官",autoplay=True)

    microphone.change(
        user_speaks,
        inputs=[microphone, chatbot],
        outputs=[microphone, chatbot],
    ).then(
        fn=llm_answer,
        inputs=[chatbot],
        outputs=[chatbot],
    ).then(
        fn=play_audio,
        inputs=[chatbot],
        outputs=[
            interviewer_microphone,
        ],
    )

demo.launch()
```

初始界面如下：

上方为ChatBot聊天框，下方左侧为面试候选人的语音录入， 工作流程如下：

- 面试候选人录入语音
- STT服务将语音转化成文字
- 将转化后的文字传给LLM 并生成 文本回复
- TTS服务将文本生成语音，并返回给界面
- 界面下方右侧的面试官Audio组件加载语音流之后自动播放

启动界面的方式为：

```
python gradio_模拟面试.py
```

tts服务 `tts_service.py`

TTS服务的详情参照上面的章节，这里只列出做项目代码：

```python
import os
from pydantic import BaseModel
from typing import Union
from fastapi import FastAPI, HTTPException
from fastapi.responses import FileResponse, JSONResponse

from cosyvoice.cli.cosyvoice import CosyVoice
from cosyvoice.utils.file_utils import load_wav
import io
import torchaudio
from scipy.io.wavfile import read as wav_read

import glob
import torch
import uvicorn

"""
这是一个用fastApi构建的 文字转语音的服务
"""
app = FastAPI()


def combine_audio_files(
    file_pattern="save_zero_shot_*.wav",
    output_filename="combined_audio.wav",
):
    # 获取所有匹配的文件
    files = sorted(glob.glob(file_pattern))

    if not files:
        raise FileNotFoundError("没有找到匹配的文件")

    # 初始化一个空的列表来存储音频数据
    audio_segments = []
    sample_rate = None

    for file in files:
        # 加载音频文件
        waveform, current_sample_rate = torchaudio.load(file)

        # 确保所有音频文件的采样率一致
        if sample_rate is None:
            sample_rate = current_sample_rate
        elif sample_rate != current_sample_rate:
            raise ValueError("所有音频文件的采样率必须一致")

        # 将音频数据添加到列表中
        audio_segments.append(waveform)

    # 拼接所有音频片段
```

```
    combined_waveform = torch.cat(audio_segments, dim=1)

    # 保存拼接后的音频数据到指定文件
    torchaudio.save(output_filename, combined_waveform, sample_rate)

    # 删除原来的多个文件
    delete_files(file_pattern)

    # 返回合并后的文件路径
    return output_filename # 如何得到这个文件的绝对路径


def delete_files(file_pattern="save_zero_shot_*.wav"):
    # 获取所有匹配的文件
    files = glob.glob(file_pattern)

    if not files:
        print("没有找到匹配的文件，无需删除。")
        return

    # 遍历并删除每个文件
    for file in files:
        try:
            os.remove(file)
            print(f"已删除文件: {file}")
        except Exception as e:
            print(f"删除文件 {file} 时出错: {e}")

        # 这里我采用国产的CosyVoice模型来进行文本转语音的操作
        # 一定要记得在启动TTS服务器之前，设置路径 export PYTHONPATH=third_party/Matcha-
TTS


import os
import torchaudio


def check_audio_file_validity(audio_file_path):
    """
    检查音频文件是否正常。

    :param audio_file_path: 音频文件的路径
    :return: 如果文件正常返回 True，否则返回 False 并打印错误信息
    """
    # 检查文件是否存在
    if not os.path.exists(audio_file_path):
        print(f"错误: 文件不存在: {audio_file_path}")
        return False

    # 检查文件大小是否为零
    file_size = os.path.getsize(audio_file_path)
```

```
    if file_size == 0:
        print(f"错误: 文件大小为零: {audio_file_path}")
        return False

    # 尝试获取文件信息
    try:
        info = torchaudio.info(audio_file_path)
        print(f"文件信息: {info}")
    except RuntimeError as e:
        print(f"错误: 无法读取文件信息: {e}")
        return False

    # 尝试加载文件
    try:
        waveform, sample_rate = torchaudio.load(audio_file_path)
    except Exception as e:
        print(f"错误: 无法加载音频文件: {e}")
        return False

    # 检查波形数据是否为空
    if waveform.numel() == 0:  # numel() 返回张量中的元素总数
        print(f"错误: 波形数据为空: {audio_file_path}")
        return False

    # 检查采样率是否合理
    if sample_rate <= 0:
        print(f"错误: 采样率不合法: {sample_rate}")
        return False

    # 如果所有检查都通过，返回 True
    print(f"文件正常: {audio_file_path}")
    return True


cosyvoice = CosyVoice("third_party/Matcha-TTS/pretrained_models/CosyVoice-300M")
prompt_speech_16k = load_wav(
    "zero_shot_prompt.wav",
    16000,
)  # 定义一个声纹文件，让生成的语音模拟这个声纹


class TextInput(BaseModel):
    text: str


@app.post("/tts/generate_audio/")
def generate_audio(text_input: TextInput):
    try:
        print("输入的文本是", text_input)

        # 执行生成音频的具体过程
```

```
        for i, j in enumerate(
            # 参数2 和 参数3的作用是，为生成语音提供一个示例，两者是映射关系，参数2是文本，参数3是语音，提高生成语音的准确性
            cosyvoice.inference_zero_shot(
                text_input.text,  # 即将转化的语音
                "希望你以后能够做的比我还好哟。",  # 语音文件提示词，这是为最终生成的语音提供一个参照
                prompt_speech_16k,  # 语音文件
                stream=False,
            )
        ):
            torchaudio.save("save_zero_shot_{}.wav".format(i), j["tts_speech"], 22050)

        final_file_name = "combined_audio.wav"
        # 将所有的声音文件组合成一个
        final_result_file = combine_audio_files(output_filename=final_file_name)

        # 找到一个正确的方式
        if os.path.exists(final_file_name):
            # 如果文件存在，返回音频文件
            print(f"{final_file_name} 存在,正常返回给前端")
            return FileResponse(final_file_name, media_type="audio/wav")
        else:
             # 如果文件存在，返回音频文件
            print(f"{final_file_name} 不存在,正常返回给前端404")
            # 如果文件不存在，返回一个包含错误码和信息的 JSON 响应
            return JSONResponse(
                status_code=404,
                content={"error_code": 404, "message": "Audio file not found"}
            )

    except Exception as e:
        raise HTTPException(status_code=500, detail=str(e))


if __name__ == "__main__":
    uvicorn.run(app, host="0.0.0.0", port=8030, reload=False)
```

注意这里的 generate_audio 方法，我最终选择了用 FileResponse 的方式将 音频流返回给外界，这是因为 上面章节中的waveform的方式我没有试验成功。

启动TTS服务的方式为：

```
source activate cosyvoice
export PYTHONPATH=third_party/Matcha-TTS
uvicorn tts_service:app --reload --port 8030
```

LLM服务 `llm_service.py`

```python
from fastapi import FastAPI, Request
from fastapi.responses import JSONResponse
from modelscope import AutoModelForCausalLM, AutoTokenizer
import torch
import json

app = FastAPI()

# 加载模型和分词器
model_name = "Qwen/Qwen2.5-7B-Instruct"
model = AutoModelForCausalLM.from_pretrained(
    model_name,
    torch_dtype="auto",
    device_map="auto"
)
tokenizer = AutoTokenizer.from_pretrained(model_name)

# 初始化历史记录
history = []

@app.post("/llm/generate_text/")
async def generate_text(request: Request):
    global history
    data = await request.json()
    prompt = data.get("prompt", "")

    if not prompt:
        return JSONResponse(content={"error": "Prompt is required"}, status_code=400)

    # 将用户的新提示添加到历史记录中
    history.append({"role": "user", "content": prompt})

     # 构建消息列表，包含历史记录
    messages = [
        {"role": "system", "content": "You are Qwen, created by Alibaba Cloud. You are
a helpful assistant."},
    ] + history

    formatted_json = json.dumps(messages, indent=4, sort_keys=True,
ensure_ascii=False)
    print("history",formatted_json)

    text = tokenizer.apply_chat_template(
        messages,
        tokenize=False,
        add_generation_prompt=True
    )

    model_inputs = tokenizer([text], return_tensors="pt").to(model.device)
```

```
    generated_ids = model.generate(
        **model_inputs,
        max_new_tokens=512
    )

    generated_ids = [
        output_ids[len(input_ids):] for input_ids, output_ids in
zip(model_inputs.input_ids, generated_ids)
    ]

    response = tokenizer.batch_decode(generated_ids, skip_special_tokens=True)[0]

    # 将AI的响应添加到历史记录中
    history.append({"role": "assistant", "content": response})

    return JSONResponse(content={"response": response})

if __name__ == "__main__":
    import uvicorn
    uvicorn.run(app, host="0.0.0.0", port=8050)
```

启动LLM服务的方式为：

```
uvicorn llm_service:app --reload --port 8050
```

STT服务 `stt_service.py`

```python
from fastapi import FastAPI, UploadFile
from fastapi.responses import JSONResponse
from funasr import AutoModel
from funasr.utils.postprocess_utils import rich_transcription_postprocess
import os

app = FastAPI()

model_dir = "iic/SenseVoiceSmall"

model = AutoModel(
    model=model_dir,
    trust_remote_code=True,
    remote_code="./model.py",
    vad_model="fsmn-vad",
    vad_kwargs={"max_single_segment_time": 30000},
    device="cuda:0",
)

@app.post("/stt/transcribe/")
async def transcribe(file: UploadFile):
    # 读取上传的文件内容
    audio_bytes = await file.read()

    temp_file_name = "temp_audio.wav"

    # 将二进制数据保存为临时文件
    with open(temp_file_name, "wb") as f:
        f.write(audio_bytes)

    # 使用 funasr 进行语音识别
    res = model.generate(
        input=temp_file_name,
        cache={},
        language="auto",  # "zn", "en", "yue", "ja", "ko", "nospeech"
        use_itn=True,
        batch_size_s=60,
        merge_vad=True,
        merge_length_s=15,
    )

    # 进行后处理
    text = rich_transcription_postprocess(res[0]["text"])

    # 删除临时文件
    os.remove("temp_audio.wav")

    return JSONResponse(content={"text": text})

if __name__ == "__main__":
```

```
    import uvicorn
    uvicorn.run(app, host="0.0.0.0", port=8040, reload=False)
```

启动STT服务的方式为：

```
uvicorn stt_service:app --reload --port 8040
```

总结

启动上述4个服务之后，可以体验完整的 AI语音沟通交流。

完整代码打包到zip文件。