OS-Genesis Automating GUI Agent Trajectory Construction via Reverse Task Synthesis

Yang

2025 年 6 月 27 日

1 初读思考

1. OS-Genesis 轨迹生成步骤：

- GUI 代理感知环境并进行逐步交互
- 追溯性探索高质量任务，轨迹探索
- 轨迹奖励模型用于确保轨迹的质量

2. GUI 代理自动执行复杂计算任务所必备的能力，可以通过用 trajuctory 训练来提高：

- 理解用户意图
- 规划执行任务
- 执行动作

3. GUI 的 trajectory 包括：

- high level instruction: 类似于用户给出的总指令，定义完成任务的总目标
- low level instruction: 经过拆解之后完成的小任务
- action: 具体的操作比如点击
- state: 分为 visual(截屏 (screeshot)) 和 textual(用 json 文件等形式格式化地描述状态信息比如窗口大小，颜色等)

4. 传统用于生成 trajectory 的方法：

- human supervision: 需要人工标注 high level task 和注释，高人力成本且劳动耗费大
- model based synthesis: 模型脚本生成，限制多样性，降低质量

5. OS-Genesis 的创新之处：采用反向任务合成技术 (reverse synthesis)，首先通过与环境交互探索可以执行什么样的功能，观察到的状态和动作被转化为 low level instruction。然后将 low level instruction 转化为 high level instruction 从而形成 trajectory。然后引入 reward model 来保证 data quality

6. 传统与 OS 之间的区别总结：

- task-driven: 传统的方法，首先制定任务 (发个邮件)，然后拆解执行，自上向下
- interaction-driven: OS，探索可以执行的操作 (界面上有 file edit save send)，然后组合操作形成 high level instruction，更像是自下向上，探索更多的未知组合

2 OS-Genesis 生成 trajectory 的具体步骤

2.1 interaction-driven functional discovery

首先进行 human-free 探索在 $\epsilon = web, mobile, etc.$ 的环境中，从中探索交互功能得到操作集 $A = \{CLICK, SCROLL, TYPE\}$

这一步骤和人类实际交互步骤类似，只是收集了潜在的操作可能而无需预定义任务

在这个探索过程中 triplet(三元组)$< s_{pre}, a, s_{post} >$ 会被保存，s_{pre} 表示执行操作前的状态 (比如滑动前屏幕截图)，s_{post} 表示执行操作后的状态

2.2 reverse task synthesis

利用 annotation model(注释模型) 生成 low level instruction，具体的操作是将收集到的 triplet 作为输入输入到某个 annotation model(如 GPT)，根据执行操作前后的状态变化，模型会输出一个 instruction。比如当下的操作是向下滑动，那么就可能会生成一个任务：向下滑动菜单展示选项，也就是这个操作可能的应用场景。

将 low level instruction 转化为 high level instruction，利用模型进行操作，结合上下文语意转化为更广泛的任务

最后这些 high level instruction 在环境中被模型 (GPT) 执行以获得 trajectory 并保存起来

2.3 trajectory reward model

因为上述提到的反向合成利用了模型，为了避免模型的不可靠性，引入奖励机制。

传统的筛选方法是引入 labeler function(标签函数)，直接丢弃掉不可靠的 trajectory。但是任何 trajectory 中都包含最基层的交互探索，直接丢弃很浪费，因此引入轨迹特征的评价标准，利用 GPT 进行评分 $R \in [1,5]$，评价标准是 completeness(完整执行任务) 和 coherence(避免冗余操作，效率)

设计了一个算法，首先得到轨迹的集合 G，遍历其中的元素 g_i，取 g_i 的最后三个状态作为 S_{last}，取所有的 low level instruction 作为 L_i，则 $R_i = RM(S_{last}, L_i)$ 得到相应的分数，然后概率化：$P(g_i) = \frac{R_i}{\sum_{j=1}^{n} R_j}$

首先明确模型输入的含义：因为我们想要更高的 completeness 和 coherence，因此我们需要知道它完成了没有，操作是否简洁高效，那么最后三个状态可以看到任务是否完成，而操作的完全集合可以判断简洁性和冗余性，而 reward model 当下可以作为一个黑盒，可以很好地进行评判。

接下来是分数的意义，分数越高的轨迹说明更好，应该作为优质数据，而分数越低的轨迹说明很差，比如只是在胡乱操作。在训练 GUI Agent 的时候可以使用 RL，把 reward score 作为 reward 策略，概率分布越高的采取训练的次数越多

3 实验操作

3.1 背景

选择 Android Control 作为评估标准，Android World 展示 Agent 完成任务的可行性，针对的环境有 mobile 和 web，以上两个是针对 mobile

3.2 Baseline Construction And Training

Zero-shot(零样本训练，即模型还未经过训练直接考试)：利用 CoT 等进行指导提示

Task-Driven：传统的生成轨迹方式

Self-instruction(在没有人为总结的前提下模型对自己的行为进行解释总结，形成任务说明)：利用 GPT 生成

3.3 Evaluation

采用两个指标：SR(success rate 指完成任务的成功率) 和 Type(执行操作种类与预期一致的概率)，GPT 的 Zero-shot 作为对照组，对其余三个模型分别进行上述三种操作的训练得到轨迹，和 OS-Genesis 得到的轨迹训练结果进行对比

简单来说变量可以分为两类，一类是模型比如 Qwen，InternVL，另一类是 trajectory 数据来源 (Task-Driven，Self-Instruction 和 OS-Genesis)，其中 zero-shot 作为空白对照，通过用 AndroidWorld 等工具测试模型执行任务的准确率和冗余性来判断 trajectory 的质量，进而得出 OS-Genesis 的优越性

3.4 Training

采用两种 Training 策略：

- Planning Training: 输入 s, h_i, c 分别表示多模态输入，high level 和历史文本，用 $L_1 = -\sum log(p_\theta(l|s, h_i, c) * p_\theta(a|s, h_i, c))$ 来提高模型的 plan 能力
- Action Training: $L_2 = -\sum log p_\theta(a|l, s, c)$，强化基于 l 的正确操作

值得一提的是这两个形式很像熵，因此可以通过最大化这两个指标得到最佳性能

3.5 Conclusion

从结果来看，OS-Genesis 可以缩小其余模型与 GPT 之间的差距，用不同的工具进行评判，虽然 OS 不一直是最佳，但是表现非常突出，最坏也是第二，平均表现水平第一

4 补充工作

4.1 保证多样性

使用 Sentence-BERT 算法，计算嵌入指令的平均余弦距离，OS 的平均余弦距离最大，因此多样性最高。另一个结论是人为参与能够保证高 intruction diversity，但是 trajectory diversity 很低

4.2 TRM 模型的作用

引入轨迹奖励模型取代传统的标签过滤，再加上什么都不干形成三种方式进行结果对比，发现 TRM 在 high level 上表现出巨大优势，且 low level 表现也很不错

4.3 Scale Trajectories

当 trajectory 变大的时候模型的 SR 也在逐渐增大

4.4 与 human annotation 之间的差距

从两个角度比：high level instruction 和 trajectory，前者 OS 做得更好，原因是有时候模型会会错意且人类注释可能无法在环境中进行；后者人类更优，但是 OS 也能做到 80%

5 总结

本篇论文讲述了关于 OS-Genesis 这种新的产生 trajectory 的方法，与传统的 Task-Driven 相比效率更高，人力资源消耗更少且效果也很不错。主要分为探索交互可能，利用模型 (GPT) 总结 low level 和 high level，引入 reward model 筛选 trajectory，两种 training 策略进行训练，保证了多样性、准确性。