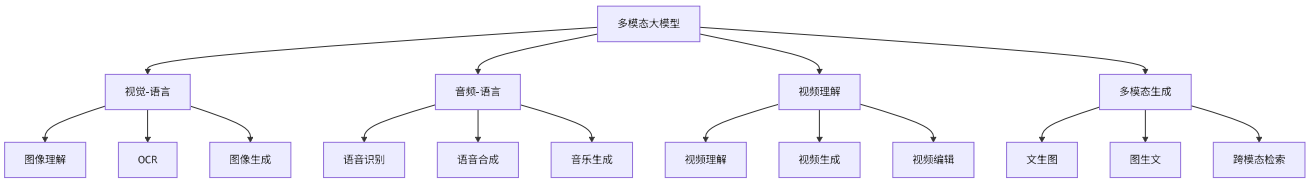
第14章 多模态大模型

> 文字、图像、音频、视频——打破模态边界

14.1 多模态概述

14.1.1 什么是多模态



模态对比：

模态	输入形式	输出形式	典型模型	应用场景
视觉-语言	图像+文字	文字描述	CLIP, BLIP	图像问答、图片搜索
文本-图像	文字	图像	DALL-E, SD	AI绘画、设计
语音-文本	音频	文字	Whisper	语音转写、字幕
文本-语音	文字	音频	VALL-E	语音合成、配音
视频理解	视频	文字/标签	VideoLLaMA	视频问答、审核

14.2 视觉-语言模型

14.2.1 CLIP原理

CLIP (Contrastive Language-Image Pre-training)

```
import torch
import torch.nn as nn
from transformers import CLIPProcessor, CLIPModel

class CLIP:
    """
    CLIP模型：图像-文本对比学习
    """
    def __init__(self):
        self.model = CLIPModel.from_pretrained("openai/clip-vit-base-patch32")
        self.processor = CLIPProcessor.from_pretrained("openai/clip-vit-base-patch32")

    def encode_image(self, image):
        """
        图像编码
        """
```

```
        inputs = self.processor(images=image, return_tensors="pt")
        image_features = self.model.get_image_features(**inputs)

        # 归一化
        image_features = image_features / image_features.norm(dim=-1, keepdim=True)

        return image_features

    def encode_text(self, text):
        """
        文本编码
        """
        inputs = self.processor(text=text, return_tensors="pt", padding=True)
        text_features = self.model.get_text_features(**inputs)

        # 归一化
        text_features = text_features / text_features.norm(dim=-1, keepdim=True)

        return text_features

    def compute_similarity(self, image, texts):
        """
        计算图像与文本的相似度

        Args:
            image: PIL Image
            texts: List[str]

        Returns:
            probs: 每个文本的概率
        """
        # 编码
        image_features = self.encode_image(image)
        text_features = self.encode_text(texts)

        # 计算相似度（余弦相似度）
        logits = (image_features @ text_features.T) * self.model.logit_scale.exp()

        # Softmax得到概率
        probs = logits.softmax(dim=-1)

        return probs


# 使用示例：零样本图像分类
def zero_shot_classification(image, candidate_labels):
    """
    零样本图像分类
    """
```

```
    clip = CLIP()

    # 构造文本prompt
    texts = [f"a photo of a {label}" for label in candidate_labels]

    # 计算相似度
    probs = clip.compute_similarity(image, texts)

    # 返回结果
    results = [
        {"label": label, "score": prob.item()}
        for label, prob in zip(candidate_labels, probs[0])
    ]

    results.sort(key=lambda x: x["score"], reverse=True)

    return results


# 示例
from PIL import Image

image = Image.open("cat.jpg")
labels = ["cat", "dog", "bird", "car"]

results = zero_shot_classification(image, labels)
print(results)
# [{'label': 'cat', 'score': 0.92}, {'label': 'dog', 'score': 0.05}, ...]
```

14.2.2 BLIP-2：图像问答

```
from transformers import Blip2Processor, Blip2ForConditionalGeneration

class ImageQA:
    """
    图像问答系统
    """
    def __init__(self):
        self.processor = Blip2Processor.from_pretrained(
            "Salesforce/blip2-opt-2.7b"
        )
        self.model = Blip2ForConditionalGeneration.from_pretrained(
            "Salesforce/blip2-opt-2.7b"
        )

    def answer_question(self, image, question):
        """
        回答图像相关问题
```

```
        Args:
            image: PIL Image
            question: str

        Returns:
            answer: str
        """
        # 处理输入
        inputs = self.processor(
            images=image,
            text=question,
            return_tensors="pt"
        )

        # 生成答案
        outputs = self.model.generate(**inputs, max_length=50)

        # 解码
        answer = self.processor.decode(outputs[0], skip_special_tokens=True)

        return answer

    def generate_caption(self, image):
        """
        生成图像描述
        """
        inputs = self.processor(images=image, return_tensors="pt")

        outputs = self.model.generate(**inputs, max_length=20)

        caption = self.processor.decode(outputs[0], skip_special_tokens=True)

        return caption


# 使用示例
image_qa = ImageQA()

image = Image.open("scene.jpg")

# 图像描述
caption = image_qa.generate_caption(image)
print(f"Caption: {caption}")

# 问答
questions = [
    "What is in the image?",
    "What color is the car?",
```

```
        "How many people are there?"
    ]

    for q in questions:
        answer = image_qa.answer_question(image, q)
        print(f"Q: {q}")
        print(f"A: {answer}\n")
```

14.2.3 LLaVA：视觉指令微调

```
    class LLaVA:
        """
        LLaVA: Large Language and Vision Assistant

        架构:
        1. Vision Encoder (CLIP ViT)
        2. Projection Layer
        3. Language Model (LLaMA)
        """
        def __init__(self):
            from llava.model import LlavaLlamaForCausalLM

            self.model = LlavaLlamaForCausalLM.from_pretrained(
                "liuhaotian/llava-v1.5-7b"
            )
            self.tokenizer = AutoTokenizer.from_pretrained("liuhaotian/llava-v1.5-7b")
            self.image_processor = CLIPImageProcessor.from_pretrained(
                "openai/clip-vit-large-patch14"
            )

        def chat(self, image, conversation_history):
            """
            多轮对话

            Args:
                image: PIL Image
                conversation_history: List[Dict]
                    [
                        {"role": "user", "content": "What is this?"},
                        {"role": "assistant", "content": "This is a cat."},
                        {"role": "user", "content": "What color is it?"}
                    ]
            """
            # 处理图像
            image_tensor = self.image_processor.preprocess(image, return_tensors='pt')

            # 构建prompt
            prompt = self._build_prompt(conversation_history)
```

```
        # 生成
        with torch.inference_mode():
            output_ids = self.model.generate(
                input_ids=self.tokenizer(prompt, return_tensors="pt").input_ids,
                images=image_tensor,
                max_new_tokens=512,
                use_cache=True
            )

        response = self.tokenizer.decode(output_ids[0], skip_special_tokens=True)

        return response

    def _build_prompt(self, history):
        """
        构建对话prompt
        """
        prompt = "<image>\n"  # 图像占位符

        for turn in history:
            role = turn["role"]
            content = turn["content"]

            if role == "user":
                prompt += f"USER: {content}\n"
            elif role == "assistant":
                prompt += f"ASSISTANT: {content}\n"

        prompt += "ASSISTANT:"

        return prompt


# 使用示例
llava = LLaVA()

image = Image.open("complex_scene.jpg")

conversation = [
    {"role": "user", "content": "Describe this image in detail."}
]

response = llava.chat(image, conversation)
print(response)

# 继续对话
conversation.append({"role": "assistant", "content": response})
conversation.append({"role": "user", "content": "What is the person doing?"})
```

```
response = llava.chat(image, conversation)
print(response)
```

14.3 文本生成图像

14.3.1 Stable Diffusion原理

```
from diffusers import StableDiffusionPipeline
import torch

class TextToImage:
    """
    文本生成图像
    """
    def __init__(self, model_id="stabilityai/stable-diffusion-2-1"):
        self.pipe = StableDiffusionPipeline.from_pretrained(
            model_id,
            torch_dtype=torch.float16
        )
        self.pipe = self.pipe.to("cuda")

    def generate(
        self,
        prompt,
        negative_prompt="",
        num_images=1,
        steps=50,
        guidance_scale=7.5,
        width=512,
        height=512,
        seed=None
    ):
        """
        生成图像

        Args:
            prompt: 正面提示词
            negative_prompt: 负面提示词（不想要的元素）
            num_images: 生成数量
            steps: 推理步数（越多越精细但越慢）
            guidance_scale: 引导系数（越大越符合prompt但可能过拟合）
            width, height: 图像尺寸
            seed: 随机种子
        """
        # 设置随机种子
        if seed is not None:
```

```
            generator = torch.Generator("cuda").manual_seed(seed)
        else:
            generator = None

        # 生成
        images = self.pipe(
            prompt=prompt,
            negative_prompt=negative_prompt,
            num_images_per_prompt=num_images,
            num_inference_steps=steps,
            guidance_scale=guidance_scale,
            width=width,
            height=height,
            generator=generator
        ).images

        return images

    def img2img(self, init_image, prompt, strength=0.75):
        """
        图像到图像（基于已有图像生成）

        Args:
            init_image: 初始图像
            prompt: 提示词
            strength: 变化强度（0-1，越大变化越大）
        """
        from diffusers import StableDiffusionImg2ImgPipeline

        img2img_pipe = StableDiffusionImg2ImgPipeline.from_pretrained(
            "stabilityai/stable-diffusion-2-1",
            torch_dtype=torch.float16
        ).to("cuda")

        images = img2img_pipe(
            prompt=prompt,
            image=init_image,
            strength=strength
        ).images

        return images


# 使用示例
t2i = TextToImage()

# 示例1: 基础生成
prompt = "A beautiful sunset over mountains, oil painting style, highly detailed"
negative_prompt = "ugly, blurry, low quality"
```

```
images = t2i.generate(
    prompt=prompt,
    negative_prompt=negative_prompt,
    num_images=4,
    steps=50,
    guidance_scale=7.5,
    seed=42
)

for i, img in enumerate(images):
    img.save(f"output_{i}.png")


# 示例2: 图像编辑
init_img = Image.open("photo.jpg")
new_prompt = "Same scene but in winter, snowy"

edited_imgs = t2i.img2img(init_img, new_prompt, strength=0.6)
edited_imgs[0].save("edited.png")
```

14.3.2 ControlNet：精确控制

```
from diffusers import StableDiffusionControlNetPipeline, ControlNetModel
from controlnet_aux import OpenposeDetector

class ControlNetGenerator:
    """
    ControlNet: 使用条件图像控制生成

    支持的控制类型:
    - Canny边缘
    - 深度图
    - 人体姿态
    - 法线贴图
    - 分割图
    """
    def __init__(self):
        # 加载ControlNet
        self.controlnet = ControlNetModel.from_pretrained(
            "lllyasviel/sd-controlnet-openpose",
            torch_dtype=torch.float16
        )

        # 加载SD pipeline
        self.pipe = StableDiffusionControlNetPipeline.from_pretrained(
            "runwayml/stable-diffusion-v1-5",
            controlnet=self.controlnet,
```

```
            torch_dtype=torch.float16
        ).to("cuda")

        # 姿态检测器
        self.pose_detector = OpenposeDetector.from_pretrained("lllyasviel/ControlNet")

    def generate_with_pose(self, reference_image, prompt):
        """
        根据参考图像的姿态生成新图像

        Args:
            reference_image: 参考人物姿态的图像
            prompt: 生成提示词
        """
        # 1. 提取姿态
        pose_image = self.pose_detector(reference_image)

        # 2. 使用姿态控制生成
        images = self.pipe(
            prompt=prompt,
            image=pose_image,
            num_inference_steps=20
        ).images

        return images[0], pose_image


# 使用示例
controlnet = ControlNetGenerator()

ref_image = Image.open("person_dancing.jpg")
prompt = "a superhero in the same pose, Marvel style, high quality"

generated, pose = controlnet.generate_with_pose(ref_image, prompt)

# 保存结果
pose.save("pose_map.png")
generated.save("generated.png")
```

14.3.3 Prompt工程技巧

```
class PromptEngineer:
    """
    Prompt工程工具
    """
    def __init__(self):
        self.style_keywords = {
            "realistic": "photorealistic, highly detailed, 8k uhd, dslr",
```

```
            "anime": "anime style, manga, Studio Ghibli",
            "oil_painting": "oil painting, impressionist, brushstrokes",
            "3d_render": "3d render, octane render, unreal engine",
            "watercolor": "watercolor painting, soft colors",
        }

        self.quality_boosters = [
            "masterpiece",
            "best quality",
            "highly detailed",
            "professional",
        ]

        self.negative_common = [
            "ugly", "blurry", "low quality", "distorted",
            "bad anatomy", "extra limbs", "disfigured"
        ]

    def build_prompt(
        self,
        subject,
        style="realistic",
        details=None,
        lighting=None,
        camera=None
    ):
        """
        构建高质量prompt

        Args:
            subject: 主体内容
            style: 风格
            details: 细节描述
            lighting: 光照描述
            camera: 相机视角
        """
        parts = [subject]

        # 添加风格
        if style in self.style_keywords:
            parts.append(self.style_keywords[style])

        # 添加细节
        if details:
            parts.extend(details)

        # 添加光照
        if lighting:
            parts.append(lighting)
```

```
        # 添加相机
        if camera:
            parts.append(camera)

        # 添加质量提升词
        parts.extend(self.quality_boosters[:2])

        prompt = ", ".join(parts)

        return prompt

    def get_negative_prompt(self, additional=None):
        """
        获取负面prompt
        """
        negatives = self.negative_common.copy()

        if additional:
            negatives.extend(additional)

        return ", ".join(negatives)


# 使用示例
engineer = PromptEngineer()

prompt = engineer.build_prompt(
    subject="a cat sitting on a window sill",
    style="realistic",
    details=["fluffy fur", "green eyes"],
    lighting="golden hour lighting, warm tones",
    camera="shot on Canon EOS, shallow depth of field"
)

negative = engineer.get_negative_prompt(additional=["cartoon", "painting"])

print(f"Prompt: {prompt}")
print(f"Negative: {negative}")

# 生成图像
t2i = TextToImage()
images = t2i.generate(prompt=prompt, negative_prompt=negative)
```

14.4 语音处理

14.4.1 Whisper：语音识别

```python
import whisper

class SpeechToText:
    """
    语音转文字
    """
    def __init__(self, model_size="base"):
        """
        Args:
            model_size: tiny, base, small, medium, large
        """
        self.model = whisper.load_model(model_size)

    def transcribe(self, audio_path, language=None, task="transcribe"):
        """
        转录音频

        Args:
            audio_path: 音频文件路径
            language: 语言代码（如 'en', 'zh'）
            task: 'transcribe'（转录）或 'translate'（翻译成英文）

        Returns:
            result: 包含文本、时间戳等信息
        """
        result = self.model.transcribe(
            audio_path,
            language=language,
            task=task,
            verbose=False
        )

        return result

    def transcribe_with_timestamps(self, audio_path):
        """
        带时间戳的转录（用于字幕）
        """
        result = self.transcribe(audio_path)

        # 提取段落和时间戳
        segments = []
        for segment in result["segments"]:
            segments.append({
                "start": segment["start"],
                "end": segment["end"],
                "text": segment["text"]
            })
```

```
        return segments

    def generate_srt(self, audio_path, output_path):
        """
        生成SRT字幕文件
        """
        segments = self.transcribe_with_timestamps(audio_path)

        with open(output_path, 'w', encoding='utf-8') as f:
            for i, seg in enumerate(segments, 1):
                # 序号
                f.write(f"{i}\n")

                # 时间戳
                start = self._format_timestamp(seg["start"])
                end = self._format_timestamp(seg["end"])
                f.write(f"{start} --> {end}\n")

                # 文本
                f.write(f"{seg['text'].strip()}\n\n")

    def _format_timestamp(self, seconds):
        """
        格式化时间戳为 HH:MM:SS,mmm
        """
        hours = int(seconds // 3600)
        minutes = int((seconds % 3600) // 60)
        secs = int(seconds % 60)
        millis = int((seconds % 1) * 1000)

        return f"{hours:02d}:{minutes:02d}:{secs:02d},{millis:03d}"


# 使用示例
stt = SpeechToText(model_size="medium")

# 转录
result = stt.transcribe("audio.mp3", language="zh")
print(result["text"])

# 生成字幕
stt.generate_srt("audio.mp3", "subtitles.srt")

# 翻译成英文
result_en = stt.transcribe("chinese_audio.mp3", task="translate")
print(result_en["text"])
```

14.4.2 语音合成 (TTS)

```
from transformers import VitsModel, AutoTokenizer

class TextToSpeech:
    """
    文字转语音
    """
    def __init__(self):
        self.model = VitsModel.from_pretrained("facebook/mms-tts-eng")
        self.tokenizer = AutoTokenizer.from_pretrained("facebook/mms-tts-eng")

    def synthesize(self, text, output_path="output.wav"):
        """
        合成语音
        """
        inputs = self.tokenizer(text, return_tensors="pt")

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

        # 保存音频
        import scipy.io.wavfile as wavfile

        sample_rate = self.model.config.sampling_rate
        wavfile.write(output_path, rate=sample_rate, data=output.squeeze().numpy())

        return output_path


# 使用示例
tts = TextToSpeech()

text = "Hello, this is a text-to-speech example."
audio_file = tts.synthesize(text, "hello.wav")
print(f"Audio saved to {audio_file}")
```

14.5 视频理解

14.5.1 视频问答

```
class VideoQA:
    """
    视频问答系统
    """
    def __init__(self):
        from transformers import VideoMAEForVideoClassification
```

```
        self.video_model = VideoMAEForVideoClassification.from_pretrained(
            "MCG-NJU/videomae-base"
        )
        self.llm = load_language_model()

    def answer_video_question(self, video_path, question):
        """
        回答视频相关问题
        """
        # 1. 提取视频帧
        frames = self._extract_frames(video_path, num_frames=16)

        # 2. 视频编码
        video_features = self._encode_video(frames)

        # 3. 生成描述
        video_description = self._generate_video_description(video_features)

        # 4. 基于描述回答问题
        answer = self._answer_with_llm(video_description, question)

        return answer

    def _extract_frames(self, video_path, num_frames=16):
        """
        提取视频帧
        """
        import cv2

        cap = cv2.VideoCapture(video_path)
        frame_count = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))

        # 均匀采样
        indices = np.linspace(0, frame_count - 1, num_frames, dtype=int)

        frames = []
        for idx in indices:
            cap.set(cv2.CAP_PROP_POS_FRAMES, idx)
            ret, frame = cap.read()
            if ret:
                frames.append(frame)

        cap.release()

        return frames

    def _generate_video_description(self, features):
        """
```

```
        生成视频描述
        """
        # 使用LLM生成描述
        prompt = f"Describe what is happening in this video based on the visual features
        description = self.llm.generate(prompt, context=features)

        return description
```

14.6 本章小结

本章介绍了多模态大模型的核心技术：

✅ **视觉-语言**：CLIP、BLIP、LLaVA ✅ **文生图**：Stable Diffusion、ControlNet ✅ **语音处理**：Whisper、TTS ✅ **视频理解**：视频问答、内容分析

关键要点：

- 多模态对齐是核心
- 不同模态需要不同处理
- Prompt工程很重要
- 控制生成质量是挑战

完成！ 第五部分（领域应用篇）全部完成。