Day 1

在google搜索Omni model是什么, Nvidia网站中给出的一句话介绍是:

Omni models go beyond image-text understanding to support additional modalities such as audio, video, or a combination of all — text, image, audio, and video in a single unified model.

> https://docs.nvidia.com/nemo/automodel/latest/model-coverage/omni/index.html

Omni属于multi-modality领域. 我立刻想到的是: 这和MLLM的关联与区别是什么(因为MLLM也是涉及处理多个模态的模型)? 因此, 我打算先阅读两方面的综述文章, high-level地明确Omni模型自己的问题定义, 发展路线, 常见评测方式和行业痛点, 以及相比于MLLM来说, Omni的相似和特殊之处在哪里.

对于MLLM的入门综述, 我选择傅老师的A Survey on Multimodal Large Language Models文章; 而MLLM到Omni model的转变, 我选择From Specific-MLLMs to Omni-MLLMs: A Survey on MLLMs Aligned with Multi-modalities这篇文章.

MLLM 综述

A Survey on Multimodal Large Language Models　　精读

> https://github.com/BradyFU/Awesome-Multimodal-Large-Language-Models

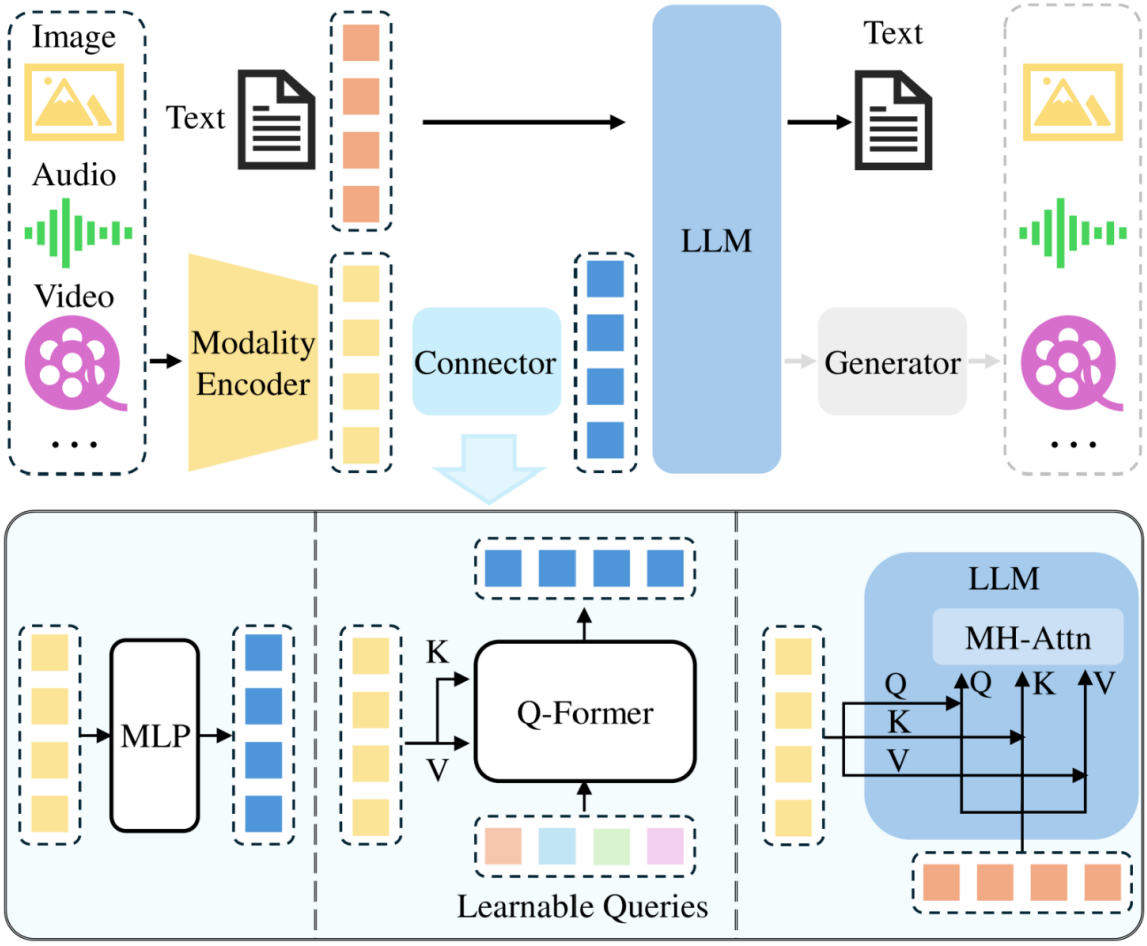
边读边记

Overall

首先，我们阐述了 MLLM 的**基本概念**，并解释了其相关概念，包括**架构、训练策略、数据以及评估方法。**然后，我们介绍了如何**扩展 MLLM** 以支持更细粒度、更多模态、更多语言和更多场景的研究方向。我们继续探讨**多模态幻觉及其扩展技术**，包括多模态 ICL（M-ICL）、多模态 CoT（M-CoT）和 LLM 辅助视觉推理（LAVR）。文章最后，我们讨论了当前面临的**挑战**，并指出了有前景的研究方向

形式上，MLLM 指的是基于 LLM 的模型，它能够接收、推理和输出多模态信息。MLLM之前, 分为判别式和生成式, 分别致力于将视觉和文本信息投影到一个统一的表示空间和以序列到序列的方式统一处理多模态任务。MLLM可以属于后者.

MLLM充分利用LLM强大的推理能力和巨大的参数量优势, 并且有自己的pretrain and post-training范式来进行优化.



典型的MLLM架构有encoder, 包含encoder, connecter, LLM等. 其中, connector处理encode好的其他模态的信息, 让LLM更容易理解. connector分为三种, projection-based, query-based, and fusion-based.

Modality Encoder

一种常见的做法是使用已与其他模态对齐的预训练编码器，而不是从头开始训练, 例如使用在大规模图像-文字训练对其后的CLIP来处理图像. 一些encoder variant, 都是encoder设计上有更多的inductive bias或者是新架构.

值得注意的是，许多研究已通过实验验证，**使用更高的分辨率可以显著提升性能**. 提升输入分辨率的方法可以分为直接缩放法和图像块分割法. 实验证明, 参数大小和训练数据组成的重要性远低于输入分辨率. 类似的编码器也适用于其他模态.

Pretrained LLM

通过在网络语料库上进行大量的预训练，LLM 嵌入了丰富的世界知识，并展现出强大的泛化和推理能力. 值得注意的是，增加 LLM 的参数规模也能带来额外的性能提升，类似于提高输入分辨率的情况. 并且一些LLM中的特殊架构技巧, 如MoE, 能够在不增加计算成本下扩展参数规模, 给LLM带来能力提升, 进而带来MLLM的能力提升.

Modality Interface

就参数规模而言，接口与encoder和 LLM 相比通常只占很小一部分。

由于 LLM 只能感知文本，因此弥合自然语言与其他模态之间的鸿沟至关重要。然而，以端到端的方式训练一个大型多模态模型成本高昂。一种更实用的方法是在预训练的视觉编码器和 LLM 之间引入一个可学习的连接器。另一种方法是借助专家模型将图像翻译成各种语言，然后将翻译后的语言发送给 LLM。

Connector: token-level and feature-level fusion两种. Token-level中, Q-Former利用一组可学习的查询词元以基于查询的方式提取信息, MLP方法则更为简单.

MM1发现对于词元级融合而言，模态适配器的类型远不如视觉词元的数量和输入分辨率重要. 同时, Zeng *et al*. 比较了token-level和feature-level的性能，并通过实验表明，token-level融合在 VQA 基准测试中表现更佳。

而feature level fusion通过插入额外的模块来实现文本特征和视觉特征之间的深度交互和融合. 如Flamingo在frozen transformer block之间有额外的交叉attn layer. 一般新的QKV权重矩阵由预训练的LLM初始化以方便训练.

Expert Model

Modality Interface之外, 专家模型也是一种方式, 直接将多模态输入转化为语言, 比如说使用图像描述模型. 很直接, 但是灵活性不如Modality Interface, 且会导致信息损失.

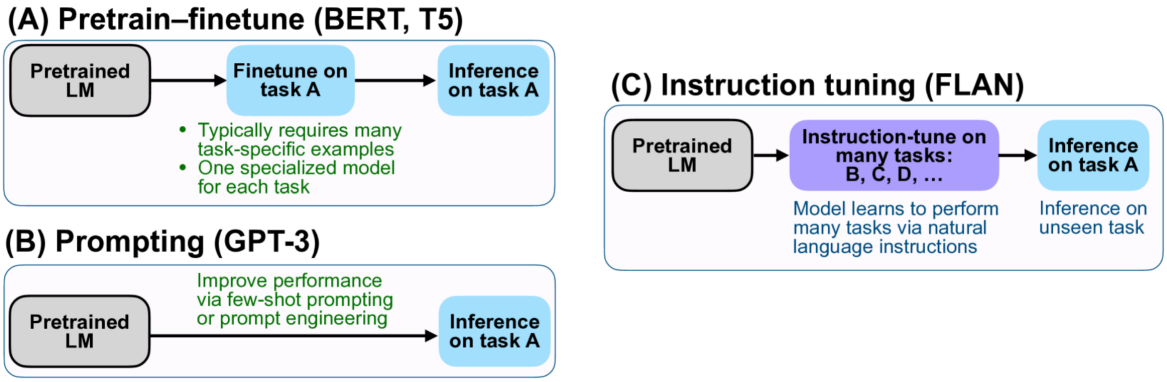
Training

一个完整的MLLM要经历三个训练阶段： 预训练、指令调优和对齐调优。每个训练阶段所需的数据类型不同，目标也各不相同。

预训练的主要目标是协调不同的模态并学习多模态世界知识。预训练阶段通常涉及大规模的文本配对数据， 例如图像描述数据。通常，图像描述对使用自然语言句子来描述图像/音频/视频。如输入image, 要求输出caption, 然后使用交叉熵损失. 并且常见的预训练里面, 预训练的模块就是尽可能freeze的, 比如说encoder and LLM, 训练的只有connector. 当然, freeze的部分并不是固定的, 可以考虑解冻更多的部分. 需要注意的是，训练方案与数据质量密切相关, 对于较短且噪声较大的图像描述数据，可以采用较低的分辨率; 而对于较长且更清晰的图像描述数据，最好使用较高的分辨率. ShareGPT4V发现, 在预训练阶段使用高质量的图像描述数据时，解锁视觉编码可以促进更好的对齐效果

Instruction Tuning

指令调优旨在教会模型更好地理解用户指令并完成所需任务。通过这种方式调优，LLM 可以通过遵循新的指令泛化到未见过的任务，从而提升零样本性能。



指令样本通常包含一条可选指令和一个输入-输出对。指令通常是描述任务的自然语言句子, 输入可以是图像-文本对（VQA)或者是仅包含图像, 输出是根据输入条件对指令的回答. 指令模板template可以灵活设计, 下面是一个文章中的例子; 指令模板也可以推广到多轮对话的情况. 训练目标和范式是teacher forcing的next token prediction.

```
Below is an instruction that describes a task. Write a response
that appropriately completes the request
Instruction: <instruction>
Input: {<image>, <text>}
Response: <output>
```

指令数据的格式更加灵活，任务形式也更加多样化，因此收集数据样本通常更加棘手且成本更高. 有三种方法:

Data Adaptation: 许多研究利用现有的高质量数据集构建指令格式的数据集. 以 VQA 数据集的转换为例，原始样本是一个输入-输出对，其中输入包含一张图像和一个自然语言问题，输出是基于该图像对该问题的文本答案; 任务描述可以源自人工设计，也可以源自 GPT 辅助的半自动生成，甚至一些研究手工构建了一个候选指令池，并在训练过程中从中抽取一个指令作为样本。

需要注意的是，由于现有VQA和图像描述数据集的答案通常较为简洁，直接使用这些数据集进行指令调优可能会限制MLLM的输出长度。目前有两种常见的策略可以解决这个问题。第一种是在指令中明确指定。第二种策略是扩展现有答案的长度, 如通过向 ChatGPT 提供原始问题、答案以及图像的上下文信息（ 如图像描述和 OCR 信息）来重述原始答案。

Self-Instruction: 利用LLM和少量人工标注的样本生成文本指令示范数据, 之后引导 GPT以这些示例为指导生成更多指令样本

Data Mixture: 我们希望MLLM 不只是“看图”，还必须像GPT 一样会聊天。LaVIN 通过从仅包含语言数据和多模态数据中随机采样，直接构建小批量数据, 这样MLLM训练的时候有时只能看到文本, 有时只能看到图片, 本质是用文本数据保持 LLM 的聊天能力, 用多模态数据学习视觉能力。本质上也可以缓解灾难性遗忘聊天问题. 混合策略也别有洞天, 常见的包含: 混合指令调优（将两种类型的数据组合并随机打乱）和顺序指令调优（先训练文本数据，再训练多模态数据）

指令调优样本的数据质量与数据量同样重要. 在大规模但噪声较大的图像-文本对上预训练的模型，其性能不如在规模较小但更干净的数据集上预训练的模型. 指令的多样性和指令的复杂性(任务覆盖率)都很重要, 研究表明越大越好.

Alignment tuning

对齐调优通常用于模型需要与特定人类偏好保持一致的场景, 而且这个技术很大程度上来源于LLM的post training or finetuning(RL). 目前，强化学习与人类反馈（RLHF）和直接偏好优化（DPO）是两种主要的对齐调优技术, 其应用和LLM中二者的应用类似, 在此不过多赘述.

Evaluation

MLLM 的评估展现出几个新的特点：（1）由于 MLLM 通常具有很强的通用性，因此对其进行全面评估至关重要（2）MLLM 展现出许多需要特别关注的新兴能力（ 无需 OCR 的数学推理），因此需要新的评估方案。根据问题类型，MLLM 的评估可以大致分为两类：封闭集评估和开放集评估(close set and open set).

封闭集问题是指答案选项预先定义且仅限于有限集合的问题类型, 通常在特定任务数据集上面做, 且能够很轻松地使用Metric来评判, 如accuracy这种.

开放式问题的回答更加灵活, 因此评估难度更高. 可以人工评分, case study, 或者是LLM as Judge(如GPT). 这个LLM Judge如果能够访问图像, 那就更好了, 评估结果会更加准确而可靠.

Multimodal Hallucination

MLLM产生的反应与图像内容不一致的现象, 那么被认为是幻觉. 目前对多模态幻觉的研究可以进一步分为三种类型:

- Existence Hallucination: 错误地声称图像中存在某些物体
- Attribute Hallucination: 对某些物体的属性描述错误
- Relationship Hallucination: 对物体之间关系的错误描述

目前的方法大致可以分为三类：预校正、过程校正和后校正

Pre-correction: 收集专门的数据，并使用这些数据进行微调

In-process-correction: 探究幻觉产生的原因，并设计相应的补救措施，以在幻觉产生过程中减轻其影响

Post-correction: 以补救后的方式减轻幻觉，并在输出生成后纠正幻觉.

Multimodal In-Context Learning

ICL 的核心在于类比学习, LLM 仅需少量示例以及可选指令即可学习并外推至新的问题，从而以少样本的方式解决复杂且未知的任务. ICL 通常以无需训练的方式实现, 因此可以在推理阶段灵活地集成到不同的框架中. 与 ICL 密切相关的技术是指令调整.

MLLM中信息上下文关联学习（ICL）已扩展到更多模态，从而产生了多模态信息上下文关联学习. 推理时, 可以给样本加演示集, 从而引导MLLM类比思考, 激发推理潜力. 越来越多的研究致力于提升 ICL 在各种场景下的性能.

Multimodal Chain of Thought

CoT 是“一系列中间推理步骤”, 主要思想是促使 LLM 不仅输出最终答案，还要输出得出该答案的推理过程，类似于人类的认知过程

获得 M-CoT 能力的方法大致有三种: 微调, 无需训练的少样本, 零样本学习。这三种方法所需的样本量按降序排列. 微调方法通常涉及为 M-CoT 学习构建特定的数据集. 与微调相比，少样本/零样本学习的计算效率更高, 少样本学习通常需要精心设计一些上下文相关的示例，以便模型能够更容易地学习逐步推理。相比之下，零样本学习不需要任何特定的示例来进行上下文推理学习. 模型无需显式指导，只需提供诸如“让我们逐帧思考”或“这两个关键帧之间发生了什么”之类的预设指令.

Challenges and Future Directions

- 多模态语言模型在处理长上下文的多模态信息方面存在局限性
- M-ICL 和 M-CoT 等技术仍有很大的改进空间
- Trustworthy MLLM, MLLM可能容易受到精心设计的攻击

From MLLM to Omni综述

From Specific-MLLMs to Omni-MLLMs: A Survey on MLLMs Aligned with Multi-modalities 精读

> https://github.com/threegold116/Awesome-Omni-MLLMs

边读边记

Overall

为了应对现实世界场景中的复杂任务, Omni-MLLMs目标是实现全模态**理解**和**生成**, 它突破了特定非语言模态的限制, 将各种非语言模态映射到LLM的嵌入空间. 文章将介绍Omni-MLLMs 的四个核心组成部分，然后介绍两阶段训练, 讨论了相应的数据集和评估方法, 最后总结Omni-MLLM遇到的挑战.

MLLM 通过将LLM与针对特定模态的预训练模型相结合而扩展了 LLM, 但是不足以应对现实世界中同时涉及多种模态的复杂任务. Omni-MLLMs扩展了基于特定模态Specific-MLLMs的理解乃至生成模态. Omni-MLLMs将非语言模态视为不同的外语, 在统一的空间内实现跨不同模态的信息交互和理解. Omni-MLLMs允许单个模型处理任意模态组合理解和生成.

Omni-MLLM在三个方向上不断扩展:

- **处理**的模态类型不断增加
- 跨模态交互能力也得到了扩展, 朝着"任意模态到任意模态"的模式发展
- 应用场景不断拓展

Architecture

Omni-MLLMs需要解决四个问题: 如何编码? 之后如何对齐到LLM空间? 那么在LLM里面, 如何进行交互和推理? 之后如何进行生成? 因此文章认为Omni具有编码、对齐、交互和生成组成的架构，并扩展了所涉及的非语言模态类型

Multi-modalities Encoding

Omni-MLLM 编码方法分为三类：连续编码, 离散编码和混合编码.

连续编码是指将模态编码到连续特征空间中, 这里的Encoder可以是specific的也可以是PreAlign的.

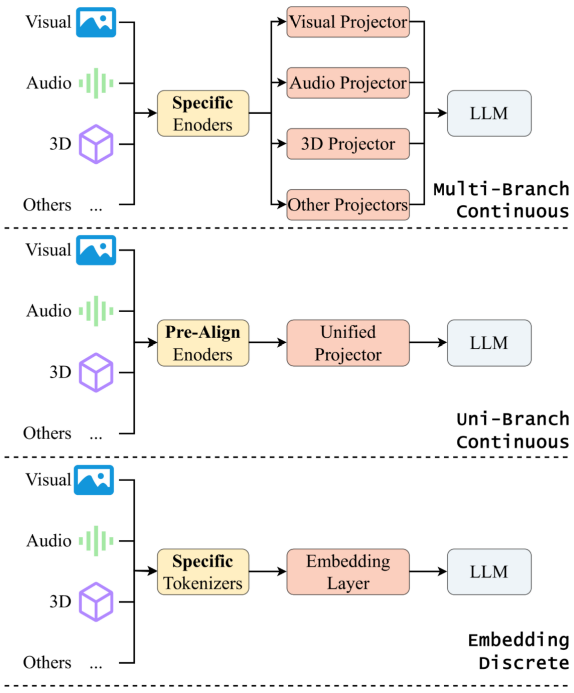
为了更好地促进新的非语言模态的无缝集成和生成, 离散编码将不同的原始模态编码到相同的离散token空间中.

尽管与连续编码相比，离散编码更有利于统一处理不同的非语言模态和文本，但离散模态标记通常难以捕捉原始连续模态中固有的细节信息. 一些Omni-MLLM并非完全离散化，而是结合了两种编码方法，针对不同的模态选择不同的编码方式

Multi-modalities Alignment

Omni-MLLMs 将各种非语言模态的编码特征与LLMs的embedding space进行对齐, 可以分为两种方法：投影对齐和嵌入对齐

在编码器和 LLM 之间插入适配器，称为投影器. 这个概念和上一篇中的Token-Level Connector很相似. 不同模态可以使用不同的projector, 也可以使用Unified projector用于不同的modality. MLP, Q-former是常见的方式; 在不同模态使用不同projector的场景下, 可以考虑不同的projector使用不同的方式.



对于离散编码的Omni-MLLM, 非语言模态的特征被表示为量化码(quantized codes)，这些量化码与文本标记位于同一个离散空间. 新的模态特定离散标记, 通过修改 LLM 的词汇表和相应的嵌入层，嵌入到连续特征空间.

这样一来, 更新后的embedding是对应于不同模态的统一嵌入层, 通常是通过将来自不同模态的离散码本添加到词汇表中，并扩展 LLM 的嵌入层来实现的. (如AnyGPT)

对于混合编码, 自然同时采用投影方法和嵌入方法来实现对齐

Multi-modalities Interaction

为了实现交互，大多数Omni-MLLM, 在输入层将对齐的非语言模态特征与文本特征连接起来，从而以渐进的方式实现交互. 有些方法在特定的层插入非语言模态特征以减轻模态信息的丢失. Omni-MLLM支持涉及两种以上非语言模态的全模态交互.

Multi-modalities Generation

Omni-MLLMs能够输出文本，并通过集成不同的生成模型生成非语言模态. 多模态生成分为三种类型：text-based generation, representation-based generation, and modality-token-based generation.

Text-based: 利用 LLM 的离散文本输出，根据文本内容输入到文本to对应模态的生成模型. 可以是文本出语音或者是图片, 文本起到指定使用模型和描述的作用.

Modality-Token-based: 通过引入来自不同模态分词器的码本，扩展相应的 LLM head, 从而生成特定模态的离散词元; 然后，使用相应的反分词器

Representation-based: 为了减轻离散标记引入的潜在噪声, 离散词汇表依然是扩充的, 但是最终拿到LLM出来的latent embedding, 喂给latent diffusion model.

Omni-MLLM Training

Input Alignment: 在不同模态对齐的训练顺序方面，多分支 Omni-MLLM 对每种模态特定的投影算子进行单独的对齐训练，直接将每种非语言模态与文本对齐，并利用文本作为桥梁来对齐不同的非语言模态; 单分支(Unified)则使用统一的投影算子对不同模态进行对齐，这可能会导致不同模态之间的对齐性能相互干扰. 离散编码则混合来自不同模态的对齐数据并同时进行对齐

Output Alignment: 输出对齐的训练通常使用与输入对齐相同的 X-text 配对数据集，并遵循相同的训练顺序. 同时，输出对齐的训练目标会根据多模态生成方法而有所不同. 基于标记的生成式 Omni-MLLM 通过最小化与特定模态离散标记相关的文本生成损失来优化扩展的 LLM 头; 基于表示的生成式 Omni-MLLM 通常通过最小化包含三个分量的复合损失来优化其输出投影器, 有信号标记的文本生成损失, 输出表示与相应解码器的条件向量之间的 L2 距离, 与条件潜在去噪损失; 基于文本的生成式 Omni-MLLM，一般不需要输出对齐训练

Challenges and Future Directions

- 大多数Omni-MLLM只能处理 2-3 种非语言模态, 通过额外的对齐预训练和指令微调来引入新模态的常用方法会导致显著的训练成本, 并且有灾难性遗忘现象(当扩展新的模态的时候)
- 当输入包含多种序列模态（视频、语音等）时，多模态标记序列的长度可能超过 LLM 的上下文窗口，导致内存溢出; 同时, 由于训练数据量不平衡以及不同模态编码器之间的性能差异, 模型可能倾向于关注某些模态
- 在处理具有时间依赖性的不同模态时，保留它们的时间对齐信息对于后续的跨模态理解至关重要

思考和总结

根据这一篇From MLLM to Omni的综述来看，**Omni Model想处理的场景或问题的formulation可以概括为**：利用LLM的能力，对于多种模态的数据进行理解和生成，或者换而言之：Any Modality -> Any Modality。

在多模态的理解上，Omni有很多方法和MLLM中的路线一致，但是因为要兼顾到最终多模态的生成，因此还发展出了离散token化，从而使得输入输出的token表得以统一。在理解语义和LLM空间的对齐上，离散token扩展LLM vocab使得不同模态在Omni视角下更为统一，而对于连续token来说，Q-Former and MLP和MLLM中的一致。在跨模态推理上，离散token化的巨大优势此时体现出来了：LLM中的attention天生跨模态，因为token space是unfied的。最后generation——也是使Omni和MLLM强调的不同之处——的三种方式，也是让我终于了解了Omni是如何利用LLM的扩模态推理结果来进行生成的。Omni的pretrain和post training和MLLM类似，但是由于涉及到的模态数量和不同的架构选择，相关的训练会更为的困难，也需要更精细的设计。总的来说，我了解到了Omni模型的overall pipeline和architecture taxnomy，也对其预训练和后训练有了初步的概念，最重要的是，我终于对Omni如何进行生成有了初步的感知。

Omni来源于MLLM，但是又超脱于MLLM：

- MLLM在理解层面支持多模态，但是在生成方面并没有过度关注其他模态的生成。经典的MLLM任务，如VQA，caption等，都在生成端未涉及到其他模态内容的生成。但是Omni的哲学是任意模态理解到任意模态生成。
- MLLM视角中，语言模态仍然是中心，但是Omni中强调多个模态之间的融合，强调平等地位，甚至尝试把token space进行统一，想要做到的是AnyGPT那种效果。
- Omni训练难度很大，要精细地处理modality alignment、cross-modality generation、modality balancing等细节，需要精巧的数据集构造和使用，还有很多LLM领域里面的问题，如Long Context问题，也会被继承过来