Computer-Use Agent：从 Prompting 到 RL

目录

想回答的问题 2
零、基础 3
一、前史与 Prompting 4
二、基模线：里程碑与 GUI 专门预训练 11
三、Grounding 与感知数据 13
四、Benchmark 演进 16
五、模型谱系 23
六、RL：reward 与环境 26
七、CUA × coding 28
八、对比与小结 28
附录：人与组谱系 28

工作	时间	研究问题	核心方法	核心结论
MiniWoB++	2018.02	能否在网页小任务上用 RL 训 agent	合成网页 + 示范引导探索	前史：LLM 之前的 GUI 环境
SoM → SeeAct	2023.10–2024.01	通用 VLM 能否靠 prompt 操作网页	截图叠编号；GPT-4V 规划	人工 grounding 下 51.1%，瓶颈在 grounding
SeeClick	2024.01	grounding 能否靠专门预训练补上	从网页 HTML 自动造 grounding 数据	提出 ScreenSpot [待核]
OSWorld	2024.04	怎么在真实电脑上评测	真实 VM + 每题一个执行式检查脚本	人类 72.36%，最好模型 12.24%
Claude 3.5 Sonnet CU	2024.10	前沿模型首次公开 computer use	截图 + 坐标动作	OSWorld 截图类 14.9% [待核]
UI-TARS ‖ OpenAI CUA	2025.01	原生 agent 能否胜过框架 + 通用模型	端到端截图 → 动作	OSWorld 24.6（50 步）‖ 38.1 [待核]
UI-TARS-2	2025.09	多轮 RL 的 reward 与环境	三类 reward + 数千 VM	OSWorld 47.5 [待核]
CUA-Gym / Qwen-CUA	2026.05–08	可验证任务与 reward 如何规模化	自动生成任务、状态与奖励函数	[待核]

> 我的复述
>
> **起点**：我听过的 [CUA] paper 什么 Marker 啊，包括什么字节 [Seed] 的，秦宇佳 [秦禹嘉] 做那个什么，PAR 什么 UI [UI-TARS] 之类的。还有比如说那一开始使用手机的，包括 Qwen2.5VL，我至少似乎是 [CUA] 这种视觉能力的预训练基础的来源的一个代表作。

想回答的问题

整理自 2026-11-26 的口述，范围随讨论扩充：

- **A. 演进路径**：这个能力最早怎么 prompt 出来，怎么进入开源，交互方式怎么变。
- **B. 基础能力预训练**：截图数据、HTML 数据各自怎么构建；有哪些 study 类工作。
- **C. Benchmark**：从最早的 benchmark 到 OSWorld 等，怎么串成一条线；与环境搭建强相关。
- **D. RL**：现在有没有 RL；重点是多步、长时序任务的 reward 怎么给，环境怎么构建（并行 VM rollout）。算法不是重点。
- **E. 视觉定位**：像素级定位在 CUA 里是否同样核心（类比 DeepSeek 的 thinking with visual primitives）。
- **F. CUA 与 coding**：两种能力的关系有没有人研究过。
- **横切**：里程碑工作、study 类工作、模型版本三张清单。

约定：闭源模型以 system card / model card 为主要信息源；不展开 3D 空间感知；不强行对照 RUA。

零、基础

符号约定

沿用 OSWorld 的写法 [1]：一个 computer-use 任务是一个 POMDP $(\mathcal{S}, \mathcal{O}, \mathcal{A}, \mathcal{T}, \mathcal{R})$。$s_t \in \mathcal{S}$ 是电脑的真实状态（整个桌面、文件、应用内部数据）；$o_t \in \mathcal{O}$ 是 agent 看到的部分观测（自然语言指令 + 截图 / accessibility tree）；$a_t \in \mathcal{A}$ 是一条可执行动作；$\mathcal{T}$ 是状态转移；$\mathcal{R}$ 是奖励；$T_{\max}$ 是步数上限。

$$o_t \xrightarrow{\text{agent}} a_t \xrightarrow{\mathcal{T}} s_{t+1} \to o_{t+1}, \qquad \text{直到} a_t \in \{\texttt{DONE}, \texttt{FAIL}\} \text{ 或 } t = T_{\max}$$

观测空间：模型「看」什么？

OSWorld 同时提供两种原始观测 [1]：

- **截图**：完整桌面画面，含鼠标位置和各窗口，屏幕分辨率设为 1920×1080。和人看到的一样。
- **Accessibility tree（a11y tree）**：操作系统给辅助功能用的 XML 结构树（Ubuntu 上经 ATSPI，Windows 上经 PyWinAuto），列出可见元素的类型、名字、文本、位置和大小。相当于桌面版的 DOM。

在这两种原始观测上，论文测了四种输入设定：

设定	输入	模型要输出
A11y tree	过滤后的 a11y tree（纯文本）	坐标（可从树里的位置读出）
Screenshot	原始截图	精确像素坐标
Screenshot + a11y tree	两者都给	精确像素坐标
Set-of-Mark (SoM)	用 a11y tree 在截图上画编号框，再附元素文本	元素编号，后处理映射成坐标

关键区别在最后一列：SoM 和 a11y tree 把「找到点哪里」的一部分工作交给了外部结构信息，纯截图设定要模型自己从像素里定位。

OSWorld 的基线结果（GPT-4V 时代，15 步）[1]：

- 纯截图：最强的 GPT-4V 和 Gemini-Pro-Vision 也只有 5.26%–5.80%；
- 只给 a11y tree：最强一批 LLM 为 2.37%–12.24%；
- GPT-4V 用 SoM 反而比「截图 + a11y tree」差。作者推测：桌面任务分辨率高、元素多（比如表格的每个单元格），编号框成了噪声；有些操作要坐标级精度，框表达不了。
- 截图降采样后性能下降：分辨率是瓶颈之一。

作者的判断是：纯截图虽然最差，**长期看应该是最终形态**——它不依赖每个应用都暴露好的结构树。

动作空间：模型「做」什么？

OSWorld 的动作空间是 `pyautogui` 的鼠标键盘函数，加三个特殊动作 [1]：

动作	含义
`click(x, y)`、`moveTo(x, y)`、`dragTo(x, y)`	按屏幕坐标点击、移动、拖拽
`write('text')`、`press('enter')`、`hotkey('ctrl','c')`	打字、按键、组合键
`scroll(±200)`	滚动
`WAIT` / `FAIL` / `DONE`	等待 / 判定任务不可行 / 判定完成

坐标动作覆盖人能做的所有键鼠操作，但要求模型输出准确的像素位置。这就是后面「grounding」一章的起点：SeeAct 发现，GPT-4V 的文字计划如果由人工落到页面元素上，能在真实网站完成 51.1% 的任务，而它自己做 grounding 时差距很大 [2]。

常见的「OSWorld xx%」是怎么算出来的?

1. 只看最后状态。奖励函数 $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to [0, 1]$ 只在最后一步给分：最终状态满足任务目标得 1（部分达成可得小于 1 的分），对不可行任务正确输出 `FAIL` 也得分，其余为 0[1]。

2. 每题一个检查脚本。检查不是比对动作序列，而是去 VM 里读结果：读应用内部文件、取浏览器 cookie、读当前 a11y tree 等。369 个任务共用到 134 个不同的评测函数 [1]。每个任务还有一份初始状态配置，从「做了一半的工作」开始。

3. 分数受步数上限影响。OSWorld 原文的基线用 $T_{\max} = 15$。UI-TARS 在 15 步时 22.7，50 步时 24.6；同表 Claude 是 14.9 和 22.0[3]。同一模型，步数一放宽分数就变。

4. 分数受基准版本影响。 2025-07 的 OSWorld-Verified 修了 300+ 处问题（网页结构变化、指令歧义、评测脚本鲁棒性），并迁到 AWS 并行评测（10+ 小时 → 几分钟）[4]。之后还有 OSWorld 2.0 [待核]。不同版本的分数不能直接比。

> **一句话：**CUA 是一个以「截图 / a11y tree → 键鼠动作」为接口的 POMDP。选什么观测和动作，决定了 grounding 由谁负责；benchmark 分数只看最终状态，而且必须连同步数上限和基准版本一起读。

> 我的复述

> 自测
>
> 1. SoM 让模型输出元素编号而不用输出坐标，听起来更容易。为什么在 OSWorld 上 GPT-4V 用 SoM 反而比「截图 + a11y tree」差?
> 2. 看到「模型 A 在 OSWorld 上 40%，模型 B 45%」，至少要先确认哪两件事才能比较?
> 3. OSWorld 的奖励只在最后一步给。如果拿它直接做 RL，会遇到什么问题?（第六章再展开）

一、前史与 Prompting

这一章回答「能力最早怎么 prompt 出来」。主线是：合成网页上的 RL → LLM 读 HTML 做选择题 → VLM 看带编号的截图 → 发现瓶颈在 grounding。

1. 前史：MiniWoB++ 上的 RL（2018.02）

Reinforcement Learning on Web Interfaces Using Workflow-Guided Exploration（Liu et al. 2018）在 MiniWoB++ 和 World of Bits 这类合成网页小任务上训 RL agent[5]。它点出的问题后来一直没消失：

- **奖励稀疏**：订机票、回邮件这类任务，一步错整条动作序列就作废，agent 很难靠随机探索撞到成功。
- **常见做法是先模仿示范再 RL**，但容易过拟合示范。
- **它的做法**：从每条示范里归纳出高层 workflow（如「第 1 步点文本框；第 2 步输入文字」），只允许探索满足 workflow 的动作。样本效率比行为克隆高 100 倍以上。

稀疏奖励 + 长序列这个难点，第六章 RL 部分会再遇到。

2. LLM 读 HTML：Mind2Web 与 MindAct（2023.06）

Mind2Web（Deng et al. 2023，OSU）是第一个面向「任意真实网站」的通用 web agent 数据集；MindAct 是论文里配套提出的基线方法 [6]。

数据集：每个实例三件套

- **任务描述**：只写高层目标，不给逐步指令。
- **动作序列**：每步是一对（目标元素，操作）。操作只有三种：`Click`（含 Hover、回车）、`Type`、`Select Option`，后两种带一个值。
- **网页快照**：原始 HTML（MHTML）、带布局和截图的 DOM 快照、网络流量（HAR）、完整交互轨迹，可离线回放。

采集流程：5 个一级领域细分成 31 个二级领域，每个领域挑 3–5 个美国热门网站，共 137 个。众包标注员提任务（用 ChatGPT 生成种子任务启发，但不许照抄），再用基于 Playwright 的工具演示——每步先在浏览器里点选元素，再确认操作。作者逐条核验：2,411 个任务保留 2,350 个，其中 390 个改了任务描述、187 个删了多余步骤。

评测是离线、逐步的：每一步都给模型真实的历史动作，单独判这一步对不对 [6]。具体说，第 t 步的输入是标注时录下的第 t 页快照 + 标注员前 $t-1$ 步的动作；模型的预测只拿来和标注比对，**不执行、不改变页面**，第 $t+1$ 步照样用录好的下一页。所以这不是 rollout，而是对每一步做一次分类，再汇总：

$$\text{Task SR} = \frac{1}{|\text{tasks}|} \sum_{\text{task}} \prod_{t=1}^{T} \mathbf{1}\left[\hat{a}_t = a_t^\star \mid s_t^\star,\, a_{<t}^\star\right]$$

网页本身不需要能交互。代价是：模型在某一步错了，不会走偏、也没机会补救；另一条正确路径也无法被认可。

指标	定义
Element Accuracy	选中的元素是否在可接受元素集合里
Operation F1	操作（含输入值）的 token 级 F1
Step SR	元素和操作都对，这一步才算成功
Task SR	所有步都成功，整个任务才算成功

测试集按泛化难度切三份：**Cross-Task**（252 个任务，网站训练时见过）、**Cross-Website**（177 个任务，网站没见过但领域见过）、**Cross-Domain**（912 个任务，Information 和 Service 两个一级领域整体留出）。训练集 1,009 个任务。

MindAct：小 LM 排序 + 大 LM 做选择题

真实网页的原始 HTML 塞不进 LLM。先用启发式只保留可见、有语义的元素，平均元素数从 1,135 降到 580，目标元素仍保留 94.7%。然后分两阶段：

1. **候选生成**：DeBERTa-base（86M）做 cross-encoder。查询 = 任务描述 + 已执行动作；每个元素表示成「标签 + 文本 + 关键属性 + 父子元素文本」。训练时目标元素为正例、随机采负例，sigmoid + 二元交叉熵；推理时给页面所有元素打分，取 top-50。Recall@50 为 88.9% / 85.3% / 85.7%（三个测试集）。
2. **动作预测**：用候选元素及其邻居裁出一段 HTML 交给 LLM。**不让 LLM 生成目标元素，而是做多选题**：每组最多 5 个候选 + 一个 `None` 选项。top-50 分成多组；一轮下来选中多于一个，就把选中的重新分组再选，直到只剩一个或全部选 `None`。操作和输入值由 LLM 直接生成。Flan-T5 用标准语言模型目标微调；GPT-3.5 / GPT-4 用 in-context 示例。

当时的基模与训练

- **没有视觉**：MindAct 只用网页的文本（HTML）。论文发布于 2023-06-09，早于 GPT-4V system card（2023-09-25）[7]。数据集里存了渲染快照，作者把「利用视觉信息」列为未来方向。
- **基模没有专门训过这个能力**：Flan-T5-XL 不微调（零样本）基本做不了这个任务；GPT-3.5 用 3 个示例时元素准确率只有约 20%，常常选 `None`。能力来自在训练集（1,009 个任务）上微调。
- **超参**：DeBERTa-v3-base 候选排序，batch 32、5 epoch、lr 3×10^{-5}，用 Sentence-Transformers 的 cross-encoder 实现；Flan-T5（base / large / XL）动作预测，batch 32、5 epoch、lr 5×10^{-5}，全参微调；GPT-3.5 / GPT-4 温度 0、3 个示例。XL 和 large 在 4×A100 80GB 上训，其余单卡 A6000。
- **离线评测的代价**：一个任务常有多条正确路径，但数据只记录了一条；模型走另一条路时动作不在缓存里，会被误判为失败。作者把同一页面上的等价元素都算作正确，只能部分缓解。

结果（Step SR，Cross-Task / Cross-Website / Cross-Domain）：

方法	Step SR	Task SR（Cross-Task）
生成式基线（Flan-T5-base 直接生成目标元素）	17.5 / 11.0 / 11.9	0.0
MindAct + Flan-T5-XL（微调）	52.0 / 38.9 / 39.6	5.2
MindAct + GPT-4（in-context，每份 50 任务、top-10 候选）	36.2 / 30.1 / 26.4	2.0

两点结论：

- **把元素选择改成多选题，比让模型生成元素好得多**（同为 Flan-T5-base，Step SR 41.0 vs 17.5）。
- **单步能对一半，整题只对 5% 左右**：只要错一步整题就算失败，长序列的错误会累积。

同期的 WebArena（Zhou et al. 2023）搭了四类可交互的自托管网站，按功能正确性检查结果：最好的 GPT-4 agent 端到端成功率 14.41%，人类 78.24%[8]。（细节放第四章。）

插曲：2023 年的全景

引用数取自 Semantic Scholar（2026-11-26 查询），只作影响力的粗略参考。

同期的 benchmark / 数据集（按引用数排）

名称	时间	场景	环境 / 评测	引用
WebArena	2023.07	Web	自托管可交互网站，查最终结果	2613
AgentBench	2023.08	8 个环境的 LLM-as-agent 综合	交互式	1727
Mind2Web	2023.06	真实网站	离线快照，逐步比对	1722
WebShop	2022.07	模拟电商	可交互，按商品属性自动打分	1718
GAIA	2023.11	通用助手问答	只看最终答案	1475
Android in the Wild (AitW)	2023.07	Android	离线演示数据：715k 条轨迹、30k 条指令、8 种设备	429
MiniWoB++	2018.02	合成网页小任务	可交互，环境内置奖励	397

AitW 值得单独记一句：它的动作不是「点某个元素」，而是精确手势（如横向滑动轮播图），必须从视觉外观推断能做什么 [9]。

同期的方法：除了 MindAct，2023 年还有两条线。

- **HTML + 代码**：Google 的 WebAgent（2023.07）把指令拆成子指令，用专门预训练的 HTML-T5 把长 HTML 摘成片段，再让 Flan-U-PaLM **生成 Python 程序**去操作网站 [10]。这是「用代码当动作」的早期例子（第七章）。
- **纯像素**：Google 的 Pix2Act（2023.05）只看截图、只用通用键鼠动作，基于 Pix2Struct 的像素预训练，首次在 MiniWoB++ 上超过众包人类 [11]；WebGUM（2023.05）同时看截图和 HTML，离线训练在 MiniWoB 上超过此前最好的离线方法 45.8%[12]。**纯像素 agent 在 2023 年就有，但只在 MiniWoB 这种简单合成环境里成立**，真实网站仍然靠 HTML。

开源 VLM 基模当时到了什么程度，放在第二章。

3. VLM 看截图：Set-of-Mark（2023.10）

Set-of-Mark Prompting（Yang et al. 2023，Microsoft 等）本身不是 GUI 论文，是一个通用的视觉 prompting 方法 [13]。

动机：GPT-4V 的视觉理解很强，但细粒度定位很弱——让它直接输出一串坐标基本不准。作者认为，让 LLM 在文本里吐坐标数字，既别扭，也用不上视觉编码器本身的空间能力。所以换个方向：**不改模型，改输入图片**，在图上画出「说得出口」的标记。

流程四步

1. **切区域**：用现成的分割模型把图像切成 K 个区域，表示为 K 张二值掩码 $R \in \{0,1\}^{K\times H\times W}$。按需求选工具：MaskDINO（精度高）、SEEM（开放词表）、SAM / Semantic-SAM（可切到物体局部）。
2. **选标记类型**：主要用数字或字母——占地小，GPT-4V 能靠 OCR 认出来；框和掩码边界作为辅助。标记要避开和图像内容冲突：满是数字的算术图不用数字，文档截图不用字母。
3. **放标记位置**：直接放在掩码中心会重叠（两个物体中心挨着、凹形区域中心落在外面）。他们的分配算法是：
   - 按面积从小到大排序，先放小区域；
   - 第 k 个掩码先扣掉前 $k-1$ 个已覆盖的部分；
   - 对剩下的掩码做**距离变换**，取「离边界最远」的点放标记；
   - 区域太小、标记会盖住整个区域时，把标记稍微挪到区域外。
4. **提问**：可以照常提问（不提标记），也可以在文字里直接引用标记（「3 号里的水果能放进 2 号吗?」）。

实验：不训练任何模型。当时没有 GPT-4V API，作者是手动往 ChatGPT 网页里贴图测的，每张图开新对话，所以每个任务只测了约 100 张图的子集。

任务（数据集）	**GPT-4V 直接输出坐标**	**GPT-4V + SoM**	**专用模型参考**
指代理解 REC（RefCOCOg，ACC@0.5）	25.7	86.4	PolyFormer 85.8*、Grounding DINO 86.1*
指代分割 RES（RefCOCOg，mIoU）	—	75.6	PolyFormer 67.2（同子集）
短语定位（Flickr30K，R@1）	—	89.2	Grounding DINO 90.5
通用分割分类（COCO，precision）	—	75.7	MaskDINO 80.7（给 GT 框）

* 为原论文在完整验证集上的数字，与 SoM 的子集结果不完全可比。

两个消融值得记：

- **标记类型**：Flickr30K 上「数字 + 掩码」84.4，再加框 89.2。
- **区域来源决定上限**：RefCOCOg 上用 MaskDINO 切的掩码 75.6，换成真值掩码 90.1。**SoM 的效果被「候选区域切得准不准」卡住**——和 MindAct 被候选召回率卡住是同一个结构。

本质：把 grounding 从「回归坐标」改成「在候选区域里做分类」。模型只需要说出编号，定位精度交给外部的候选生成器。

搬到 GUI 上的变化：GUI 里候选区域一般不用分割模型，而用 DOM、XML 或 a11y tree 里现成的元素框（AppAgent、OSWorld 的 SoM 设定）。但后来两篇评测都发现它在 GUI 上不总是有效：

- SeeAct：SoM 式图像标注比「HTML 文本选项 + 截图」最多低 10 个点 [2]；
- OSWorld：GPT-4V 用 SoM 反而比「截图 + a11y tree」差，因为桌面元素太密（表格的每个单元格都要标号），而且有些操作要坐标级精度，框表达不了 [1]。

4.（附）手机端：AppAgent 与 Mobile-Agent（2023.12–2024.01）

附属小节：两篇本质上是同一批基模能力（GPT-4V、外部检测器）在手机上的应用，不改变主线。

两篇代表了手机端的两种做法：

	AppAgent[14]	**Mobile-Agent**[15]
时间	2023.12	2024.01
观测	截图 + App 的 XML（交互元素列表）	只用截图
定位	按 XML 给每个元素编号，半透明数字叠在截图上	外接视觉感知工具识别、定位屏幕上的文字和图标
动作	`Tap(element:int)`、`Long_press(element:int)`、`Swipe(element, direction, dist)` 等基本函数	按定位结果逐步操作
特点	先自主探索或看人类示范，生成每个 App 的知识库，执行时查阅；测了 10 个 App 的 50 个任务	不依赖 XML 等系统元数据，换系统不用定制；提出 Mobile-Eval

AppAgent 是 SoM 思路在手机上的直接应用；Mobile-Agent 走「纯视觉」，但定位仍交给外部检测工具，不是模型自己做。

5. SeeAct：瓶颈在 grounding（2024.01）

GPT-4V(ision) is a Generalist Web Agent, if Grounded（Zheng et al. 2024）把 web agent 拆成两步：**动作生成**（用文字描述下一步要做什么）和 **grounding**（把这句话落到页面上的具体元素）[2]。

- **如果 grounding 由人工完成**（oracle grounding），GPT-4V 在真实网站上能完成 51.1% 的任务，远高于 GPT-4 文本模型的 13.3% 和 FLAN-T5 的 8.9%。
- 它比较了三种自动 grounding：

策略	**做法**
元素属性	让模型描述目标元素的类型和文字，再在 DOM 里启发式搜索匹配
文本选项	把候选 HTML 元素列成选项，模型结合截图做选择（同时用 HTML 和视觉）
图像标注	在截图上画编号（即 SoM），模型选编号

- **最好的策略（同时用 HTML 和视觉）仍比 oracle 低 20–30 个点**；SoM 式图像标注比它最多再低 10 个点。

结论：GPT-4V 已经会规划，卡在「把计划落到像素 / 元素上」。

SeeAct 的在线评测：在 Mind2Web 离线评测之外，它用 Playwright 做了一个在线评测工具，让 agent 直接在线上真实网站执行动作；只做不需要登录的任务，并人工盯着，防止下单、改资料这类有影响的操作。在三个测试集里抽了 90 个任务（网站改版导致失效的会重抽）。结果是**在线的整任务成功率明显高于离线**——作者的解释正是离线评测只认一条参考路径，会低估成功率 [2]。

和 WebArena 的结论对得上：两边都用 GPT-4 系模型当基线，都发现整任务成功率远低于人类，也都指向「离线逐步比对不够，要在真实可交互环境里看结果」。分歧在归因：WebArena 认为缺的是探索和失败恢复，SeeAct 认为缺的是 grounding。

同月的 WebVoyager（He et al. 2024）在 15 个真实网站上做端到端多模态 agent，任务成功率 59.1%；它还用 GPT-4V 自动判定任务是否完成，与人工判断一致率 85.3%[16]。**用 VLM 当裁判**这个做法，第六章讲 reward 时会再出现。

小结：**Prompting 时代谁负责 grounding?**

阶段	模型看到的	「点哪里」由谁解决	代表
RL 前史	合成网页 DOM	小策略网络 + 示范约束	MiniWoB++
LLM + HTML	过滤后的 HTML 候选	小 LM 排序 + LLM 多选	MindAct
VLM + 编号	截图 + 编号框	外部结构（DOM / XML / a11y）给编号	SoM、AppAgent
VLM + 检测器	截图	外部 OCR / 图标检测	Mobile-Agent
诊断	截图 + HTML	人工 oracle vs 自动策略，差 20–30 点	SeeAct

2023 年的整体格局：

- **观测以文本为主**：真实网站上的 agent 几乎都读 HTML / a11y tree；纯像素 agent（Pix2Act、WebGUM）只在 MiniWoB 这类合成环境里成立。原因是 VLM 还不够：开源模型分辨率低、没见过 GUI；GPT-4V 到 2023-09 才出现，且自己定位很差。
- **动作以「选元素」为主**：元素 ID、候选多选、SoM 编号；直接输出坐标的很少。用代码当动作（WebAgent）是少数。
- **评测以离线为主**：Mind2Web、AitW 都是录好的演示数据，逐步比对。可交互、查最终结果的环境（WebShop、WebArena）是少数，其中 WebArena 是真实网站规模的第一个（见第四章）。

> **一句话**：Prompting 时代，通用模型的规划已经够用；「点哪里」一直靠外部结构（HTML 候选、XML 编号、检测器）兜底。SeeAct 量出了没有兜底时的差距——这就是第三章 grounding 预训练要补的东西。

> 我的复述

> 自测
>
> 1. MindAct 和 AppAgent 都把「点哪里」变成了选择题。两者的候选分别从哪来？

2. SeeAct 的 51.1% 是在什么条件下测出来的？这个数字说明了什么，没说明什么？
3. MiniWoB++ 时代的 RL 卡在哪里？workflow 约束是怎么缓解的？

二、基模线：里程碑与 GUI 专门预训练

CUA 的每一次跳变，背后几乎都是基模的跳变。本章只保留和 CUA 直接相关的两件事：谁是当时 benchmark 的基线，以及专门为 GUI 做的预训练。视觉编码器、分辨率、像素信息是否无损、grounding 的通用做法，拆到了单独的 **VLM 笔记**（网页版，本地 http://127.0.0.1:8790/VLM/）。

1. 基模发布里程碑：谁是当时的基线?

日期取自各家官方发布页；「被谁当基线」只列本笔记已读过的论文，后续章节逐步补。

日期	基模	对 CUA 的意义	当基线 / 骨干的工作
2023-03-14	GPT-4 [17]	最强文本推理；agent 读 HTML / a11y tree	Mind2Web（3 示例）、WebArena（14.41%）
2023-09-25	GPT-4V system card [7]	图像输入，但自己定位很差（RefCOCOg 直接输出坐标 25.7）	SoM（只能在 ChatGPT 网页里手动测）
2023-11-06	DevDay：GPT-4 Turbo with vision 进 API [18]	VLM 可以程序化调用，agent 才能批量跑	SeeAct、VisualWebArena（gpt-4-1106-vision-preview）、OSWorld
2023-12-06	Gemini 1.0 [19]	第二家闭源多模态	VisualWebArena（Gemini-Pro）、OSWorld（Gemini-Pro-Vision）
2024-03-04	Claude 3 [20]	第三家闭源多模态	OSWorld（Claude-3 Opus）
2024-05-13	GPT-4o [21]	OSWorld v2 补测：a11y tree 11.36%、纯截图 5.03%，没有超过 GPT-4 / GPT-4V	OSWorld
2024-06-21	Claude 3.5 Sonnet [22]	—	待补
2024-09-12	o1 [23]	推理模型	待补
2024-10-22	Claude 3.5 Sonnet（new）+ computer use [24]	首个公开 beta 的前沿 computer use；OSWorld 仅截图 14.9%	UI-TARS 的对比基线
2025-01-23	OpenAI CUA / Operator [25]	GPT-4o 视觉 + RL 推理；OSWorld 38.1%（200 步）	UI-TARS-1.5 的对比基线

日期	基模	对 CUA 的意义	当基线 / 骨干的工作
2025-01-26	Qwen2.5-VL [26]	开源、原生分辨率、框和点定位、原生 computer / phone use；自报 OSWorld 8.83、ScreenSpot 87.1、ScreenSpot-Pro 43.6	OpenCUA（7B / 32B / 72B 骨干）
2025-04-16	o3 [27]	—	待补

开源一侧在 2023 年被拿来当基线的：Flan-T5（Mind2Web）、CogVLM / CogAgent、IDEFICS-80B（VisualWebArena）、Mixtral、Llama-3、CogAgent（OSWorld）。在 VisualWebArena 上这些开源 VLM 都在 1% 左右或以下（见第四章）。

读法：2023 年的 CUA 论文几乎都是「新 benchmark + 用 GPT-4 / GPT-4V 跑一遍」；基线换成谁，往往决定了那篇论文能得出什么结论。

2. 读结构还是读像素：GUI 专门预训练

工作	时间	输入	预训练目标
MarkupLM（Microsoft）	2021.10	HTML / XML 文本	文本和标记联合预训练，面向网页类文档理解 [28]
Pix2Struct（Google）	2022.10	网页截图	把遮挡后的截图解析成简化 HTML；可变分辨率输入 [29]
HTML-T5（Google，WebAgent 内）	2023.07	长 HTML	局部 + 全局注意力，长跨度去噪目标 [10]
CogAgent	2023.12	截图	用大规模 GUI 和 OCR 数据继续预训练 [30]
ScreenAI（Google）	2024.02	截图	PaLI 架构 + Pix2Struct 的灵活切块；核心任务是「屏幕标注」：识别 UI 元素的类型和位置 [31]

Pix2Struct 的思路最关键：网页天然带着 HTML，HTML 就是截图的免费结构化标注。用 HTML 监督像素，后来 SeeClick、OS-Atlas 等 grounding 数据管线（第三章）都沿用这个思路。

> **一句话**：2023 年的 CUA 论文几乎都是「新 benchmark + 用 GPT-4 / GPT-4V 跑一遍」，GPT-4V 进 API（2023-11）才让 agent 能批量评测；GUI 专门预训练的关键思路是用网页 HTML 监督截图（Pix2Struct）。

> 我的复述

自测

1. 为什么 GPT-4V 进 API 比 GPT-4V 发布对 CUA 研究更关键?
2. Pix2Struct 的预训练数据为什么「免费」?

三、Grounding 与感知数据

1. GUI 上的「输准像素」

CUA 的核心能力是把一句话落到屏幕上的一个像素位置。通用的部分——patch 内的位置怎么编码、哪些环节丢信息、坐标表示怎么演化、预训练里怎样放位置监督——写在 **VLM 笔记**（网页版）第四、五章。这里只记 GUI 特有的事实。

目标到底有多小。ScreenSpot-Pro 里目标元素平均只占截图面积的 0.07%，ScreenSpot 是 2.01%[32]。按 4K 屏（3840×2160）粗算，0.07% 约等于边长 76 像素的方块，而这只是平均值。GUI-Actor 的作者也承认：Qwen2-VL 的视觉 token 是 28×28 像素，小于 10×10 像素的图标「可能表示不充分」，在 CAD 这类专业软件里尤其明显 [33]。

监督怎么和视觉 token 对齐：三派做法。

做法	监督的粒度	代表	局限
文本坐标	像素级：答案就是像素数值，按数字 token 做交叉熵	SeeClick、UGround、Qwen2.5-VL、UI-TARS	交叉熵不懂距离：把 512 写成 513 和写成 900 罚得一样重；RL 阶段的高斯奖励（GUI-G²，第六章）正是为补上「距离」
补丁注意力	补丁级：凡与标注框有重叠的补丁都记为正例（框坐标按补丁网格向外取整），注意力分布与归一化后的掩码之间用 KL 散度	GUI-Actor	精度上限就是一个补丁（28 像素）；用验证器在多个候选里挑
放大再定位	不改监督，改输入：把候选区域裁出来重新编码，让目标占更多补丁	ScreenSeekeR（免训练，ScreenSpot-Pro 18.9→48.1）、ZoomClick（免训练，UI-Venus-72B 在 ScreenSpot-Pro 上 73.1）[32][34]	多次前向，延迟增加

评测本身的容差。 ScreenSpot 一类基准判对的标准是预测点落在目标元素框内，所以「点准」需要的精度是元素大小，不是单个像素。元素比补丁大时，补丁级信息够用；元素比补丁小时，要么提高分辨率或放大重看，要么依赖模型从补丁特征里读出补丁内部的位置。

一句话：GUI 目标极小（ScreenSpot-Pro 平均只占截图 0.07%），而一个视觉 token 覆盖约 28×28 像素；对齐监督有三派：文本坐标（像素级）、补丁注意力（补丁级，GUI-Actor）、放大再定位（ScreenSeekeR、ZoomClick）。

我的复述

自测

1. ScreenSpot-Pro 的目标平均多大？和一个视觉 token 比呢？
2. GUI-Actor 的补丁级监督是怎么构造正例的？

2. Grounding 数据：SeeClick → OS-Atlas / UGround（2024.01–2024.10）

三篇回答同一个问题：截图 + 一句描述 → 坐标，这种训练数据从哪来、要多少？共同的答案是：网页自带 **HTML**，**HTML** 就是截图的免费标注（第二章 Pix2Struct 的思路），再想办法补上桌面和手机。

SeeClick（2024.01，南京大学 + 上海 AI Lab） SeeClick（Cheng et al. 2024）[35] 的出发点：通用 VLM 说自己会在自然图像上定位，但放到 GUI 上很差，**GUI grounding 是纯视觉 agent 的关键瓶颈**。

- 坐标格式：之前 Pix2Seq、Pix2Act 等把图像切成 1,000 个 bin、新增 `<p0>...<p999>` 词表；SeeClick 改为直接把数字当自然语言，输出如 `click (0.49, 0.40)` 的相对坐标，用普通交叉熵训练。
- 数据（共约 100 万条）：
  - 网页：从 Common Crawl 抓约 30 万个网页，从 HTML 里取两类元素——**显示可见文字的元素**，和带 **`title` 属性（鼠标悬停时显示说明）的元素**。文字或 title 当指令，元素位置当答案；另加反向任务（给坐标、说出文字，即 web OCR）。
  - 手机：widget captioning 数据（约 2 万截图、4 万控件、10 万条描述），**把「控件 → 描述」反过来当「描述 → 控件」用**；再加 RICO 自动收集的数据和界面摘要数据。
  - 通用：LLaVA 的指令数据，防止通用能力退化。
- 训练：在 Qwen-VL（448×448）上继续预训练约 1 万步（约 1 个 epoch），视觉编码器和 LLM 都用 LoRA。
- **ScreenSpot 基准**：600+ 截图、1,200+ 条指令，覆盖 iOS、Android、macOS、Windows、网页；特意包含大量图标 / 控件（比文字难定位）。

模型	ScreenSpot 平均	图标类（手机 / 桌面 / 网页）
Qwen-VL（原版）	5.2	4.8 / 5.0 / 2.4
GPT-4V	16.2	24.5 / 11.8 / 8.8
CogAgent（18B）	47.4	24.0 / 20.0 / 28.6
SeeClick（9.6B）	53.4	52.0 / 30.0 / 32.5

作者在三个 agent 基准上得出的结论：**grounding 提升与 agent 任务表现直接相关**。

OS-Atlas（2024.10，上海 AI Lab 等） OS-Atlas（Wu et al. 2024，一作 Zhiyong Wu，也是 SeeClick 的作者）[36] 把数据扩到五个平台：

数据集	截图（网页 / 手机 / 桌面）	元素数
SeeClick	270K / 94K / —	3.3M
CogAgent（未开源）	400K / — / —	70M
OS-Atlas	1.9M / 285K / 54K	13.58M

- 网页：从 FineWeb 的 URL 抓约 400 万个网页，取所有可见、可点击的元素（按钮、滚动条、搜索框、链接、带 title 的 SVG）；渲染整页再切成 **1920×1080** 的截图（SeeClick 主要只取页面上部）；规则过滤掉渲染不全、元素分布异常的页面，每页最多 10 个元素，最后 160 万截图、770 万元素。
- 桌面和手机：难点是要在真实系统里自动造出各种界面状态。Android 用 AndroidEnv，Linux 用 OSWorld 的环境；Windows 和 macOS 虚拟化困难，直接部署在实体机上。用各系统的 a11y tree 取元素（Ubuntu 用 pyatspi、Windows 用 pywinauto、macOS 用 ApplicationServices），再按 a11y tree 采样动作、用深度优先搜索和随机游走模拟操作，改变界面状态。
- 指令型数据：用 GPT-4o 给已有轨迹数据补「子指令」——给它动作前后两张截图和用 SoM 标出的被操作元素，让它推断这一步在做什么。
- 统一动作空间：多个数据集直接混合会冲突（桌面的 click 等价于手机的 tap），所以统一动作格式。
- **ScreenSpot-V2**：评测时发现 ScreenSpot 约 11.32% 的标注有错，修正后发布 V2。
- 底座：Qwen2-VL-7B（训练时见过 GUI 数据）和 InternVL-2-4B（没见过）。

两条消融结论：

1. 「描述 → 元素」这种指代表达数据几乎足够训练出强 **grounding** 模型，而且比指令型数据好扩展；
2. 只用网页数据训练，难以泛化到桌面和手机——这是要花大工程搭跨平台采集设施的理由。

ScreenSpot（不加规划器）：OS-Atlas-Base-7B 82.47、UGround-7B 73.30、Qwen2-VL-7B 42.89、SeeClick 53.40。接到 OSWorld 上当「定位模块」（GPT-4o 负责规划、OS-Atlas 负责给坐标）：GPT-4o 单独 5.03% → 加 SeeClick 9.21% → 加 OS-Atlas-Base-7B 14.63%，同表 GPT-4o + SoM 4.59%。

UGround（2024.10，OSU） UGround（Gou et al. 2024，OSU NLP，Yu Su、Huan Sun 组——Mind2Web、SeeAct 的同一组）[37] 主张 agent 应该像人一样只看画面、直接做像素级操作：HTML 和 a11y tree 有噪声和缺失（例如图片缺 alt 文字），HTML 编码的 token 数可达视觉的 10 倍，桌面和手机上取 a11y tree 本身也慢。配套提出 SeeAct-V：规划器出指令，UGround 出坐标。

- 数据：共 1,000 万个元素、130 万张截图，90% 来自自家的 **Web-Hybrid** 合成管线（规则 + LLM）。
- 指代表达分三类：视觉型（文字、图像内容、元素类型、形状、颜色）、位置型（「页面左上角」「X 右边」，以及「Item A 的那个按钮」这类上下文指代）、功能型（元素的用途）。前两类主要用规则从 HTML 属性和 DOM 结构生成，并用 Llama-3-8B 把描述改得更简洁；再用 GPT-4o 直接合成一小批开放式和功能型描述（Web-Direct）。手机端补入已有的 Android 数据。
- 模型：在 LLaVA-NeXT 7B 上小改。

  - **坐标不归一化**，直接输出像素数值，如 (1344，1344)，预测元素中心点；
  - 把 AnyRes 的切片上限提到 36 块，编码器换成 CLIP@224px 以便切得更灵活，最大支持 1344×1344（横屏）/ 896×2016（竖屏）；LLM 换成 16K 上下文的 Vicuna-1.5；
  - 去掉 AnyRes 常用的低分辨率全图融合模块——224px 的全图太小，对 GUI 定位没有用。
- 之后也发布了基于 Qwen2-VL 的 UGround-V1。

3. 三篇对照

	SeeClick	**OS-Atlas**	**UGround**
时间 / 机构	2024.01，南大 + 上海 AI Lab	2024.10，上海 AI Lab 等	2024.10，OSU
数据来源	网页 HTML + 手机公开数据	网页 + 实机 / 模拟器上的 a11y tree（五平台）	网页 HTML 规则 + LLM 合成，补 Android
规模	约 100 万条	1,358 万元素	1,000 万元素
坐标	0–1 相对坐标文本	点或框	**不归一化的像素坐标**
底座	Qwen-VL	Qwen2-VL-7B / InternVL-2-4B	LLaVA-NeXT 7B（后有 Qwen2-VL 版）
关键发现	grounding 提升 ↔ agent 提升	指代表达数据够用；只用网页泛化不到桌面 / 手机	纯视觉 + 简单配方就够强

人员连线：Kanzhi Cheng、Zhiyong Wu 从 SeeClick 到 OS-Atlas；Kanzhi Cheng 之后又署名 GUI-Actor（第三章第 1 节）。UGround 出自做 Mind2Web、SeeAct 的 OSU 组——这个组从「读 HTML」（Mind2Web）→「GPT-4V 看截图但定位要靠 HTML」（SeeAct）→「只看截图、自己定位」（UGround），一步步走到了纯视觉。

> **一句话**：grounding 数据的主力是「网页 HTML 当免费标注」——SeeClick 证明它有效，UGround 把网页合成做到 1,000 万并改成像素坐标，OS-Atlas 发现只靠网页泛化不到桌面和手机，于是在真实系统里用 a11y tree 采数据。到 2024 年底，专门训练过的 7B grounding 模型在 ScreenSpot 上已到 70–80 分，接到 OSWorld 上能把 GPT-4o 的成功率提高近两倍。

> 我的复述

> 自测
>
> 1. SeeClick 从网页 HTML 里取哪两类元素？为什么 `title` 属性特别有用？
> 2. OS-Atlas 为什么要在实体机上跑 Windows 和 macOS？它的消融说明了什么？
> 3. UGround 为什么去掉了 AnyRes 的低分辨率全图模块？为什么坐标不归一化？

四、Benchmark 演进

1. WebArena：可交互、查结果的真实网站（2023.07）

WebArena（Zhou et al. 2023，CMU）[8] 要同时做到两件事：**可复现**（不依赖线上网站，避开验证码、内容变化、改版）和**真实**（用真实网站背后的开源软件搭，再导入真实数据）。

环境

- **选站**：作者分析了自己浏览器历史里约 200 个例子，归出四类最常见的网站，每类搭一个：电商（OneStopShop）、论坛（Reddit 式）、协作开发（GitLab）、内容管理（网店后台 CMS）。
- **工具与知识**：地图、计算器、草稿本；英文 Wikipedia 和各网站的用户手册。
- **交付**：Docker 镜像 + 重置脚本，每次评测前把环境恢复到确定的初始状态。状态转移是确定性的，由网站本身的实现决定。

观测：当前 URL、所有已打开的标签页、当前标签页的内容。内容可选三种表示：原始 HTML（DOM 树）、截图、accessibility tree。它是第一个考虑多标签页任务的 web 环境。

动作（12 种，分三组）：

组	动作
元素操作	`click(elem)`、`hover(elem)`、`type(elem, text)`、`press(key_comb)`、`scroll(dir)`、`noop`
标签页	`tab_focus(i)`、`new_tab`、`tab_close`
URL 导航	`goto(URL)`、`go_back`、`go_forward`

元素可以用屏幕坐标 (x, y) 指定，也可以用遍历 DOM / a11y tree 时给每个元素加上的唯一 ID 指定（如 `click [1582]` 点「Add to Cart」）。**用 ID 时，选元素就变成 n 选 1 的分类问题**——和 MindAct、SoM 是同一个思路。

任务：标注员先熟悉网站，再写高层、有创意的意图，并写成模板（如「在 {site1} 上建一个和我 {site2} 上一样的账号」），每个模板实例化多次。共 241 个模板、812 个任务，平均每个模板 3.3 个。三类：

类别	例子
信息查询	「我上次买洗发水是什么时候?」（要翻自己的订单历史）
站内导航	「查看分配给我的 merge request」
内容与配置	「发帖问在纽约是否需要车」「删掉骗子 Yoke 的评论」

另有一部分**不可完成的任务**（证据不足、权限不够、网站没这个功能），正确做法是停下并说明。

评测：只看结果，不看路径

- **信息查询** $r_{\text{info}}(\hat{a}, a^\star)$：把模型的回答和标注答案比，三种函数——`exact_match`（完全一致）、`must_include`（包含关键内容）、`fuzzy_match`（用 gpt-4-0613 判语义是否等价，处理「2h58min」和「2:58」这类格式差异）。
- **导航和内容操作** $r_{\text{prog}}(\mathbf{s})$：程序化检查执行过程中的状态。每题配一个 **locator** 取出关键内容——可以是数据库查询、网站 API 调用，或在页面上跑一段 JavaScript 选择器；再检查标注的关键词是否出现。例如「在 nyc 版块发帖」：取最新帖子的 URL，打开后用 JS 读帖子正文，检查 URL 是否在 nyc 版块、正文是否包含要求的内容。

和 Mind2Web 的根本区别：Mind2Web 问「每一步是否和标注员一样」，WebArena 问「订单下了没、帖子发了没」。走哪条路都行，中途走错还能补救。

基线与结果：用 a11y tree + 元素 ID 作观测，给 2 个示例，比较是否先做 CoT、是否提示「做不到就停」（UA hint）。

模型	CoT	UA hint	成功率
text-bison-001	✓	✓	5.05
GPT-3.5	✓	✓	8.75
GPT-4	✓	✓	11.70
GPT-4	✓	□	14.41
人类	—	—	78.24

一个典型失败：加了 UA hint 后，**GPT-4 把 54.9% 本可完成的任务判成不可能，提前放弃**。作者认为模型缺的是主动探索和从失败中恢复的能力。

那时候哪种表示最好？ WebArena 自己的基线只用 a11y tree（文本）。半年后同组的 VisualWebArena（Koh et al. 2024.01，CMU）专门做了比较 [38]：910 个必须看图才能做的任务，三个环境（新增的分类广告站 Classifieds，以及沿用 WebArena 的购物站和 Reddit），25.2% 的任务输入里还带图片。

设定	最好模型	成功率
纯文本（a11y tree）	GPT-4	7.25
a11y tree + 图片描述（BLIP-2 生成）	GPT-4	12.75
截图 + 描述 + a11y tree	GPT-4V	15.05
截图 + 描述 + SoM 编号	GPT-4V	16.37
人类	—	88.7

同表里开源 VLM 几乎为零：CogVLM 0.33，IDEFICS-80B-Instruct 不到 1。结论是：**纯文本在需要看图的任务上明显不够，多模态 + SoM 最好**；但这个提升只在 GPT-4V 这种闭源模型上成立，开源 VLM 当时还用不起来。

一句话：WebArena 把评测从「逐步模仿标注」改成「在可重置的真实网站里做完任务、用程序检查最终状态」，这套「Docker 环境 + 重置 + 每题一个检查函数」的范式，后来被 OSWorld 搬到整台电脑上。

我的复述

自测

1. WebArena 为什么不直接用线上的真实网站？代价是什么？
2. 「在 nyc 版块发帖」这道题是怎么判对错的？和 Mind2Web 判一步对错有什么不同？
3. GPT-4 加了「做不到就停」的提示，成功率反而从 14.41 降到 11.70，为什么？

2. OSWorld：把 WebArena 的范式搬到整台电脑（2024.04）

OSWorld（Xie et al. 2024，HKU XLang 牵头，联合 CMU、Salesforce、Waterloo）[1]。「零、基础」已经用它讲过观测、动作和计分，这里补完整。

和 WebArena 的对应关系

	WebArena	OSWorld
环境单元	Docker 里的自托管网站	虚拟机里的完整操作系统
重置	重置脚本	VM 快照 + 配置文件
观测	URL、标签页、HTML / 截图 / a11y tree	整屏截图、a11y tree、终端输出等
动作	12 种网页动作，可按元素 ID	`pyautogui` 键鼠代码字符串，按坐标
评测	每题 locator + 关键词	每题一个执行式评测脚本，共 134 个评测函数
人类 / 最好模型	78.24% / 14.41%	72.36% / 12.24%

基础设施（§2.2）：宿主机上的 Coordinator 读任务配置文件 → 自动起一个 VM 实例 → Task Manager 恢复指定快照并完成初始化 → agent 交互，每个动作以代码字符串送进 VM 由 Simulator 执行 → 任务结束后 Task Manager 做后处理（保存文件、重开应用），把数据取回宿主机，运行评测脚本。一台宿主机可同时跑多个 VM，**训练和评测都能并行**。

从「做了一半」开始：真实求助往往发生在中途（软件已打开、电脑刚崩过）。每个任务都用快照 + 配置模拟这种中间状态。不给每题单独存快照（会存下大量无用的硬件状态，每题几个 GB），而是三步混合：启动 VM → 准备文件（从云端下载，可选）→ 执行预处理命令（打开文件或标签页、调窗口大小，可选）。共 302 种初始状态。

评测脚本长什么样（论文表 1，简化）：

任务	怎么判
清除 Amazon 存下的所有 cookie	从 VM 取 cookie 数据，检查 `.amazon.com` 域的已删除
把 Sheet 1 改名、复制一份放到 Sheet 2 前并加后缀	从 VM 取结果文件、从云端取期望文件，按表名和表数据规则逐项比
从缴费记录里找出没交学费的人，填进邮件收件人	取 Thunderbird 当前的 a11y tree，用选择器检查收件人栏里的地址

为了能读到这些内部状态，要为每个软件搭脚手架：给 Chrome、VLC 开远程调试端口，给 VS Code 写扩展，Thunderbird 的账户信息甚至要用逆向工具解密。

任务构成

统计	数量
Ubuntu 任务	369
— 跨应用工作流	101（27.4%）
— 单应用	268（72.6%）

统计	数量
— 从其他基准整合（NL2Bash、Mind2Web、SheetCopilot 等）	84（22.8%）
— 不可完成	30（8.1%）
Windows 补充任务	43

任务是人写的，不是合成的：9 位计算机背景的作者逐题标注指令、初始状态配置和评测脚本；只有 84 题是从其他基准整合进来的。应用以 8 个为主：Chrome、VLC、Thunderbird、VS Code、LibreOffice（Calc / Writer / Impress）、GIMP，加上终端、文件管理器等系统应用。任务来源是官方教程、TikTok / YouTube 技巧视频、WikiHow、Reddit / StackOverflow 等问答、Coursera 课程；跨应用任务网上少，作者自己拼。不可完成的任务来自真实用户问过的「已废弃功能」或「想象出来的功能」。质检：每题由两位非标注者当 agent 做一遍，四轮检查共 400 多人时。

人类实验：计算机专业本科生，没见过这些任务和部分软件。中位用时 111.94 秒（WebArena 抽样 100 题为 35.38 秒），不少任务超过 900 秒；准确率 72.36%（同条件下 WebArena 约 88%）。

完整结果（论文 v2，15 步，总成功率）：

输入	代表模型	成功率
A11y tree	GPT-4 / GPT-4o / Qwen-Max	12.24 / 11.36 / 6.87
纯截图	Gemini-Pro-Vision / GPT-4V / GPT-4o / CogAgent	5.80 / 5.26 / 5.03 / 1.11
截图 + a11y tree	GPT-4V / GPT-4o	12.17 / 11.21
SoM	GPT-4V / Claude-3-Opus	11.77 / 6.72
人类	—	72.36

按类别看，Office（LibreOffice）最难，Calc 子集常常是 0；跨应用工作流最好也只有 6.57%（GPT-4V SoM），单应用 13.74%。

分析里最值得记的五条

1. **按人类用时分难度**：0–60 秒 / 60–180 秒 / 180 秒以上，GPT-4V（SoM）成功率 16.78 / 13.12 / 4.59，人类 84.91 / 81.08 / 49.57。超过 3 分钟的任务对模型几乎不可能。
2. **分辨率**：截图降采样后性能下降（第二章分辨率一节的直接证据）。
3. **历史**：文本形式的历史轨迹越长越好，截图形式的历史没有这个效果；但 a11y tree 很长，单个观测约需 6,000 token 上下文才能覆盖 90% 的情况。
4. **窗口扰动**：挑 28 个模型做得较好的任务（50.79%），开局时移动窗口、把窗口缩到最小、或打开无关软件铺满屏幕，成功率下降 60%–80%。模型会切窗口，但不会先把窗口最大化。
5. **失败原因**：抽样 550 个失败案例，**超过 75% 有鼠标点击不准**——代码注释里计划写得很对，点的坐标却错了，「规划强、执行弱」。点错又引出重复点击和误触弹窗；对专业软件缺乏先验，走偏后不知道怎么回到正常状态。

一年后的 OSWorld-Verified（2025-07）[4]：

- **可靠的 reward 比想象中费人力**：发布前已投入 400 多人时，社区仍报告了约 300 个问题，来自 Moonshot AI、OpenAI、ByteDance、Anthropic、Simular 等。修复时尽量只改评测脚本、不改任务描述，以保持分数连续。

- **问题类型**：网站反爬和验证码、IP 封锁、网页改版；初始化的时序依赖（软件没加载完就执行预处理命令）；任务描述有歧义（「紫色背景」是哪种紫）；**模型找到了标注者没想到的正确解法**（用扩展改 VS Code 背景、用跨应用方式完成单应用任务），被误判失败。
- **基础设施演进**：VMware（分发 `.vmdk` 镜像，占用个人电脑资源，Broadcom 收购后难下载）→ 2024 年夏 Docker 内跑 QEMU，单服务器 8–16 个环境并行 → 参照 WindowsAgentArena 上 AWS，最多 50 个环境并行，评测从 10 小时以上缩到几分钟。
- **去中心化评测的问题**：各家自己修环境、甚至改任务刷分，分数越来越不可比，所以推公开统一的评测平台。

一句话：OSWorld 把 WebArena 的「可重置环境 + 每题一个检查函数」搬到整台电脑：VM 快照做中间状态，134 个评测函数读软件内部状态判结果。2024 年的模型在它上面「规划强、点不准」；一年后 Verified 版本说明，**可靠的 reward 本身就是最贵的部分**——这正是第六章 RL 要面对的问题。

我的复述

自测

1. OSWorld 为什么不给每个任务存一份完整快照？它用什么办法构造「做了一半」的初始状态？
2.「清除 Amazon cookie」和「填写邮件收件人」两题的评测脚本分别读了什么？为什么要给 Chrome 开远程调试端口？
3. OSWorld-Verified 发现了哪几类让分数失真的问题？其中哪一类说明「检查脚本越严格越好」不成立？

3. OSWorld 分数是怎么被刷到超过人类的（2024.04–2026.08）

出现的时间点：OSWorld 发布于 2024-04-11——GPT-4V 进 API（2023-11）和 Claude 3（2024-03）之后，GPT-4o（2024-05）和 Claude computer use（2024-10）之前。它正好赶上前沿模型开始做 computer use，于是成了各家发布页的标准分。

时间	事件	OSWorld 分数	设定
2024-04-11	OSWorld 发布 [1]	GPT-4 12.24；纯截图最好 5.80	原版，15 步
2024-10-22	Claude 3.5 Sonnet computer use[24]	14.9（仅截图类；下一名 7.8）	15 步；50 步时 22.0[3]
2025-01-21	UI-TARS[3]	22.7 / 24.6	15 步 / 50 步
2025-01-23	OpenAI CUA[25]	38.1（此前 SOTA 22.0，人类 72.4）	原版
2025-02-24	Claude 3.7 Sonnet	28.0	原版（Sonnet 4.6 页图表）
2025-04-16	UI-TARS-1.5	42.5	100 步（GitHub）
2025-05-22	Claude Sonnet 4	42.2	原版（Sonnet 4.5 页文字）[39]
2025-07-28	**OSWorld-Verified** 发布 [4]	—	修评测脚本、AWS 并行

时间	事件	OSWorld 分数	设定
2025-08	OpenCUA-72B；CoAct-1（GUI + 代码）	45.0；60.76	Verified；原文设定
2025-09-02	UI-TARS-2	47.5	—
2025-09-29	Claude Sonnet 4.5[39]	61.4	**Verified，100 步，4 次平均**
2025-11-24	Claude Opus 4.5	66.3	Verified（发布页图表）
2026-02-05 / 17	Claude Opus 4.6 / Sonnet 4.6[40]	72.7 / 72.5	Verified（图表）；**追平人类 72.36**
2026-03-05	GPT-5.4[41]	75.0	Verified；同表 GPT-5.3-Codex 74.0（此前报 64.7，换了评测方式）
2026-04-16	Claude Opus 4.7	82.3	Opus 4.8 页脚注：改了运行方式后更新的分数 [42]
2026-06-28	**OSWorld 2.0 发布** [43]	最好 20.6	新基准，500 步
2026-08	Qwen-CUA（397B-A17B）	86.2	Verified（摘要）

带「图表」的分数取自发布页的图片表格，由检索时读图得到；2026 年开源模型只核对了摘要。

分数涨上去的几条原因（都有原文依据）：

1. **步数上限放宽**：15 步 → 50 步 → 100 步。同一个 Claude 3.5，15 步 14.9、50 步 22.0。
2. **基准本身修了**：Verified 修掉约 300 处问题，大量是评测脚本把正确解法判错（假阴性）。Anthropic 从 Sonnet 4.5 起改报 Verified，并注明之前的分数是原版。
3. **评测方式改了**：GPT-5.3-Codex 从 64.7 改报 74.0，Opus 4.7 改报 82.3，都是「改了怎么跑」，不是换了模型。
4. **模型真的变强**：原生 computer use 模型（UI-TARS、OpenAI CUA），之后是数千台 VM 上的多轮 RL（UI-TARS-2）、几万个可验证任务的 RL（Qwen-CUA）——第五、六章展开。Verified 博客的判断是：当时领先模型的成功主要来自大规模人类轨迹数据。
5. **动作空间变了**：加入代码 / 终端（CoAct-1 60.76）——第七章展开。

刷满之后：同一组（HKU XLang，Tianbao Xie、Tao Yu 署名）在 2026-06 发布 OSWorld 2.0：108 个长程工作流，人类中位用时约 1.6 小时；Opus 4.7（最大思考）平均要 318 次工具调用，OSWorld 1.0 约 30 次。500 步下最好的 Opus 4.8 二值完成率只有 20.6%（部分得分 54.8%），GPT-5.5 约 13%。作者认为失败已经不在基本 GUI 操作或写代码，而在：丢失约束、漏掉中途到来的信息、该问用户时去猜、跳过验证、恢复不了隐藏状态 [43]。

> **一句话**：OSWorld 从 12% 到超过人类用了约两年，其中一部分是步数、基准版本、评测方式的变化，一部分是原生模型和大规模 RL 带来的真实进步；读任何一个 OSWorld 分数都要先看版本和步数。刷满之后，难点从「点得准」转到了「长程、隐藏状态、何时提问」。

五、模型谱系

本章讲 2024-10 到 2025-09 的模型线：闭源两家起点（Claude、OpenAI CUA）和字节 Seed 的 UI-TARS 系列。开源学术一侧（HKU XLang × Qwen × Moonshot：Aguvis → OpenCUA → Qwen-CUA）整理在附录「XLang 组谱系」。

1. 闭源起点：Claude computer use（2024-10-22）

升级版 Claude 3.5 Sonnet 是**第一个以公开 beta 提供 computer use 的前沿模型**，通过 API 开放，官方自己说「仍是实验性的，有时笨拙、容易出错」[24]。同日的研发博客给了几条做法 [44]：

- **只看截图、只用键鼠**：模型看用户屏幕的截图，**数出光标要在横向、纵向移动多少像素**，再点击。「把像素数准是关键」——否则模型很难给出鼠标指令，类似模型数不清 banana 里有几个 a。
- **训练范围很小**：只在计算器、文本编辑器等几个简单软件上训练，出于安全考虑训练时不让模型上网；泛化之快出乎他们意料，模型还会在遇到障碍时自我纠正、重试。
- **结果**：OSWorld 仅截图类别 14.9%，同类别第二名 7.7%，人类一般 70%–75%。

这正好对上 OSWorld 的失败分析（75% 以上是点不准）：闭源一侧解决「点不准」的办法，是专门训练坐标 / 像素计数能力。

2. OpenAI CUA / Operator（2025-01-23）

Operator 背后的 Computer-Using Agent（CUA）= **GPT-4o 的视觉能力 + 通过强化学习得到的推理能力** [25]：

环节	做法
感知	截图加入上下文
推理	结合当前和过去的截图、动作做 chain-of-thought（内心独白），评估观察、追踪中间步骤
动作	点击、滚动、输入，直到判断完成或需要用户输入；输入登录信息、遇到验证码等敏感操作时请用户确认

同一套「屏幕 + 鼠标 + 键盘」接口跑三个基准：OSWorld 38.1%（此前 SOTA 22.0%，人类 72.4%）、WebArena 58.1%、WebVoyager 87%。UI-TARS-1.5 的 README 注明这个 38.1 是 200 步的结果，100 步时为 36.4[45]。

3. UI-TARS 系列（字节 Seed）

UI-TARS（2025-01-21） UI-TARS（Qin et al. 2025，一作 Yujia Qin）[3]：**在 Qwen2-VL 上继续训练约 50B token**，得到 2B / 7B / 72B 三个规格。论文把 GUI agent 的演进分四个阶段：规则式（RPA）→ agent 框架（GPT-4 / GPT-4o + 提示词和工作流）→ **原生 agent 模型**（端到端，感知、推理、记忆、动作都在一个模型里）→ 主动与终身学习的 agent（展望）。UI-TARS 自定位在第三阶段。

模块	做法
感知数据	从网站、App、操作系统大规模收集截图，用解析工具抽元素类型、层级、边界框等元数据；构造元素描述、密集描述、**状态转移描述**（前后两帧发生了什么）、问答、SoM 等任务
统一动作空间	把各平台的操作统一成一套动作；收集多步动作轨迹
Grounding	**直接预测要操作元素的坐标**；grounding 数据比多步轨迹好扩展，靠元素的视觉和位置属性即可合成
System-2 推理	用 GUI 教程丰富推理；为动作补写「思考」
从经验中学	在线轨迹自举 + 反思微调 + Agent DPO（见下）

在线轨迹自举（在几百台虚拟机上）：第 n 轮的模型 M_n 执行一批指令（人工写的 + 模型生成的），得到原始轨迹；三层过滤：

1. **规则奖励**：删掉明显异常的轨迹（如不改变环境的冗余动作）；
2. **VLM 打分**：低于阈值的删掉；
3. **人工审查**：标注员找到出错的那一步，只保留之前的有效前缀。

$$M_{n+1} = \text{FineTune}\left(M_n,\ \text{Filter}(\mathcal{T}_{\text{raw},n}, \mathcal{J}_n)\right)$$

- **反思微调**：离线数据都是标注员保证正确的理想轨迹，模型学不到「出错后怎么恢复」。于是拿模型自己犯的错 + 标注员的纠正来训练。
- **Agent DPO**：SFT 只用纠正后的正样本，丢掉了大量错误步骤；DPO 把「纠正后的动作」作为偏好、「原来的错误动作」作为非偏好，一起用上。

训练三阶段：持续预训练（全部数据，恒定学习率）→ 退火（高质量子集，含反思数据）→ DPO，得到 UI-TARS-DPO。结果：OSWorld 24.6（50 步）/ 22.7（15 步），同条件 Claude 22.0 / 14.9；AndroidWorld 46.6，GPT-4o 34.5。

UI-TARS-1.5（2025-04-16） 在 UI-TARS 架构上**加入强化学习带来的「先想后做」推理**，强调推理时扩展（inference-time scaling）[45]。OSWorld 100 步 42.5（同表 OpenAI CUA 36.4、Claude 3.7 28.0），AndroidWorld 64.2；还能玩游戏。开源的 UI-TARS-1.5-7B（OSWorld 27.5）**基于 Qwen2.5-VL**，README 专门提醒其 grounding 用绝对坐标。

UI-TARS-2（2025-09-02） UI-TARS-2[46] 针对四个开放问题：数据扩展、多轮 RL、只靠 GUI 的局限、环境稳定性。

- **基座换了**：不再基于 Qwen，而是从字节自己的 **Seed-thinking-1.6** 初始化——532M 视觉编码器 + 230B 总参数 / 23B 激活的 MoE。
- **数据飞轮**：持续预训练 → SFT → RL 三段循环，「更好的模型产出更好的数据，更好的数据训出更好的模型」。

- **混合环境**：GUI 之外接入文件系统和终端（GUI-SDK），可调工具、写代码。
- **沙箱平台**：自建分布式 VM 平台，覆盖 Windows、Ubuntu、Android（PyAutoGUI + ADB）；集群几千台实例，VM Manager 支撑每秒几千次请求；会话按 ID 绑定任务和环境，可用 VNC 实时查看，租约机制在任务结束或失败后自动回收资源。
- **奖励分三类**：游戏用 JavaScript 校验脚本读运行时变量（分数、关卡、剩余生命）给确定性奖励；GUI 浏览类有参考答案，用 LLM 当裁判比对；通用 GUI 任务**既没有验证器也没有参考答案**，就用 UI-TARS-2 自己做生成式结果奖励模型（输入完整文本历史 + 最后 5 张截图，输出成功分数）。
- **算法**：多轮 RL 用 PPO，论文称 PPO 稳定地明显优于 GRPO；不同垂直领域训出的 agent 用参数插值合并。
- **结果**：OSWorld 47.5、WindowsAgentArena 50.6、AndroidWorld 73.3、Online-Mind2Web 88.2；15 款游戏平均归一化分 59.8（约人类的 60%）；接上终端和工具后 BrowseComp 29.6、Terminal Bench 45.3。

4. 对照：底座和路线

模型	时间	底座	关键做法	**OSWorld**
Claude 3.5 Sonnet CU	2024-10	自家	训练像素计数	14.9（15 步）
Aguvis	2024-12	Qwen2-VL	先 grounding 后规划，纯视觉	10.26（Qwen2.5-VL 报告中的数字）
UI-TARS	2025-01	Qwen2-VL	50B token 持续训练 + 在线自举 + DPO	24.6（50 步）
OpenAI CUA	2025-01	GPT-4o	视觉 + RL 推理	38.1（200 步）
Qwen2.5-VL	2025-01	自家	基模内置 agent 数据	8.83
UI-TARS-1.5	2025-04	7B 版基于 Qwen2.5-VL	加 RL 推理	42.5（100 步）
OpenCUA-72B	2025-08	Qwen2.5-VL	开放数据 AgentNet	45.0（Verified）
UI-TARS-2	2025-09	Seed-thinking-1.6（自家 MoE）	数据飞轮 + 多轮 PPO + 混合环境	47.5

两个规律：一是 2025 年上半年，开源 / 学术的 CUA 模型几乎都以 Qwen-VL 为底座；字节到 UI-TARS-2 换成了自家基座。二是从 UI-TARS 到 UI-TARS-2，训练信号从「人工审查过滤的 SFT + DPO」走到「多轮 RL + 三类 reward」——第六章展开。

> **一句话**：解决「点不准」有两条路——闭源（Claude）专门训练像素级坐标，开源（UI-TARS、Aguvis）在 Qwen-VL 上用大规模 grounding 数据直接预测坐标；之后各家都转向用在线交互数据和 RL 继续提升，底座也从「借 Qwen」走向「自家基模 + 自建 VM 集群」。

> 我的复述

自测

1. Anthropic 说「把像素数准是关键」，这和 OSWorld 失败分析里的哪条发现对应？
2. UI-TARS 的在线自举里，「规则奖励 + VLM 打分 + 人工审查」三层各过滤什么？反思微调为什么必须用模型自己的错误？
3. UI-TARS-2 的三类 reward 分别用在什么任务上？「通用 GUI 任务」为什么只能用模型自己当裁判？

六、RL：reward 与环境

本章从最简单的情形讲起：**单步 grounding**——给一张截图和一句指令，模型输出一个动作；奖励直接拿标注好的元素框来算，不需要真的跑环境。之后再讲多步、长时序任务的 reward 和环境（DigiRL、WebRL、UI-TARS-2、ComputerRL、CUA-Gym、Qwen-CUA）。

1. 单步 grounding 的 RL：从「点中得 1 分」到高斯奖励（2025.03–07）

背景是 DeepSeek-R1：用**规则就能判对错**的奖励（数学答案对不对）做 RL，推理能力就能长出来。GUI 的单步 grounding 天然满足这个条件——点击坐标落没落在标注框里，一行代码就能判。2025 年春天，一批工作几乎同时把 R1 的配方搬过来，底座基本都是 Qwen2.5-VL，算法基本都是 GRPO。

UI-R1（Lu et al. 2025）[47]，自称第一个把规则式 RL 用到 GUI 动作预测上的框架。

UI-R1(2025-03-27)

$$R = R_{\mathcal{T}} + R_{\mathcal{C}} + R_{\mathcal{F}}, \qquad R_{\mathcal{C}} = \begin{cases} 1 & \text{点击坐标落在标注框内} \\ 0 & \text{否则} \end{cases}$$

- $R_{\mathcal{T}}$：动作类型对不对（Click / Scroll / Back / Open_App / Input_Text，对得 1）；$R_{\mathcal{C}}$：坐标是否落在框内；$R_{\mathcal{F}}$：输出格式。作者特意不用检测任务常用的 IoU，因为 GUI 关心的是**点击位置**，不是框得多准。
- **数据只有 136 条**：仿照 s1，按质量、难度、多样性三条标准，从公开的手机数据里筛出来。
- 底座 Qwen2.5-VL-3B。相对底座：ScreenSpot +22.1、ScreenSpot-Pro +6.0、AndroidControl +12.7；只用手机数据训练，在桌面和网页上也涨了。3B 的效果可比肩用 7.6 万条数据 SFT 的 OS-Atlas-7B。
- 消融结论（原话）：**对 grounding 这种简单任务，推理不是必需的**。于是有了加速版 UI-R1-E：先按「回答长度和预算的偏差」调整奖励训练，再做不思考（NOTHINK）训练。

GUI-R1（Luo et al. 2025）[48] 把同样的思路扩到五个平台和高层任务：

GUI-R1(2025-04-14)

$$R_{\text{acc}} = R_{\text{act}} + R_{\text{point}} + R_{\text{text}}$$

另加格式奖励（`<think>` 推理 + `<answer>` 答案）。关键是**统一动作空间**，让各平台数据能放在一起训。数据 3K 条，摘要称是 OS-Atlas（13M）的 0.02%（正文另一处写作 14M、0.2%，两处不一致）。最直接的对照：**同样的数据，3B 模型用 SFT 和用 RL 训**——ScreenSpot 63.55 vs 80.08，ScreenSpot-Pro 13.80 vs 25.23。

InfiGUI-R1（2025-04-19） InfiGUI-R1（Liu et al. 2025）[49] 关心的是从「条件反射式」到「先想后做」：第一阶段从教师模型蒸馏带空间推理的轨迹；第二阶段 RL，**奖励模型生成的中间子目标是否准确**，并从容易出错的步骤构造「出错—恢复」训练场景。它已经开始碰多步任务的问题。

GUI-G1（2025-05-21）：拆解 R1 配方在 grounding 上的三个坑 GUI-G1（Zhou et al. 2025）[50] 逐项检查「在线 RL + 先写思维链再定位」这套 R1-Zero 式流程：

环节	问题	改法
输入	模板鼓励写思维链，但**思维链越长，定位越差**	快思考模板：直接给答案
奖励	「中心点命中」和「IoU」两种奖励都能**靠改变框的大小来钻空子**（reward hacking），定位质量反而变差	奖励里加框大小约束
更新	GRPO 的长度偏差鼓励更长的错误回答；难度偏差让模型偏向简单样本	调整长度归一化，加难度感知的缩放

结果：17K 条公开数据、Qwen2.5-VL-3B，ScreenSpot 90.3、ScreenSpot-Pro 37.1，超过 UI-TARS-7B。

GUI-G²（2025-07-21）：把二值奖励换成高斯奖励 GUI-G²（Tang et al. 2025）[51] 的出发点：「点中 / 没点中」的二很远得分一样，没有方向信息。他们统计 AitW 里人类的真实点**高斯分布**。于是把每个元素建模成二维高斯：

$$\mu = \left(\tfrac{x_1+x_2}{2}, \tfrac{y_1+y_2}{2}\right), \qquad \sigma_x = \alpha\,(x_2 - x_1), \quad \sigma$$

- **点奖励**：预测中心在目标高斯下的取值，离中心越远越小，平滑衰减；
- **覆盖奖励**：预测框对应的高斯和目标区域的重叠程度；
- **自适应方差**：σ 和元素宽高成正比（$\alpha = 0.5$）——小图标要求精确到几个像素，大按钮允许更大偏差；
- 总奖励 $R_{\text{total}} = \nu R_{\text{point}} + \gamma R_{\text{coverage}}$（默认 $\nu = \gamma = 1$）。

底座 Qwen2.5-VL-7B，8 张 A100 训一个 epoch：ScreenSpot 92.0、ScreenSpot-v2 93.3、**ScreenSpot-Pro 47.5**，比 UI-TARS-72B 最多高 24.7%。

小结：单步 grounding RL 学到了什么

工作	奖励	数据	关于推理的结论
UI-R1	动作类型 + 点中框（0/1）+ 格式	136 条	grounding 不需要推理
GUI-R1	动作类型 + 点 + 文本 + 格式	3K	同数据下 RL 明显好于 SFT
InfiGUI-R1	加子目标奖励、出错恢复场景	—	需要「先想后做」（偏多步任务）

工作	奖励	数据	关于推理的结论
GUI-G1	点中 / IoU + 框大小约束	17K	思维链越长定位越差
GUI-G²	连续高斯点奖励 + 覆盖奖励	—	—

三条共识：**数据可以很少**（几百到几万条就有效）；**奖励设计决定上限**（二值 → 防钻空子 → 连续稠密）；**单步定位不需要长思维链**。但这些都只是「一步」——奖励靠标注框免费得到，不需要环境。多步任务里，「这条轨迹算不算完成」只能在环境里执行后才知道，这才是下面几节的难点。

一句话：单步 grounding 是 CUA 里最容易做 RL 的部分——标注框就是免费的验证器。2025 年的进展集中在奖励形状：从「点中得 1 分」到加框大小约束防止钻空子，再到按元素大小自适应的高斯奖励；同时多篇发现定位任务不需要长思维链。

我的复述

自测

1. UI-R1 为什么用「点中框」而不用 IoU 作奖励?
2. GUI-G1 说「点中」和「IoU」奖励都能靠框的大小钻空子，模型具体可能怎么钻?
3. GUI-G² 的方差为什么要和元素宽高成正比？如果所有元素用同一个固定方差会怎样?

七、CUA × coding

八、对比与小结

附录：人与组谱系

按组追踪，随章节推进补充。机构取自论文首页署名；标「待核」的尚未在原文确认。

组 / 机构	核心人物（论文署名）	CUA 相关工作
Stanford（Percy Liang）	Evan Zheran Liu、Kelvin Guu、Panupong Pasupat	MiniWoB++ / WGE（2018.02）
Google DeepMind / Google Research	Kenton Lee、Mandar Joshi、Panupong Pasupat、Kristina Toutanova	Pix2Struct（2022.10）→ Pix2Act（2023.05）
	Izzeddin Gur、Hiroki Furuta、Aleksandra Faust（+ 东京大学 Yutaka Matsuo）	WebGUM（2023.05）、WebAgent / HTML-T5（2023.07）
	Christopher Rawles、Alice Li、Oriana Riva、Timothy Lillicrap	Android in the Wild（2023.07）
OSU NLP（Yu Su、Huan Sun）	Xiang Deng、Boyuan Zheng、Boyu Gou	Mind2Web（2023.06）→ SeeAct（2024.01）→ UGround（2024.10）

组 / 机构	核心人物（论文署名）	CUA 相关工作
CMU（Graham Neubig、Daniel Fried、Yonatan Bisk、Ruslan Salakhutdinov）	Shuyan Zhou、Frank F. Xu、Jing Yu Koh	WebArena（2023.07）→ VisualWebArena（2024.01）；Shuyan Zhou 也署名 OSWorld；Graham Neubig、Frank F. Xu 之后署名 OpenHands（2024.07，编程 agent 平台）[52]
HKU XLang（Tao Yu）	Tianbao Xie、Danyang Zhang（+ CMU、Salesforce、Waterloo）	OSWorld（2024.04）；之后 OSWorld-Verified、Jedi、OpenCUA（待核）
Microsoft Research Redmond（Jianfeng Gao）	Chunyuan Li、Jianwei Yang、Hao Zhang、Feng Li	SoM（2023.10）；Chunyuan Li 同时是 LLaVA 共同作者
UW–Madison（Yong Jae Lee）	Haotian Liu（+ Chunyuan Li）	LLaVA（2023.04）→ LLaVA-1.5（2023.10）→ LLaVA-NeXT（2024.01，博客）
ByteDance + NTU S-Lab（Ziwei Liu）	Bo Li、Yuanhan Zhang、Chunyuan Li、Feng Li、Hao Zhang	LLaVA-OneVision（2024.08）
Alibaba Qwen	Jinze Bai、Shuai Bai、Junyang Lin、Jingren Zhou	OFA（2022.02，Peng Wang、An Yang、Junyang Lin、Shuai Bai、Jingren Zhou 署名）→ Qwen-VL（2023.08）→ Qwen2-VL（2024.09）→ Qwen2.5-VL（2025.02）
Alibaba X-PLUG（+ 北京交通大学 Jitao Sang）	Haiyang Xu、Jiabo Ye、Ming Yan、Junyang Wang	mPLUG-Owl（2023.04）→ Mobile-Agent（2024.01）
清华 + 智谱（Jie Tang）	Wenyi Hong、Weihan Wang、Ming Ding、Yuxiao Dong	CogVLM（2023.11）→ CogAgent（2023.12）
ByteDance Seed	Yujia Qin、Haoming Wang、Junjie Fang、Guang Shi	UI-TARS（2025.01）→ UI-TARS-1.5（2025.04）→ UI-TARS-2（2025.09）
南京大学 + 上海 AI Lab	Kanzhi Cheng、Zhiyong Wu、Qiushi Sun	SeeClick（2024.01）→ OS-Atlas（2024.10）；Kanzhi Cheng 后署名 GUI-Actor（2025.06，Microsoft）
Tencent	Chi Zhang、Gang Yu	AppAgent（2023.12）
Tencent AI Lab + 浙大 / 西湖大学	Hongliang He、Dong Yu	WebVoyager（2024.01）
SenseTime + 北航	Keqin Chen、Zhao Zhang	Shikra（2023.06）

XLang 组（Tao Yu）谱系：从 benchmark 到 Qwen-CUA

只按 arXiv 作者列表整理；机构取自论文首页。

时间	工作	一作	组内其他署名	合作方	做了什么
2022.10	Binder[53]	Zhoujun Cheng	Tianbao Xie	—	LLM 绑定符号语言（SQL / Python）
2023.09	Text2Reward[54]	Tianbao Xie	—	—	用 LLM 为 RL 写奖励函数（第六章会对照）
2023.10	Lemur[55]	Yiheng Xu	Zhoujun Cheng	Salesforce	兼顾自然语言和代码的 agent 底座模型
2023.10	OpenAgents[56]	Tianbao Xie	Zhoujun Cheng	Salesforce	数据 / 插件 / Web 三种 agent 的开放平台
2024.04	**OSWorld**[1]	Tianbao Xie	Danyang Zhang、Yiheng Xu、Zhoujun Cheng	CMU（Shuyan Zhou）、Salesforce、Waterloo	真实电脑环境 + 执行式评测
2024.07	Spider2-V[57]	Ruisheng Cao	Danyang Zhang、Tianbao Xie	—	数据科学与工程工作流的多模态 agent 基准，494 题
2024.12	**Aguvis**[58]	Yiheng Xu	Junli Wang、Dunjie Lu、Tianbao Xie	Salesforce	纯视觉 GUI agent；先 grounding 再规划推理两阶段；以 **Qwen2-VL 为骨干**，理由是其 NaViT 式编码器原生支持动态分辨率
2024.12	AgentTrek[59]	Yiheng Xu	Dunjie Lu、Junli Wang	Salesforce	从网上教程合成 web agent 轨迹，VLM 执行、VLM 评判

时间	工作	一作	组内其他署名	合作方	做了什么
2025.02	**Qwen2.5-VL**[26]	（Qwen 团队）	**Yiheng Xu**、**Tianbao Xie** 在作者列表中	Alibaba Qwen	agent 数据：每步推理过程的标注方式引用 Xu et al. 2024（即 Aguvis）；agent 表格里拿 Aguvis-72B 作对比
2025.05	Jedi / OSWorld-G[60]	Tianbao Xie	Junli Wang、Xinyuan Wang	Salesforce	400 万条 grounding 合成数据 + 564 条 grounding 基准
2025.07	OSWorld-Verified[4]	—	—	多家机构反馈	修复约 300 个问题、上 AWS
2025.08	**OpenCUA**[61]	Xinyuan Wang	Tianbao Xie、Junli Wang、Dunjie Lu	**Moonshot AI**（Zhilin Yang、Flood Sung）	标注工具 + AgentNet 数据集（3 个 OS、200+ 应用）；骨干为 **Qwen2.5-VL-7B / 32B / 72B**，也试了 Qwen2-VL 和 Kimi-VL-A3B
2025.10	VideoAgentTrek[62]	Dunjie Lu	Yiheng Xu、Junli Wang	**Qwen 团队**（Jingren Zhou、Junyang Lin、Binyuan Hui 署名）	从 3.9 万个 YouTube 教程视频自动挖出 152 万步交互数据，用于 computer use 预训练
2025.11	Qwen3-VL[63]	（Qwen 团队）	Tianbao Xie、Yiheng Xu 在作者列表中	Alibaba Qwen	—
2026.02	Kimi K2.5	—	Xinyuan Wang、Tao Yu 在作者列表中	Moonshot AI	—

时间	工作	一作	组内其他署名	合作方	做了什么
2026.05	CUA-Gym[64]	Bowen Wang	Dunjie Lu、Junli Wang	Salesforce、**Qwen** 团队（Junyang Lin、Dayiheng Liu）	自动同时生成任务、环境状态和奖励函数；在 Qwen3.5 MoE 上做 RLVR，OSWorld-Verified 72.6%
2026.06	OSWorld 2.0[43]	Mengqi Yuan	Tianbao Xie、Tao Yu	—	长程工作流基准
2026.08	**Qwen-CUA**[65]	Dunjie Lu	Tianbao Xie、Yiheng Xu、Danyang Zhang、Junli Wang、Tao Yu	**Qwen** 团队，**Alibaba**	397B-A17B 原生 computer-use 模型；近 10 万 vCPU、几万个并发环境、约 4 万个可验证任务

这条线的结构：benchmark（OSWorld）→ grounding 数据（Jedi）→ 开放模型与数据（Aguvis、OpenCUA）→ 轨迹合成（AgentTrek、VideoAgentTrek）→ RL 环境（CUA-Gym）→ 产品级模型（Qwen-CUA）→ 新 benchmark（OSWorld 2.0）。同一批人把「评测—数据—训练—模型」整条链都做了一遍。

和 Qwen2.5-VL、OpenCUA 的关系：

1. Aguvis（2024.12）建在 **Qwen2-VL** 上；
2. Qwen2.5-VL（2025.02）的作者列表里有 Yiheng Xu、Tianbao Xie，agent 数据的推理标注方式沿用 Aguvis，并在 agent 评测表里直接和 Aguvis-72B 比（ScreenSpot：Aguvis 89.2、Qwen2.5-VL 87.1；OSWorld：Aguvis 10.26、Qwen2.5-VL 8.83）；
3. OpenCUA（2025.08）再以 **Qwen2.5-VL** 为骨干，并和 Moonshot 合作；
4. 之后 VideoAgentTrek、CUA-Gym 有 Qwen 核心成员署名，最终 Qwen-CUA 由 Qwen 团队发布、XLang 成员署名。

也就是一个「开源基模 → 学术组做 CUA 数据和模型 → 经验回流进下一代基模」的循环。

合作方的变化：2023–2025 年主要和 Salesforce（Caiming Xiong）合作；2025 年起分别与 Moonshot（OpenCUA、Kimi K2.5）和 Qwen（Qwen2.5-VL、Qwen3-VL、VideoAgentTrek、CUA-Gym、Qwen-CUA）合作。

两位贯穿这条线的作者：Junli Wang 署名了 Aguvis、AgentTrek、Jedi、OpenCUA、VideoAgentTrek、CUA-Gym、Qwen-CUA，几乎贯穿这条线的数据和训练部分；Zhoujun Cheng 署名了 Binder、Lemur、OpenAgents、OSWorld，之后有 CocoaBench（2026.04，与 Junli Wang 同署名）。

已能看出的连线：

- **Panupong Pasupat**：Stanford 的 MiniWoB++（2018）→ Google DeepMind 的 Pix2Act（2023）。

- **OSU NLP**：Mind2Web 的作者 Boyuan Zheng 一作了 SeeAct，同一组从「读 HTML」走到「GPT-4V 看截图」。
- **Shuyan Zhou**：WebArena 一作，也在 OSWorld 作者列表里——「可交互环境 + 程序检查」范式从 web 搬到整台电脑，有人员上的直接连接。
- **Zhe Gan**、**Yinfei Yang**（Apple）：Ferret（2023.10）→ Ferret-UI（2024.04）→ MM1（2024.03），通用 VLM 和移动 UI 模型出自同一圈人。
- **Yiheng Xu**：MarkupLM（MSRA，2021）第二作者，OSWorld（2024）作者列表里也有同名作者，是否同一人待核。
- **Chunyuan Li**：LLaVA、LLaVA-1.5、SoM、LLaVA-NeXT、LLaVA-OneVision 都署名，横跨 Microsoft Research 和 ByteDance 两段。

参考资料

[1] Xie et al.: OSWorld: Benchmarking Multimodal Agents for Open-Ended Tasks in Real Computer Environments（§2.1–2.4、§4.1–4.2）

[2] Zheng et al.: GPT-4V(ision) is a Generalist Web Agent, if Grounded（SeeAct）

[3] Qin et al.: UI-TARS: Pioneering Automated GUI Interaction with Native Agents

[4] XLANG Lab: Introducing OSWorld-Verified（2025-07-28）

[5] Liu et al.: Reinforcement Learning on Web Interfaces Using Workflow-Guided Exploration（MiniWoB++）

[6] Deng et al.: Mind2Web: Towards a Generalist Agent for the Web

[7] OpenAI: GPT-4V(ision) system card（2023-09-25）

[8] Zhou et al.: WebArena: A Realistic Web Environment for Building Autonomous Agents

[9] Rawles et al.: Android in the Wild: A Large-Scale Dataset for Android Device Control

[10] Gur et al.: A Real-World WebAgent with Planning, Long Context Understanding, and Program Synthesis

[11] Shaw et al.: From Pixels to UI Actions: Learning to Follow Instructions via Graphical User Interfaces（Pix2Act）

[12] Furuta et al.: Multimodal Web Navigation with Instruction-Finetuned Foundation Models（WebGUM）

[13] Yang et al.: Set-of-Mark Prompting Unleashes Extraordinary Visual Grounding in GPT-4V

[14] Zhang et al.: AppAgent: Multimodal Agents as Smartphone Users（§3.1）

[15] Wang et al.: Mobile-Agent: Autonomous Multi-Modal Mobile Device Agent with Visual Perception

[16] He et al.: WebVoyager: Building an End-to-End Web Agent with Large Multimodal Models

[17] OpenAI: GPT-4（2023-03-14）

[18] OpenAI: New models and developer products announced at DevDay（2023-11-06）

[19] Google: Introducing Gemini（2023-12-06）

[20] Anthropic: Introducing the next generation of Claude（2024-03-04）

[21] OpenAI: Hello GPT-4o（2024-05-13）

[22] Anthropic：Introducing Claude 3.5 Sonnet（2024-06-21）

[23] OpenAI：Introducing OpenAI o1（2024-09-12）

[24] Anthropic：Computer use, Claude 3.5 Sonnet, and Claude 3.5 Haiku（2024-10-22）

[25] OpenAI：Computer-Using Agent（2025-01-23）

[26] Bai et al.：Qwen2.5-VL Technical Report；博客 Qwen2.5 VL（2025-01-26）

[27] OpenAI：Introducing OpenAI o3 and o4-mini（2025-04-16）

[28] Li et al.：MarkupLM: Pre-training of Text and Markup Language for Visually-rich Document Understanding

[29] Lee et al.：Pix2Struct: Screenshot Parsing as Pretraining for Visual Language Understanding

[30] Hong et al.：CogAgent: A Visual Language Model for GUI Agents

[31] Baechler et al.：ScreenAI: A Vision-Language Model for UI and Infographics Understanding

[32] Li et al.：ScreenSpot-Pro: GUI Grounding for Professional High-Resolution Computer Use

[33] Wu et al.：GUI-Actor: Coordinate-Free Visual Grounding for GUI Agents

[34] Jiang et al.：Zoom in, Click out: Unlocking and Evaluating the Potential of Zooming for GUI Grounding（ZoomClick）

[35] Cheng et al.：SeeClick: Harnessing GUI Grounding for Advanced Visual GUI Agents

[36] Wu et al.：OS-ATLAS: A Foundation Action Model for Generalist GUI Agents

[37] Gou et al.：Navigating the Digital World as Humans Do: Universal Visual Grounding for GUI Agents（UGround）

[38] Koh et al.：VisualWebArena: Evaluating Multimodal Agents on Realistic Visual Web Tasks

[39] Anthropic：Introducing Claude Sonnet 4.5（2025-09-29）

[40] Anthropic：Introducing Claude Sonnet 4.6（2026-02-17）

[41] OpenAI：Introducing GPT-5.4（2026-03-05）

[42] Anthropic：Introducing Claude Opus 4.8（2026-05-28）

[43] Yuan et al.：OSWorld 2.0: Benchmarking Computer Use Agents on Long-Horizon Real-World Tasks

[44] Anthropic：Developing a computer use model（2024-10-22）

[45] ByteDance Seed：UI-TARS GitHub README（UI-TARS-1.5，2025-04-16）

[46] Wang et al.：UI-TARS-2 Technical Report: Advancing GUI Agent with Multi-Turn Reinforcement Learning

[47] Lu et al.：UI-R1: Enhancing Efficient Action Prediction of GUI Agents by Reinforcement Learning

[48] Luo et al.：GUI-R1: A Generalist R1-Style Vision-Language Action Model For GUI Agents

[49] Liu et al.：InfiGUI-R1: Advancing Multimodal GUI Agents from Reactive Actors to Deliberative Reasoners

[50] Zhou et al.: GUI-G1: Understanding R1-Zero-Like Training for Visual Grounding in GUI Agents

[51] Tang et al.: GUI-G²: Gaussian Reward Modeling for GUI Grounding

[52] Wang et al.: OpenHands: An Open Platform for AI Software Developers as Generalist Agents

[53] Cheng et al.: Binding Language Models in Symbolic Languages（Binder）

[54] Xie et al.: Text2Reward: Reward Shaping with Language Models for Reinforcement Learning

[55] Xu et al.: Lemur: Harmonizing Natural Language and Code for Language Agents

[56] Xie et al.: OpenAgents: An Open Platform for Language Agents in the Wild

[57] Cao et al.: Spider2-V: How Far Are Multimodal Agents From Automating Data Science and Engineering Workflows?

[58] Xu et al.: Aguvis: Unified Pure Vision Agents for Autonomous GUI Interaction

[59] Xu et al.: AgentTrek: Agent Trajectory Synthesis via Guiding Replay with Web Tutorials

[60] Xie et al.: Scaling Computer-Use Grounding via User Interface Decomposition and Synthesis（Jedi / OSWorld-G）

[61] Wang et al.: OpenCUA: Open Foundations for Computer-Use Agents

[62] Lu et al.: VideoAgentTrek: Computer Use Pretraining from Unlabeled Videos

[63] Bai et al.: Qwen3-VL Technical Report

[64] Wang et al.: CUA-Gym: Scaling Verifiable Training Environments and Tasks for Computer-Use Agents

[65] Lu et al.: Qwen-CUA: Native Computer Use for (almost) Everything