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

Anonymous ACL submission

Abstract

Graphical User Interface (GUI) agents powered by Vision-Language Models (VLMs) have demonstrated human-like computer control capability. Despite their utility in advancing digital automation, the development of such agents faces a critical bottleneck: collecting high-quality trajectory data for training. Common practices for collecting such data rely on human supervision or synthetic data generation through executing pre-defined tasks, which are either resource-intensive or unable to guarantee data quality. Further, these approaches exhibit significant gaps between the generated data and online environments, alongside limited data diversity. To address this issue, we introduce *OS-Genesis*, a novel GUI data synthesis pipeline that overcomes the challenges above. Unlike prior methods that rely on preset tasks, *OS-Genesis* reverse engineers the GUI trajectory construction process. Agents first perceive environments and perform step-level interactions, then retrospectively derive high-quality tasks to enable trajectory-level exploration. A trajectory reward model is then employed to ensure the quality of the generated trajectories. We demonstrate that training GUI agents with *OS-Genesis* significantly improves their performance on highly challenging online benchmarks. In-depth analysis further validates *OS-Genesis*'s cost-effectiveness and its superior data quality and diversity compared to existing synthesis methods.

1 Introduction

Recent advancements in Vision-Language Models (VLMs; Chen et al., 2024b; Wang et al., 2024b) have driven researchers to build a variety of language agents (Sumers et al., 2024). As an emerging class of AI systems, these agents are being explored for their potential to automate complicated computer tasks on Graphical User Interfaces (GUIs), aiming to achieve digital automation (Anthropic, 2023; Hu et al., 2024). To complete GUI tasks autonomously, an agent must possess key capabilities: understanding user intentions, planning tasks, and executing actions. Therefore, using high-quality trajectories for training is essential for improving their agentic capabilities (Zheng et al., 2024c).

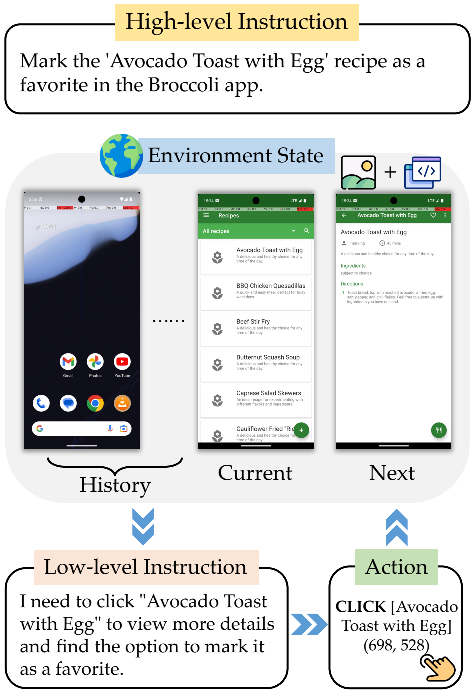


Figure 1: Ideal GUI trajectory format, including High-Level Instructions, States (visual + textual representation), Low-Level Instructions, and Actions.

As illustrated in Figure 1, ideal GUI agent trajectories contain the following key components: (1) a high-level instruction that defines the overall goal the agent aims to accomplish, (2) a series of low-level instructions that each describe specific steps required, (3) actions (*e.g.*, `CLICK`, `TYPE`) and (4) states, which include visual representations like screenshots and textual representations such as `a11ytree`[1]. Such data enable end-to-end training

[1] `a11ytree`: Accessibility (`a11y`) trees are informative structures in software or web applications, each `a11ytree` node corresponds to a UI element on the screen.

of GUI agents, extending their capabilities from automating actions (Cheng et al., 2024) to achieving full-process autonomy (Zhang et al., 2024a).

However, collecting such trajectories is far from trivial. Existing *task-driven* methods, which rely on humans or machines executing predefined tasks, face the following limitations: human collection requires annotators to label entire trajectories and predefine high-level tasks manually (Li et al., 2024; Lù et al., 2024), making it both costly and labor-intensive. Model-based synthesis also faces critical challenges: (1) it heavily depends on pre-defined high-level tasks (Lai et al., 2024), which not only limit the scalability of synthesized data but also constrain its diversity; and (2) it relies on a model having strong agentic ability and domain-specific app knowledge—both of which are often inadequate (Murty et al., 2024b; Patel et al., 2024). Above mentioned issues pose a bottleneck for advancing GUI agents. These issues lead to a critical bottleneck for advancing GUI agents. Thus, effective trajectory construction methods are a clear desideratum to address these challenges.

In this paper, we present *OS-Genesis*, a pipeline for synthesizing high-quality and diverse GUI agent trajectories without involving human supervision or pre-defined tasks. Recognizing the limitations of the aforementioned *task-driven* methods, we draw inspiration from how humans learn to interact with GUI applications and adopt an *interaction-driven* approach. *OS-Genesis* begins by exploring the functionality of GUI environments through traversing interactive UI elements with actions (*e.g.*, `CLICK`). This forms the basis for reverse task synthesis, where observed states and actions are retroactively transformed into low-level instructions. These low-level instructions are then derived into high-level instructions, which can seed the collection of GUI trajectories. By uncovering considerable functionalities, reverse task synthesis facilitates the creation of meaningful and executable tasks. Moreover, it naturally bridges the gap between abstract instructions and the dynamic nature of GUIs. Once synthesized tasks are explored and transformed into trajectories, we leverage a trajectory reward model to ensure data quality.

Experiments on two challenging online benchmarks, AndroidWorld and WebArena, demonstrate the effectiveness of *OS-Genesis*. It surpasses task-driven methods by a large margin, nearly doubling the performance from 9.82% to 17.41% on AndroidWorld. This highlights the high quality of the agent trajectories synthesized by *OS-Genesis* and its great potential to transform general-purpose VLMs into specialized GUI agents.

Our primary contributions are as follows:

- By shifting from *task-driven* approaches to *interaction-driven* GUI agent data construction, we introduce reverse task synthesis to improve trajectory quality and diversity.
- We propose a novel pipeline, *OS-Genesis*, capable of efficiently synthesizing high-quality trajectory data. Without human supervision, *OS-Genesis* supports end-to-end training of GUI agents across environments.
- Extensive experiments across mobile and web tasks on dynamic benchmarks demonstrate the superior performance of *OS-Genesis* over a suite of strong baselines.

2 Related Works

Agents for Digital Automation. The recent proliferation of LLMs has significantly boosted researchers' interest in developing language agents (Durante et al., 2024) to explore the digital world (Feng et al., 2024). One line of work leverages the capabilities of fixed LLMs to create agents using methods like prompt engineering, model collaboration (Wu et al., 2023; Sun et al., 2023), code or tool use (Sun et al., 2024), self-improvement (Shinn et al., 2024; Wu et al., 2024a), or integration with world or agent models (Hu and Shu, 2023; Jin et al., 2024; Zhang et al., 2023). Another line focuses on fine-tuning to augment models with agentic abilities, including (1) the ability to perceive the state of the computer, such as understanding screens or application UI trees (Xie et al., 2024; Zheng et al., 2024a; Cheng et al., 2024; Wu et al., 2024b), (2) the ability to generate actions (click, type, scroll, *etc*. Chen et al., 2024a), and (3) the flexibility to operate across diverse environments, including web (Yao et al., 2022; Deng et al., 2023), desktop (Kapoor et al., 2024; Niu et al., 2024), and mobile platforms (Li et al., 2024; Wang et al., 2024a). Collectively, these efforts pave the way for digital automation, with general-purpose agents engaging across a diverse digital landscape.

Data for Building Computer Agents. High-quality data are essential for building general-purpose computer agents, enabling VLMs to propose plans, execute appropriate actions, and navigate themselves across diverse environments (Zeng et al., 2024; Pan et al., 2024). Rico (Deka et al.,

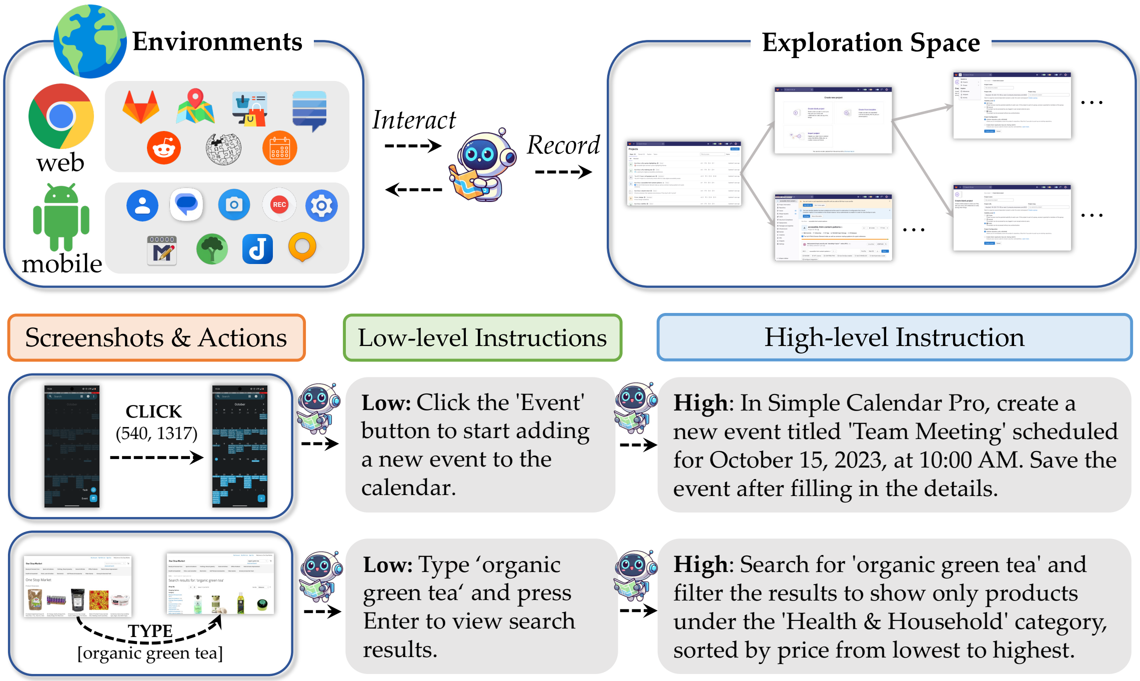


Figure 2: An overview of how we generate instruction data without relying on predefined tasks or human annotations. *OS-Genesis* begins with a model-free, interaction-driven traversal in online environments (*e.g.*, a web browser). This process produces massive triples consisting of actions and their corresponding pre- and post-interaction screenshots. Reverse task synthesis leverages these triples to generate low-level instructions and associates them with broader objectives to construct high-level instructions.

2017) first introduces sequential GUI data for mobile apps, while MiniWob (Shi et al., 2017) provides low-level keyboard and mouse actions for web-based tasks. Since then, several works have expanded the availability of such data for mobile (Rawles et al., 2023; Zhang et al., 2024b; Lu et al., 2024; Chai et al., 2024), web (Liu et al., 2018; Lù et al., 2024; Murty et al., 2024a), and desktop (Chen et al., 2024a) applications. To effectively build computer agents, the best approach is to use trajectory data, which should consist of sequences containing GUI information, both low-level and high-level instructions, as well as corresponding actions (Li et al., 2024; Zhang et al., 2024a; Zheng et al., 2024b). However, acquiring such trajectories poses significant challenges. First, existing datasets often lack essential components. Second, current datasets are mainly curated using manual methods, which are costly. Finally, current works are usually tailored to specific GUI (*e.g.*, web-only), restricting their applicability in different scenarios.

3 *OS-Genesis*

In this section, we present the pipeline of *OS-Genesis*, detailing the process from automated data collection to the construction of complete GUI agent trajectories.

3.1 Interaction-Driven Functional Discovery

As illustrated in Figure 2, *OS-Genesis* begins with human-free exploration in dynamic environments $\mathcal{E} = \{\text{mobile}, \text{web}\}$, systematically traversing most interactive elements through actions $a \in \mathcal{A} = \{\texttt{CLICK}, \texttt{TYPE}, \texttt{SCROLL}\}$. With the goal of constructing mobile and web agents, this process is conducted in both the Android emulator and a chrome browser [2]. It to some extent mirrors human interaction with GUIs, uncovering potential functionalities without requiring pre-defined tasks.

The entire exploration phase is rule-based, except when interacting with input fields, where `GPT-4o` is invoked to generate contextually appropriate contents. At the end of this phase, massive triplets $\langle s_{\text{pre}}, a, s_{\text{post}} \rangle$ are collected, where s_{pre} and s_{post} denote the pre- and post-action states (*i.e.*, screenshots of the interface before and after the action, and a denotes the executed action.

3.2 Reverse Task Synthesis

Following the exploration phase, *OS-Genesis* leverages collected triplets $\langle s_{\text{pre}}, a, s_{\text{post}} \rangle$ to construct meaningful task instructions. This process involves generating low-level tasks using an anno-

[2] We build dynamic environments on the basis of Zhou et al. (2024) and Rawles et al. (2024).

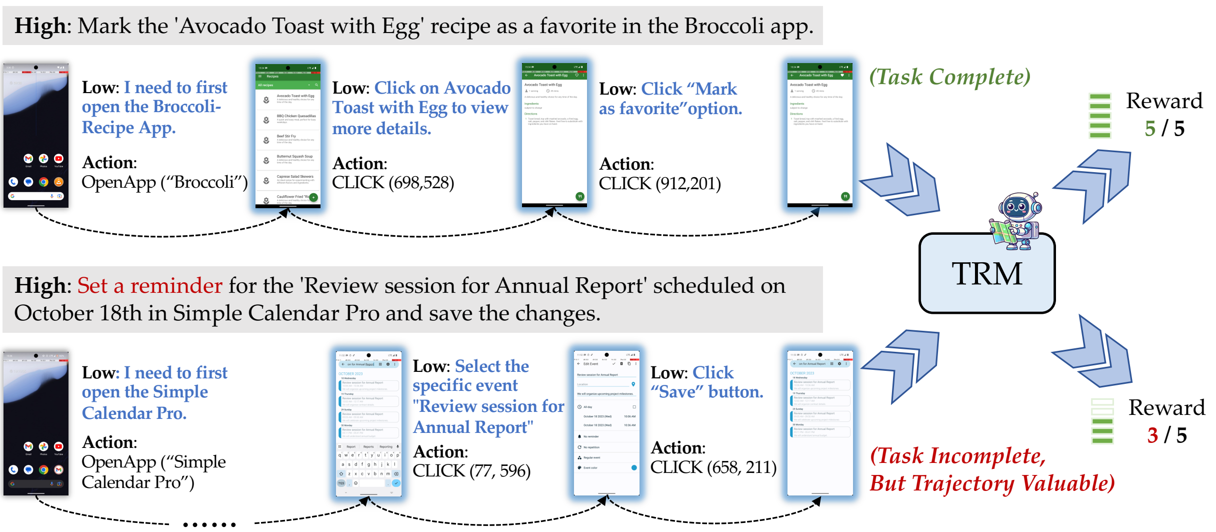


Figure 3: An overview of collecting complete trajectories through exploring high-level instructions generated by reverse task synthesis. Low-level instructions and the last three states of the trajectory (indicated in light blue) are used by the Trajectory Reward Model (TRM) to assign reward scores.

tation model and subsequently transforming them into high-level tasks. The annotation model $\mathcal{M}$ transforms each triplet $\langle s_{\text{pre}}, a, s_{\text{post}} \rangle \in \mathcal{T}$ into a specific low-level task instruction:

$$f_{\text{low}} : \langle s_{\text{pre}}, a, s_{\text{post}} \rangle \xrightarrow{\mathcal{M}} \tau_{\text{low}}.$$

Here, τ_{low} represents an atomic, executable operation derived from the observed state transition caused by the action a. For example, if the action $a = \texttt{CLICK}$ reveals a dropdown menu, the corresponding task might be "click the dropdown to display options." The annotation model integrates visual, contextual, and action semantics to ensure that τ_{low} aligns with the functions of $\mathcal{E}$.

Building on the synthesized low-level tasks, *OS-Genesis* constructs high-level tasks by associating each low-level task τ_{low} with broader objectives that could plausibly encompass it. This process, performed by the annotation model $\mathcal{M}$, maps individual low-level steps to high-level tasks by leveraging contextual information and domain knowledge:

$$f_{\text{high}} : \tau_{\text{low}} \xrightarrow{\mathcal{M}} \tau_{\text{high}}.$$

Here, τ_{high} represents a goal-oriented instruction that contextualizes the low-level operation within a larger user intent. For instance, a low-level task such as "click the dropdown to display options" might be linked to a high-level task like "configure application settings," as the dropdown interaction is often a prerequisite for such configurations. Details and prompts for transforming triples into high-level instructions are provided in Appendix C.

After this reverse task synthesis process, *OS-Genesis* generates a diverse set of high-level instructions $\mathcal{T} = \{\tau_1, \tau_2, \ldots, \tau_N\}$ that are aligned with dynamic environments and semantically rich. This entire process is completed without any human intervention.

Subsequently, these synthetic instructions $\mathcal{T}$ are executed in environment $\mathcal{E}$ by a model like `GPT-4o`, producing a complete set of trajectories, denoted as $\mathcal{G} = \{g_1, g_2, \ldots, g_N\}$.

3.3 Trajectory Reward Model

Considering the potential limitations of a model's agentic ability, errors or incomplete steps may arise when using high-level instructions to explore and generate trajectories. To address this, we incorporate a Trajectory Reward Model (TRM) to ensure the quality and utility of trajectories synthesized by *OS-Genesis*, as illustrated in Figure 3. Previous methods commonly rely on labeler functions (He et al., 2024; Murty et al., 2024a, *inter alia*), which discard trajectories deemed incomplete directly. (Pan et al., 2024). However, even incomplete trajectories often contain valuable exploration of the GUI environment. Given their large proportion of the data, discarding them wastes critical opportunities to enhance the model's agentic capabilities. Thus, diverging from binary evaluation, we leverage the characteristics of trajectory. The TRM, built upon `GPT-4o`, aims to perform graded evaluation to assist in sampling for training, focusing on the following features:

- **Completion**: Measures the extent to which the trajectory successfully fulfills the instructed task, considering completeness and proper handling of interactions.
- **Coherence**: Evaluates whether the trajectory follows a logical sequence of actions toward achieving the high-level task, avoiding redundant or irrelevant steps.

Algorithm 1 Reward-Based Trajectory Sampling

Require: Trajectory set $\mathcal{G} = \{g_1, g_2, \ldots, g_N\}$, where $g_i = \{s_{i,1}, l_{i,1}, s_{i,2}, \ldots, s_{i,K_i}\}$ represents a trajectory with K_i steps, including states $s_{i,j}$ and low-level instructions $l_{i,j}$. Reward model $\mathcal{RM}$.

Ensure: Trajectories are sampled for training according to their rewards.

```
for each trajectory g_i ∈ G do
    Initialize trajectory reward R_i ← 0
    Extract low-level instructions L_i = {l_{i,1}, l_{i,2}, ..., l_{i,K_i}}
    Extract the last three states S_last = {s_{i,K_i−2}, s_{i,K_i−1}, s_{i,K_i}}
    Compute trajectory reward: R_i = RM(L_i, S_last)
end for
for each training iteration do
    Compute sampling probabilities P(g_i) = R_i / (∑_{k=1}^{N} R_k) for all g_i
    Sample a trajectory g_i based on P(g_i) for each training step
end for
```

The whole process is shown in Algorithm 1. By leveraging TRM, *OS-Genesis* ensures that synthesized trajectories are utilized effectively, allowing the training process to benefit from both high-quality data and diverse task scenarios.

4 Experiments

4.1 Experimental Settings

Evauation Benchmarks. For mobile tasks, we select (1) AndroidControl (Li et al., 2024), which evaluates the ability of GUI agents to perform both low- and high-level tasks, and (2) AndroidWorld (Rawles et al., 2024), a challenging online benchmark running in Android emulators, to demonstrate the practicability of our agents in solving human daily tasks. Regarding web tasks, More information about the benchmark settings and and evaluation details are presented in Appendix A.

Model Settings. We primarily use `GPT-4o` for reverse task synthesis and reward modeling. As for the backbone models used to construct agents, we consider (1) InternVL2-4B/8B (Chen et al., 2024b), which is trained without GUI data, and (2) Qwen2-VL-7B-Instruct (Wang et al., 2024b), which claims to possess certain agentic capabilities to conduct thorough and comparative experiments. All training is performed as VLM full fine-tuning on interconnected clusters of 8 × A100 80GB GPUs, with detailed training settings provided in Appendix B and prompt settings in Appendix D.

4.2 Baseline Construction and Training

Baselines. As a pioneering study in synthesizing GUI agent data, we design the following baselines to demonstrate the superiority of trajectories obtained through *OS-Genesis*. All settings uniformly accept `a11ytree` and screenshots as inputs.

- **Zero-Shot**: This baseline leverages CoT (Wei et al., 2022) prompting to guide the model in perceiving environments and taking actions. For AndroidWorld tasks, we follow Rawles et al. (2024) to adopt M3A agent setup with multimodal input for this setting.
- **Task-Driven**: We build this baseline to compare with the common approach for agent data synthesis (Lai et al., 2024, *inter alia*). Given the initial screenshots of the app/web page and task examples, use `GPT-4o` to generate high-level instructions and explore the environment to collect trajectories. These trajectories are then used for training.
- **Self-Instructions**: Building upon the task-driven baseline, this approach employs `GPT-4o` to perform self-instruction (Wang et al., 2023), generating additional high-level tasks for exploration and trajectory collection. Together with the previously collected trajectories, they are then used for training.

Details of the baseline construction are provided in Appendix E. All these data and resources will be made public to accelerate future research.

Trajectory Training. Training GUI Agents based on VLMs using trajectory data is essentially a supervised fine-tuning (SFT) process. Nevertheless, we devise two training objectives to maximize the utility of synthesized trajectories:

- **Planning Training.** This objective aims to enhance agents' planning ability. For each trajectory $g_i \in \mathcal{G}$, given multimodal input s,

Base Model	Strategies	AndroidWorld	AndroidControl-High		AndroidControl-Low	
			SR	Type	SR	Type
`GPT-4o`	Zero-Shot (M3A)	23.70	53.04	69.14	69.59	80.27
InternVL2-4B	Zero-Shot	0.00	16.62	39.96	33.69	60.65
	Task-Driven	4.02	27.37	47.08	66.48	90.37
	Task-Driven w. Self Instruct	7.14	24.95	44.27	66.70	90.79
	OS-Genesis	**15.18**	**33.39**	**56.20**	**73.38**	**91.32**
InternVL2-8B	Zero-Shot	2.23	17.89	38.22	47.69	66.67
	Task-Driven	4.46	23.79	43.94	64.43	89.83
	Task-Driven w. Self Instruct	5.36	23.43	44.43	64.69	89.85
	OS-Genesis	**16.96**	**35.77**	**64.57**	**71.37**	**91.27**
Qwen2-VL-7B	Zero-Shot	0.89	28.92	61.39	46.37	72.78
	Task-Driven	6.25	38.84	58.08	71.33	88.71
	Task-Driven w. Self Instruct	9.82	39.36	58.28	71.51	89.73
	OS-Genesis	**17.41**	**44.54**	**66.15**	**74.17**	**90.72**

Table 1: Evaluations on AndroidControl and AndroidWorld. SR represents the task success rate. Type measures the exact match score between the predicted action types (*e.g.*, `CLICK`, `SCROLL`) and the ground truth.

high-level instruction h_i, and history context c, the agent θ predict the low-level instruction ℓ and the corresponding action a.

$$\mathcal{L}_1 = -\sum_{t_i \in \mathcal{T}} \log \left(p_\theta(\ell \mid s, h_i, c) \cdot p_\theta(a \mid s, h_i, c, \ell) \right) \tag{1}$$

- **Action Training.** This objective strengthens the agent's ability to execute appropriate actions based on the low-level instruction ℓ. given s, h_i, c, the agent predicts the action a.

$$\mathcal{L}_2 = -\sum_{t_i \in \mathcal{T}} \log p_\theta(a \mid s, c, \ell) \tag{2}$$

After trajectory training, agents will generate ReAct-style (Yao et al., 2023) outputs, with their step-by-step thoughts recorded in the history. To ensure a fair comparison, both Task-Driven baseline and *OS-Genesis* use 1K trajectories for training, while Self-Instructions baseline uses 1.5K trajectories, with an average trajectory length of 6.4 steps.

4.3 Main Results

AndroidWorld. To prove the effectiveness of *OS-Genesis* under dynamic environment, we evaluate it on AndroidWorld (Rawles et al., 2024) that leverages a Pixel 6 phone simulator as testbed. As shown in Table 1, *OS-Genesis* significantly narrows the performance gap between open-source agents and the SOTA `GPT-4o`-based M3A agent. Compared to task-driven methods, training with *OS-Genesis* achieves performance improvements that are often double those of the baselines. Even self-instruct baseline utilize 1.5 $\times$ the amount of data compared to *OS-Genesis*, they fail to match the quality of data generated by *OS-Genesis*. underscoring the importance of using high-quality trajectory data in online settings.

Beyond improvements in planning and action, some gains also stem from *OS-Genesis* 's ability to cover subtle yet critical app functionalities during the reverse task synthesis process. These functionalities, often overlooked by task-driven methods, are essential for completing intricate tasks.

AndroidControl. We then evaluate *OS-Genesis* on AndroidControl (Li et al., 2024). Out of the 833 apps covered by AndroidControl, only 20 have been directly encountered during data synthesis, making this evaluation a test of *OS-Genesis* 's out-of-distribution (OOD) performance. In the high-level setting, the agent is required to autonomously plan and execute actions to complete a given task. For the low-level setting, agents will follow human instructions and only need to determine the next step. As shown in Table 1, *OS-Genesis* consistently improves both action and planning abilities across various backbones. Compared to `GPT-4o`, *OS-Genesis* achieves substantial gains, especially in the low-level setting where it consistently outperforms. While maintaining an edge over other task-driven trajectory synthesis methods, *OS-Genesis* excels particularly in the high-level setting. This validates that exploration-first task construction produces more meaningful and logically coherent tasks. Additionally, it highlights *OS-Genesis* 's

Model	Strategies	Shopping	CMS	Reddit	Gitlab	Maps	Overall
GPT-4o	Zero-Shot	14.28	21.05	6.25	14.29	20.00	16.25
InternVL2-4B	Zero-Shot	0.00	0.00	0.00	0.00	0.00	0.00
	Task-Driven	5.36	1.76	0.00	**9.52**	5.00	4.98
	Task-Driven w. Self-Instruct	5.36	3.51	0.00	**9.52**	**7.50**	5.81
	OS-Genesis	**10.71**	**7.02**	**3.13**	7.94	**7.50**	**7.88**
InternVL2-8B	Zero-Shot	0.00	0.00	0.00	0.00	0.00	0.00
	Task-Driven	3.57	7.02	0.00	6.35	2.50	4.56
	Task-Driven w. Self-Instruct	**8.93**	10.53	6.25	**7.94**	0.00	7.05
	OS-Genesis	7.14	**15.79**	**9.34**	6.35	**10.00**	**9.96**
Qwen2-VL-7B	Zero-Shot	**12.50**	7.02	6.25	6.35	5.00	7.47
	Task-Driven	8.93	7.02	6.25	6.35	5.00	7.05
	Task-Driven w. Self-Instruct	8.93	1.76	3.13	4.84	**7.50**	5.39
	OS-Genesis	7.14	**8.77**	**15.63**	**15.87**	5.00	**10.79**

Table 2: Evaluations on WebArena with success rate reported.

generalization ability to unseen OOD scenarios compared to task-driven approaches.

WebArena. We choose WebArena (Zhou et al., 2024), a highly challenging benchmark running on functional websites to evaluate *OS-Genesis* on web environments. We follow similar baseline settings as in mobile tasks. Results in Table 2 show that training with *OS-Genesis* data generally leads to notable performance improvements. For InternVL2-4B and 8B that can hardly generate outputs in correct formats under zero-shot settings, *OS-Genesis* enables a remarkable leap in performance after training. For Qwen2-VL-7B, which has already been trained on GUI agent data, further training with *OS-Genesis* results in substantial performance gains. Notable edges over task-driven baselines highlight that, in web environments rich with interactive elements, reverse task synthesis can derive more meaningful explorations.

5 Analysis

5.1 How Diverse is Our Synthesized Data?

Ensuring the diversity of synthetic data is crucial for effective model training. Traditional approaches that rely on pre-defined high-level tasks are inherently constrained, as it is practically impossible to enumerate and cover the full spectrum of potential interactions within a complex environment. In contrast, *OS-Genesis* employs an exploration-driven method that naturally adapts to the environment by interacting with diverse interface elements, systematically uncovering a broader range of functional capabilities.

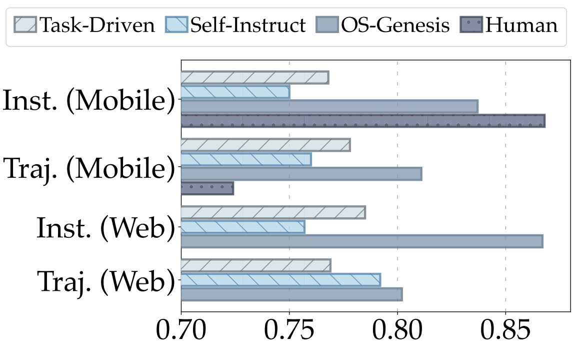


Figure 4: Comparison of instruction diversity and trajectory diversity between different synthetic data and human data, measured by average cosine distance.

To validate the effectiveness of our method in generating more diverse data, we examine both instruction diversity and trajectory diversity. We begin by analyzing the variety of generated instructions. Using Sentence-BERT (Reimers and Gurevych, 2019), we embed each instruction and compute the average cosine distance among these embeddings. As illustrated in Figure 4, *OS-Genesis* achieves the greatest average distance across both mobile and web environments among different synthetic data, indicating a broader range of task types beyond those pre-defined at the outset. We then apply the same approach to the low-level actions taken in the generated trajectories. *OS-Genesis* demonstrates the highest trajectory diversity, suggesting that our interaction-driven strategy more thoroughly exploits the available operations within different environments.

Interestingly, while human-annotated data displays high instruction diversity, it shows low trajectory diversity. This suggests that while humans can imagine a wide variety of instructions, they tend to rely on a narrower set of familiar, well-practiced actions for execution. In contrast, *OS-Genesis* achieves high diversity in both instructions and trajectories, enabling a more comprehensive exploration of the environment.

5.2 How TRM Impacts Performance?

We introduce a Trajectory Reward Model (TRM) for data quality control and exploitation, substituting traditional labeler filtering methods (He et al., 2024; Murty et al., 2024a). To analyze its impact and for ablation purposes, we include additional settings for comparison: (1) training without an RM, where all synthesized data is treated equally during training, and (2) using a labeler, similar to previous approaches where only complete trajectories are retained for training.

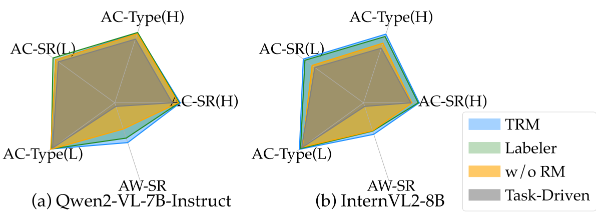
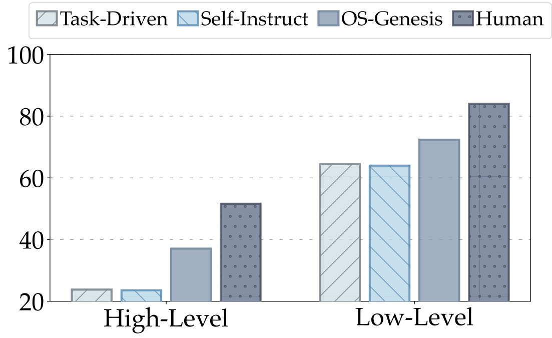


Figure 5: Comparison of different reward modeling strategies.

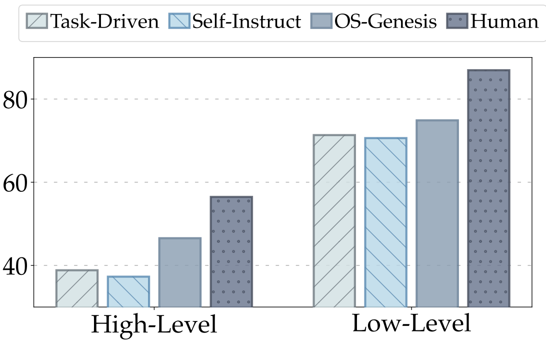
As shown in Figure 5, the relative performance across different reward strategies demonstrates the effectiveness of TRM, notably in enhancing high-level capabilities (*e.g.*, AndroidControl-High and AndroidWorld). While using a labeler provides slight gains in high-level tasks, it comes at the cost of reduced performance in low-level tasks. For low-level scenarios, since *OS-Genesis* data—even individual steps—is inherently more meaningful and of good quality, all training strategies yield consistent improvements.

5.3 How Far are We from Human Data?

Here, we investigate the gaps between synthetic data and human demonstrations in GUI agent training. We select 10K crowdsourced data provided by AndroidControl (Li et al., 2024) dataset for comparison. As shown in Figure 6, *OS-Genesis* significantly narrows the performance gap between synthetic trajectories and human-annotated trajectories. This is notably evident in high-level tasks, demonstrating that agents trained on *OS-Genesis* trajectories can plan and solve problems more closely aligned with human manners. In terms of average success rate, viewing human-annotated data as the gold standard, the performance retention rate of *OS-Genesis* data surpasses 80%.



(a) InternVL2-8B



(b) Qwen2-VL-7B-Instruct

Figure 6: Comparison of training effectiveness between different synthetic data and human-annotated data.

6 Conclusion

We introduce *OS-Genesis*, a data synthesis pipeline to fuel diversified computer control agents. By leveraging a novel interaction-driven approach, *OS-Genesis* overcomes the critical bottlenecks of constructing meaningful and diverse GUI tasks in previous practices. Through extensive evaluations on challenging online benchmarks, we demonstrate that *OS-Genesis*-synthesized data has led to a breakthrough in GUI agents' planning and action capabilities. Moreover, our synthesized trajectories exhibit greater diversity and substantially narrow the quality gap between synthetic data and human annotations. *OS-Genesis* provides a promising direction for generating high-quality trajectory data for GUI agent training, bringing the community one step closer to achieving digital automation.

Limitations

While *OS-Genesis* demonstrates the potential to overcome critical challenges in acquiring GUI trajectory data, it is important to acknowledge certain limitations:

Proprietary Models. We build our GUI agents upon open-source VLMs, but for data quality, we leverage `GPT-4o` for exploration and reward modeling in the annotation process. The reason we did not replace this process with open-source counterparts is that existing open-source VLMs lack the ability to follow user instructions and proactively complete exploration in online environments. We believe that in the future, more capable action models can bridge this gap and replace proprietary components in this pipeline.

Data usage. Throughout this work, we employ textual and visual representations to train and evaluate our GUI agents. This is designed to (1) maximize agents' planning and action capabilities in semantically rich environments, and (2) ensure evaluation consistency across different environments. We are aware that using either textual or visual data alone could also contribute to constructing GUI agents, provided that the I/O format and training strategies are appropriately adjusted. We leave the partial use of full trajectory data as future works.

Broader Impacts

Computer agents operating in an OS environment could potentially affect the normal functioning of the system. However, considering that all settings in this work are conducted within virtual environments, we do not view this as a concern.

References

Anthropic. 2023. The claude 3 model family: Opus, sonnet, haiku.

Yuxiang Chai, Siyuan Huang, Yazhe Niu, Han Xiao, Liang Liu, Dingyu Zhang, Peng Gao, Shuai Ren, and Hongsheng Li. 2024. Amex: Android multi-annotation expo dataset for mobile gui agents. *arXiv preprint arXiv:2407.17490*.

Wentong Chen, Junbo Cui, Jinyi Hu, Yujia Qin, Junjie Fang, Yue Zhao, Chongyi Wang, Jun Liu, Guirong Chen, Yupeng Huo, Yuan Yao, Yankai Lin, Zhiyuan Liu, and Maosong Sun. 2024a. Guicourse: From general vision language models to versatile gui agents. *Preprint*, arXiv:2406.11317.

Zhe Chen, Jiannan Wu, Wenhai Wang, Weijie Su, Guo Chen, Sen Xing, Muyan Zhong, Qinglong Zhang, Xizhou Zhu, Lewei Lu, Bin Li, Ping Luo, Tong Lu, Yu Qiao, and Jifeng Dai. 2024b. Internvl: Scaling up vision foundation models and aligning for generic visual-linguistic tasks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 24185–24198.

Kanzhi Cheng, Qiushi Sun, Yougang Chu, Fangzhi Xu, Li YanTao, Jianbing Zhang, and Zhiyong Wu. 2024. SeeClick: Harnessing GUI grounding for advanced visual GUI agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9313–9332, Bangkok, Thailand. Association for Computational Linguistics.

Biplab Deka, Zifeng Huang, Chad Franzen, Joshua Hibschman, Daniel Afergan, Yang Li, Jeffrey Nichols, and Ranjitha Kumar. 2017. Rico: A mobile app dataset for building data-driven design applications. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology*, UIST '17, page 845–854, New York, NY, USA. Association for Computing Machinery.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Samuel Stevens, Boshi Wang, Huan Sun, and Yu Su. 2023. Mind2web: Towards a generalist agent for the web. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

Zane Durante, Qiuyuan Huang, Naoki Wake, Ran Gong, Jae Sung Park, Bidipta Sarkar, Rohan Taori, Yusuke Noda, Demetri Terzopoulos, Yejin Choi, et al. 2024. Agent ai: Surveying the horizons of multimodal interaction. *arXiv preprint arXiv:2401.03568*.

Tao Feng, Chuanyang Jin, Jingyu Liu, Kunlun Zhu, Haoqin Tu, Zirui Cheng, Guanyu Lin, and Jiaxuan You. 2024. How far are we from agi. *arXiv preprint arXiv:2405.10313*.

Hongliang He, Wenlin Yao, Kaixin Ma, Wenhao Yu, Hongming Zhang, Tianqing Fang, Zhenzhong Lan, and Dong Yu. 2024. Openwebvoyager: Building multimodal web agents via iterative real-world exploration, feedback and optimization. *arXiv preprint arXiv:2410.19609*.

Siyuan Hu, Mingyu Ouyang, Difei Gao, and Mike Zheng Shou. 2024. The dawn of gui agent: A preliminary case study with claude 3.5 computer use. *Preprint*, arXiv:2411.10323.

Zhiting Hu and Tianmin Shu. 2023. Language models, agent models, and world models: The law for machine reasoning and planning. *arXiv preprint arXiv:2312.05230*.

Chuanyang Jin, Yutong Wu, Jing Cao, Jiannan Xiang, Yen-Ling Kuo, Zhiting Hu, Tomer Ullman, Antonio Torralba, Joshua B Tenenbaum, and Tianmin Shu. 2024. Mmtom-qa: Multimodal theory of mind question answering. *arXiv preprint arXiv:2401.08743*.

Raghav Kapoor, Yash Parag Butala, Melisa Russak, Jing Yu Koh, Kiran Kamble, Waseem Alshikh, and Ruslan Salakhutdinov. 2024. Omniact: A dataset and benchmark for enabling multimodal generalist autonomous agents for desktop and web. *arXiv preprint arXiv:2402.17553*.

Hanyu Lai, Xiao Liu, Iat Long Iong, Shuntian Yao, Yuxuan Chen, Pengbo Shen, Hao Yu, Hanchen Zhang, Xiaohan Zhang, Yuxiao Dong, and Jie Tang. 2024. Autowebglm: A large language model-based web navigating agent. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 5295—-5306.

Wei Li, William E Bishop, Alice Li, Christopher Rawles, Folawiyo Campbell-Ajala, Divya Tyamagundlu, and Oriana Riva. 2024. On the effects of data scale on UI control agents. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

Evan Zheran Liu, Kelvin Guu, Panupong Pasupat, and Percy Liang. 2018. Reinforcement learning on web interfaces using workflow-guided exploration. In *International Conference on Learning Representations*.

Quanfeng Lu, Wenqi Shao, Zitao Liu, Fanqing Meng, Boxuan Li, Botong Chen, Siyuan Huang, Kaipeng Zhang, Yu Qiao, and Ping Luo. 2024. Gui odyssey: A comprehensive dataset for cross-app gui navigation on mobile devices. *Preprint*, arXiv:2406.08451.

Xing Han Lù, Zdeněk Kasner, and Siva Reddy. 2024. Weblinx: Real-world website navigation with multi-turn dialogue. *arXiv preprint arXiv:2402.05930*.

Shikhar Murty, Dzmitry Bahdanau, and Christopher D Manning. 2024a. Nnetscape navigator: Complex demonstrations for web agents without a demonstrator. *arXiv preprint arXiv:2410.02907*.

Shikhar Murty, Christopher D Manning, Peter Shaw, Mandar Joshi, and Kenton Lee. 2024b. BAGEL: Bootstrapping agents by guiding exploration with language. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 36894–36910. PMLR.

Runliang Niu, Jindong Li, Shiqi Wang, Yali Fu, Xiyu Hu, Xueyuan Leng, He Kong, Yi Chang, and Qi Wang. 2024. Screenagent: A vision language model-driven computer control agent. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence, IJCAI-24*, pages 6433–6441. International Joint Conferences on Artificial Intelligence Organization. Main Track.

Jiayi Pan, Yichi Zhang, Nicholas Tomlin, Yifei Zhou, Sergey Levine, and Alane Suhr. 2024. Autonomous evaluation and refinement of digital agents. In *First Conference on Language Modeling*.

Ajay Patel, Markus Hofmarcher, Claudiu Leoveanu-Condrei, Marius-Constantin Dinu, Chris Callison-Burch, and Sepp Hochreiter. 2024. Large language models can self-improve at web agent tasks. *arXiv preprint arXiv:2405.20309*.

Christopher Rawles, Sarah Clinckemaillie, Yifan Chang, Jonathan Waltz, Gabrielle Lau, Marybeth Fair, Alice Li, William Bishop, Wei Li, Folawiyo Campbell-Ajala, et al. 2024. Androidworld: A dynamic benchmarking environment for autonomous agents. *arXiv preprint arXiv:2405.14573*.

Christopher Rawles, Alice Li, Daniel Rodriguez, Oriana Riva, and Timothy P Lillicrap. 2023. Androidinthewild: A large-scale dataset for android device control. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3982–3992, Hong Kong, China. Association for Computational Linguistics.

Tianlin Shi, Andrej Karpathy, Linxi Fan, Jonathan Hernandez, and Percy Liang. 2017. World of bits: An open-domain platform for web-based agents. In *International Conference on Machine Learning*, pages 3135–3144. PMLR.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2024. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36.

Theodore Sumers, Shunyu Yao, Karthik Narasimhan, and Thomas Griffiths. 2024. Cognitive architectures for language agents. *Transactions on Machine Learning Research*. Survey Certification.

Qiushi Sun, Zhirui Chen, Fangzhi Xu, Kanzhi Cheng, Chang Ma, Zhangyue Yin, Jianing Wang, Chengcheng Han, Renyu Zhu, Shuai Yuan, et al. 2024. A survey of neural code intelligence: Paradigms, advances and beyond. *arXiv preprint arXiv:2403.14734*.

Qiushi Sun, Zhangyue Yin, Xiang Li, Zhiyong Wu, Xipeng Qiu, and Lingpeng Kong. 2023. Corex: Pushing the boundaries of complex reasoning through multi-model collaboration. *arXiv preprint arXiv:2310.00280*.

Junyang Wang, Haiyang Xu, Jiabo Ye, Ming Yan, Weizhou Shen, Ji Zhang, Fei Huang, and Jitao Sang. 2024a. Mobile-agent: Autonomous multi-modal mobile device agent with visual perception. In *ICLR 2024 Workshop on Large Language Model (LLM) Agents*.

Peng Wang, Shuai Bai, Sinan Tan, Shijie Wang, Zhihao Fan, Jinze Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Yang Fan, Kai Dang, Mengfei Du, Xuancheng Ren, Rui Men, Dayiheng Liu, Chang Zhou, Jingren Zhou, and Junyang Lin. 2024b. Qwen2-vl: Enhancing vision-language model's perception of the world at any resolution. *arXiv preprint arXiv:2409.12191*.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi. 2023. Self-instruct: Aligning language models with self-generated instructions. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 13484–13508, Toronto, Canada. Association for Computational Linguistics.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed H. Chi, Quoc V Le, and Denny Zhou. 2022. Chain of thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Shaokun Zhang, Erkang Zhu, Beibin Li, Li Jiang, Xiaoyun Zhang, and Chi Wang. 2023. Autogen: Enabling next-gen llm applications via multi-agent conversation framework. *arXiv preprint arXiv:2308.08155*.

Zhiyong Wu, Chengcheng Han, Zichen Ding, Zhenmin Weng, Zhoumianze Liu, Shunyu Yao, Tao Yu, and Lingpeng Kong. 2024a. OS-copilot: Towards generalist computer agents with self-improvement. In *ICLR 2024 Workshop on Large Language Model (LLM) Agents*.

Zhiyong Wu, Zhenyu Wu, Fangzhi Xu, Yian Wang, Qiushi Sun, Chengyou Jia, Kanzhi Cheng, Zichen Ding, Liheng Chen, Paul Pu Liang, et al. 2024b. Os-atlas: A foundation action model for generalist gui agents. *arXiv preprint arXiv:2410.23218*.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. 2024. OSWorld: Benchmarking multimodal agents for open-ended tasks in real computer environments. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. 2022. Webshop: Towards scalable real-world web interaction with grounded language agents. In *Advances in Neural Information Processing Systems*, volume 35, pages 20744–20757. Curran Associates, Inc.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*.

Aohan Zeng, Mingdao Liu, Rui Lu, Bowen Wang, Xiao Liu, Yuxiao Dong, and Jie Tang. 2024. AgentTuning: Enabling generalized agent abilities for LLMs. In *Findings of the Association for Computational Linguistics ACL 2024*, pages 3053–3077, Bangkok, Thailand and virtual meeting. Association for Computational Linguistics.

Chi Zhang, Zhao Yang, Jiaxuan Liu, Yucheng Han, Xin Chen, Zebiao Huang, Bin Fu, and Gang Yu. 2023. Appagent: Multimodal agents as smartphone users. *Preprint*, arXiv:2312.13771.

Jianguo Zhang, Tian Lan, Rithesh Murthy, Zhiwei Liu, Weiran Yao, Juntao Tan, Thai Hoang, Liangwei Yang, Yihao Feng, Zuxin Liu, et al. 2024a. Agentohana: Design unified data and training pipeline for effective agent learning. *arXiv preprint arXiv:2402.15506*.

Jiwen Zhang, Jihao Wu, Yihua Teng, Minghui Liao, Nuo Xu, Xiao Xiao, Zhongyu Wei, and Duyu Tang. 2024b. Android in the zoo: Chain-of-action-thought for gui agents. *Preprint*, arXiv:2403.02713.

Boyuan Zheng, Boyu Gou, Jihyung Kil, Huan Sun, and Yu Su. 2024a. GPT-4V(ision) is a generalist web agent, if grounded. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 61349–61385. PMLR.

Longtao Zheng, Zhiyuan Huang, Zhenghai Xue, Xinrun Wang, Bo An, and Shuicheng Yan. 2024b. Agentstudio: A toolkit for building general virtual agents. *arXiv preprint arXiv:2403.17918*.

Longtao Zheng, Rundong Wang, Xinrun Wang, and Bo An. 2024c. Synapse: Trajectory-as-exemplar prompting with memory for computer control. In *The Twelfth International Conference on Learning Representations*.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. 2024. Webarena: A realistic web environment for building autonomous agents. In *The Twelfth International Conference on Learning Representations*.

A Details of Benchmarks

A.1 Dataset and Benchmarks

Here we present more information about the benchmarks involved in evaluating *OS-Genesis*.

AndroidControl. AndroidControl is a benchmark designed to evaluate real-world mobile control agents, created from human-collected tasks within the Android environment, consisting of 7,708 tasks across 1,412 trajectories. It includes two SeqIO tasks: (i) SeqIO HL (high-level), where the prompt contains only a high-level instruction, and (ii) SeqIO LL (low-level), where both a low-level instruction and its corresponding high-level instruction are included. In terms of evaluation metrics, AndroidControl calculates the success rate (SR) and action type accuracy (Type) based on ground truth action labels. In our experimental setup, we add the screenshot's accessibility tree and historical actions from the current trajectory as additional observation space to better simulate the agent's execution environment.

AndroidWorld. AndroidWorld is an online benchmark for evaluating autonomous agents in Android environments, featuring 116 tasks across 20 real-world apps. Tasks are parameterized with randomized inputs, enabling diverse scenarios and robust evaluations. Success rates (SR) are assessed using system state inspections without modifying app source code. Due to app unavailability, a total of 112 tasks are actually used. Tasks marked as "NaN" are re-tested, and those that remain incomplete after re-testing are uniformly marked as false to ensure fair comparisons.

WebArena. WebArena is a realistic web benchmark for autonomous digital agents, comprising 812 challenging web navigation tasks across multiple domains, including maps, e-commerce, Reddit forums, and software development. It features robust evaluation programs that assess the success rate (SR) based on functional correctness. We follow the standard practices of WebArena by using the default action space (including actions such as clicks and inputs) and employing screenshots and the accessibility tree as the observation space for multimodal GUI agents.

B Experimental Details

The instructions we employed for evaluating baselines and *OS-Genesis* on AndroidWorld and AndroidControl are listed in Prompt 13 and Prompt 14 respectively.

C Reverse Task Synthesis Details

Our reverse task synthesis process simulates how humans explore new tasks in an unknown GUI environment. After performing actions on random elements, humans infer possible subsequent actions by observing changes on the screen, thus continuing their exploration to construct a complete trajectory for executing a particular task. In our reverse task synthesis, we provide `GPT-4o` with the current action being executed, before-and-after screenshots of the screen changes, and a red bounding box highlighting the interacted element in the screenshots. This allows `GPT-4o` to first comprehend the action being performed and then associate the possible high-level task based on the observed screen changes. The detailed association prompts for synthesizing high-level instruction data for both Android and Web are provided in Prompt 11 and Prompt 12 respectively.

D Training Details

InternVL2-{4B,8B}. InternVL2 (Chen et al., 2024b) utilizes Dynamic Aspect Ratio Matching to handle dynamic high-resolution inputs. In our training setting, we set the `max_dynamic_patch` parameter to 24 to comprehensively capture the fine-grained details of the image. Consequently, the resized input image is partitioned into a maximum of 24 tiles, each of 448×448 pixels, while a thumbnail of the entire image is included to preserve global contextual information.

Qwen2-VL-7B-Instruct. Qwen2-VL (Wang et al., 2024b) introduces the Naive Dynamic Resolution mechanism, which is capable of handling images of any resolution by mapping them into a dynamic number of visual tokens, providing a more human-like visual processing experience. Through our experiments, we found that configuring the `image_resolution` parameter to 1024 for both training and inference produces outstanding results in GUI agent tasks, while also contributing to the optimization of the model's training and inference costs.

Accessibility Tree. The accessibility tree represents the hierarchical relationships and attributes of all interactive or accessible elements on a screen, providing rich GUI information in text form to train GUI agents. In constructing the training data, we filter the accessibility tree to retain only the position or index information of elements visible on the screen, reducing the interference of excessive redundant text in model training.

Data Format. We follow the data formats of AndroidWorld and WebArena to construct our training data, ensuring consistency in formatting between the training and evaluation phases. The detailed training instructions for Android and Web data are listed in Prompt 9 and Prompt 10 respectively.

E Baseline Settings

E.1 Task-Driven

Following prior work (He et al., 2024; Lai et al., 2024) on collecting tasks for GUI agents, we guide GPT-4o to infer possible high-level instructions based on the initial GUI interface (e.g., the homepage of a social forum like Reddit). Some examples of initial screens are demonstrated in Figure 7 (mobile) and Figure 8 (web).

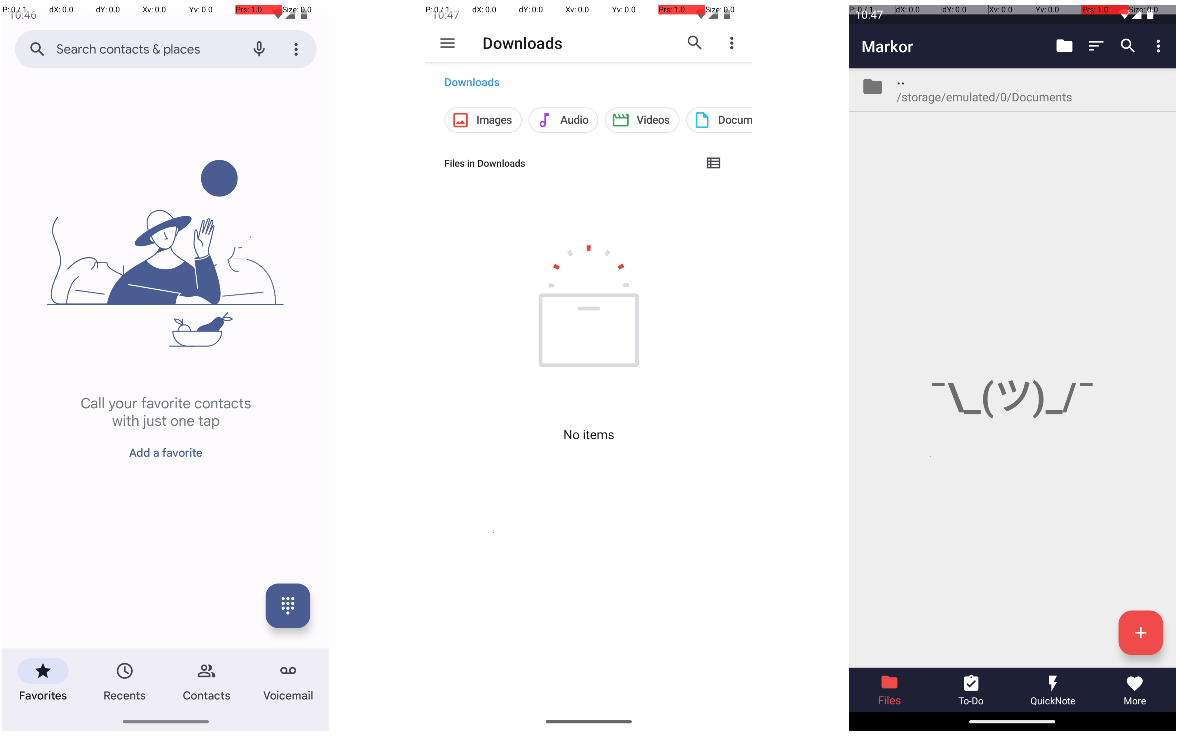
E.2 Task-Driven w. Self Instruct

Building upon the task-driven baseline in E.1, we incorporate self-instruction (Wang et al., 2023) data as a second baseline. This is constructed by randomly sampling 3 demonstrations from the above task-driven high-level instructions as in-context examples for each synthesis iteration.

Notably, we make certain that the total number of trajectories for the baseline is at least equal to that of our method to avoid data imbalance and maintain fairness in comparisons.

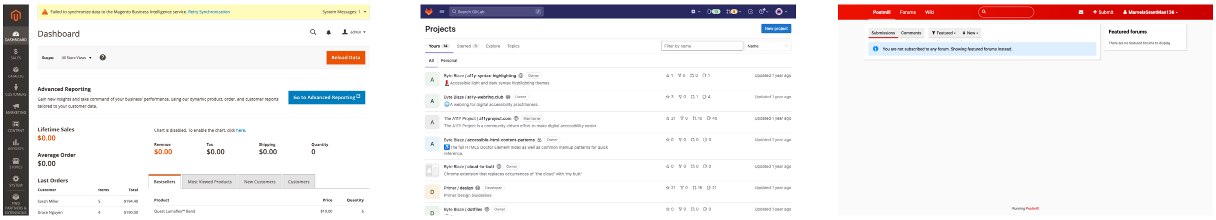
F Details of Trajectory Reward Model

The Trajectory Reward Model (TRM) primarily assesses the quality of agent trajectories by focusing on completion and coherence. Based on a high-level instruction to complete, the agent's entire action history (e.g., low-level instructions), and screenshots from the last three timesteps, GPT-4o is prompted to assign a score between 1 and 5 for the trajectory. Instead of providing in-context learning examples, we include in the prompt specific aspects of coherence and completion to consider, along with detailed descriptions of what each score from 1 to 5 represents. Given the similarity between mobile and web tasks, we apply the same TRM to both, as shown in prompt 18.



(a) Contacts (b) Files (c) Markor

Figure 7: Examples of initial screens employed in building task-driven baselines for mobile tasks.



(a) CMS (b) GitLab (c) Reddit

Figure 8: Examples of initial screens employed in building task-driven baselines for web tasks.

Prompt for Planning Training
<image>
You are a GUI task expert, I will provide you with a high-level instruction, an action history, a screenshot with its corresponding accessibility tree.
High-level instruction: {high_level_instruction}
Action history: {action_history}
Accessibility tree: {a11y_tree}
Please generate the low-level thought and action for the next step.

Prompt for Action Training
<image>
You are a GUI task expert, I will provide you with an action history, a screenshot with its corresponding accessibility tree, and a low-level thought.
Action history: {action_history}
Accessibility tree: {a11y_tree}
Low-level thought: {low_level_thought}
Please generate the action for the next step.

Prompt 9: Prompts for training our agents on Android.

Prompt for Planning Training
<image>
Task Description
You are an intelligent agent completing web-based tasks.
Based on the user's objective (i.e. instruction), current interface information (i.e. screenshot and its corresponding accessibility tree), and action history, determine the next action.

Available Actions
- click [id]: This action clicks on an element with a specific id on the webpage.
- type [id] [content] [press_enter_after=0|1]: Use this to type the content into the field with id. By default, the "Enter" key is pressed after typing unless press_enter_after is set to 0.
- hover [id]: Hover over an element with id.
- press [key_comb]: Simulates the pressing of a key combination on the keyboard (e.g., Ctrl+v).
- scroll [direction=down|up]: Scroll the page up or down.
- new_tab: Open a new, empty browser tab.
- tab_focus [tab_index]: Switch the browser's focus to a specific tab using its index.
- close_tab: Close the currently active tab.
- goto [url]: Navigate to a specific URL.
- go_back: Navigate to the previously viewed page.
- go_forward: Navigate to the next page (if a previous go_back action was performed).
- stop [answer]: Issue this action when you believe the task is complete. If the objective is to find a text-based answer, provide the answer in the bracket. If you believe the task is impossible to complete, provide the answer as "N/A" in the bracket.

Output Format
First, generate the reasoning process for the action. Then, generate the action in the correct format. Start with a "In summary, the next action I will perform is" phrase, followed by action inside ``````.
For example:
"Let's think step-by-step. To add a product to the shopping cart, I need to navigate to the catalog or product section. The "CATALOG" link is available with ID [1234]. In summary, the next action I will perform is ```click [1234]```".

Instruction: {instruction}
Accessibility tree: {a11y_tree}
Action History: {action_history}
What's the next action?

Prompt for Action Training
<image>
You are an intelligent agent completing web-based tasks. I will provide you with available actions, a screenshot with its corresponding accessibility tree, and a low-level thought.

Available Actions
- click [id]: This action clicks on an element with a specific id on the webpage.
- type [id] [content] [press_enter_after=0|1]: Use this to type the content into the field with id. By default, the "Enter" key is pressed after typing unless press_enter_after is set to 0.
- hover [id]: Hover over an element with id.
- press [key_comb]: Simulates the pressing of a key combination on the keyboard (e.g., Ctrl+v).
- scroll [direction=down|up]: Scroll the page up or down.
- new_tab: Open a new, empty browser tab.
- tab_focus [tab_index]: Switch the browser's focus to a specific tab using its index.
- close_tab: Close the currently active tab.
- goto [url]: Navigate to a specific URL.
- go_back: Navigate to the previously viewed page.
- go_forward: Navigate to the next page (if a previous go_back action was performed).
- stop [answer]: Issue this action when you believe the task is complete. If the objective is to find a text-based answer, provide the answer in the bracket. If you believe the task is impossible to complete, provide the answer as "N/A" in the bracket.

Accessibility tree: {a11y_tree}
Low-level thought: {low_level_thought}
Please generate the action inside `````` for the next step.

Prompt 10: Prompts for training our agents on Web.

Prompt for Associating High-Level Tasks

You are an expert at envisioning specific tasks corresponding to changes in mobile screenshots. I will provide you with the following:
1. The type of action currently being executed. The type of action currently being executed, which can be one of five types: CLICK, SCROLL, TYPE, PRESS_BACK, and LONG_PRESS. If the action is TYPE, an additional value representing the input will be provided. If the action is SCROLL, an additional scroll direction will be provided.
2. Screenshots of the interface before and after the current action is performed. If the action is CLICK, the pre-action screenshot will include a red bbox highlighting the element being interacted with (if applicable). Pay particular attention to the content of the element corresponding to the red bbox.
3. The name of the app where the current screenshot is located.

Your task is to envision a specific task based on the current action and the corresponding changes in screenshots. The output should include three parts:
1. Sub-Instruction: Based on the interface change caused by the current action, generate a corresponding natural language instruction for the current action. The instruction should be concise, clear, and executable. It must include specific details critical to the operation, such as file names, times, or other content as they appear in the screenshots. For example: “Scroll left to open the app drawer, displaying all installed applications on the devic”, “Click the chat interface, allowing the user to view and participate in conversation”, “Type the username ‘Agent’, preparing for the next step in logging into the account”.
2. Analysis: Based on the interface changes and the current action instructions, analyze the possible subsequent operations. This analysis should involve step-by-step reasoning, considering the potential changes on the screen and the actions that can be taken after these changes. For example: “After clicking the plus button, a dropdown menu appears with an option to create a document. I can select this option to create a new document. First, I need to name the document, then enter any content into the document, and finally save the document and exit”.
3. High-Level-Instruction: Based on the analysis results, envision a high-level task that can be completed within the current interface. There are two types of High-Level-Instruction:
Task-Oriented: Completing a series of operations to achieve a specific goal.
Question-Oriented: Performing a series of operations and deriving an answer to a specific question.
For example: {examples}.
Ensure that the High-Level-Instruction is executable by including all critical specifics, such as file names, relevant timings, or required details.

You ONLY need to return a dictionary formatted as follows:
{
“Sub-Instruction”: “xxx”,
“Analysis”: “xxx”,
“High-Level-Instruction”: “xxx”
}

Current Action: {current_action}
App Name: {app_name}
RETURN ME THE DICTIONARY I ASKED FOR.

Prompt 11: Prompts for associating high-level tasks on mobile.

Prompt for Associating High-Level Tasks

You are a GUI (Graphical User Interface) expert capable of analyzing interface changes and envisioning executable tasks or instructions. Given a GUI interface change caused by an action (e.g., clicking or typing) and the corresponding element highlighted in red boxes, you are required to analyze the interface and generate related tasks.
Your task is to envision tasks based on the current action and the resulting changes in the screenshots.
The output should include three components:
1. **Sub-Instruction**: Create a natural language instruction for the current action based on the interface changes it caused. The instruction should be concise, clear, and actionable, incorporating specific details critical to the task, such as elements, file names, timestamps, or other relevant content visible in the screenshots. For example:
- "Click on the 'Add to Cart' button next to the product to add it to your shopping cart."
- "Type 'OpenAI' into the search bar to find relevant articles."
- "Scroll down to view the latest blog posts on the homepage."
2. **Analysis**: Carefully analyze the before-and-after screenshots step by step, focusing on the changes caused by the action. Then, examine key elements in both screenshots and consider possible operations based on these elements. For example: "The previous screen displayed the main interface of a shopping website, featuring multiple product categories and several showcased items. After clicking the 'Sign Up' button, the interface transitioned to a login page where an email and password can be entered to log into an account. The login page also provides other options, such as recovering a password, creating a new account, or logging in with a Google account".
3. **High-Level Instruction**: Based on the before-and-after screenshots, the action, and the analysis, generate a high-level task that you believe can be completed within the current interface. There are three types of tasks:
- Information seeking: The user wants to obtain certain information from the webpage, such as product details, reviews, map information, or route comparisons. Please propose clear and specific questions that need an explicit answer, and avoid asking for summary-type questions, such as "summarize the information about a product".
- Site navigation: The user wants to navigate to a specific page or state.
- Content modification: The user wants to modify the content of a webpage or its settings.
The high-level instruction should be creative. You need to deeply analyze the elements and executable actions on the interface to generate realistic, valuable, and executable tasks that can be completed within the current GUI. The instruction should be specific, actionable, and goal-oriented, ensuring the task can be completed on the current GUI by including all critical specifics such as file names, relevant timings, or required details.

Below is a brief description of the current website: {website_intro}
Here are some examples of High-Level Instruction for reference: {task_examples}
Please generate tasks that can be completed on the current platform, and avoid tasks that are unrelated to the current website.

You ONLY need to return a dictionary formatted as follows:
{
"Sub-Instruction": "xxx",
"Analysis": "xxx",
"High-Level-Instruction": "xxx"
}

Current Action: {current_action}
Website Name: {website_name}
RETURN ME THE DICTIONARY I ASKED FOR.

Prompt 12: Prompts for associating high-level tasks on web.

Evaluation Prompt for AndroidWorld

You are a GUI task expert, I will provide you with a high-level instruction, an action history, a screenshot with its corresponding accessibility tree.
High-level instruction: {high_level_instruction}
Action history: {action_history}
Accessibility tree: {a11y_tree}
Please generate the low-level thought and action for the next step.

Prompt 13: Prompts for evaluating our agents on AndroidWorld.

Evaluation Prompt for AndroidControl: High-Level Settings
<image>
You are a GUI task expert, I will provide you with a high-level instruction, an action history, a screenshot with its corresponding accessibility tree.
High-level instruction: {high_level_instruction}
Action history: {action_history}
Accessibility tree: {a11y_tree}
Please generate the low-level thought and action for the next step.

Evaluation Prompt for AndroidControl: Low-Level Settings
<image>
You are a GUI task expert, I will provide you with a high-level instruction, an action history, a screenshot with its corresponding accessibility tree, and a low-level thought.
High-level instruction: {high_level_instruction}
Action history: {action_history}
Accessibility tree: {a11y_tree}
Low-level thought: {low_level_thought}
Please generate the action for the next step.

Prompt 14: Prompts for evaluating our agents on AndroidControl.

Evaluation Prompt for AndroidControl: High-Level Settings

```
<image>
You are a GUI task expert, I will provide you with a high-level instruction, an action history,
a screenshot with its corresponding accessibility tree.
High-level instruction: {high_level_instruction}
Action history: {action_history}
Accessibility tree: {a11y_tree}
Please generate the low-level thought and action for the next step.
Candidate Actions:
“action_type”: “type”, “text”: <text_input>, “x”: <x_coordinate>, “y”: <y_coordinate>
“action_type”: “navigate_home”
“action_type”: “navigate_back”
“action_type”: “scroll”, “direction”: <up, down, left, or right>
“action_type”: “open_app”, “app_name”: <app_name>
“action_type”: “wait”
“action_type”: “dismiss”, “x”: <x_coordinate>, “y”: <y_coordinate>
“action_type”: “long_press”, “x”: <x_coordinate>, “y”: <y_coordinate>
“action_type”: “get_text”, “x”: <x_coordinate>, “y”: <y_coordinate>
You need to generate a script in the form:
thoughts:
{THOUGHTS}
actions:
{ACTION}
Make sure to consider the details in the screenshot and the task requirements to create an accurate
and functional script.
```

Evaluation Prompt for AndroidControl: Low-Level Settings

```
<image>
You are a GUI task expert, I will provide you with a high-level instruction, an action history,
a screenshot with its corresponding accessibility tree, and a low-level thought.
High-level instruction: {high_level_instruction}
Action history: {action_history}
Accessibility tree: {a11y_tree}
Low-level thought: {low_level_thought}
Please generate the action for the next step.
Candidate Actions:
“action_type”: “type”, “text”: <text_input>, “x”: <x_coordinate>, “y”: <y_coordinate>
“action_type”: “navigate_home”
“action_type”: “navigate_back”
“action_type”: “scroll”, “direction”: <up, down, left, or right>
“action_type”: “open_app”, “app_name”: <app_name>
“action_type”: “wait”
“action_type”: “dismiss”, “x”: <x_coordinate>, “y”: <y_coordinate>
“action_type”: “long_press”, “x”: <x_coordinate>, “y”: <y_coordinate>
“action_type”: “get_text”, “x”: <x_coordinate>, “y”: <y_coordinate>
You need to generate a script in the form:
thoughts:
{THOUGHTS}
actions:
{ACTION}
Make sure to consider the details in the screenshot and the task requirements to create an accurate
and functional script.
```

Prompt 15: Prompts for evaluating base models (Zero-Shot) on AndroidControl.

Evaluation Prompt for WebArena
<image>
Task Description
You are an intelligent agent completing web-based tasks.
Based on the user's objective (i.e. instruction), current interface information (i.e. screenshot and its corresponding accessibility tree), and action history, determine the next action.

Available Actions
- click [id]: This action clicks on an element with a specific id on the webpage.
- type [id] [content] [press_enter_after=0|1]: Use this to type the content into the field with id. By default, the "Enter" key is pressed after typing unless press_enter_after is set to 0.
- hover [id]: Hover over an element with id.
- press [key_comb]: Simulates the pressing of a key combination on the keyboard (e.g., Ctrl+v).
- scroll [direction=down|up]: Scroll the page up or down.
- new_tab: Open a new, empty browser tab.
- tab_focus [tab_index]: Switch the browser's focus to a specific tab using its index.
- close_tab: Close the currently active tab.
- goto [url]: Navigate to a specific URL.
- go_back: Navigate to the previously viewed page.
- go_forward: Navigate to the next page (if a previous go_back action was performed).
- stop [answer]: Issue this action when you believe the task is complete. If the objective is to find a text-based answer, provide the answer in the bracket. If you believe the task is impossible to complete, provide the answer as "N/A" in the bracket.

Output Format
First, generate the reasoning process for the action. Then, generate the action in the correct format. Start with a "In summary, the next action I will perform is" phrase, followed by action inside ```.
For example:
"Let's think step-by-step. To add a product to the shopping cart, I need to navigate to the catalog or product section. The "CATALOG" link is available with ID [1234]. In summary, the next action I will perform is ```click [1234]```".

Instruction: {instruction}
Accessibility tree: {a11y_tree}
Action History: {action_history}
What's the next action?

Prompt 16: Prompts for evaluating our agents on WebArena.

Evaluation Prompt for WebArena

prompt = { “intro”: “““You are an autonomous intelligent agent tasked with navigating a web browser. You will be given web-based tasks. These tasks will be accomplished through the use of specific actions you can issue.
Here’s the information you’ll have: The user’s objective: This is the task you’re trying to complete. The current web page’s accessibility tree: This is a simplified representation of the webpage, providing key information. The current web page’s URL: This is the page you’re currently navigating. The open tabs: These are the tabs you have open. The previous action: This is the action you just performed. It may be helpful to track your progress. The screenshot of current webpage: This .png image will be input as base64 format and the image is for you to better understand the web page, providing key information.
The actions you can perform fall into several categories:
Page Operation Actions:
`click [id]`: This action clicks on an element with a specific id on the webpage. Note that you CAN ONLY answer the id (a number) instead of clicking a text like ‘click [month]’.
`type [id] [content] [press_enter_after=0|1]`: Use this to type the content into the field with id. By default, the “Enter” key is pressed after typing unless press_enter_after is set to 0.
`hover [id]`: Hover over an element with id.
`press [key_comb]`: Simulates the pressing of a key combination on the keyboard (e.g., Ctrl+v).
`scroll [down/up]`: Scroll the page up or down. You need to output the command like scroll [down] to scroll down.
Tab Management Actions:
`new_tab`: Open a new, empty browser tab.
`tab_focus [tab_index]`: Switch the browser’s focus to a specific tab using its index.
`close_tab`: Close the currently active tab.
URL Navigation Actions:
`goto [url]`: Navigate to a specific URL.
`go_back`: Navigate to the previously viewed page.
`go_forward`: Navigate to the next page (if a previous ‘go_back’ action was performed).
Completion Action:
`stop [answer]`: Issue this action when you believe the task is complete. If the objective is to find a text-based answer, provide the answer in the bracket. If you believe the task is impossible to complete, provide the answer as “N/A” in the bracket.
Homepage: If you want to visit other websites, check out the homepage at http://homepage.com. It has a list of websites you can visit. http://homepage.com/password.html lists all the account names and passwords for the websites. You can use them to log in to the websites.
To be successful, it is very important to follow the following rules:
1. You should only issue an action that is valid given the current observation.
2. You should only issue one action at a time.
3. You should follow the examples to reason step by step and then issue the next action.
4. Generate the action in the correct format. Start with a “In summary, the next action I will perform is” phrase, followed by the action inside ``````. For example,“In summary, the next action I will perform is ```click [1234]```”.
5. Issue stop action when you think you have achieved the objective. Don’t generate anything after stop.”””
“examples”: [(“““OBSERVATION: [1744] link ‘HP CB782A#ABA 640 Inkjet Fax Machine (Renewed)’, [1749] StaticText ’$279.49’, [1757] button ‘Add to Cart’, [1760] button ‘Add to Wish List’, [1761] button ‘Add to Compare’, URL: http://onestopmarket.com/office-products/office-electronics.html OBJECTIVE: What is the price of HP Inkjet Fax Machine PREVIOUS ACTION: None”””, “Let’s think step-by-step. This page lists the information of HP Inkjet Fax Machine, which is the product identified in the objective. Its price is $279.49. I think I have achieved the objective. I will issue the stop action with the answer. In summary, the next action I will perform is ```stop [$279.49]```”,), (“““OBSERVATION: [164] textbox ‘Search’ focused: True required: False [171] button ‘Go’ [174] link ‘Find directions between two points’ [212] heading ‘Search Results’ [216] button ‘Close’ URL: http://openstreetmap.org OBJECTIVE: Show me the restaurants near CMU PREVIOUS ACTION: None”””, “Let’s think step-by-step. This page has a search box whose ID is [164]. According to the Nominatim rule of OpenStreetMap, I can search for the restaurants near a location by “restaurants near”. I can submit my typing by pressing Enter afterwards. In summary, the next action I will perform is ```type [164] [restaurants near CMU] [1]```”,),],
“template”: “““OBSERVATION: observation, URL: url, OBJECTIVE: objective, PREVIOUS ACTION: previous_action”””,
“meta_data”: { “observation”: “accessibility_tree”, “action_type”: “id_accessibility_tree”, “keywords”: [“url”, “objective”, “observation”, “previous_action”], “prompt_constructor”: “CoTPromptConstructor”, “answer_phrase”: “In summary, the next action I will perform is”, “action_splitter”: “```” }, }

Prompt 17: Prompts for evaluating base models (Zero-Shot) on WebArena.

Trajectory Reward Model Prompt

You are an expert in evaluating GUI agent task trajectories. Your task is to assess the quality and effectiveness of task trajectories for GUI manipulation tasks.
A trajectory consists of the following components:
1. High-level Instruction: Describes the user's intended task (e.g., "Create a new blank project name 'OS-Genesis'").
2. Action History: Includes two key parts:
- Reasoning and Action for Each Step: A sequence of actions performed by the agent, including the reasoning thought and final executed action.
- GUI Screenshots: Screenshots of the last state: (if there are at least three states; otherwise, include all states).

When evaluating a trajectory, consider these key aspects:
Evaluation Criteria:
1. Trajectory Coherence:
- Do the low-level steps and corresponding actions follow a logical sequence toward the goal?
- Are the actions clearly described and specific?
- Are there redundant or unnecessary actions?

2. Task Completion:
- Does the trajectory successfully achieve the instructed task?
- Are all necessary interactions completed?
- Are error cases handled appropriately?

Scoring Guidelines:
Rate the trajectory on a scale of 1 to 5 based on the evaluation criteria:
- 5: The task is perfectly completed, successfully executing multiple actions to achieve the goal. The sequence is logically clear with no noticeable redundancies.
- 4: The task is mostly completed, successfully executing multiple actions. However, due to challenges or ambiguities in the instructions, the completion is not perfect, or there are inefficiencies in the process.
- 3: The task is partially completed, with some successful actions executed. However, due to task or environmental constraints, the goal is not fully achieved, or the sequence ends in a loop or error.
- 2: Only a few actions are executed. Although there is an attempt to complete the task, the trajectory deviates from the goal early on or demonstrates significant inefficiencies in execution and logic.
- 1: The task fails completely, with no meaningful actions executed at the start. The sequence either falls into an immediate deadlock, a repetitive loop, or demonstrates no value in completing the task. Or the tasks are completely inaccessible.

Note: If the task is relatively complex, but the trajectory demonstrates valuable attempts, even if the task is not fully completed, consider adjusting the score upward. However, if the task is complex but the trajectory fails to perform actions that contribute meaningfully to task completion, no extra points should be awarded.
You need to judge the score based on the agent's actions and screenshots combined.
Response Format:
Format your response into two lines as shown below:
Reason: <your thoughts and reasoning process for the score>
Score: <your score from 1-5>

Prompt 18: Prompts for Trajectory Reward Model