VLA基线模块

🔢 为什么要加 log 和负号？

第一步：为什么用 log？

概率 $\pi_\theta(a|s)$ 的范围是 $[0, 1]$

$\pi_\theta(a|s)$	$\log \pi_\theta(a|s)$
0.99（很自信）	-0.01（接近0）
0.7（还行）	-0.36
0.1（不自信）	-2.3
0.01（很不自信）	-4.6（很负！）

规律：概率越高，log 值越接近 0；概率越低，log 值越负。

第二步：为什么加负号？

我们做深度学习习惯**最小化** Loss。

- $\log \pi_\theta(a|s)$ 是负数（因为概率<1）

损失函数自然是越低越好，自然我们要来降低它，加上这个负号即可体现损失属性

这个模型的意思就是给定一个模版（专家模版），测定机器在这个相同状态下和专家保持相同选择，行为动作的概率，log与负，变为损失函数，E更加具有统计学意义，平均值

$$\mathcal{L}_{BC} = -\mathbb{E}_{(s,a)\sim D}[\log \pi_\theta(a|s)]$$

🧩 逐个符号拆解

符号	读法	中文意思	生活比喻
$\mathcal{L}_{BC}$	Loss BC	BC的损失函数	我们要**最小化**的"扣分项"
$\mathbb{E}$	Expectation	期望/平均值	所有样本算一遍，取平均
$(s,a) \sim D$	sampled from D	从数据集D中采样	从专家录像里随便挑一条
s	state	状态	机器人"看到"的东西
a	action	动作	专家在这个状态下做的动作
π_θ	policy	策略网络	我们训练的神经网络
$\pi_\theta(a\|s)$	—	给定状态s，模型认为动作a的概率	
log	log	对数	数学工具，把乘法变加法
$-$	negative	负号	把"最大化"变成"最小化"

论文：OpenVLA (2024)

• 链接：https://arxiv.org/abs/2406.09246
• 代码：https://github.com/openvla/openvla

组件	中文名	擅长什么	生活比喻
DinoV2	自监督视觉模型	看**空间细节**：物体在哪、边缘在哪、形状是什么	像一个**验房师**，专门看房子的结构、角落、裂缝
SigLIP	视觉-语言对齐模型	看**语义含义**：这是茄子、那是碗	像一个**房产经纪**，知道"这是厨房"、"那是卧室"

为什么要两个？

单独用一个不够好：

- 只用 DinoV2：知道"紫色的东西在左边"，但不知道那是茄子
- 只用 SigLIP：知道"这是茄子"，但不知道精确位置在哪

物体信息和位置信息，两个信息合并成一个特征

```
### ③ Llama Tokenizer: 语言指令的"切词器"
```

```
"Put eggplant in bowl"
          ↓
    ┌─────────────┐
    │Llama Tokenizer│
    └─────────────┘
          ↓
 [Put] [egg] [plant] [in] [bowl]
   ↓     ↓     ↓      ↓    ↓
 [152] [893] [1234] [297] [476]  ← 变成数字 ID
```

```
**一句话解释**：把人类的句子切成一个个"词块"（token），然后变成数字

**生活比喻**：
> 就像你发微信，输入法会把你的话拆成一个个字/词来处理
```

你将这个Tokenizer理解为这个词块化更加的合理，不仅仅是前面的将这个文本和图像分割，更是将这个文本和图像变为这个token信息单元

```
                    ↓
              7 个连续的数值:
              [Δx, Δy, Δz, Δroll, Δpitch, Δyaw, Δgripper]
              [0.02, -0.01, 0.05, 0.0, 0.1, -0.05, 1.0]
```

这 7 个维度是什么意思？

维度	符号	中文	含义	例子
1	Δx	x位移	左右移动	+0.02 = 向右移 2cm
2	Δy	y位移	前后移动	-0.01 = 向后移 1cm
3	Δz	z位移	上下移动	+0.05 = 向上移 5cm
4	Δroll	翻滚角	绕 x 轴旋转	手腕"翻转"
5	Δpitch	俯仰角	绕 y 轴旋转	手腕"点头"
6	Δyaw	偏航角	绕 z 轴旋转	手腕"摇头"
7	Δgripper	夹爪	开/合	1.0 = 闭合，0.0 = 张开

好一个解释，这样的话我就能够看懂这个被这个模型处理后的这个输出的特点，什么特点呢，就是这个动作向量，真是非常精彩且具体

让我们跟着图中的例子走一遍：

```
步骤 1：输入

┌──────────────────────────────────┐
│ 图像：桌上有茄子、碗、篮子          │
│ 指令："Put eggplant in bowl"      │
└──────────────────────────────────┘
                ↓

步骤 2：视觉编码

┌──────────────────────────────────┐
│ DinoV2 看到：紫色物体在 (x=0.3, y=0.2) │
│ SigLIP 理解：那是"茄子"，旁边是"碗"   │
│ 合并 → 图像特征向量                 │
└──────────────────────────────────┘
                ↓

步骤 3：投影对齐

┌──────────────────────────────────┐
│ MLP Projector 把图像特征           │
│ "翻译"成 Llama 能理解的格式         │
└──────────────────────────────────┘
```

好一个投影对齐，真是形象的翻译

所以这个Llama本身就能够理解这个语言token对吗，只有图像需要通过这个多层感知机进行投影

对，你这句话抓到核心了——**Llama 本身天生就"懂"它自己的语言 token**；真正需要"翻译/对齐"的，是**图像这边出来的视觉特征。**

我们把它拆成两个通道看就很清楚：

Llama 在 OpenVLA 里="把图像+语言融合后，生成动作 token 的核心 Transformer"，既负责理解/对齐，也负责最终控制输出。

你可以这样理解这个Llama的作用

1.对齐两个模态，融合两个模态，处理信息

2.处理后输出这个动作序列

原来如此，这个动作模块输出的依旧是这个tensor向量，需要进一步解码到这个具体的动作序列

数据清洗（很重要！）

论文发现原始数据有很多问题，做了以下清理：

```
原始 OpenX 数据 → 过滤 → 清洗后的训练数据
                  |
                  ├── 去除全零动作（机器人没动但记录了数据）
                  ├── 去除异常轨迹
                  └── 统一动作空间格式
```

有意思，这帮我更加深刻的理解了这个数据清洗

模型	视觉编码器策略	效果
VLM（视觉-语言模型）	冻结 ❄️	好 ✅
VLA（视觉-语言-动作模型）	微调 🔥	好 ✅

关键问题是：**为什么会有这个差异？**

🎯 核心原因：任务对"视觉精度"的要求不同

VLM 的任务：理解图片、回答问题

```
输入：一张猫的图片
问题："这是什么动物？"
输出："这是一只猫"
```

需要的视觉信息：

- 知道这是"猫"这个**类别** ✅
- 大概知道猫在图片**哪个区域** ✅
- **不需要**知道猫鼻子精确在像素 (234, 567) 的

难怪需要微调，而不冻结，原来是对于这个VLA模型所需要的这个精度不同，这个VLA精度要是这个工业机器臂，旋转移动都需要这个视觉的配合

而前者VLM模型只是需要识别这个物体即可

- 显存大（40GB/80GB）
- 训练吞吐高（特别适合 Transformer/大模型）

如果你告诉我你手上是什么 GPU（比如 3090/4090/4060 或者云上什么卡），我也可以按你的硬件给你换算：**OpenVLA LoRA 大概能不能跑、需要哪些省显存技巧（8bit/4bit、梯度累积、冻结视觉编码器等）。**

真是精彩，减少了这个数据训练量是吗

最终效果：老木匠 + 小学徒 = 既会传统家具，又会日式家具

LoRA 的做法：

```
┌──────────────────────────────┐
│  老木匠 W（冻结，不动）       │
│     +                        │
│  小学徒 ΔW（只训练这个）      │
│     =                        │
│  新技能!                     │
└──────────────────────────────┘
```

所以说这个lora降低这个训练量是挑选这个局部的参数进行这个训练

1）为什么输入和输出是“相乘”关系?

因为线性层（`nn.Linear`）做的事情就是：

$$y = Wx \quad (\text{或 } y = xW^\top)$$

直觉：输出的每一个维度，其实都是“输入各个维度的加权求和”。

举个最小例子：输入是 3 维，输出是 2 维：

$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}, \quad W = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{bmatrix}$$

输出：

$$y_1 = w_{11}x_1 + w_{12}x_2 + w_{13}x_3$$

$$y_2 = w_{21}x_1 + w_{22}x_2 + w_{23}x_3$$

你看到了吗?

“相乘”不是为了炫技，而是为了把一堆加权求和用一↓一的符号表示出来——这就是矩阵乘法的意义。

从矩阵乘法的角度来看，输出的每一个元素都是这个输入元素，通过这个权重矩阵的加权求和

“相乘”不是为了炫技，而是为了把一堆加权求和用一个统一的符号表示出来——这就是矩阵乘法的意义。

看来线性代数，我学的还是不够精深，需要继续将其应用到这个深度学习领域

2）把“书架”比喻对上：为什么 4096×4096?

在大模型里常见 `Linear(4096, 4096)`：

- 输入向量长度：4096（你手里有 4096 本“输入书”）
- 输出向量长度：4096（你要产出 4096 本“输出书”）

权重矩阵 W 的形状是：

$$W \in \mathbb{R}^{4096\times4096}$$

为什么是这个大小?
因为它要为**每一个输出维度**提供一套“怎么从 4096 维输入里加权组合”的规则。

- 一共 4096 个输出
- 每个输出需要 4096 个权重去加权输入

所以总共是 4096×4096 个格子（权重参数）。
这就是“买一整面墙书架”的由来。

3）LoRA 的“中转站”为什么也是相乘?

LoRA 不直接学一个巨大的 ΔW（形状同样是 4096×4096），而是把它拆成两个小矩阵：

$$\Delta W \overset{\downarrow}{=} BA$$

一个大的权重矩阵被分成了两个小的权重矩阵，通过这个，我们通过这个两步，走，乘32，再次乘32

这样看，参数量确实大大降低了，卧槽，当时想到这个方法的人真是一个天才，将这个权重矩阵从4096x4096到4096x32+4096x32，这个总的就是这个参数的数量

天的学习计划

第三层：数学公式

原始的线性层：

$$y = Wx$$

LoRA 改造后：

$$y = Wx + \Delta W \cdot x = Wx + (A \times B) \cdot x$$

其中：

- W：原始权重，shape = $[d_{out}, d_{in}]$，**冻结**
- A：降维矩阵，shape = $[d_{out}, r]$，**训练**
- B：升维矩阵，shape = $[r, d_{in}]$，**训练**
- r：秩，一个很小的数（如 32）

而这分割，或者说替代而成的这个权重矩阵就是这个它所说的这个降维矩阵和这个升维矩阵，真是厉害，而这个公式就是这个思想的数学形式的表达

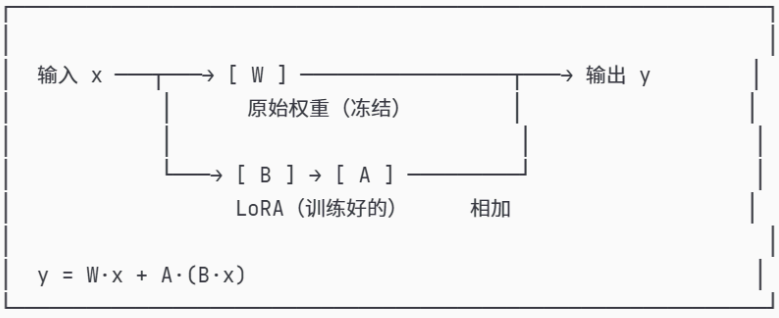
三种使用/保存方式

方式一：分开保存，推理时"叠加"（最常用）

保存的文件：

```
├── base_model/ (原始 OpenVLA, 28GB) ← 大家共用，只下载一次
└── lora_adapter/ (只有 A 和 B，约 400MB) ← 你自己的，很小!
```

推理时动态合并：



优点：

- 原始模型只存一份，省空间
- 可以随时"插拔"不同的 LoRA（换任务）
- 可以同时加载多个 LoRA

Reply...

原来如此，原始的这个权重没有发生这个变化，而我们进行这个验证的时候是通过这2份矩阵的叠加

```
$$W_{new} = W + A \times B$$
```

合并过程：

PyTorch 代码：

```python
```

直接相加，确实可以说是叠加方式，因为这个结果本就是这个加权求和，而这个加权求和是具有这个分配率的

一句话

量化推理 = “把模型里的数字用更少的位数来存和算”，从而省显存、可能更快。

生活比喻：把“高精度尺子”换成“粗一点的刻度”

- bf16 像是 **毫米刻度**：量得更细
- int8 像是 **厘米刻度**：没那么细，但大多时候够用
- int4 像是 **2~3 厘米刻度**：更粗，但如果你只是要把箱子放进门里，不需要毫米级精度，照样能干活

推理时模型做的主要事情是大量的矩阵乘法（linear/attention），**参数权重是一堆数字**。量化就是把这些数字“用更粗的刻度表示”。

真是精彩至极

> **🔗 与我的课题的联系**
>
> **课题：工业机械臂模糊指令语义消歧**
>
> **OpenVLA 能给我什么？**
>
> | 用途 | 具体做法 |
> | --- | --- |
> | **强基线** | 用 OpenVLA 做"无消歧"的基线，对比你的"有消歧"方法 |
> | **动作执行器** | 你的消歧模块输出明确指令后，交给 OpenVLA 执行 |
> | **微调起点** | 在工业场景数据上 LoRA 微调 OpenVLA |

好一个基线，原来这个基线就是没有增加这个消歧情况时候的标准成功率

我们要用这个openvla对比我们有消歧方法的成功概率，没想到这个gpt都给我考虑好了

Step B：离散整数 → tokenizer 里的 token

- 问题来了：Llama tokenizer 默认只留了 **100 个 special token**给你新增，但你这里需要 **256 个动作 token。**
- 论文的做法很"工程直觉"：**直接把 tokenizer 词表里最不常用的 256 个 token（词表最后 256 个）覆盖掉，改成动作 token。** Kim 等 - 2024 - OpenVLA An Open-...

牺牲掉这个次要的，换取更大的效益，好一个工程直觉

3. **现代 PEFT/量化落地**：OpenVLA 论文强调自己**首次在 VLA 上系统验证**参数高效微调（如 LoRA）与量化推理的有效性。 Kim 等 - 2024 - OpenVLA An Open-...
4. **效果对比（在他们实验里）**：他们声称通过"强的开放 VLM + 更丰富的机器人预训练数据"等设计，在实验中**能超过 RT-2-X**，同时模型更小。 Kim 等 - 2024 - OpenVLA An Open-...

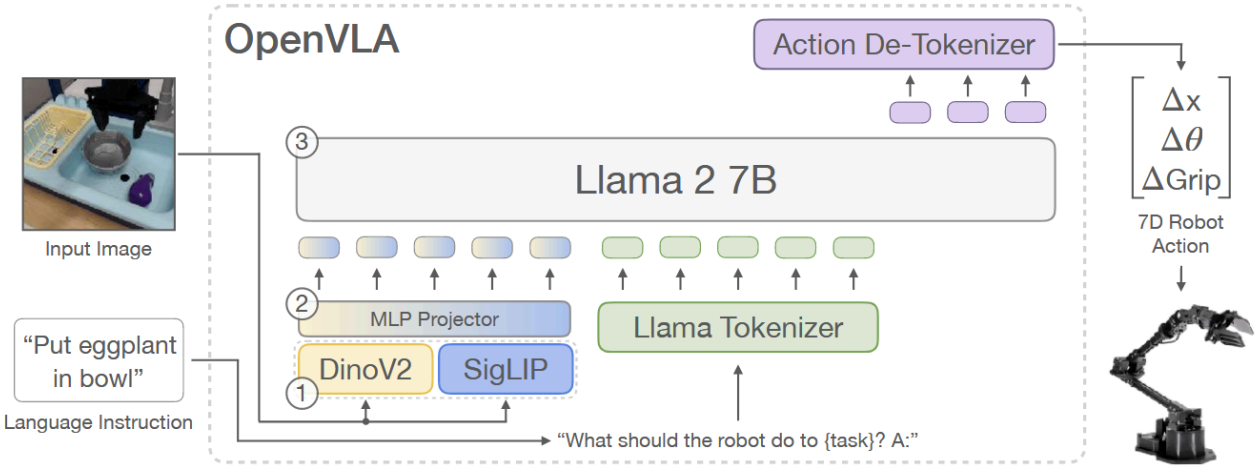
确实如此，我觉得这篇文章里面最有意思的就是这个量化特征int4，和这个高效微调lora

OpenVLA 的三大组件是什么？各自的作用？

OpenVLA 的架构可以理解成：**"看得见（视觉）→ 翻译成 LLM 听得懂的语言 → LLM 输出动作"**，三大组件是：

1. **视觉编码器（Vision Encoder）**：把相机图像提取成特征；OpenVLA把 **DINOv2 的空间细节** + **SigLIP 的语义信息**拼在一起用。 Kim 等 - 2024 - OpenVLA An Open-...
2. **投影器（Projector / MLP Projector）**：把"视觉特征"映射到 **语言模型的词向量空间**，让 LLM 能把图像当成"可读的 token 表示"来处理。 Kim 等 - 2024 - OpenVLA An Open-...
3. **LLM 主干（LLM Backbone）**：用 **Llama 2 7B** 来做"下一 token 预测"，但这里预测的 token 不是词，而是"动作 token"。Kim 等 - 2024 - OpenVLA An Open-...

> 你可能还会看到图里写了 **Action De-Tokenizer**：它的作用就是把 LLM 预测出来的"动作 token 序列"再还原成真实可执行的 **7 维连续控制量**（比如末端位移/旋转/夹爪开合）。Kim 等 - 2024 - OpenVLA An Open-...



这个才是最形象的，论文里面的这个图片