Ponder & Press: Advancing Visual GUI Agent towards General Computer Control

Yiqin Wang* **Haoji Zhang*** **Jingqi Tian** **Yansong Tang†**
Shenzhen International Graduate School, Tsinghua University
{yq-wang23@mails.,tang.yansong@sz.}tsinghua.edu.cn

Abstract

Most existing GUI agents typically depend on non-vision inputs like HTML source code or accessibility trees, limiting flexibility across diverse software environments and platforms. Current multimodal large language models (MLLMs), though excel at using vision to ground real-world objects, often struggle with accurately localizing GUI elements – a critical requirement for effective GUI automation – due to the semantic gap between real-world objects and GUI elements. In this work, we introduce Ponder & Press, a divide-and-conquer framework for general computer control that uses only visual input. Our approach combines a general-purpose MLLM as an 'interpreter', responsible for translating high-level user instructions into detailed action descriptions, with a GUI-specific MLLM as a 'locator' that precisely locates GUI elements for action placement. By leveraging a purely visual input, our agent offers a versatile, human-like interaction paradigm applicable to various applications. Ponder & Press locator outperforms existing models by +22.5% on the ScreenSpot GUI grounding benchmark. More offline and interactive agent benchmarks across various GUI environments – including web pages, desktop software, and mobile UIs – demonstrate that the Ponder & Press framework achieves state-of-the-art performance, highlighting the potential of visual GUI agents.

1 Introduction

Researchers have long pursued the development of autonomous agents to assist humans in interacting with various GUI devices (Shi et al., 2017; Yao et al., 2022; Li et al., 2020). With recent advances in Large Language Models (LLMs) (Brown et al., 2020; Achiam et al., 2023; Touvron et al., 2023; Anthropic, 2024), agents for Web browsing (Gur et al., 2024), office automation (Wu et al., 2024; Tan et al., 2024), and mobile apps (Rawles et al., 2024b) have been proposed to streamline user interactions and improve productivity. Major technology companies have also contributed to this development by creating agents that facilitate user experiences, such as Apple Siri, Microsoft 365 Copilot, and Capcut smart video editor.

Despite these advancements, existing GUI automation approaches face limitations in generalizability and adaptability across software environments. First, software-specific agents from tech companies often operate beneath the GUI layer, bypassing user-facing elements, and thus sacrificing generalization by interacting directly with underlying code. Second, most GUI agents (Shi et al., 2017; Humphreys et al., 2022; Gur et al., 2024; Yao et al., 2022; Li et al., 2020; Zhou et al., 2023; Deng et al., 2024) developed by the research community rely on additional information such as HTML, DOM, or accessibility trees, making them specific to certain platforms and software environments. Human interaction with GUIs, on the contrary, relies exclusively on visual input and interaction through actions such as mouse clicks, keyboard input, and screen taps. **Therefore, a robust GUI agent designed for broad applicability must ideally be able to operate using only visual input, similar to human perception, and output actions in a human-like manner.**

Developing vision-only general agents capable of human-like interactions with GUIs presents significant challenges as follows:

- **Task Decomposition:** Interpreting and breaking down high-level task instructions into a series of executable actions within a software environment, ensuring that the GUI agent executes the correct action.
- **Precise GUI Localization:** Accurately localizing GUI elements to facilitate correct action placement, such as clicks or text inputs.

*Equal contribution. † Correspondence to Yansong Tang.

Findings of the Association for Computational Linguistics: ACL 2025, pages 1461–1473

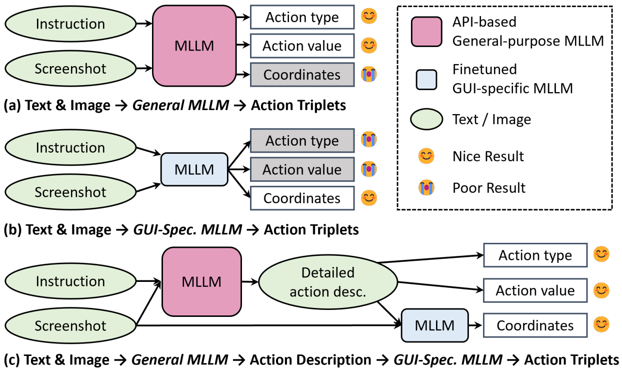

Figure 1: **Different types of frameworks for vision-based GUI agents.**

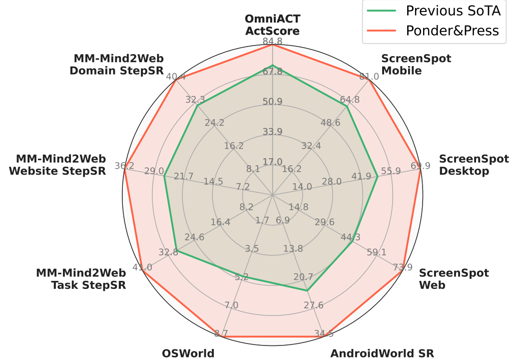

Figure 2: ***Ponder&Press* improves vision-based GUI agents on a broad range of tasks.**

As shown in Figure 1 (a) and (b), previous efforts have sought to build end-to-end models that address both challenges simultaneously. High-level user instructions are directly mapped to action types, action values, and pixel coordinates in a single inference. However, this approach struggles due to the significant difference between the textual nature of actions and values, and the numerical nature of pixel coordinates. As shown in Figure 1 (a), general-purpose end-to-end multimodal models (MLLMs) (Achiam et al., 2023; Anthropic, 2024; Wang et al., 2024) often suffer from poor grounding performance (refer to Table 1 for proof). As shown in Figure 1 (b), GUI-specific models (Cheng et al., 2024), though specialized in GUI grounding, struggle to effectively decompose complex user instructions. As a result, these types of models suffer from poor accuracy in predicting the action type (e.g. TYPE or CLICK) and action value (e.g. the typed content). The claims made above are proved in the experiment section.

In this paper, we introduce a divide-and-conquer framework called Ponder & Press. It follows the design presented in Figure 1 (c), leveraging the user-instruction interpretation ability of general-purpose MLLM, as well as the grounding ability of GUI-specific MLLM.

As further shown in Figure 3, the framework is composed of two distinct stages that deal with the two challenges separately: **(1) The 'Ponder' stage**, involves an **Instruction Interpreter** that converts high-level user goals into executable steps. For instance, as shown in Figure 3, when tasked with finding the stock price of 'Netflix' on Google Finance, the interpreter outputs: 'To find the latest price of Netflix stock, I need to search for Netflix in the Google Finance platform. The search bar is visible at the top of the page, so I'll use that to enter [Netflix]', along with a structured output "Action: TYPE, Value: 'Netflix', Element Description: 'Search bar with placeholder text [Search for stocks, ETFs & more]'". This stage leverages the commonsense knowledge embedded in MLLMs to bridge the gap between high-level user instructions and textual, structured action descriptions. **(2) The 'Press' stage**, where we train a **Visual Element Locator** to map the 'Element Description' to pixel coordinates, requiring only a small labeled dataset while achieving state-of-the-art performance.

This modular design allows the agent to accurately understand user intent and execute precise actions (Liu et al., 2025), maintaining flexibility for general software control. Furthermore, by relying solely on visual inputs—without the need for HTML, accessibility trees, or other supplementary data—our purely visual GUI agent enhances generalizability across various platforms, avoiding the need for software-specific modifications.

Our main contributions are as follows:

- We propose Ponder & Press, a divide-and-conquer GUI agent framework that only relies on visual input to mimic human-like interaction with GUIs. It guarantees generalizability across diverse environments.
- We evaluate Ponder & Press locator on the GUI grounding benchmark *ScreenSpot*, outperforming previous state-of-the-art model by +22.5%.
- We further conducted extensive evaluations of our framework on 4 widely used GUI agent benchmarks, demonstrating the effectiveness of our agent in offline, online, desktop, webpage, and mobile settings.

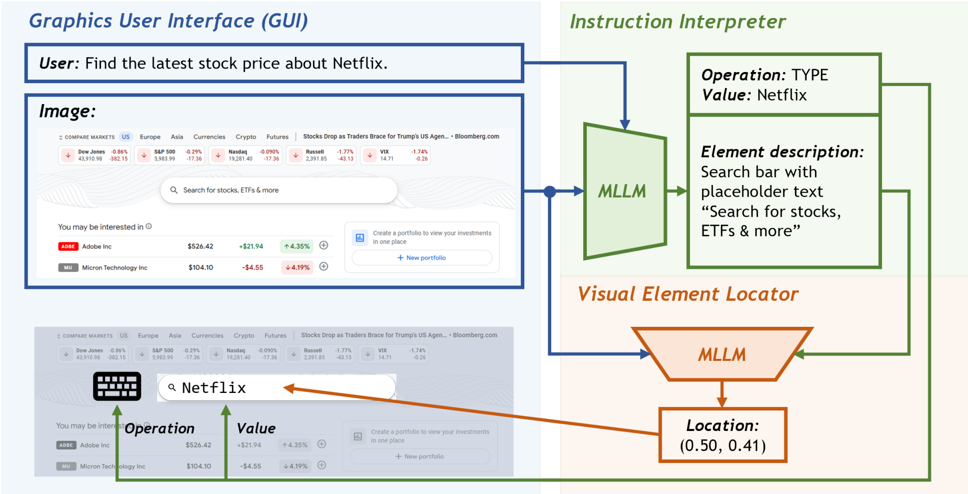

Figure 3: **The framework of *Ponder&Press* agent.** The framework consists of two core components: an **Instruction Interpreter** that translates high-level user instructions into actionable steps, and a **Visual Element Locator** that localizes GUI elements for interactions such as clicking or typing. Our method ensures that complex instructions can be decomposed and precisely executed within diverse GUIs.

2 Related Work

2.1 Autonomous Agents for GUI Devices

System-specific agents developed by technology companies, such as Apple Siri and Microsoft Copilot, are typically integrated beneath the GUI layer and lack a mechanism to generalize to arbitrary software interfaces without internal system access. In contrast, many GUI agents developed by the research community (Yao et al., 2022; Kim et al., 2023; Zhou et al., 2023; Deng et al., 2024) are designed to work with various GUIs but often rely on HTML, DOM, or accessibility trees as input sources to locate elements. This reliance on non-visual data sources limits their ability to generalize to GUIs that do not expose internal structural data.

Efforts to build human-like vision-only GUI agents aim to overcome these limitations (Shaw et al., 2023; Hong et al., 2024; Cheng et al., 2024; Zheng et al., 2024; Lu et al., 2024). However, existing vision-only agents often face challenges with task decomposition and localization precision. Single-step end-to-end models that predict both actions and pixel coordinates in a single inference tend to struggle with the modality gap between action description and numerical coordinates, leading to restricted planning ability (Cheng et al., 2024). Our work addresses these issues by adopting a divide-and-conquer approach to separate task planning from localization, enhancing both the generalizability and precision of our GUI agent.

2.2 Multimodal Large Language Models

General-purpose commercial MLLMs(Achiam et al., 2023; Anthropic, 2024) excel in common-sense reasoning and high-level planning, making them suitable for interpreting complex instructions within a GUI context. Open-source MLLMs such as LLaVA (Liu et al., 2023, 2024b) and Qwen2-VL (Wang et al., 2024) are designed to solve various vision-related tasks. These models are particularly effective when applied to familiar visual domains that match their training distribution, such as grounding real-world objects. However, their performance declines in out-of-distribution (OOD) scenarios, such as novel GUI layouts, due to limited training data and a narrow generalization capacity.

In our approach, we leverage the instruction interpretation ability of general-purpose MLLMs for task decomposition while addressing GUI localization with a dedicated visual grounding model, leveraging the commonsense GUI knowledge as well as GUI-specific grounding ability.

2.3 Visual Grounding

Due to the semantic gap between real-world images and GUI images, general visual grounding models (Wang et al., 2024) suffer from severe performance drops on GUI data, as further shown in our experiment section. Screenshot marks (Yang et al., 2023a; Liu et al., 2024a), chain-of-thought methods (Wei et al., 2022) , or explicitly extract GUI elements from screenshot (Lu et al., 2024) serve as training-free workarounds to help MLLM understand the relative position between different visual elements. Still, they rely on explicitly performing another stage of image segmentation, object tracking, or API-based model calling, which is inconvenient and may involve additional errors.

In order to enhance grounding performance on GUI data, (Bai et al., 2021; Qian et al., 2024)build datasets to bridge the gap between natural language, GUI element, and its location. CogAgent (Hong et al., 2024) conducted large-scale pretraining on datasets including 400k webpage screenshots and further finetuned on human-annotated restricted internal datasets. SeeClick (Cheng et al., 2024) open-sourced a GUI visual grounding training set consisting of 1M data. Our approach builds on these advances by training a GUI-specific grounding model with a labeled dataset, translating structured action descriptions into precise pixel coordinates. This modular design facilitates robust and efficient GUI localization across diverse environments.

3 Method

3.1 Task Formulation

Consider a GUI environment $\mathcal{E}$ (e.g., an office software, a web page, a mobile app interface, etc.) and a task $\mathcal{T}$ (e.g., 'Find the latest news about Netflix stock.'). The agent's goal is to produce a sequence of executable actions $\mathcal{A} = [\alpha_1, \alpha_2, \ldots, \alpha_m]$ to complete the task. At each step k, the agent ρ must generate an action α_k based on the current visual observation o_k, previous actions $\{\alpha_1, \alpha_2, \ldots, \alpha_{k-1}\}$, and the task $\mathcal{T}$:

$$\alpha_k = \rho(o_k, \mathcal{T}, \{\alpha_1, \alpha_2, \ldots, \alpha_{k-1}\})$$

In this setting, the observation o_k is purely visual. We have $o_k = i_k$ at each step k, with i_k representing the screenshot input. No structured HTML code, DOM tree, accessibility tree, or any other text-based information is available. All environment understanding must be derived from the current screenshot. The state of the GUI environment $\mathcal{E}$ updates after each action as follows:

$$o_{k+1} = \mathcal{E}(\alpha_k)$$

Each action α corresponds to an application or system event within the environment, represented as a triplet:

$$\alpha = (\eta, \omega, \nu)$$

Here, η represents a target location (e.g., '[0.50, 0.20]') as a pixel coordinate on the screen, denoting the position where the 'Click', 'Type', or 'Select' operation should be executed. $\omega \in \mathcal{O}$ specifies the intended operation type (e.g., 'Type'), and ν provides any additional value required for the action (e.g., the type content 'Netflix'). The set $\mathcal{O}$ encompasses all allowable operations in $\mathcal{E}$.

3.2 Framework Design

It is challenging for multimodal language models (MLLMs) to produce the action triplet (η, ω, ν) in a single inference step. Specifically, generating ω and ν requires strong planning abilities, contextual reasoning, and domain-specific knowledge of the GUI, while determining η demands precise and accurate grounding of GUI elements. As shown in the experiments section, existing end-to-end models exhibit relatively low performance on this challenging task. To address this issue, we introduce an intermediate variable $\mathcal{D}$, a textual description of the target element that serves as a reliable and interpretable bridge for accurate grounding.

Our approach follows a two-stage process:

1. **Instruction Interpretation**: The first model ϕ functions as an instruction interpreter. Given the current screenshot o_k, previous actions $\{\alpha_1, \alpha_2, \ldots, \alpha_{k-1}\}$, and the task $\mathcal{T}$, it generates the intermediate output $(\mathcal{D}, \omega, \nu)$:

$$(\mathcal{D}, \omega, \nu) = \phi(o_k, \mathcal{T}, \{\alpha_1, \alpha_2, \ldots, \alpha_{k-1}\}),$$

where ϕ encapsulates task interpretation abilities.

2. **GUI Element Localization**: The second model ψ functions as a visual GUI element locator. Given the current screenshot o_k and the textual description $\mathcal{D}$, it determines the relative coordinates η on the screen:

$$\eta = \psi(o_k, \mathcal{D}),$$

where ψ represents the grounding function for locating GUI elements.

The final action triplet (η, ω, ν), thus obtained, can be executed within the GUI environment to get the next observation o_{k+1}.

3.3 Instruction Interpreter

The Instruction Interpreter translates high-level task instructions into structured components for GUI interaction, producing $(\mathcal{D}, \omega, \nu)$ in a single inference. Here, $\mathcal{D}$ is a textual description of the target element, ω denotes the intended operation (e.g., Click, Type), and ν provides additional input required for the action, such as specific text or dates.

We employ two multimodal models—GPT-4o and Claude 3.5 Sonnet—that process screenshots alongside the task $\mathcal{T}$ and prior actions $\{\alpha_1, \alpha_2, \ldots, \alpha_{k-1}\}$. Given these inputs, each model generates a text output containing $(\mathcal{D}, \omega, \nu)$:

$$(\mathcal{D}, \omega, \nu) = \phi(o_k, \mathcal{T}, \{\alpha_1, \alpha_2, \ldots, \alpha_{k-1}\}),$$

where ϕ represents the instruction interpreter. Each output is extracted directly from the model's single-text response.

3.4 Visual Element Locator

The Visual Element Locator module is tasked with accurately identifying and locating GUI elements within a screenshot, positioning this as a GUI visual grounding task. The objective is to produce the normalized coordinates (x, y) of the target element, with values constrained to $0 \leq x, y \leq 1$.

For this purpose, we use Qwen2-VL-Instruct (Wang et al., 2024) as the pretrained model and further finetune it with LoRA (Hu et al., 2021) on a GUI-specific data subset sampled from (Cheng et al., 2024). This finetuning enhances the model's capacity to localize GUI elements effectively across diverse interfaces.

The Locator computes the coordinates $\eta = (x, y)$ based on the current screenshot observation o_k and the textual description $\mathcal{D}$ generated by the Instruction Interpreter, using the function ψ:

$$\eta = \psi(o_k, \mathcal{D}).$$

To train the model to output these coordinates, we avoid explicit numerical loss. Instead, we treat the prediction as a natural language next-token-prediction task. We prompt the model following the prompt template presented in (Cheng et al., 2024) as follows:

> *"In this UI screenshot, what is the position of the element corresponding to the description {DESCRIPTION } (with point)?"*

This setup encourages the model to generate (x, y) as part of a structured textual response, effectively supporting GUI-specific localization in a multimodal environment.

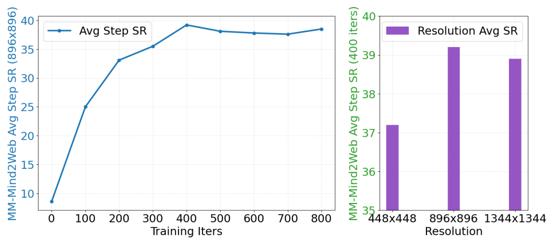

Figure 4: **The learning curve of *Ponder&Press* locator.** The performance peaked at 400 iters and the best input screenshot resolution is 896x896. Tests are conducted on Multimodal-Mind2Web (Deng et al., 2024).

3.5 Training Details

Ponder & Press locator is based on the Qwen2-VL (Wang et al., 2024) model, leveraging its initial multimodal grounding capabilities. In order to fit its output space to GUI data, we apply LoRA (Hu et al., 2021) to adapt both the visual encoder and language model layers using the same training set as SeeClick (Cheng et al., 2024).

With batch size = 64, training converged after 400 steps on 8 NVIDIA A100 GPUs as shown in Figure 4, consuming approximately 2 hours. This utilizes merely 25,600 data samples, which is a 2.5% subset of the original SeeClick training set, revealing the high data-efficiency of parameter-efficient fine-tuning. We use the AdamW optimizer with a learning rate of 3×10^{-5} and apply a cosine annealing scheduler to manage learning rate decay. Best input resolution are 896×896, further increasing the resolution bring no performance gain as shown in Figure 4. Dynamic Resolusion (Dehghani et al., 2023) is employed to deal with extended input resolution, enabling resolution scaling without additional retraining, which is advantageous for GUI tasks requiring high visual detail.

4 Experiments

4.1 GUI Grounding Benchmark

ScreenSpot. To assess the grounding capabilities of Ponder & Press's Visual Element Locator, we evaluate it on the ScreenSpot dataset (Cheng et al., 2024), a benchmark specifically designed for GUI element localization. ScreenSpot encompasses over 600 diverse screenshots from mobile (iOS, Android), desktop (macOS, Windows), and web platforms, along with more than 1,200 instructions tied to actionable elements.

As shown in Table 1, Non-GUI-specific models (such as Qwen2-VL and GPT-4o) struggle with lo-

Table 1: **Comparisons with pure-vision methods on GUI grounding benchmark *ScreenSpot* (Cheng et al., 2024).** I/W denotes Icon/Widget. Results with * are from (Cheng et al., 2024). Ponder & Press's locator exhibits state-of-the-art performance in precisely locating GUI elements while maintaining a smaller model size.

Methods	Model Size	Commercial Model	GUI Specific	Mobile Text	Mobile I/W	Desktop Text	Desktop I/W	Web Text	Web I/W	Avg.
GPT-4o (Achiam et al., 2023)	N/A	w.	✗	23.4%	25.8%	17.5%	21.4%	10.9%	9.7%	18.1%
Claude 3.5 Sonnet (Anthropic, 2024)	N/A	w.	✗	37.6%	26.1%	29.0%	26.3%	17.4%	8.4%	24.1%
OmniParser (GPT-4V + GD) (Lu et al., 2024)	N/A	w.	✓	**94.8%**	53.7%	**89.3%**	44.9%	**83.0%**	45.1%	68.7%
Qwen2-VL (Wang et al., 2024)	7 B	w.o.	✗	41.4%	16.2%	25.3%	5.7%	12.2%	6.3%	17.8%
Fuyu* (Bavishi et al., 2023)	8 B	w.o.	✓	40.6%	1.6%	33.6%	6.7%	48.4%	2.9%	22.3%
CogAgent* (Hong et al., 2024)	18 B	w.o.	✓	66.5%	26.7%	73.7%	19.3%	78.0%	21.4%	47.6%
SeeClick* (Cheng et al., 2024)	9.6 B	w.o.	✓	78.0%	52.0%	72.2%	30.0%	55.7%	32.5%	53.4%
Ponder&Press locator	7 B	w.o.	✓	88.6%	**73.4%**	80.4%	**59.3%**	82.6%	**65.1%**	**74.9%**

Table 2: **Comparisons with pure-vision methods on web agent benchmark *Multimodal-Mind2Web* (Deng et al., 2024), in a zero-shot manner.** Claude denote Claude 3.5 Sonnet (Anthropic, 2024), Naive Guess denotes always ground on the center point of the screen, Ele.Acc denotes element accuracy, Step SR denotes step success rate. Results with * are from their original paper.

Visual Locator	Instruction Interpreter	GUI Specific	Cross-Task Ele.Acc	Cross-Task Step SR	Cross-Website Ele.Acc	Cross-Website Step SR	Cross-Domain Ele.Acc	Cross-Domain Step SR	Avg. Step SR
Naive Guess	Claude	✗	0.6%	0.5%	1.6%	1.4%	1.2%	0.9%	0.9%
Qwen2-VL	Claude	✗	9.1%	8.4%	11.2%	9.7%	8.7%	7.8%	8.6%
OmniParser(GPT4V + GD)*	GPT	✓	42.3%	38.7%	41.5%	36.1%	44.9%	36.8%	37.2%
SeeClick*	w/o	✓	26.3%	23.7%	21.9%	18.8%	22.1%	20.2%	20.9%
SeeClick	Claude	✓	34.9%	30.2%	32.9%	26.5%	36.1%	31.4%	29.4%
Ponder&Press	Claude	✓	**46.7%**	**41.0%**	**44.1%**	**36.2%**	**47.0%**	**40.4%**	**39.2%**

calization due to their lack of targeted knowledge about GUI structures and common interface patterns. GUI-specific fine-tuning equips the model with essential prior knowledge, enhancing its ability to navigate and precisely localize elements in varied and complex GUIs. **The Ponder&Press locator model achieves state-of-the-art performance across all GUI categories—mobile, desktop, and web—outperforming previous methods by a substantial margin.** In particular, Ponder & Press-7B locator surpasses the previous SoTA SeeClick (Cheng et al., 2024) with an average accuracy increase of over 20%, and even outperforms the commercial GUI locator model OmniParser (Lu et al., 2024) that utilizes GPT-4V and Grounding DINO. Our model demonstrates particular strength in locating icon and widget elements, underscoring the model's robust learning of such GUI-specific visual features.

4.2 Offline GUI Agent Benchmark

Multimodal Mind2Web (Deng et al., 2024). The benchmark includes three test splits: (1) Cross-Domain, (2) Cross-Website, and (3) Cross-Task. **We only focus on two core evaluation metrics: element accuracy (Ele.Acc)** and **step success rate (Step SR), omitting operation F1.** While operation F1 measures action type correctness, simply

Table 3: **Comparisons with pure-vision methods on desktop and web agent benchmark *OmniACT* (Kapoor et al., 2024).** Seq. denote sequence, Act. denote action, Claude denote Claude 3.5 Sonnet (Anthropic, 2024), Naive Guess denote always ground on the center point of the screen.

Visual Locator	Instruction Interpreter	Seq. Score	Act. Score	Click Penalty
Naive Guess	GPT-4o	30.9	55.8	36.7
	Claude	39.3	58.8	32.8
QWen2-VL	GPT-4o	30.9	70.4	22.1
	Claude	39.3	70.0	21.5
SeeClick	GPT-4o	30.9	73.0	19.5
	Claude	39.3	73.1	18.4
Ponder&Press	GPT-4o	30.9	**84.8**	**7.7**
	Claude	39.3	**82.9**	**8.6**

setting every operation as 'CLICK' yields high F1 scores across all splits (over 80%), but fails to capture the nuance required for varied interaction types, resulting in poor step success rates. Thus, the operation F1 metric is included only in the appendix to avoid misleading conclusions about the agent's effectiveness.

Table 2 illustrates that Ponder & Press achieves state-of-the-art performance on both element accuracy and step success rate across all three splits. With the same interpreter, the locator of Ponder & Press demonstrates a substantial

Table 4: **Comparisons with visual methods on online GUI agent benchmark *OSWorld* (Xie et al., 2024).** Results with * are from OSWorld (Xie et al., 2024). Ponder & Press exhibit a unified performance boost across all subsets compared to the GPT-4o baseline.

Methods			Office (117)	OS (24)	Daily (78)	Workflow (101)	Professional (49)	All (369)
Human*			71.8	75.0	70.5	73.3	73.5	72.4
			Single-stage method					
CogAgent*			0.9	4.2	2.7	0.0	0.0	1.1
GPT-4o*			3.6	8.3	6.1	5.6	4.1	5.0
Agent	Locator	Interpreter	Two-stage method					
Ponder&Press	Naive Guess	GPT-4o	0.0	0.0	0.0	0.0	0.0	0.0
Ponder&Press	SeeClick	GPT-4o	5.1	16.7	7.7	5.0	6.1	6.5
Ponder&Press	**Ponder&Press**	**GPT-4o**	**6.8**	**16.7**	**12.8**	**5.0**	**10.2**	**8.7**

Table 5: **Comparisons with methods on interactive mobile GUI agent benchmark *AndroidWorld* (Rawles et al., 2024a). P&P denote Ponder&Press.** Results with * are from AndroidWorld (Rawles et al., 2024a). With only visual input, the Ponder & Press agent, equipped with the visual locator, exhibits state-of-the-art performance on Success Rates.

Agent	Visual Locator	Instruction Interpreter	Success Rates
Human*	-	-	80%
Input: A11y tree			
M3A*	GPT-4 Turbo	GPT-4 Turbo	30.6%
Input: Screenshot + A11y tree			
M3A*	GPT-4 Turbo	GPT-4 Turbo	25.4%
Input: Screenshot			
P&P	Naive Guess	GPT-4o	0.0%
P&P	SeeClick	GPT-4o	23.3%
P&P	**P&P**	**GPT-4o**	**34.5%**

improvement (+9.8%) compared to the previous SoTA SeeClick(Cheng et al., 2024). Moreover, when we equip SeeClick's original model with our Claude interpreter, the model also demonstrates a significant boost in performance (+8.5%). This underscores the generalizability of our interpretation stage, which enhances GUI-specific grounding accuracy and task success across diverse and dynamic web contexts. Intuitively, our locator model outperforms non-GUI-specific models (such as Qwen2-VL) by a significant margin (+30.6%) and even outperforms the commercial GUI locator model OmniParser (+2.0%) that utilizes GPT-4V and Grounding DINO for locating, confirming the effectiveness of its grounding capabilities honed on GUI-specific data. This also clearly proves that a strong locator is necessary to fully unleash the potential of the instruction interpreter.

OmniACT(Kapoor et al., 2024). OmniACT benchmark (Kapoor et al., 2024) consists of over 9.8K pairs of images and instructions from various OS and the web. It employs two evaluation metrics: **sequence score** and **action score**. The sequence score measures whether the predicted action sequence matches the ground truth, while the action score evaluates how well the generated code snippet performs the task. Note that the sequence score is only impacted by the accuracy of the action sequence produced, making it an independent metric from the action score that measures action placement precision.

As shown in Table 3, our results demonstrate that Ponder & Press achieves state-of-the-art performance in terms of the action placement. The superiority in action scores is primarily driven by a significant reduction in click penalties. When equipped with GPT-4o as the interpreter, Ponder & Press demonstrates a 11.8% increase in Action Score while simultaneously decreasing exactly 11.8% in Click Penalty, compared to the previous SoTA SeeClick. This confirms that our model excels at precisely locating the specific GUI element that needs to be clicked—an essential component of effective grounding. Furthermore, the Claude 3.5 Sonnet interpreter consistently outperforms GPT-4o in both action prediction and element description, which underscores the importance of a strong interpreter for effective task execution.

It is worth noting that **the evaluation script open-sourced by OmniACT does not align with the formula provided in Section 4 of their paper(Kapoor et al., 2024)**, where the action score should only be calculated for those action sequences that match. Instead, their script calculates the action score across all samples, including those where the sequence is mismatched. To avoid confusion and ensure a fair comparison, we do not directly compare our results with those reported in the OmniACT paper as those results are much lower than ours due to the mis-implementation of the formula. **Our independent evaluation, following the correct action score formula, highlights the superior grounding and planning abilities of Ponder & Press. Please refer to the appendix for detailed discussion.**

4.3 Interactive Online GUI Agent Benchmark

OSWorld (Xie et al., 2024). OSWorld is a computer environment designed to evaluate multimodal GUI agents in an execution-based manner. OSWorld proposed a benchmark of 369 computer tasks, involving OS-related tasks, office-related tasks, as well as desktop application operating tasks. As shown in Table 4, Ponder&Press outperforms the GPT-4o baseline in all task categories, with notable performance gains in office-related, OS-related, and daily tasks. This unified improvement across the benchmark solidifies the effectiveness of our agent in handling a wide range of GUI-related tasks. Furthermore, the agent equipped with the Ponder&Press locator surpasses the one with the SeeClick locator, which proves the precise localization capabilities of Ponder&Press and aligns the results on the visual grounding benchmark. When equipping our agent with the SeeClick locator, it still surpasses the GPT-4o baseline, further proving the generalizability and effectiveness of our divide-and-conquer framework.

AndroidWorld (Rawles et al., 2024a). AndroidWorld is a dynamic Android environment that evaluates interactive mobile GUI agents across 116 tasks from 20 real-world apps. It generates tasks expressed in natural language, offering a flexible and realistic testing suite. As shown in Table 5, the Ponder& Press agent equipped with the Ponder&Press visual locator achieves a success rate of 34.5%, a notable improvement over the SeeClick locator variant (23.3%). This result validates the effectiveness of our locator model. Even when compared to the strong baseline M3A which utilizes additional A11y trees input, our agent demonstrates a superior performance, proving the effectiveness of our vision-only framework.

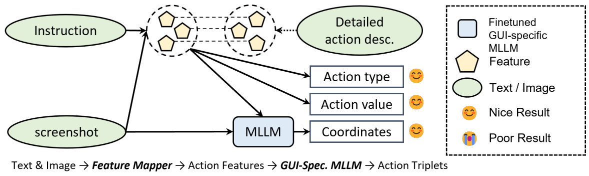

Figure 5: **A possible end-to-end visual GUI agent framework.**

4.4 Generalization on OOD data

Locator generalization in Out-of-Distribution (OOD) scenario is important for consistent user experience. To address this concern, we construct a fully OOD benchmark set for GUI grounding based on ScreenSpot called **ScreenSpot-P&P-OOD**, by applying Automatic Icon Removal with CLIP Similarity Filtering. Specifically, we apply our Ponder&Press locator to detect and extract all distinct icons from each screenshot. Then we compute the CLIP embedding of each detected icon and compare it against the icon set from the Ponder & Press training data. If the cosine similarity between a detected icon and any training icon exceeds a pre-defined threshold=0.85, it is removed from the benchmark.

From Table 6 we conclude that Ponder & Press locator performance slightly drops under OOD scenario but still remain competitive comparing to method without GUI-specific finetuning, proving its generalizing ability.

4.5 Inference latency

Inference latency is important for a smooth user experience. We test Ponder&Press on two benchmarks and list the latency in Table 7. The interpreter (Claude 3.5 Sonnet API) has a relatively stable inference time of 1.2s per step. The locator (Qwen2-VL-7B on a single NVIDIA RTX4090 with vLLM) has an inference time of 0.8s per step. Note that Multimodal-Mind2Web is an easier benchmark, requiring around 8 steps per task, while Android-world is harder, requiring around 15 steps per task.

5 Limitations

Just like other agents that involve API-based model (Rawles et al., 2024a; Tan et al., 2024; Lu et al., 2024), our agent inevitably involves inference latency and API cost. Previous single-stage methods suffer from either poorly mapping high-level user instruction into action description (Cheng

Table 6: **Performance comparisons between original *ScreenSpot* (Cheng et al., 2024) and *ScreenSpot-P&P-OOD*.** I/W denotes Icon/Widget. Ponder & Press's locator performance slightly drops under OOD scenario but still remain competitive.

Methods	Benchmark	Mobile		Desktop		Web		Avg.
		Text	I/W	Text	I/W	Text	I/W	
Claude 3.5 Sonnet	ScreenSpot	37.6%	26.1%	29.0%	26.3%	17.4%	8.4%	24.1%
Claude 3.5 Sonnet	ScreenSpot-P&P-OOD	36.8%	27.3%	28.5%	24.9%	16.9%	9.2%	23.9%
Ponder&Press locator	ScreenSpot	88.6%	73.4%	80.4%	59.3%	82.6%	65.1%	74.9%
Ponder&Press locator	ScreenSpot-P&P-OOD	84.8%	69.7%	76.5%	55.6%	78.4%	61.3%	71.1%

Table 7: **Inference latency of Ponder&Press.**

Benchmark	Interpreter Avg Step Time	Locator Avg Step Time	Avg Steps per Task
MM-Mind2Web	$1.231s$	$0.820s$	7.8
Android-world	$1.193s$	$0.804s$	15.2

et al., 2024; Hong et al., 2024), or poorly predicting the coordinates for action placement (Achiam et al., 2023; Anthropic, 2024). To resolve these problems while not involving an API-based model, we further propose a possible end-to-end visual GUI agent framework as future work as shown in Figure 5. We could align the features between (high-level instruction + screenshot) and (detailed action description) in the first training stage, and train the GUI-specific MLLM to enhance grounding ability in the second training stage. This framework could possibly encompass a strong GUI locating capability while retaining the task decomposition ability. The main challenge may lie in the collection of large-scale, high-quality training data.

6 Conclusion

We introduced Ponder&Press, a novel divide-and-conquer framework that enables general computer control using only visual input. **The 'Ponder' stage** interprets high-level instructions into actionable steps, **and the 'Press' stage** localizes GUI elements for precise action placement. By relying solely on visual input, Ponder&Press ensures generalizability across diverse software environments, eliminating the need for additional inputs such as HTML or accessibility trees. Ponder&Press achieved SoTA performance through extensive evaluations conducted across desktop, web, and mobile environments, demonstrating both precise localization capabilities and effective task decomposition. Our work highlights the potential of vision-based GUI agents for robust general-purpose automation, paving the way for human-like interactions with a wide range of software systems.

7 Acknowledgments

We thank the anonymous reviewers for carefully reading our paper and their insightful comments and suggestions.

This work was supported by Guangdong Natural Science Funds for Distinguished Young Scholar (No. 2025B1515020012).

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

AI Anthropic. 2024. Claude 3.5 sonnet model card addendum. *Claude-3.5 Model Card*.

Chongyang Bai, Xiaoxue Zang, Ying Xu, Srinivas Sunkara, Abhinav Rastogi, Jindong Chen, and Blaise Agüera y Arcas. 2021. Uibert: Learning generic multimodal representations for ui understanding. In *IJCAI*, pages 1705–1712.

Rohan Bavishi, Erich Elsen, Curtis Hawthorne, Maxwell Nye, Augustus Odena, Arushi Somani, and Sağnak Taşırlar. 2023. Introducing our multimodal models.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *NeurIPS*, pages 1877–1901.

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 *ACL*, pages 9313–9332.

Mostafa Dehghani, Basil Mustafa, Josip Djolonga, Jonathan Heek, Matthias Minderer, Mathilde Caron, Andreas Steiner, Joan Puigcerver, Robert Geirhos, Ibrahim M Alabdulmohsin, Avital Oliver, Piotr Padlewski, Alexey Gritsenko, Mario Lucic, and Neil Houlsby. 2023. Patch n' pack: Navit, a vision transformer for any aspect ratio and resolution. In *NeurIPS*, pages 2252–2274.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Sam Stevens, Boshi Wang, Huan Sun, and Yu Su. 2024. Mind2web: Towards a generalist agent for the web. *NeurIPS*.

Izzeddin Gur, Hiroki Furuta, Austin V Huang, Mustafa Safdari, Yutaka Matsuo, Douglas Eck, and Aleksandra Faust. 2024. A real-world webagent with planning, long context understanding, and program synthesis. In *ICLR*.

Wenyi Hong, Weihan Wang, Qingsong Lv, Jiazheng Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan Wang, Yuxiao Dong, Ming Ding, et al. 2024. Cogagent: A visual language model for gui agents. In *CVPR*, pages 14281–14290.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*.

Peter C Humphreys, David Raposo, Tobias Pohlen, Gregory Thornton, Rachita Chhaparia, Alistair Muldal, Josh Abramson, Petko Georgiev, Adam Santoro, and Timothy Lillicrap. 2022. A data-driven approach for learning to control computers. In *ICML*, pages 9466–9482.

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*.

Geunwoo Kim, Pierre Baldi, and Stephen McAleer. 2023. Language models can solve computer tasks. *arXiv preprint arXiv:2303.17491*.

Yang Li, Jiacong He, Xin Zhou, Yuan Zhang, and Jason Baldridge. 2020. Mapping natural language instructions to mobile ui action sequences. In *ACL*.

Benlin Liu, Yuhao Dong, Yiqin Wang, Yongming Rao, Yansong Tang, Wei-Chiu Ma, and Ranjay Krishna. 2024a. Coarse correspondence elicit 3d spacetime understanding in multimodal language model. *arXiv preprint arXiv:2408.00754*.

Haotian Liu, Chunyuan Li, Yuheng Li, and Yong Jae Lee. 2024b. Improved baselines with visual instruction tuning. In *CVPR*, pages 26296–26306.

Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. 2023. Visual instruction tuning. In *NeurIPS*, pages 34892–34916.

Jinpeng Liu, Wenxun Dai, Chunyu Wang, Yiji Cheng, Yansong Tang, and Xin Tong. 2025. Plan, posture and go: Towards open-vocabulary text-to-motion generation. In *ECCV*, pages 445–463.

Yadong Lu, Jianwei Yang, Yelong Shen, and Ahmed Awadallah. 2024. Omniparser for pure vision based gui agent. *Preprint*, arXiv:2408.00203.

Yijun Qian, Yujie Lu, Alexander Hauptmann, and Oriana Riva. 2024. Visual grounding for user interfaces. In *NAACL*, pages 97–107.

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

Christopher Rawles, Alice Li, Daniel Rodriguez, Oriana Riva, and Timothy Lillicrap. 2024b. Androidinthewild: A large-scale dataset for android device control. *NeurIPS*.

Peter Shaw, Mandar Joshi, James Cohan, Jonathan Berant, Panupong Pasupat, Hexiang Hu, Urvashi Khandelwal, Kenton Lee, and Kristina N Toutanova. 2023. From pixels to ui actions: Learning to follow instructions via graphical user interfaces. *NeurIPS*, pages 34354–34370.

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

Weihao Tan, Wentao Zhang, Xinrun Xu, Haochong Xia, Ziluo Ding, Boyu Li, Bohan Zhou, Junpeng Yue, Jiechuan Jiang, Yewen Li, Ruyi An, Molei Qin, Chuqiao Zong, Longtao Zheng, Yujie Wu, Xiaoqiang Chai, Yifei Bi, Tianbao Xie, Pengjie Gu, Xiyun Li, Ceyao Zhang, Long Tian, Chaojie Wang, Xinrun Wang, Börje F. Karlsson, Bo An, Shuicheng Yan, and Zongqing Lu. 2024. Cradle: Empowering foundation agents towards general computer control. *arXiv preprint arXiv:2403.03186*.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Peng Wang, Shuai Bai, Sinan Tan, Shijie Wang, Zhihao Fan, Jinze Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, et al. 2024. Qwen2-vl: Enhancing vision-language model's perception of the world at any resolution. *arXiv preprint arXiv:2409.12191*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *NeurIPS*, pages 24824–24837.

Zhiyong Wu, Chengcheng Han, Zichen Ding, Zhenmin Weng, Zhoumianze Liu, Shunyu Yao, Tao Yu, and Lingpeng Kong. 2024. Os-copilot: Towards generalist computer agents with self-improvement. *arXiv preprint arXiv:2402.07456*.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, et al. 2024. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *arXiv preprint arXiv:2404.07972*.

Jianwei Yang, Hao Zhang, Feng Li, Xueyan Zou, Chunyuan Li, and Jianfeng Gao. 2023a. Set-of-mark prompting unleashes extraordinary visual grounding in gpt-4v. *arXiv preprint arXiv:2310.11441*.

Jianwei Yang, Hao Zhang, Feng Li, Xueyan Zou, Chunyuan Li, and Jianfeng Gao. 2023b. Set-of-mark prompting unleashes extraordinary visual grounding in gpt-4v. *arXiv preprint arXiv:2310.11441*.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. 2022. Webshop: Towards scalable real-world web interaction with grounded language agents. *NeurIPS*, pages 20744–20757.

Boyuan Zheng, Boyu Gou, Jihyung Kil, Huan Sun, and Yu Su. 2024. GPT-4V(ision) is a generalist web agent, if grounded. In *ICML*, pages 61349–61385.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, et al. 2023. Webarena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*.

A Evaluation Details

A.1 ScreenSpot

Deal with redundant output. ScreenSpot (Cheng et al., 2024) evaluates a model's visual grounding performance based on its predicted relative coordinates (x, y), where $0 \leq x, y \leq 1$. For models not fine-tuned on GUI-specific data (Achiam et al., 2023; Anthropic, 2024; Wang et al., 2024), their grounding outputs often include redundant descriptive text alongside the (x, y) coordinates. To evaluate these models on ScreenSpot and accurately report their GUI grounding performance, we use Regular Expression (Regex) to extract the coordinates while discarding any extraneous information. The following Python function performs this coordinate extraction:

```
def extract_two_float_tuple(s):
    pattern = r'[\(\[\s]*([-+]?\d*\.\d+|\d+)\s*,\s*([-+]?\d*\.\d+|\d+)\s*[\)\]\s]*|-\s*[Xx]:\s*([-+]?\d*\.\d+|\d+)\s*(?:\([^\)]*\))?\s*-\s*[Yy]:\s*([-+]?\d*\.\d+|\d+)\s*(?:\([^\)]*\))?|-\s*[Tt]op:\s*([-+]?\d*\.\d+|\d+)\s*-\s*[Ll]eft:\s*([-+]?\d*\.\d+)|\(\s*[Xx]:\s*([-+]?\d*\.\d+|\d+)\s*,\s*[Yy]:\s*([-+]?\d*\.\d+|\d+)\s*\)'
    match = re.search(pattern, s, re.VERBOSE)
    if match:
        if match.group(1) and match.group(2):
            return float(match.group(1)), float(match.group(2))
        elif match.group(3) and match.group(4):
            return float(match.group(3)), float(match.group(4))
        elif match.group(5) and match.group(6):
            return float(match.group(6)), float(match.group(5))
        else:
            return float(match.group(8)), float(match.group(9))
    else:
        raise ValueError("String does not contain a valid '(float, float)', '[float, float]', '- X: float - Y: float', '- X: float (xxxxxx) - Y: float (xxxxxx)', '- Top: float - Left: float', or '(X: float, Y: float)' pattern")
```

In cases where the function raises an error due to no match, we fall back to the default result of $(0.5, 0.5)$, representing the center point. As tested, over 95% of cases match successfully. Consequently, the ScreenSpot results we report for non GUI specific models (Achiam et al., 2023; Anthropic, 2024; Wang et al., 2024) in main paper Table.1 reliably reflect their GUI grounding performance.

A.2 Multimodal-Mind2Web

Discussion on metrics. Mind2Web adopt 3 metrics: (1) Element Accuracy measures whether the selected GUI element is one of the acceptable elements, reflecting the grounding accuracy. (2) Operation F1 calculates token-level F1 score for the predicted action. For those action type without value (such as 'CLICK'), this is the same as action type accuracy. For those action type with value (such as 'TYPE' needs a content), this is calculated between the predicted value and GT value. (3) Step Success Rate is measured among all steps, a step is regarded as successful only if both the selected element and the predicted operation are correct.

In our agent setting, action type is fully determined by the instruction interpreter. As shown in the first, the third, the forth, and the sixth line of appendix Table 8. However, due to the highly biased action type distribution in Mind2Web, simply setting every operation as 'CLICK' yields a high F1 scores across all splits (over 80%) as shown in the second and fifth line of appendix Table 8. This brute force manner results in failure of all 'non-CLICK' steps, harming the step success rate. This reflects that simply comparing the Op.F1 metric is not enough to determine the action predicting ability. We claim that the gold metric should always be the Step Accuracy, which relies on the correctness of both element selection and action prediction.

A.3 OmniACT

Calculation details of Action Score. According to the formula presented in OmniACT (Kapoor et al., 2024) and shown below, when $\text{SeqScore}_i = 0$, the i-th case does not contribute to the final result of the Action Score. In other words, the Action Score should only be calculated based on matched sequences. Moreover, the sum of (Action Score + Click Penalty + Key Penalty + Write Penalty) is always expected to equal 100%, and mismatched sequences should not introduce penalties.

$$\textbf{Action Score} = \frac{\sum_i \max\left(\text{SeqScore}_i - \sum_j \left(M_i^j + K_i^j + W_i^j\right), 0\right)}{\sum_i \text{SeqScore}_i}$$

However, in the evaluation script provided by OmniACT, mismatched sequences still incur penalties. Specifically, the condition $\text{SeqScore}_i \geq 1$ is not properly enforced in the code. As a result, (Action Score + Click Penalty + Key Penalty + Write Penalty) equals SeqScore instead of 100%, as reflected in the experimental results table of OmniACT (Kapoor et al., 2024). This implementation error leads to a misreported Action Score that fails to accurately represent "how well a code snippet performs the correct action." The reported values are directly influenced by the SeqScore due to this mistake.

To address this issue, we report all results based on the correct formula, ensuring consistency with the intended evaluation framework.

A.4 OSWorld

When building Ponder& Press agent on OSWorld (Xie et al., 2024) benchmark, we firstly prompt the interpreter model to explicitly generate 1.action type, 2.action value, and 3.GUI element description. Then we utilize the locator model to convert GUI element description into pixel-level coordinates. With the action triplet (action type, action value, coordinates), we finally format the 'pyautogui' code required by OSWorld in a rule-based manner.

A.5 AndroidWorld

When building Ponder& Press agent on AndroidWorld benchmark, we refer to the prompts of M3A agent proposed in the AndroidWorld paper (Rawles et al., 2024a). The main difference lies in we solely utilize raw screenshot as input, neither utilizing set-of-mark labels (Yang et al., 2023b) nor utilizing additional a11y tree input. We use the output coordinate of our grounding model to decide action placement location on the screen.

A.6 Model Endpoints

We utilize api-based MLLM as the instruction interpreter. To better ensure consistent behaviors, we listed the model endpoint names as follows:

- GPT-4o: 'gpt-4o-2024-08-06'
- Claude: 'aws_claude35_sdk_sonnet'

B More examples

To better showcase the pipeline of Ponder&Press, we provide more examples at appendix Figure 6, Figure 7, Figure 8, and Figure 9.

Table 8: **Comparisons with pure-vision methods on web agent benchmark *Multimodal-Mind2Web* (Deng et al., 2024).** Claude denote Claude 3.5 Sonnet (Anthropic, 2024), Ele.Acc denote element accuracy, Op.F1 denote operation F1, Step SR denote step success rate. **Always conducting 'CLICK' action may result in higher operation F1, but harms step success rate.**

Visual Locator	Ele. Desc. Interpreter	Action Interpreter	Cross-Task			Cross-Website			Cross-Domain		
			Ele.Acc	Op.F1	Step SR	Ele.Acc	Op.F1	Step SR	Ele.Acc	Op.F1	Step SR
SeeClick	GPT-4o	GPT-4o	31.7%	72.6%	28.3%	33.0%	72.3%	27.2%	34.3%	71.6%	30.6%
Ponder&Press	GPT-4o	Always CLICK	**42.8%**	**83.5%**	32.7%	**43.8%**	**80.8%**	32.2%	**45.3%**	**83.8%**	35.5%
Ponder&Press	GPT-4o	GPT-4o	**42.8%**	72.6%	**37.0%**	**43.8%**	72.3%	**36.8%**	**45.3%**	71.6%	**39.4%**
SeeClick	Claude	Claude	34.9%	79.2%	30.2%	32.9%	76.2%	26.5%	36.1%	79.0%	31.4%
Ponder&Press	Claude	Always CLICK	**46.7%**	**83.5%**	35.7%	**44.1%**	**80.8%**	30.7%	**47.0%**	**83.8%**	35.1%
Ponder&Press	Claude	Claude	**46.7%**	79.2%	**41.0%**	**44.1%**	76.2%	**36.2%**	**47.0%**	79.0%	**40.4%**

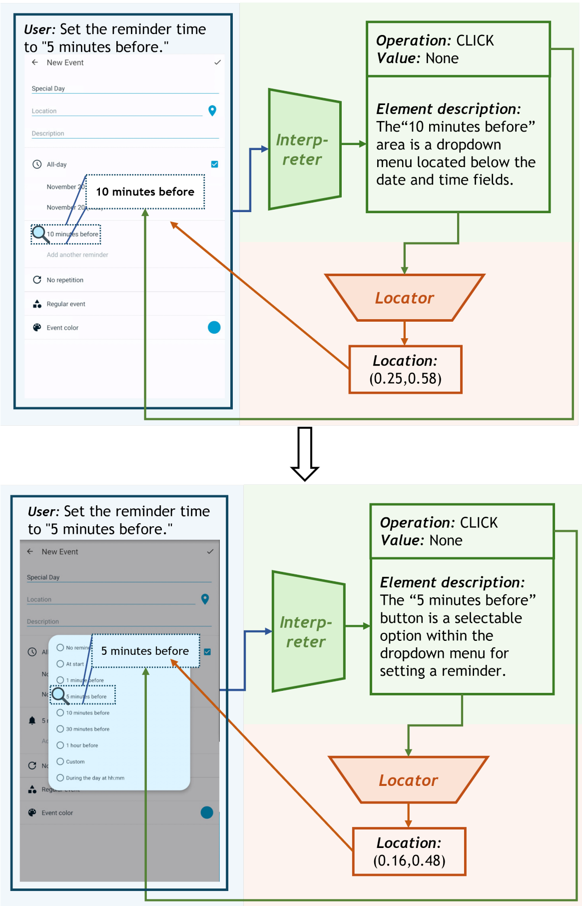

Figure 6: Example of mobile GUI task.

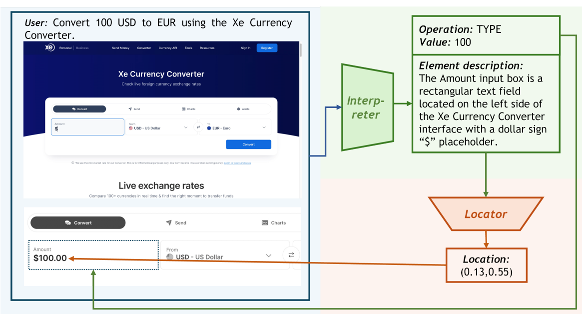

Figure 7: Example of webpage GUI task.

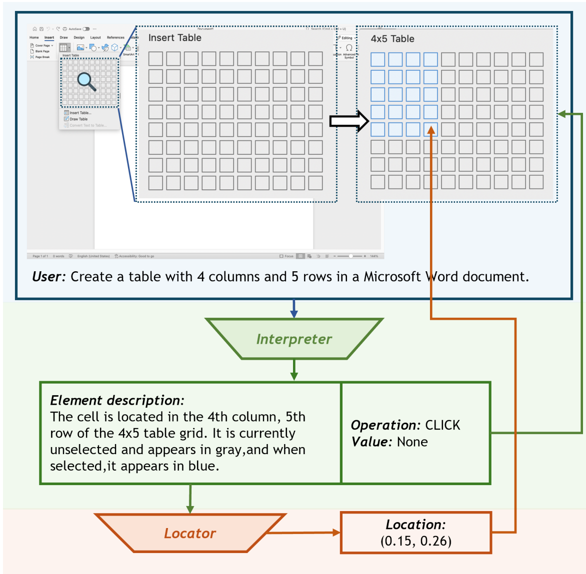

Figure 8: Example of office GUI task.

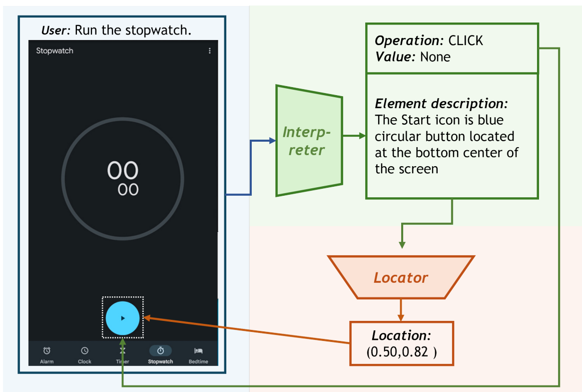

Figure 9: Example of mobile GUI task.