UI-TARS调研

1. 简介

UI-TARS：基于**视觉语言模型**（Vision-Language Model）构建的原生GUI代理(Agent)模型

**主要功能**

自然语言控制：通过自然语言指令控制电脑操作，简化用户交互。

视觉识别支持：支持截图和视觉识别功能，能够识别屏幕内容并执行相应操作。

精准控制：提供精确的鼠标和键盘控制，确保操作的准确性。

跨平台支持：支持 Windows 和 MacOS 系统，满足不同用户的需求。

实时反馈：提供实时反馈和状态显示，帮助用户了解指令执行情况。

**项目地址**：[GitHub - bytedance/UI-TARS](https://github.com/bytedance/UI-TARS)与[GitHub - bytedance/UI-TARS-desktop](https://github.com/bytedance/UI-TARS-desktop)

前者讲述UI-TARS论文和模型效果评估，后者是基于UI-TARS的GUI代理应用程序，可在本地个人设备上运行（也可直接下载客户端使用）

1. UI-TARS论文

UI-TARS整体架构->增强感知与动作能力->System2推理能力注入->经验学习迭代

2.1整体结构

给定初始任务instruction，UI-TARS迭代地接收来自设备的观察observation（即截图），并执行相应的动作action来完成任务。该过程可表示为：

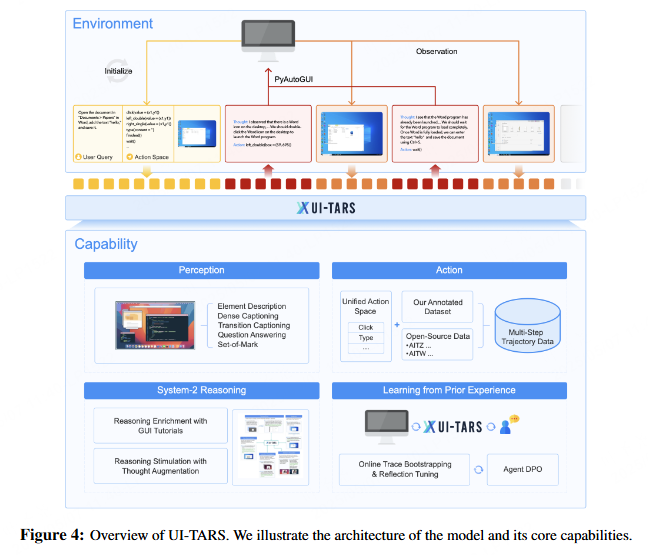
（instruction,(o1,a1),(o2,a2),…,(on,an)）,其中oi表示时间步i的观察(设备截图)，ai表示agent执行的动作。

在每个时间步timestep，UI-TARS将任务指令，先前交互的历史（o1,a1,o2,…,oi-1,ai-1）和当前的观察oi作为输入。基于该输入，模型从预定义的动作空间中输出一个动作ai。该过程迭代进行。

为了增强agent推理能力，构建了推理组件”thoughts”ti,在每个动作ai之前完成。这些ti反映了system2的反思性质，作为中介步骤引导agent在动作之前重新考虑以前的行动和观察。(instruction,(o1,t1,a1),…,（on,tn,an）)

为了优化内存使用和在有限token预算下保持效果，UI-TARS将输出限制在最后N个观察，以前的动作a和想法t的完整历史作为短期记忆保留。UI-TARS迭代预测tn，an：





2.2 GUI感知增强

与依赖独立的模块化感知模型的传统框架不同，native agent可直接处理来自GUI屏幕截图的原始输入克服(1)特定的GUI屏幕截图稀缺性（2）高信息密度和精度要求；这使得它们能够通过利用大规模、统一的数据集扩展功能。

**截屏收集**：构建大规模数据集，使用专门解析工具渲染屏幕截图提取元数据。数据以结构化格式（屏幕截图，元素框，元素元数据）记录，采用自下而上数据构建方法，从单个元素发展到整体界面理解。  
 **元素描述**：基于元数据由VLM进一步合成，涵盖元素类型、视觉描述、位置信息、元素函数（训练UI-TARS枚举屏幕截图中的所有可见元素，并根据屏幕截图生成其元素描述）

**密集字幕(dense caption)**：目标是提供全面详细的GUI屏幕截图描述，训练UI-TARS理解整个界面时保持准确性并减少幻觉。训练期间仅提供图像给UI-TARS，并输出对应的密集字幕。

**状态转换字幕**：密集字幕提供了GUI界面的全面描述但并未捕捉状态转换（例如按下一个微小按钮）。屏幕截图对来训练，即训练模型识别和描述连续屏幕截图间的差异。

**问答（QA）**：合成一组涵盖广泛任务的多样化QA数据，包括界面理解、图像解释、元素识别和关系推理，增强UI-TARS处理涉及更高抽象和推理程度的查询能力。

**标记集（SoM）**：根据GUI屏幕截图上解析的元素的空间坐标为其绘制视觉上的不同标记，为模型定位和识别特定元素提供清晰直观的视觉提示。

2.3 一致动作建模和基础

提高UI-TARS的行动能力，训练模型在任务执行中模仿人类行为。

该模块主要介绍了

构建一致动作空间：实现跨平台知识转移、跨场景一致性；

动作跟踪收集：应对任务执行训练模型的重大挑战----多步骤轨迹数据的可用性有限，历史上一直记录不足且稀疏。

提高接地(grounding)能力：准确定位并与特定GUI元素交互能力，对于点击或拖动能力至关重要。

2.4 注入系统2推理

通过GUI教程数据丰富推理：利用交织文本和图像的公开教程来演示跨不同软件和Web环境的详细用户交互。为了提取高质量的教程数据，采取以下步骤：

a.粗粒度过滤: 手动策划高质量教程的正集和来自MINT和OmniCorpus的随机样本作为负集训练了一个快速文本分类器。然后应用训练好的分类器进行初始筛选，过滤掉不相关的样本并生成候选数据集。

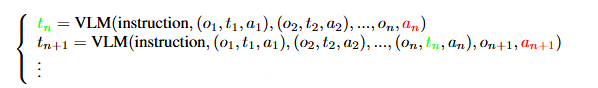
b.细粒度过滤：为进一步细化候选数据集，使用LLM来识别和去除误报，确保剩余的样本符合GUI教程的特征。粗略和精细过滤过程经过多轮迭代，以最大限度地提高高质量GUI教程的召回率。

c. 重复数据删除和数据细化：过滤后的数据集进一步细化，以解决重复、广告和残余噪声问题。重复数据删除使用基于URL和局部敏感散列（LSH）方法进行。最后，通过提示LLM重新措辞教程中的所有文本内容，细化内容，同时删除不相关或低质量的内容。

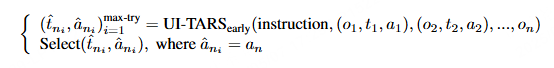
最终策划了大约6M高质量的GUI教程。平均而言，每个教程包含510个文本标记和3.3个图像。这些数据不仅增强了模型对GUI操作的理解，还为注入推理能力奠定了坚实的基础。

思维增强的推理刺激：该部分主要讲解2.1中的“thoughts”ti如何构建，如何通过思维注释增强数据集。构建thoughts采用了两个阶段：ActRe，Thought Boostrapping。

ActRe：对于每个2.3中收集的跟踪，将其拆分为多个步骤。对于每个步骤n，通过提示具有前一个上下文和当前目标动作an的VLM来迭代生成其思想tn。



Thought Bootstrapping： ActRe的反向注释可能会导致误报，因为生成的想法t可能与动作a匹配浅，通过抽样多个思想-行动对，如下所示，识别出导致正确行动的思想，确保推理与所选择的行动因果一致。

****

2.5 从长期记忆的先验经验学习

将GUI agent扩展到LLM层面面临挑战，因为GUI操作的大规模、标准化、真实世界的处理数据稀缺，详细描述GUI环境中用户交互和决策序列的流程数据很少被记录或组织，这阻碍了GUI agent在广泛任务中的拓展与泛化能力。UI-TARS通过半自动化的数据采集、过滤和细化改进模型。利用长期记忆建立UI-TARS的累积知识，随时间推移改进性能。

反思(reflection)调整：在现实应用中，agent常会由于缺乏自我反思和纠错能力陷入困难（如反复点击无效按钮），为解决该问题，文章提出了一种反映调整协议：假设在t步发生错误，动作at被认为无效，需要识别该错误并标记纠正后的想法t和动作a；创新的是，模型基于错误动作继续进行后续步骤，模拟错误已经发生的场景。当确定下一步的想法时，必须补偿前一步的错误（如将网页加入书签但错误关闭网页，下一步应该重新打开关闭的网页作为补偿）；利用正样本SFT,仅计算校正后的损失，错误步骤不用于训练。

Agent DPO：在bootstrapping过程中会产生大量错误步骤，而SFT只利用了正确样例，故而采用DPO引入修正和错误动作。定义奖励函数，评估在状态s中采取行动a的可取性（s由指令及当前步骤的历史交互组成）。



其中r是奖励函数，≻ 采取前者行动优于后者

2.6 训练

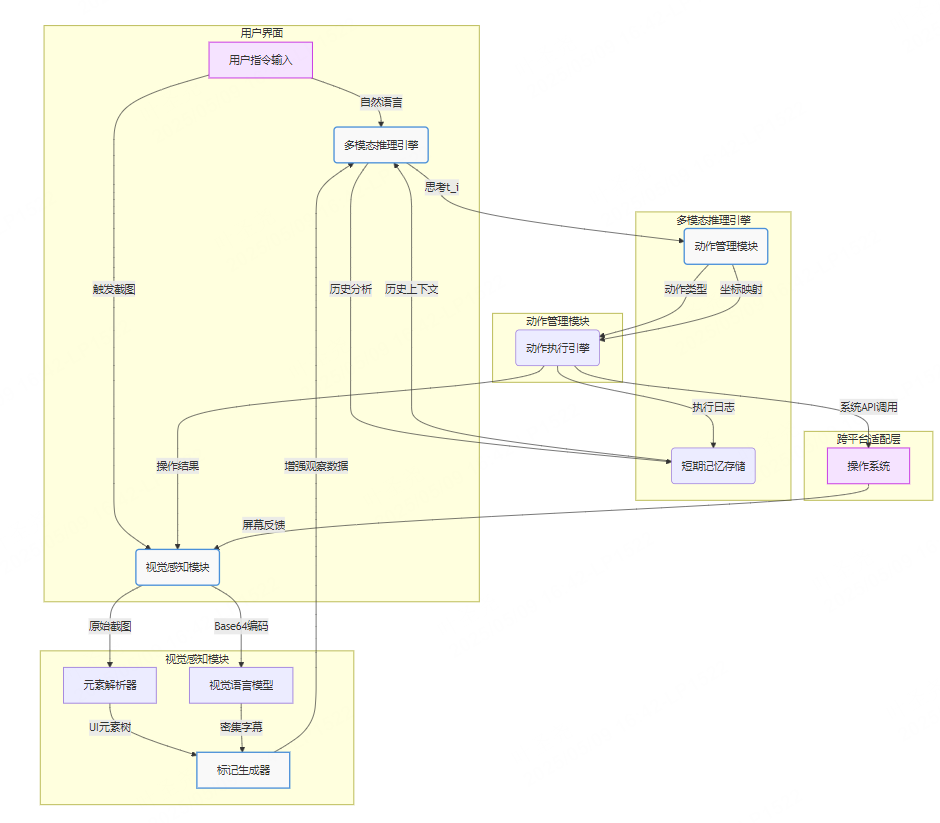
采用三阶段训练过程，在不同GUI任务中细化模型能力，总计使用50B tokens。

持续预训练阶段：以恒定学习率预训练，该基础阶段允许模型学习自动化GUI交互的所有必要知识，包括感知、接地grounding和动作跟踪action traces，确保跨不同GUI元素和交互的稳健覆盖。

退火阶段：选择感知、接地、动作跟踪、反思reflection的高质量子集进行退火，逐渐动态调整模型学习，促使其更加专注于真实GUI交互场景的决策策略。此阶段之后的训练模型表示为UI-TARS-SFT。

DPO阶段：该阶段的训练加强最优行为，同时惩罚了次优行为。这个过程提高了模型在现实世界GUI交互中做出精确、上下文感知决策的能力。最终模型表示为UI-TARS-DPO（也为目前服务器中所部署的模型）

3.算法流程图



(供参考，欢迎指正)

graph TD

subgraph 用户界面

UI[用户指令输入] -->|自然语言| MM(多模态推理引擎)

UI -->|触发截图| VP(视觉感知模块)

end

subgraph 视觉感知模块

VP -->|原始截图| OCR[元素解析器]

VP -->|Base64编码| VLM[视觉语言模型]

OCR -->|UI元素树| SOM[标记生成器]

VLM -->|密集字幕| SOM

SOM -->|增强观察数据| MM

end

subgraph 多模态推理引擎

MM -->|思考t\_i| AM(动作管理模块)

MM -->|历史分析| SM(短期记忆存储)

SM -->|历史上下文| MM

end

subgraph 动作管理模块

AM -->|动作类型| AE(动作执行引擎)

AM -->|坐标映射| AE

AE -->|操作结果| VP

AE -->|执行日志| SM

end

subgraph 跨平台适配层

AE -->|系统API调用| OS[操作系统]

OS -->|屏幕反馈| VP

end

classDef module fill:#f9f9f9,stroke:#4a90d9,stroke-width:2px;

classDef dataflow fill:#fff,stroke:#7ed321,stroke-dasharray:5 5;

classDef interface fill:#f5e3ff,stroke:#bd10e0;

class UI,OS interface;

class VP,MM,AM,SOM module;

linkStyle 0,2,3,5,6,7,8,9,10,11,12 dataflow;

1. 部署

云端部署：

首先需要在huggingface部署UI-TARS，huggingface作为VLM提供商，提供API与URL等VLM参数，将其配置到已下载的客户端设置后可以正常使用（不推荐，需要在huggingface租GPU，4$/h）

本地部署：

（vLLM或ollama）下载模型，启动兼容OpenAI的API服务，使用代码与模型进行交互。（客户端的VLM提供商基于huggingface，能否在客户端直接配置API与URL使用存疑）

以下介绍vLLM本地部署：

1. 部署前准备：
   1. 硬件要求：

GPU：推荐 NVIDIA 显卡（如 A100、H100、L40S、RTX 4090），显存 ≥ 24GB（7B模型单卡，72B需多卡）

CUDA：需确保已安装与显卡驱动的CUDA（vLLM要求CUDA ≥ 11.8）

* 1. 虚拟环境：

conda create -n ui-tars python=3.10 -y conda activate ui-tars

conda activate ui-tars

pip install vllm==0.0.6 --extra-index-url https://download.pytorch.org/whl/cu121

1. 模型下载：

git lfs install

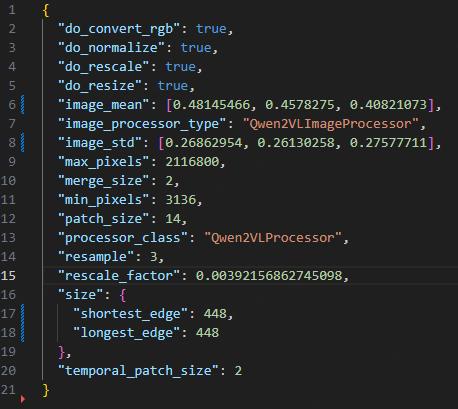
git clone https://huggingface.co/bytedance-research/UI-TARS-7B-DPO

# Hugging Face 格式：直接加载原始模型目录。

1. 模型启动

Q1由于单张RTX4090显存不够，需要多卡部署，启用原生张量并行；模型有28个注意力头，故而GPU数需要能整除28

Q2 原模型配置中多模态模型（Qwen2VL）图像处理器不兼容，需做如下处理：

1. 启用 trust\_remote\_code(已在运行指令中添加)
2. 修改 preprocessor\_config.json，替换为以下文件：

Q3 client配置：

base\_url=<http://localhost:8000/v1>

api\_key任意非空字符

终端激活ui-tars虚拟环境，运行：

CUDA\_VISIBLE\_DEVICES=0,1,2,3 python -m vllm.entrypoints.openai.api\_server   --model /root/yesy/UI-TARS-7B-DPO   --served-model-name UI-TARS-7B-DPO   --trust-remote-code   --tensor-parallel-size 4

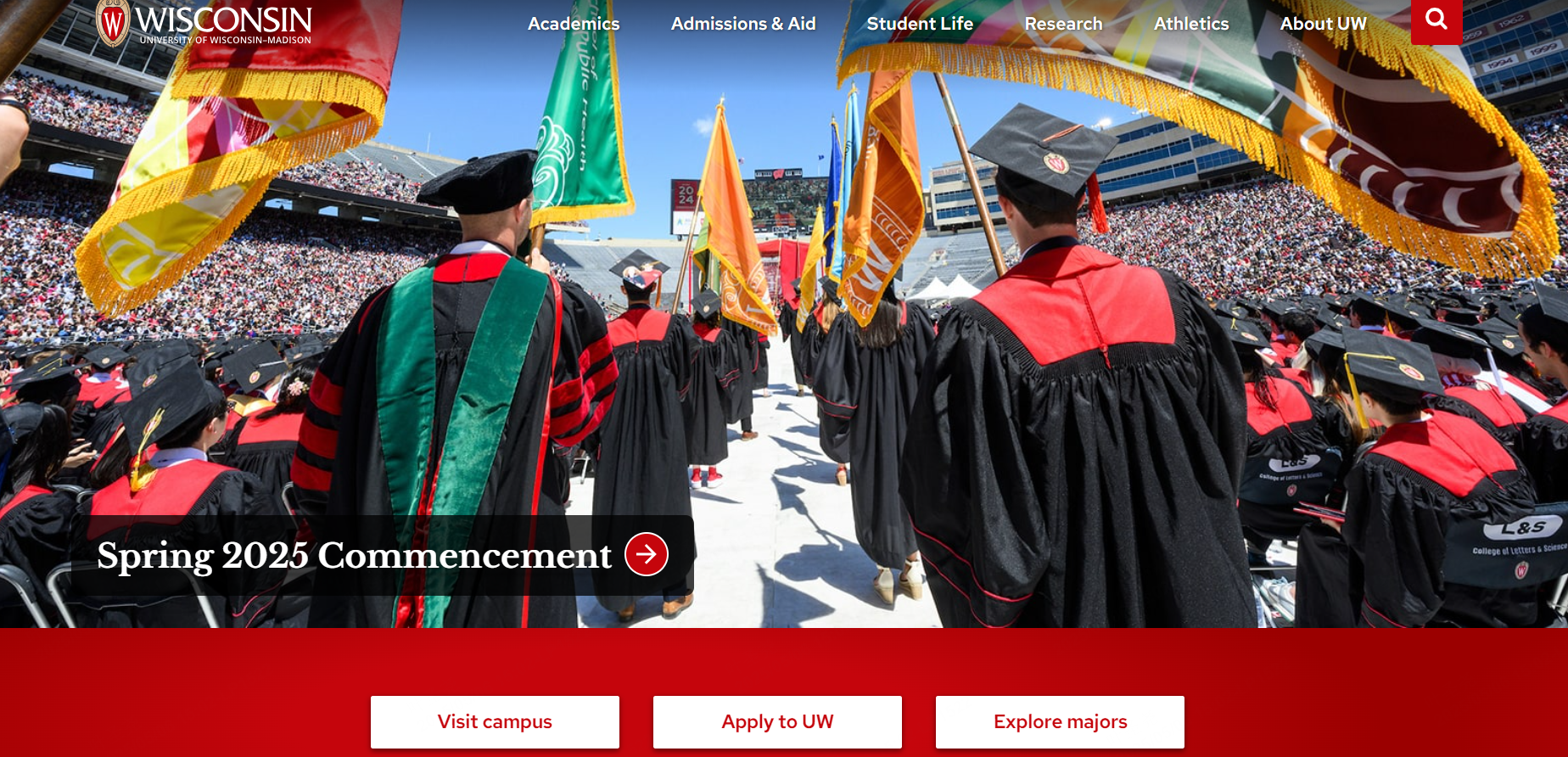
三、部署验证

1.启动模型运行文件test\_api.py：

instruction = "Help me visit the campus"

screenshot\_path = "/root/yesy/UI-TARS-7B-DPO/test\_chat/UWMadison.png"

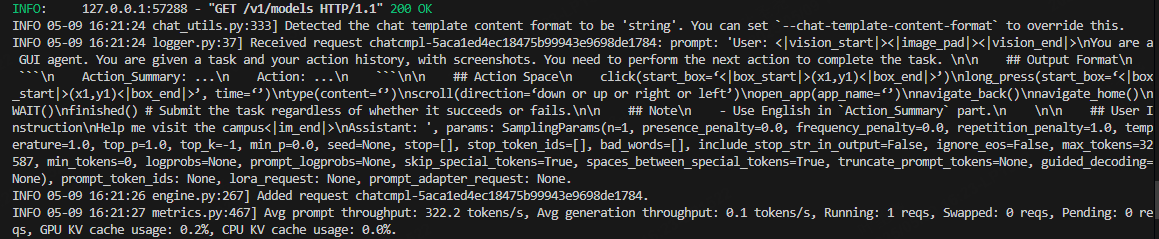
屏幕截图：



结果如此下，模型输出第一步的操作描述与Action



模型运行情况：



部署成功

1. 结合论文理论部分：

屏幕截图为o，Action为a，操作描述为t。

具体场景使用：UI-TARS会处理屏幕截图，参考t给出Action并执行；对新页面截屏然后迭代该过程。

1. 完整功能