📖 PaLM-E 论文精读笔记

> **论文标题:** PaLM-E: An Embodied Multimodal Language Model
> **作者:** Danny Driess, Fei Xia 等 (Google Robotics / TU Berlin / Google Research)
> **发表时间:** 2023年3月
> **阅读日期:** 2025年
> **所属学习阶段:** 10天VLA学习计划 — Day 9-10（VLM→Policy 范式）
> **阅读目的:** 理解"把连续传感/状态塞进语言模型"的端到端范式，为课题"工业机械臂模糊指令语义消歧"提供架构灵感

一句话总结

PaLM-E 的核心思想是：**把图像、机器人状态等连续信号"翻译"成和文字一样的 token（词元），塞进一个超大的语言模型（PaLM）里，让语言模型同时"看得见、摸得着、说得出"**。它不依赖外部的 affordance（可供性）模型，而是端到端地直接从多模态输入生成机器人的行动计划。

> **生活比喻：** 想象你是一个只会读文字的"超级大脑"（LLM）。以前别人只能用文字告诉你世界是什么样的（"桌上有红色杯子"）。PaLM-E 的做法是：给你装上眼睛（ViT视觉编码器）和触觉（状态向量），但为了让你的"文字大脑"能处理这些新信息，先把图像和触觉翻译成你能理解的"伪文字"，然后插入到句子里一起读。

这篇论文在 VLA 版图中的位置

VLA Models
└── 📋 Task Planner（高层任务规划）
  ├── 端到端：PaLM-E ⬅️ 【你现在读的这篇】
  └── 模块化
    ├── 语言驱动：SayCan ✅（已读）, Inner Monologue ✅（已读）
    └── Grounded（落地）

与之前已读论文的关系：

已读论文	核心思路	与 PaLM-E 的对比
SayCan	LLM 出计划 + affordance 过滤	PaLM-E 不需要单独的 affordance 模型，端到端搞定
Inner Monologue	闭环语言反馈修正计划	PaLM-E 也做闭环（看图→重新规划），但信息更丰富（直接看图而非文字描述）
VIMA	多模态 prompt 驱动控制	PaLM-E 范围更广（多机器人、多任务），但不直接输出低层动作
RT-2	VLM 直接输出动作 token	RT-2 是 PaLM-E 思路的"动作输出版"，PaLM-E 输出的是文字计划

核心架构（必须看懂）

📐 整体思路：多模态句子（Multi-modal Sentences）

PaLM-E 的核心设计非常优雅——**把所有模态的信息都变成 token 序列，和文字混在一起**：

输入示例（多模态句子）：
"Q: How to grasp blue block? Given <img> <emb>"
↑ ↑
图像token 状态token
（ViT编码） （MLP编码）

输出示例：
"A: First, grasp yellow block and place it on the table, then grasp the blue block."

关键公式 — 连续观测的注入（Eq. 3）：

$$x_i = \begin{cases} \gamma(w_i) & \text{如果 } i \text{ 是文本 token} \\ \phi_j(O_j)_i & \text{如果 } i \text{ 对应观测 } O_j \end{cases}$$

逐行解释：

- $\gamma(w_i)$：普通文字通过词嵌入矩阵（embedding matrix）变成向量，这是 LLM 本来就会做的事
- $\phi_j(O_j)_i$：连续观测（图像/状态）通过编码器 ϕ 映射到 **同一个向量空间**
- 关键：两种 token 的维度 k 必须一样，这样 LLM 的 Transformer 就能 **一视同仁**地处理它们

> **哲学内涵：** 这体现了"统一表示"的哲学——与其为每种信息设计专门的处理通道，不如把所有信息翻译成同一种"语言"。就像联合国开会时，不管你说什么语言，都会被翻译成统一的工作语言。这种简洁性往往意味着更强的泛化能力。

📊 四种输入编码器（Section 4 — 重要）

PaLM-E 探索了四种把"外部世界"编码成 token 的方式：

编码器	中文名	输入	输出	适用场景
State MLP	状态向量编码	机器人/物体的位姿、颜色等数值	嵌入向量	状态信息已知时（如工业场景的传感器数据）
ViT	视觉Transformer	原始图像	多个 token（类似图像patch）	通用视觉场景
ViT + Entity Referrals	带实体标注的ViT	图像 + 物体mask	每个物体一组 token	需要区分具体物体时
OSRT	物体场景表示 Transformer	多视角图像	3D感知的物体 token	需要3D几何理解时

⭐ 对你课题最重要的理解：

State MLP（状态向量编码） 是最简单但对工业场景最实用的。工业机械臂有大量传感器数据（夹爪开合角度、力矩、工位编号、工具类型等），这些都可以编码成 token 塞进语言模型：

```python
# 伪代码：状态向量编码
class StateEncoder(nn.Module):
    def __init__(self, state_dim, embed_dim):
        super().__init__()
        # 一个简单的 MLP 就够了
        self.mlp = nn.Sequential(
            nn.Linear(state_dim, 256),
            nn.ReLU(),
            nn.Linear(256, embed_dim)  # embed_dim 要和 LLM 的 token 维度一致
        )

    def forward(self, state):
        # state: [batch, state_dim] 比如 [夹爪角度, 力矩x, 力矩y, 工位ID, ...]
        return self.mlp(state)  # [batch, embed_dim] → 一个"状态token"
```

Entity Referrals（实体标注引用） 是另一个关键设计——在文本中用特殊标记 <obj1>, <obj2> 来**指代**场景中的具体物体，每个标记会被替换成该物体的视觉编码。

> **与你课题的直接联系：** 当用户说"把**那个**螺丝拧到**那里**"时，系统需要把"那个"和"那里"关联到具体的视觉实体。Entity Referrals 提供了一种范式：用特殊 token 在文本中"占位"，然后把视觉信息注入这些位置。你的消歧模块可以借鉴这个思路——先用 Grounding DINO 找到候选实体，再用类似 Entity Referrals 的方式把候选实体的特征注入到语言模型中进行消歧推理。

🔄 如何接入机器人控制回路

PaLM-E **不直接输出低层动作**（这是它和 RT-2 的关键区别），而是输出**文字形式的高层计划**：

输入: Human: bring me the rice chips from the drawer. Robot: I see <img>.
输出: 1. Go to the drawers. 2. Open the top drawer. 3. Pick the green rice chip bag...

然后由**预训练好的低层策略**（如 SayCan 中的技能库）执行每一步。PaLM-E 在控制循环中不断拍照、重新规划：

```
while 任务未完成:
    img = 拍照()
    plan_step = PaLM-E(指令, 历史步骤, img)  # 生成下一步文字指令
    低层策略.执行(plan_step)                 # 用预训练技能执行
    if PaLM-E判断(plan_step, img) == "失败":
        重新规划()
```

关键实验结论（必须记住的 3 个）

1️⃣ 正迁移效应（Positive Transfer）— Figure 3 ⭐⭐⭐

这是本文最重要的发现。

> 把机器人数据、网络图文数据（WebLI）、VQA 数据等混在一起训练，反而让**每个单独任务都变好了**。

具体数据：

- TAMP（任务与运动规划）：混合训练后，用仅 320 个机器人样本就能达到 74.1%（单独训练只有 30.6%）——**性能翻倍还多**
- Language-Table（桌面推物）：混合训练后，10-shot 就能工作
- 移动操作：混合训练显著超过 SayCan（用了 oracle affordance）

> **哲学启示：** 这印证了"博学多才"的智慧——学习看似无关的知识（VQA、图像描述）反而能帮助解决机器人任务。这类似于人类的"通识教育"：学文学的工程师可能比只学工程的更有创造力，因为不同领域的知识会发生正向迁移。

对你课题的启发： 你在做工业机械臂消歧时，不要只用机器人数据训练。混入通用的 VQA 数据、指代表达理解（RefCOCO）数据，可能会让消歧能力更强。

2️⃣ 规模与遗忘（Scale vs. Catastrophic Forgetting）— Figure 6 ⭐⭐

模型规模	多模态训练后语言能力下降（NLG）
PaLM-E-12B（120亿参数）	↓ 87.3%（几乎全忘了）
PaLM-E-84B	↓ 61.6%
PaLM-E-562B（5620亿参数）	↓ 仅 3.9%（几乎没忘）

核心发现：模型越大，学新东西时越不容易忘旧东西。

> **比喻：** 就像一个知识面很窄的人（小模型），学了新技能后原来的知识很容易被覆盖；但一个知识渊博的人（大模型），有足够的"容量"同时装下新旧知识。

3 OSRT 的数据效率 — Table 1 ⭐⭐

OSRT（物体场景表示Transformer，一种 3D 感知的编码方式）即使**不用大规模数据**，在 TAMP 任务上也能达到最好成绩（82.5% / 76.2%）。

含义： 如果你的输入表示（input representation）设计得好（比如包含 3D 几何信息），可以大大减少所需的训练数据量。

> **对工业场景的启发：** 工业场景中机器人的 3D 信息通常是可以获取的（深度相机、CAD 模型等）。好好利用这些结构化信息，可能比暴力堆数据更有效。

必看的图表清单

图表	页码	内容	重要性	看什么
Figure 1	p.1	系统全景图	⭐⭐⭐	理解多模态句子如何同时服务于机器人规划、VQA、图像描述
Figure 2	p.2	562B 模型的涌现能力展示	⭐⭐	零样本多模态推理链、多图推理——感受大模型的"涌现"
Figure 3	p.6	**正迁移效应总结**	⭐⭐⭐	混合训练如何让每个任务都变好
Figure 4	p.7	TAMP 详细消融实验	⭐⭐⭐	冻结 vs 微调 LLM、单任务 vs 混合训练的对比
Figure 5	p.8	一个模型控制两个真实机器人	⭐⭐	感受端到端规划在真实世界的效果
Figure 6	p.8	规模 vs 遗忘曲线	⭐⭐	理解为什么大模型更不容易"忘记"
Figure 7	p.9	Language-Table 真实机器人	⭐⭐	闭环规划 + 对抗干扰 + zero-shot 泛化
Table 1	p.7	TAMP 不同输入表示对比	⭐⭐⭐	理解哪种编码方式在少数据下最好
Table 2	p.7	Language-Table 结果	⭐⭐	混合训练 + 规模的效果

可以暂时跳过的部分

1. **Section 4 中 OSRT 的细节推导**：你只需要知道"它是一种 3D 感知的编码方式，数据效率高"即可，具体的 slot attention 等细节等用到时再看
2. **Appendix A（数据混合比例表）**：了解"机器人数据只占 8.9%"这个比例即可，不需要记住每个数据集的采样频率
3. **562B 模型的具体涌现能力**（Figure 2）：很酷但你无法复现，看个热闹理解"规模的力量"就行
4. **Tab. 7-9 的所有具体数字**：记住趋势（混合训练好、大模型好、OSRT 数据效率高）就够了

与我的课题的深度联系

💡 启发 1：状态信息是消歧的关键辅助

PaLM-E 证明了：**机器人自身的状态信息可以有效地编码进语言模型**。

在工业场景中，消歧不仅靠视觉和语言。比如：

- 当前夹爪里**已经夹着**一个螺丝 → "再拿一个"而不是"拿那个"
- 当前**工位编号**是 3 → "那个"更可能指工位 3 附近的物体
- 当前**力矩**异常 → 可能需要换工具

PaLM-E 的 State MLP 给了一个简单直接的做法：把这些数值全部编码成 token 塞进去。

💡 启发 2：端到端 vs 模块化的权衡

方面	PaLM-E（端到端）	SayCan（模块化）
Grounding 方式	隐式（训练中学到）	显式（affordance 模型）
灵活性	高（一个模型搞定）	高（模块可替换）
可解释性	低（黑箱）	高（知道为什么选这个）
对你课题的适用性	需要大量数据	**更适合**——消歧需要可解释性

我的建议： 对于工业机械臂消歧，**模块化方案更实际**。因为：

1. 工业场景需要**可解释性**——出错了要知道为什么
2. 你的数据量不可能达到 PaLM-E 的量级
3. 消歧本身需要**显式的候选对比**（"左边那个 vs 右边那个"），端到端做不好这件事

但 PaLM-E 的**状态编码思想**和**正迁移思想**完全可以借鉴！

💡 启发 3：Entity Referrals → 消歧占位符

PaLM-E 的 Entity Referrals 机制给了一个关键启发：

> 原始 PaLM-E 用法：
> "<obj1> is to the left of <obj2>"
> ↑ 每个标记被替换成该物体的视觉编码
>
> 你的消歧系统可以这样用：
> "把 <candidate1|螺丝A|置信度0.8> 拧到 <candidate2|孔位B|置信度0.6>"
> ↑ 每个候选实体附带视觉特征和置信度
> → 当置信度接近时触发澄清："你指的是左边还是右边的螺丝？"

💡 启发 4：闭环规划 = 渐进式消歧

PaLM-E 在执行中不断拍照、重新规划。这和"渐进式消歧"（Inner Monologue 的思想）完美结合：

> 用户: "把那个螺丝拧到那里"
> 第1轮: PaLM-E 看图 → 找到3个候选螺丝 → 置信度最高的执行
> 执行中: 拍照发现→ 夹爪接近了错误的螺丝 → 重新规划
> 第2轮: 生成澄清问题 → "你指的是M6还是M8的螺丝？"
> 用户回答后: 确认目标 → 继续执行

关键术语速查

术语	中文	解释
Multi-modal Sentences	多模态句子	把文字、图像、状态混合编排成一个 token 序列
Embodied Language Model	具身语言模型	能理解物理世界的语言模型（有"身体"感知）
Positive Transfer	正迁移	学A任务帮助了B任务（学骑自行车帮助学摩托车）
Catastrophic Forgetting	灾难性遗忘	学新任务时把旧知识忘了（学了法语忘了英语）
Entity Referrals	实体引用标注	在文本中用特殊标记指代具体的视觉物体
OSRT	物体场景表示 Transformer	一种 3D 感知的场景编码方式
TAMP	任务与运动规划	需要同时考虑"做什么"和"怎么做"的规划问题
Frozen LLM	冻结的语言模型	训练时不更新 LLM 的参数，只训练编码器

术语	中文	解释
Input-conditioned Soft-prompting	输入条件的软提示	用连续向量（而非固定文字）来引导 LLM 的行为
Co-training	联合训练	在多种任务/数据上同时训练同一个模型

与 10 天计划其他论文的串联

Day 1-2: Grounding DINO（"找出那个"的候选物体）
↓ 提供候选实体集合
Day 3: Ask-to-Clarify（"问一句"消歧策略）
↓ 决定何时/如何澄清
Day 4-5: OpenVLA（"能跑的强基线"）
↓ 端到端 VLA 的实践经验
Day 7-8: Diffusion Policy（"多模态动作"执行器）
↓ 底层动作生成
Day 9-10: PaLM-E ⬅️ 【你现在在这里】
↓ 提供了"状态编码进 LLM"的范式
↓ 证明了"混合训练正迁移"的可行性
↓ Entity Referrals → 消歧占位符的灵感

你现在的知识图谱已经基本完整了：

- **感知层**：Grounding DINO（找候选）
- **交互层**：Ask-to-Clarify（问澄清）
- **规划层**：SayCan / Inner Monologue / PaLM-E（出计划）
- **执行层**：Diffusion Policy / OpenVLA（出动作）

检验标准 ✅

读完这篇论文后，你应该能回答以下问题：

- [ ] PaLM-E 和 SayCan 的核心区别是什么？（端到端 vs 模块化 + affordance）
- [ ] "多模态句子"的设计思想是什么？为什么要把所有模态统一成 token？
- [ ] 正迁移（Positive Transfer）意味着什么？混合训练为什么能让单个任务变好？
- [ ] 冻结 LLM vs 微调 LLM 各有什么优缺点？
- [ ] 在你的消歧课题中，PaLM-E 的哪些设计可以借鉴？（至少说出 2 点）
- [ ] 为什么 OSRT 数据效率高？这对工业场景有什么启发？

下一步行动

1. **回顾串联**：把 PaLM-E 和之前读的 SayCan、Inner Monologue、RT-2 做一个对比表（已在上文提供），确保理解了端到端 vs 模块化这条主线
2. **思考你的系统架构**：基于 10 天学习的全部论文，初步勾画你的"模糊指令消歧系统"框架图
3. **动手实验**：参考学习计划中的"3 个小而硬的实验"，优先做**候选生成实验**（用 Grounding DINO）

笔记完成时间: 2025年
学习阶段: Day 9-10 / 10天VLA学习计划