CAAP: CONTEXT-AWARE ACTION PLANNING PROMPTING TO SOLVE COMPUTER TASKS WITH FRONT-END UI ONLY

Anonymous authors

Paper under double-blind review

ABSTRACT

Software robots have long been used in Robotic Process Automation (RPA) to automate mundane and repetitive computer tasks. With the advent of Large Language Models (LLMs) and their advanced reasoning capabilities, these agents are now able to handle more complex or previously unseen tasks. However, LLM-based automation techniques in recent literature frequently rely on HTML source code for input or application-specific API calls for actions, limiting their applicability to specific environments. We propose an LLM-based agent that mimics human behavior in solving computer tasks. It perceives its environment solely through screenshot images, which are then converted into text for an LLM to process. By leveraging the reasoning capability of the LLM, we eliminate the need for large-scale human demonstration data typically required for model training. The agent only executes keyboard and mouse operations on Graphical User Interface (GUI), removing the need for pre-provided APIs to function. To further enhance the agent's performance in this setting, we propose a novel prompting strategy called Context-Aware Action Planning (CAAP) prompting, which enables the agent to thoroughly examine the task context from multiple perspectives. Our agent achieves an average success rate of 94.5% on MiniWoB++ and an average task score of 62.3 on WebShop, outperforming all previous studies of agents that rely solely on screen images. This method demonstrates potential for broader applications, particularly for tasks requiring coordination across multiple applications on desktops or smartphones, marking a significant advancement in the field of automation agents. Codes and models are accessible at https://github.com/caap-agent/caap-agent.

029 030 031

027

000

001

002 003 004

005

006 007 008

010 011

012

013

014

015

016

017

018

019

021

023

024

1 INTRODUCTION

033 Artificial intelligence (AI) agents are often commercialized as Robotic Process Automation (RPA) tools, designed to streamline business processes with minimal human intervention. Traditional RPA agents use 035 rule-based algorithms to automate structured tasks, making them particularly effective for repetitive and standardized data processing. However, most desktop tasks are not well-suited to this rule-based approach, 036 as they frequently involve handling unexpected situations and exceptions that are too numerous and varied 037 to anticipate and program for in advance. In contrast, agents utilizing deep neural networks can adapt more flexibly to unforeseen situations. Studies have shown that agents trained on extensive data of expert demonstrations can effectively handle unseen tasks by imitating the decision-making styles of human 040 experts (Humphreys et al., 2022; Shaw et al., 2023). The emergence of Large Language Models (LLMs) 041 is now further advancing the capabilities of RPA agents. LLMs have demonstrated the potential to support complex decision-making processes with their advanced reasoning capabilities. Their In-Context Learning 043 (ICL) ability allows them to learn meaningful actions for specific tasks with just a few examples, significantly reducing the need for large-scale data collection of expert demonstrations. As a result, research on agents leveraging ICL has been vigorously pursued in recent years (Yao et al., 2022b; Sridhar et al., 2023; Zheng et al., 2023; Shinn et al., 2024; Lin et al., 2023; Huang et al., 2022; Liu et al., 2024). As AI chips are

increasingly integrated into computers and smartphones (Intel, March 26, 2024; Samsung, January 18, 2024), and as these AI devices are poised to become the standards in the market, this new trend in the design of AI-driven RPA tools is likely to have a profound impact.

Most prior research on agents for solving computer tasks has primarily relied on HTML or the Document
Object Model (DOM) as input sources, or has leveraged well-structured Application Programming Interfaces
(APIs) to interact with digital environments (Humphreys et al., 2022; Liu et al., 2018; Gur et al., 2022; Kim
et al., 2023; Sun et al., 2023; Furuta et al., 2023). However, the effectiveness of these methods depends on the
availability of such shortcuts. They lack general applicability to broader desktop tasks that are not web-based
or that require the use of applications with no provided APIs. Often, many applications and web services in
the real world do not maintain rigorous development standards.

A promising approach to ensure the applicability of agents for more general computer tasks is to rely solely on visual information from the screen as input and use keyboard and mouse actions as output (Shaw et al., 2023; Cheng et al., 2024). By leveraging well-established GUI environments, such agents can be readily deployed to assist humans in performing their computer tasks. Prior research in this direction has primarily focused on using Vision-Transformers, which are a natural choice for processing screenshot images. However, even state-of-the-art Vision-Language Models (VLMs) struggle to ground actions to specific pixel coordinates (Anthropic; Zheng et al., 2024). Without a robust and reliable mechanism for processing the spatial information of GUI elements, achieving a practical agent remains unattainable.

065 In this paper, we propose a novel agent architecture composed of a separate visual observation module and a 066 reasoning model (LLM), which strongly reinforces the agent's coordinate-grounding functionality. The visual 067 observation module interprets the environment and represents it in textual form, alleviating the burden on the 068 VLM to simultaneously process visual information and perform task-specific, reasoning-based planning. This 069 modular approach also greatly improves the agent's adaptability a cross various applications, as each module 070 can be trained or upgraded independently. To further support and maximize the effectiveness of our agent design, we have developed a novel prompting technique called Context-Aware Action Planning (CAAP). This 071 technique systematically organizes the contextual information required for action-proposal and identifies 072 syntactic structures that trigger the most effective Chain of Thought (CoT) (Wei et al., 2022). Our agent 073 demonstrates superior task-solving performance on the MiniWoB++ and WebShop benchmark. It surpasses 074 previous studies by utilizing only a minimal amount of human demonstration data, thereby illustrating its 075 efficiency and effectiveness in handling complex computer tasks with limited training cost. 076

077 Our contributions are as follows:

We present the first LLM-based agent for computer tasks that leverages ICL while emulating human behavior by using front-end User Interface (UI) channels as input-output sources. It processes input through an image-to-text model followed by an LLM, a combination that eliminates the need for extensive human labor in dataset collection before deploying the agent for new tasks. The agent operates exclusively through keyboard and mouse actions, enabling deployment on any GUI-based system without the need for special APIs to be provided.

We introduce an effective prompting technique named CAAP, which enhances the ICL capabilities of an LLM-based agent in managing complex desktop tasks. By systematically structuring contextual information and leveraging syntactic patterns that trigger optimal CoT reasoning, CAAP significantly outperforms other agent designs.

088 089

090 091 2 RELATED WORK

Agents with image inputs Humphreys et al. (2022) introduced CC-Net, a compound model of ResNet,
 Transformer, and LSTM that uses image and DOM data as inputs and produces policies for keyboard-mouse

094 actions. It was trained via SL and RL, covering the largest number of MiniWoB++ tasks to date (104 tasks). 095 Furuta et al. (2023) proposed WebGUM, which combines Vision Transformer (ViT) and T5 transformers. It 096 was trained using SL to solve both MiniWoB++ and WebShop tasks, utilizing both image and HTML inputs. 097 Shaw et al. (2023) introduced Pix2Act, the first agent to completely eliminate reliance on DOM or HTML for 098 solving MiniWoB++ and WebShop tasks. The Pix2Act model is built upon the Pix2Struct architecture (Lee 099 et al., 2023) and uses only image inputs. While these studies provide valuable insights and serve as important benchmarks, their methods are impractical for real-world applications as they require around a million or 100 more human demonstration data samples, demanding thousands of hours of human labor for training. Cheng 101 et al. (2024) introduced SeeClick, which is aided by the strong reasoning capabilities of the VLM. SeeClick 102 required a much smaller dataset of 2.8K human demonstrations to train for MiniWoB++ tasks. It fell short in 103 overall performance, however, achieving less than a 70% success rate on the 55 tasks it covered. A common 104 feature of all these approaches is the integration of image interpretation and action planning models within 105 a single architecture. This rigid design poses a significant limitation: when new UI elements need to be 106 recognized or additional actions supported for a new task, the entire architecture must be retrained. 107

108 **LLM-based agents with ICL** To reduce dependence on large quantities of human demonstration data, 109 several studies have explored the use of LLMs for action planning. Yao et al. (2022b) developed the ReAct 110 model, which evaluates the current state before determining subsequent actions. Kim et al. (2023) introduced 111 the Recursive Criticism and Improvement (RCI) agent, which enhances action planning through iterative 112 self-critique, highlighting the importance of advanced reasoning techniques in agents. Huang et al. (2022) proposed Inner Monologue (IM), a mechanism that integrates real-time environmental states into action 113 planning via an explicit feedback loop. Inspired by IM, Sun et al. (2023) introduced AdaPlanner, which 114 prompts LLMs to generate action plans as Python functions and refine them based on feedback from execution 115 failures. Despite their innovations, all these approaches share a common limitation: their architectures are 116 designed to operate solely with HTML inputs, restricting their applicability to constrained, well-curated web 117 environments. 118

3 MODULARIZED GUI AGENT FOR COMPUTER TASKS

Our agent is composed of three distinct modules: the visual observer, which processes and interprets image 122 inputs; the action proposer, which suggests subsequent actions; and the action executor, which carries out 123 the proposed actions. The agent operates by iteratively repeating a cycle in which these modules execute 124 sequentially. Each module is further divided into sub-modules based on its functionality, as illustrated in 125 Figure 1. This modular architecture provides substantial benefits over monolithic agent designs, such as 126 those utilizing VLMs, particularly in flexibility, resource efficiency, and scalability. Modular designs enable 127 targeted updates to specific components, ensuring more robust and stable improvements without impacting the 128 entire system. Additionally, they reduce computational complexity during training. When needed, functional 129 sub-modules can be seamlessly integrated alongside existing ones to further enhance system performance. 130

131 3.1 VISUAL OBSERVER¹

119

120 121

The visual observer module employs a two-step approach to transform visual information from a screen into the linguistic domain. Initially, in the detection stage, UI elements are identified, and their spatial locations are extracted. Subsequently, these elements undergo further analysis through advanced object understanding techniques, during which detailed attributes of each element are captured. This processing results in the information being organized into structured text, ready to be handed to the action proposer as the representation of the current state.

¹While VLMs may eventually advance enough to replace the visual observer in the future, it would still serve as
 a valuable complementary tool for interpreting screenshots. For instance, the action proposer's LLM model could be



188 **UI element understanding** We use the Pix2Struct model (Lee et al., 2023) to extract detailed attributes 189 from all UI elements identified during the detection stage. We fine-tune the original Pix2Struct model to 190 match the specific output format required for our task. During this fine-tuning process, we employ a different 191 masking technique from Pix2Struct style, as depicted in Figure 2 (a). Our masking technique highlights the 192 target UI element by outlining its region and darkening the surrounding image areas, as demonstrated in 193 Figure 2 (b). This method preserves the pixel-level details of the targeted element while helping the model learn the spatial relationships between the element and its adjacent elements. The structured output from the 194 visual observer is presented as shown in Figure 2 (c), with the range of extractable attributes for each element 195 detailed in Table 4 of Appendix A. 196

197 198

3.2 ACTION PROPOSER

199 Upon receiving the representation of the current screen, the action proposer is employed to determine possible 200 actions for completing the task. The proposal process is facilitated by its two key components: a context 201 collector and a CAAP prompter. The context collector gathers vital contextual information that aids in task 202 resolution. This includes exemplary demonstrations, the ultimate task goal, the history of actions performed 203 so far, and the range of possible actions that the language model can choose from, all formatted as text. The 204 CAAP prompter then uses this information to construct prompts that are sent to the language model, eliciting 205 responses that guide the next steps. The CAAP prompting method, one of our major contributions in this 206 paper, is discussed in detail in Section 4.

207 208

209

3.3 ACTION EXECUTOR

The actions determined by the action proposer are implemented through a low-level interface to effect changes in the environment by the last module, action executor. It first identifies a series of recommended actions by parsing the response text from the action proposer. Then, it executes these actions using the keyboard and mouse interfaces of the operating system. After each action, it receives an updated screenshot, halting further actions if a change in the environment is detected. The types of actions that can be executed include mouse actions such as clicking, pointing, and scrolling, as well as keyboard actions like typing and using common shortcuts. These are enumerated in Table 5 of Appendix B and implemented as Python functions using the PyAutoGUI package.

217 218 219

220

4 CAAP PROMPTING

The performance of LLM-based agents is significantly influenced by the structure of their prompts. Our prompting approach is specifically designed to enable the LLM to effectively utilize surrounding context information essential for decision-making. This approach aims to elicit a CoT reasoning process. The strength of the CoT mechanism lies in its ability to connect disparate pieces of context information, progressively reinforcing their cohesion and enriching the reasoning process. CAAP prompting is specifically developed to enable the effective utilization of this mechanism during the action proposal phase. To achieve this, it actively incorporates both explicit and implicit strategies to guide the LLM's CoT reasoning. The design of CAAP prompting is systematically structured as illustrated in Figure 3.

228 229

230

4.1 EXPLICIT AND IMPLICIT COT INDUCEMENT

Explicit inducement of CoT Inspired by human action planning, we identified four essential types of
 surrounding context information necessary for determining subsequent actions. These contextual elements
 are conveyed to the LLM along with CoT-inducing instructions, which explicitly guide the model to establish
 semantic connections between them. Using CoT-inducing instructions, the LLM examines the actions taken so

235	Г		
236		Human demonstrations	
237		• Exemplars of action trajectory with rationale	
238		Surrounding context information	
239		Task description	
240		Action trajectory	
240		• Visual observation description	
241		• Candidates of action types	
242		CoT-inducing instructions	
243		• Instruction for reviewing action trajectory	
244		Instruction for reviewing visual observation	
240		Instruction for deciding next actions	
246			
247		Extra guidelines	
248	_		
249		Figure 3: Content design for the CAAP prompting.	
250			
251			
252	far to uncover the relationship	ip between the task objective and the action trajectory.	It then infers the contextual
253	meaning of these actions a	and the updated state of the visual environment. Sub	sequently, referencing the
254	candidates of action types, i	it refines the subsequent action plan considering the cu	irrent situation and outputs
255	actionable guidance in a str	uctured text format.	
256			
257	Implicit inducement of Co	T In addition to CoT-inducing instructions, we enh	ance the implicit induction
258	of chain-of-thought (CoT)	reasoning in large language models (LLMs) through l	numan demonstrations. By
259	providing exemplars of few-	-shot task-solving action trajectories from experts, LLI	Vis can be guided to reason
260	step-by-step Wei et al. (202.	2). However, an over-reliance on demonstrations can lo	ead LLMs to rigidly mimic
261	the examples, making them	tiess capable of adapting to exceptional scenarios. It	A to better comprehend the
262	contextual information relev	want to the action. These rationales contain detailed na	tural language descriptions
263	of the actions and the associ	ated circumstances. The methodology for generating the	ese rationales using LLMs
264	is described in the 'human of	demo dataset' section of Appendix C	lese fationales using LEWIS
265			
266	The prompts both explicitly	and implicitly inducing CoT reasoning and the subseq	uent inference processes of
267	the LLM can be found in A	ppendix D, specifically in Figures 6 and 7, respectively	ly.
268			
269	4.2 CAAP COMPONENT	S	
270	Human demonstrations	Providing few-shot exemplars within the promot	s a common practice that
271	enhances the advanced race	roviding tew-shot exemptats within the prompt has	s a common practice that th few-shot human demon
272	strations which are sequen	ces of actions with rationales for solving tasks similar	to the given objective
273	strations, which are sequent	ces of decions with futionales for solving tasks similar	to the given objective.
274	Surrounding context infor	mation This part contains all the essential information	n for the language model to
275	propose the next actions. It i	includes a description of the target task, a record of pas	t actions, state descriptions
276	from the visual observer, an	ad the available range of actions for the agent.	accomptions
277			
278	CoT-inducing instructions	The section on CoT-inducing instructions contains	phrases that encourage the
279	LLM to link the surroundir	ng context information, thereby inducing CoT. Instru	ction for reviewing action
280	trajectory encourages the ag	gent to review its actions taken so far. Instruction for re	viewing visual observation
281	prompts the agent to examine	ne the current state's observations. Instruction for impl	roving action plan requests

the formulation of a subsequent action plan, taking into account the self-generated text thus far. Finally, the
 Instruction for deciding next actions provides guidance on the desired response format and requests a list of
 actions to be taken in the subsequent steps.

Extra guidelines The final section of the CAAP prompt primarily features phrases designed to safeguard the LLM's output. It incorporates additional directives specifically tailored to reduce common errors made by the particular LLM in use.

- 5 EXPERIMENTS
- 292 293 5.1 MINIWOB++

286

287

288

289 290

291

MiniWoB++ is a highly suitable benchmark for evaluating an agent's ability to handle a wide range of scenarios encountered during computer task execution, as it encompasses over a hundred of fundamental tasks (Liu et al., 2018; Shi et al., 2017; Min). As an extension of OpenAI MiniWoB, this benchmark includes a diverse spectrum of tasks, ranging from simple activities such as button clicks and form filling to more complex tasks like email forwarding and flight booking. We utilize the MiniWoB++ benchmark to thoroughly assess our agent's effectiveness in navigating and solving diverse tasks within a simulated computer environment.

Previous studies using the MiniWoB++ benchmark often choose subsets of tasks most suitable for demonstrating their methods. We have also selected a total of 73 MiniWoB++ tasks that meet the following criteria: (1) tasks that consist of UI elements of types recognizable by our visual observer module, as listed in Table 4 of Appendix A; (2) tasks where none of UI elements extend beyond the default screen area of 160x210 pixels; and (3) tasks that do not require color recognition, as our visual observer module is not yet trained for this capability.

For each selected task, we run at least 50 repetitions with different seeds to estimate the Success Rate (SR)
 for that task. Across all task types, we consistently use the identical CAAP prompt template without making
 any manual adjustments for specific tasks, unlike some other works (Kim et al., 2023).

Models and datasets YOLO and Pix2Struct models within the visual observer have been fine-tuned
 to enable accurate recognition of images presented in MiniWoB++ environment. The image dataset for
 training has been collected through a process of capturing screenshots and annotating the UI elements within.
 This dataset consists of 1.76K screenshot images, gathered and annotated over a span of 10 hours by two
 individuals.

For human expert demonstrations, we generate between 0 and 5 demonstrations per task, tailored to the task's complexity, amassing a total of 97 demonstrations for our experiments. We record the trajectory of each action performed by the demonstrators, alongside screenshots taken at the time of each action. These screenshots are subsequently analyzed to extract UI elements, enabling the identification of the UI component interacted with during mouse-related actions, such as clicks. This demonstration data have been collected by a single individual over a span of approximately two hours.

For the action-proposal language model, we utilize the Azure OpenAI gpt-4-0125 model. The Azure OpenAI API features the 'function calling' capability, thus our prompt does not directly include the list of action types (as depicted in Figure 3). Instead, this contextual information is conveyed to the language model through arguments in the function calling feature. An example of how the function calling is applied is illustrated in Figure 8 in Appendix E.

326

Experiment result Table 1 compares the performance of our approach with other studies reported on the MiniWoB++ benchmark. Since each study reports results over different sets of tasks (See Figure 9 of

Method	Modality	Image datasize	Demo datasize	Reported SR	Reported tasks
Human	Image	-	-	93.5%	104
CC-Net (no DOM)	Image	0	2.4M	24.0%	104
Pix2Act	Image	0	1.3M	96.2%	59
SeeClick	Image	0	2.8K	69.4%	55
CAAP	Image	1.8K	0.1K	94.5%	73

Table 1: Comparison with other image-only methods for solving MiniWoB++ problems.

Table 2: Comparison with other image-only methods for solving WebShop problems. The LLM model used for the action proposer in the CAAP method is specified in parentheses.

Method	Modality	Image datasize	Demo datasize	Task score
Human	Image	-	-	82.1
Pix2Act	Image	0	1K	46.7
CAAP (OpenAI gpt-4-0125)	Image	0.3K	1	62.3
CAAP (Claude 3.5 sonnet)	Image	0.3K	1	70.01^{3}

Appendix F.), we have included both the SRs reported in the literature and the number of tasks used in each 349 case. First, CC-Net, which covers the largest set of tasks, was trained on 2.4M human demonstration data 350 across 104 tasks. However, when using only image input (excluding DOM), its performance dropped to 24%, 351 indirectly highlighting the challenge of solving computer tasks using image inputs alone. Pix2Act, trained 352 on 1.3M human demonstrations across 59 tasks, achieved an SR of 96.2%, demonstrating the feasibility of 353 agents solving computer tasks with image input only. SeeClick, despite using a relatively small dataset of 354 2.8K demonstrations to learn click locations on screenshots, achieved an SR of only 69.4% across 55 tasks, showing limited effectiveness. In contrast, our approach, using only 97 demonstration samples, achieved a 356 high task SR of 94.5% across 73 tasks. Although an additional 1.76K samples² were required to train the 357 model for image input interpretation due to the modular nature of our approach, our method still demonstrated 358 its effectiveness on a broader range of computer tasks with about 1.86K samples in total. Task-level SRs are detailed in Table 7 of Appendix G. 359

360 Of the 73 tasks included in our experiments, six tasks—drag-circle, drag-single-shape, find-greatest, generate-361 number, odd-or-even, and sign-agreement— were excluded during the generation of training data for the 362 visual observer, making them unseen tasks for our agent. Despite this, the agent successfully completed 363 these tasks with an average success rate of 95.3%. These results underscore the benefits of our LLM-based approach compared to the prior end-to-end learning method that relied on imitation learning from human 364 demonstrations. As long as the new tasks do not introduce completely unfamiliar types of UI elements that 365 the visual observer cannot recognize, our agent can effectively manage these unseen tasks by utilizing the 366 knowledge and reasoning capabilities of the LLM. 367

369 5.2 WEBSHOP

WebShop (Yao et al., 2022a) is a simulated e-commerce web environment well-suited for assessing an agent's ability to handle real-world scenarios. To successfully complete WebShop tasks, the agent must understand textual instructions and requirements for the target product and navigate various types of web pages to locate

374

368

329

339

²These image samples were semi-automatically annotated using a custom tool shown in Figure 10 of Appendix H. ³This score is measured on 100 out of 500 test instructions, and the final score is subject to further updates.



Figure 4: The effects of different CAAP components on MiniWoB++ task performance.

389 390

414

376 377

378

379 380

381 382

383

384 385

386

387 388

it, out of a pool of 1.18 million real products scraped from amazon.com. The environment provides a total
 of 12,087 instructions, divided into 10,587 for training, 1,000 for development, and 500 for testing. Among
 them, we utilized only the 500 test instructions for evaluation purposes.

395 Models and datasets The web browser for the WebShop environment can be resized freely, and for convenience, we used a fixed resolution of 1276 × 1153 pixels for both data collection and evaluation. Unlike 396 MiniWoB++, the dataset used to train the visual observer for WebShop was generated entirely through an 397 automated process. Given an instruction, a simple helper agent we created extracted a search term, performed 398 a search, and clicked the first item in the results. Screen images were captured throughout this process, and 399 annotation data was collected using Selenium for each image, resulting in a total of 300 annotated samples. 400 The action-proposal model, supported action types, and function-calling approach are entirely consistent 401 with those employed in the MiniWoB++ experiments. For the evaluation of the CAAP agent, a single human 402 demonstration is applied universally to all instructions. 403

404 **Experiment result** There has been limited research on constructing agents that rely solely on image inputs, 405 and Pix2Act is the only method whose performance has been validated on the WebShop benchmark. Table 2 406 presents a comparison between our approach and Pix2Act on WebShop. (Comparisons with other agent-based 407 methods that rely on additional HTML input, on top of image input, are presented in Appendix I.2.) While 408 Pix2Act employed 1.012 human demonstrations to train its agent model, our approach relied on just a single human demonstration, along with 300 samples collected through a fully automated process for training the 409 visual observer module. Despite the significantly reduced manual data collection effort, our agent achieved a 410 task score of 62.3, a meaningful improvement over Pix2Act's score of 46.7, with a margin of 15.6. Although 411 this result does not yet reach human-level performance, it demonstrates the potential of our approach to solve 412 real-world computer tasks with minimal human effort in data collection. 413

415 5.3 ABLATION STUDIES

Impact of CoT inducement on task-solving performance Figure 4 presents experimental results illustrating the impact of CoT inducement on task performance. The results show that incorporating CoT-inducing instructions leads to a significant improvement in performance. Specifically, as shown in the figure, adding CoT-inducing instructions increases task SRs by 5.3% (from 79.2% to 84.5%) in the absence of human demonstrations and by 2.5% (from 92.0% to 94.5%) when human demonstrations are included. Moreover, augmenting human demonstrations with rationales further enhances task SRs by 3.5%, raising them from 91.0% to 94.5%. Additionally, removing human demonstrations entirely results in a 10.0% drop in task

Table 3: Performance comparison of the CAAP and RCI promptings. The *RCI agent with CAAP* is configured by replacing the prompting method of the original RCI agent with the CAAP approach.

	RCI agent with CAAP	RCI agent (original)
Average SR	0.840	0.769

SRs, while removing CoT-inducing instructions in addition leads to a substantial decrease of 15.3%. These findings highlight that CAAP prompting, by explicitly and implicitly guiding the LLM to think step-by-step and establish contextual relationships before proposing actions, significantly improves task SRs in solving computer-based tasks. Detailed per-task results are presented in Table 11 of Appendix J.

435 Comparison of the CAAP and the RCI prompting Since our main experiments combine GUI-based 436 perception and execution with the CAAP prompting technique, we conducted additional experiments to isolate 437 the effect of CAAP prompting as part of an ablation study. Without modifying the input/output channels, we 438 tested how the performance of the RCI agent (Kim et al., 2023) improves when its prompting mechanism is 439 replaced by our CAAP prompting. We first executed the original RCI code from the authors' official GitHub repository without modifications, testing only the 47 tasks listed in the repository by 50 times and taking 440 the average.⁴ Then, we replaced the prompting method with CAAP and reran the tests. To ensure a fair 441 comparison, we used identical seeds for task instances, the same demo set provided by the repository, and the 442 same LLM for queries. 443

444 As described in Table 3, the results demonstrate that applying CAAP prompting to the RCI agent significantly increased the average task-solving SR by 7.1%, from 76.9% to 84.0%. (Task-level results can be seen in 445 Table 12 of Appendix K.) The prompting methods of RCI and CAAP exhibit two key differences. First, the 446 RCI approach tends to modify even correct action plans during the iterative critique and revision process. In 447 contrast, the CAAP method, by focusing on thorough context review rather than critique, is able to avoid 448 this issue when formulating action plans. Second, RCI executes actions sequentially based on a fixed initial 449 plan, failing to adapt to changes in the environment that occur during execution. Conversely, the CAAP 450 method is designed to account for state changes resulting from prior actions. This experiment underscores 451 the effectiveness of our CAAP prompting, confirming its potential to enhance the performance of LLMs in 452 computer tasks.

453 454 455

431

432

433

434

6 CONCLUSION

456 In this paper, we introduce a modularly designed GUI agent model that emulates the way humans solve 457 tasks on computers. Our modular design not only effectively enhances pixel coordinate detection-a critical 458 challenge faced by integrated VLM architectures-but also addresses the limitations of existing models that 459 rely heavily on imitation learning with large amounts of human demonstrations. Furthermore, we propose the 460 CAAP prompting mechanism to improve the action-proposal capabilities of LLM-based agents for computer 461 tasks by guiding the LLM to fully utilize the CoT reasoning mechanism. Evaluations on the MiniWoB++ 462 and WebShop benchmarks demonstrate that our agent model achieves superior performance compared to 463 the previous methods while requiring significantly less data. By adopting a modular approach that separates 464 visual perception, action proposal, and action execution, our method allows for easy adaptation to new tasks or model upgrades. Additionally, our agent is designed to operate beyond web environments and across 465 inter-application settings, offering broader applicability and significant advantages in terms of generality. 466

 ⁴⁶⁷ ⁴The success rate of the RCI agent we observed deviates from the results reported in the literature, despite our diligent
 ⁴⁶⁸ efforts to replicate the experimental setup exactly. Upon inquiry, the RCI authors indicated that they no longer had access
 ⁴⁶⁹ to the complete experimental logs used in their publication.

470 REFERENCES

479

491

- 472 Miniwob++ documentation. https://miniwob.farama.org (Last accessed on 25 Nov 2024).
- Ultralytics github. https://github.com/ultralytics/ultralytics (Last accessed on 25 Nov 2024).
- Anthropic. Understand computer use limitations. https://docs.anthropic.com/en/docs/
 build-with-claude/computer-use (Last accessed on 25 Nov 2024).
- Kanzhi Cheng, Qiushi Sun, Yougang Chu, Fangzhi Xu, Yantao Li, Jianbing Zhang, and Zhiyong Wu. Seeclick: Harnessing gui grounding for advanced visual gui agents. *arXiv preprint arXiv:2401.10935*, 2024.
- Hiroki Furuta, Ofir Nachum, Kuang-Huei Lee, Yutaka Matsuo, Shixiang Shane Gu, and Izzeddin Gur. Multimodal web navigation with instruction-finetuned foundation models. *arXiv preprint arXiv:2305.11854*, 2023.
- Izzeddin Gur, Ofir Nachum, Yingjie Miao, Mustafa Safdari, Austin Huang, Aakanksha Chowdhery, Sharan Narang, Noah
 Fiedel, and Aleksandra Faust. Understanding html with large language models. *arXiv preprint arXiv:2210.03945*, 2022.
- Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor
 Mordatch, Yevgen Chebotar, et al. Inner monologue: Embodied reasoning through planning with language models.
 arXiv preprint arXiv:2207.05608, 2022.
- Peter C Humphreys, David Raposo, Tobias Pohlen, Gregory Thornton, Rachita Chhaparia, Alistair Muldal, Josh Abramson,
 Petko Georgiev, Adam Santoro, and Timothy Lillicrap. A data-driven approach for learning to control computers. In
 International Conference on Machine Learning, pages 9466–9482. PMLR, 2022.
- Intel. Intel announces new program for ai pc software developers and hardware vendors. Intel newsroom, March 26, 2024. https://www.intel.com/content/www/us/en/newsroom/news/
 intel-announces-new-program-for-ai-pc.html (Last accessed on 25 Nov 2024).
- Geunwoo Kim, Pierre Baldi, and Stephen McAleer. Language models can solve computer tasks. *arXiv preprint arXiv:2303.17491*, 2023.
- Kenton Lee, Mandar Joshi, Iulia Raluca Turc, Hexiang Hu, Fangyu Liu, Julian Martin Eisenschlos, Urvashi Khandelwal, Peter Shaw, Ming-Wei Chang, and Kristina Toutanova. Pix2struct: Screenshot parsing as pretraining for visual language understanding. In *International Conference on Machine Learning*, pages 18893–18912. PMLR, 2023.
- Kevin Lin, Christopher Agia, Toki Migimatsu, Marco Pavone, and Jeannette Bohg. Text2motion: From natural language instructions to feasible plans. *Autonomous Robots*, 47(8):1345–1365, 2023.
- Evan Zheran Liu, Kelvin Guu, Panupong Pasupat, Tianlin Shi, and Percy Liang. Reinforcement learning on web interfaces
 using workflow-guided exploration. *arXiv preprint arXiv:1802.08802*, 2018.
- Na Liu, Liangyu Chen, Xiaoyu Tian, Wei Zou, Kaijiang Chen, and Ming Cui. From llm to conversational agent: A memory enhanced architecture with fine-tuning of large language models. *arXiv preprint arXiv:2401.02777*, 2024.
- Samsung. era of mobile ai with samsung galaxy s24 series. Enter the new Sam-2024. https://news.samsung.com/global/ sung newsroom, January 18, 509 enter-the-new-era-of-mobile-ai-with-samsung-galaxy-s24-series (Last accessed on 510 25 Nov 2024). 511
- Peter Shaw, Mandar Joshi, James Cohan, Jonathan Berant, Panupong Pasupat, Hexiang Hu, Urvashi Khandelwal, Kenton
 Lee, and Kristina Toutanova. From pixels to ui actions: Learning to follow instructions via graphical user interfaces.
 arXiv preprint arXiv:2306.00245, 2023.
- Tianlin Shi, Andrej Karpathy, Linxi Fan, Jonathan Hernandez, and Percy Liang. World of bits: An open-domain platform
 for web-based agents. In *International Conference on Machine Learning*, pages 3135–3144. PMLR, 2017.

- 517
 518
 518
 519
 519
 510
 510
 511
 512
 513
 514
 514
 515
 515
 516
 517
 517
 518
 519
 518
 519
 518
 519
 519
 519
 510
 510
 510
 511
 512
 512
 513
 514
 514
 515
 516
 517
 517
 518
 518
 518
 519
 518
 519
 519
 519
 510
 510
 510
 511
 512
 512
 512
 514
 514
 514
 515
 514
 514
 514
 514
 515
 514
 514
 514
 514
 514
 514
 514
 514
 514
 514
 514
 514
 514
 514
 514
 514
 514
 514
 514
 514
 514
 514
 514
 514
 514
 514
 514
 514
 514
 514
 514
 514
 514
 514
 514
 514
 514
 514
 514
 514
 514
 514
 514
 514
 514
 514
 514
 514
 514
 514
 514
 514
 514
 514
 514
 514
 514
 514
 514
 514
 514
 514
 514
 514
 514
 514
 514
 514
 514
 514
 514
 514
 514
 514
 514
 514
 514
 514
 514
 514
 514
 514
 514
 514
 514
 514
 514
 514
 514
 514
 514
 514
 514
 514
 514
- Abishek Sridhar, Robert Lo, Frank F Xu, Hao Zhu, and Shuyan Zhou. Hierarchical prompting assists large language model on web navigation. *arXiv preprint arXiv:2305.14257*, 2023.
- Haotian Sun, Yuchen Zhuang, Lingkai Kong, Bo Dai, and Chao Zhang. Adaplanner: Adaptive planning from feedback
 with language models. *arXiv preprint arXiv:2305.16653*, 2023.
- Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35:24824–24837, 2022.
- Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. Webshop: Towards scalable real-world web interaction with grounded language agents. *Advances in Neural Information Processing Systems*, 35:20744–20757, 2022a.
- Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022b.
- Boyuan Zheng, Boyu Gou, Jihyung Kil, Huan Sun, and Yu Su. Gpt-4v (ision) is a generalist web agent, if grounded.
 arXiv preprint arXiv:2401.01614, 2024.
- Longtao Zheng, Rundong Wang, Xinrun Wang, and Bo An. Synapse: Trajectory-as-exemplar prompting with memory for computer control. In *NeurIPS 2023 Foundation Models for Decision Making Workshop*, 2023.

564 APPENDIX

A ELEMENT COVERAGE OF CAAP AGENT

Table 4 provides a comprehensive overview of the types of front-end elements handled by the agent, pre-dominantly including well-known types of front-end objects. Beyond the traditional categories, this study introduces the concept of 'tabled text' to describe texts commonly embedded within table structures, and presents 'draggable text' to represent text elements of which its text can be partially selected. These additions ensure extensive coverage of both interactive and static elements found in varying types of applications, thereby enhancing the understanding capability of the visual environmental states. Additionally, our agent covers conventional shapes and icons. The shape type includes circle, triangle, and rectangle, distinguished by their subtype values. Moreover, the icon type encompasses subtypes such as "back," "delete," "important," "forward," "reply," "search," and "send," as identified in MiniWob++.

Table 4: The front-end element coverage of CAAP agent.

	Туре	Subtype	Checked / focused / highlighted	Text	Coordinates
-	Text	-			
	Hyperlink		v v		
	Button				
	Radio	-			
	Checkbox				
	Dropdown	-			
	Input field		v v		
	Text area	-		-	
	Resize handle	-	-	-	
	Scrollbar		-	-	
	Tabled text				
	Draggable text	-	-		
	Shape		-	-	
	Icon	$\ $ $$		-	\checkmark
	Image	-	-	-	\checkmark

611 B ACTION COVERAGE OF CAAP AGENT

Tables 5 and Table 6 detail the action types supported by the CAAP agent and the requisite parameters, respectively. The action executor operates using atom actions based on mouse and keyboard interactions, specifically: mouse down, mouse move, mouse up, and type a key. However, when conveying candidate actions to an LLM, it is more effective to transmit them as high-level actions capable of altering the environment. Consequently, as indicated in Table 5, actions have been delineated into units such as clicking on an element, clicking an empty position, clicking on an element during pressing control key, and typing text.

Table 5: The action coverage of our agent. 'Element ID' is the ID number of the screen element to act on.
The number is designated during the element detection procedure. 'Text to Type' is which text to type on.
'Coordinates' are the X and Y coordinates of the mouse down location.

Name	Required	Description
click_element	element_id	This action moves the mouse pointer to a screen element and performs a left-click to activate that element.
click_new_point	х, у	This action moves the mouse pointer to a screen location and performs a left-click on the location.
control_click_element	element_id	This action click on a screen element while holding down the 'control' modifier key. It is used to select multiple elements.
type_text	string_to_type	This action makes keyboard typing actions to enter the text into a specific screen element, Clicking on the element is essential to give it focus, without which typing the text is impossible.
point_element	element_id	This action moves the mouse pointer to on top of an UI element without clicking it. This sometimes activates the element and reveals hidden menus or scrollbar.
press_control_A	-	This is 'Select All'. All text in the activated text field is highlighted.
press_control_C	-	This is 'Copy'. Highlighted text is copied to the clipboard.
press_control_V	-	This is 'Paste'. Text stored in the clipboard is pasted into the selected area.
drag_mouse_hold_down	х, у	This action initiates the drag action sequence by clicking and holding down the left mouse button. It is used to move objects on the screenor to highlight a block of text. This action marks the starting point of the dragging move.
drag_mouse_move	x, y	This action is the middle of the drag action sequence. It moves the mouse pointer while holding down the left mouse button. When used to move an object on the screen, the object will be dragged to the current mouse location. When used to highlight text, it will highlight up to the current mouse location.
drag_mouse_release	-	This action marks the end of the drag action sequence. It releases the left mouse button, indicating that the drag move is finished. starting point of the dragging move.

	Table 6	Parameter candidates listed in the 'required' column of Table 5.
Name	Туре	Description
1 1	.,	The id number (ranged from 1 to N) of the screen element to point at. A list of element
element_1d	integer	will be provided with the corresponding id numbers.
X	integer	The x coordinate of the click location.
у	integer	The y coordinate of the click location.
string_to_type	string	The text to type.
	·	

C MINIWOB++ EXPERIMENT DETAILS

Models for visual observer We utilize Ultralytics YOLOv8.0.143 (ult), employing the default hyper-parameters with the exception of 'box', 'cls', 'dfl', and 'max_patches' set to 0.9, 0.05, 0.05, and 512, respectively. For Pix2Struct, we use the base model provided at https://huggingface.co/google/pix2struct-base, without any modifications to the default configuration.

711

712 **Fine-tuning dataset for visual observer** Data collection for fine-tuning the visual observer involves a 713 two-step process: screenshot collection and annotation of the collected UI elements. Initially, the target task is manually performed while screenshots are taken at each step. Image comparisons are then conducted to 714 remove duplicate screenshots. From 10 episodes for each of the 73 tasks, a total of 1,768 screenshots were 715 collected. In the second stage, a GUI-based annotation tool, as depicted in Figure 10 of Appendix H, is 716 utilized. Each UI element within the screenshots is marked with bounding boxes, and specific attributes are 717 assigned. These attributes are subsequently stored in a JSON format. Data augmentation techniques are also 718 employed to enhance the dataset. These techniques include color adjustments, minor scaling of bounding box 719 sizes, and the random insertion of Gaussian noise. As a result, the total number of annotated data items is 720 tripled compared to the original dataset. 721

The YOLOv8 model is fine-tuned using the screenshots and their bounding boxes around the UI elements.
 The Pix2Struct model is fine-tuned using the masked screenshots (Figure 2(a)) and the corresponding element attributes, excluding the positions of bounding boxes.

725 **Human demo dataset** To further enrich the dataset, we augmented each action with a rationale statement. 726 As illustrated in Figure 4, the performance of the CAAP agent significantly improves when rationales for the 727 demonstrated actions are also provided. Instead of relying on demonstrators to articulate their reasoning-a 728 method that is impractical and unscalable for deploying our CAAP agent to a wider range of applications—we 729 employed an LLM (*i.e.* GPT-4) to generate these rationales in a fully automated way. For each action, a 730 prompt comprising both the visual state and the corresponding action was presented to the LLM, as detailed 731 in Figure 5. The rationales were then integrated with the action trajectories to produce comprehensive, 732 high-quality demonstrations. An example is shown in the 'Expert Demonstration' section of the prompt in Figure 6. 733

Train and test episode split To differentiate between the train datasets and the test dataset, we assign distinct seed numbers when generating individual task instances within MiniWoB++. Image annotation data was collected from tasks with seed numbers ranging from 1000 to 2999, while demonstration data was sourced from seeds ranging from 3000 to 3999. Testing was conducted using seeds ranging from 0 to 999.

739

- 740
- 741
- 742
- 743 744
- 745
- 746
- 747 748
- 749
- 750
- 751

752 D PROMPT EXAMPLES

_	
]	Fasks can be completed by applying appropriate actions in sequence.
1 () ()	Below is a record of a successful completion of the given task, demonstrated by an expert. Note: The trainee has added the "reason" part for each action in the record, but it may not accur lescribe the reasoning used by the expert who performed the task. Do not assume that the written reas correct.)
	FASK: Select EiTE and click Submit.
	Action History: action_1: {name: start, reason: "Initiating the task."} action_2: {name: click_element, arg: {type: radio, X: 31 [10-52], Y: 62 [55-70], checked: False, 'EiTE"}} action_3: {name: click_element, arg: {type: button, X: 50 [2-98], Y: 114 [99-130], text: "Submit", foc False}}
٧	We want to explain to a trainee why the action_2 was made.
1 (1	Before the action_2, the status of the computer screen was as the following: Note: Coordinates are given in the form: center_x [left_edge_x-right_edge_x], center_y [top_ed bottm_edge_y])
0 0 1 0	<pre>lemo_element_1: {type: radio, X: 31 [10-52], Y: 62 [55-70], checked: False, text: "EiTE", visible: ' lemo_element_2: {type: radio, X: 38 [10-67], Y: 80 [73-88], checked: False, text: "vAzBm9", vi frue} lemo_element_3: {type: button, X: 50 [2-98], Y: 114 [99-130], text: "Submit", focused: False, vi frue}</pre>
	After the action_2, the status of the computer screen was as the following: demo_element_1: {type: radio, X: 31 [10-52], Y: 62 [55-70], checked: True , text: "EiTE", visible: demo_element_2: {type: radio, X: 38 [10-67], Y: 80 [73-88], checked: False, text: "vAzBm9", vi Frue}
(]	lemo_element_3: {type: button, X: 50 [2-98], Y: 114 [99-130], text: "Submit", focused: False, vi Frue}
	First, explain how the action_2 (both its type and its arguments) was chosen by the expert, and why increasary. Since the trainee cannot view the screen, always provide a detailed description as specified whenever refer to a screen component in your response. Second, describe what happened after the action, as shown on the screen.
1	Answer in one paragraph.

799 Tasks can be completed by applying appropriate actions in sequence. 800 801 ### Expert Demonstrations ### 802 For example, given below are the demos showing the correct sequence of actions for each corresponding 803 task: 804 $DEMO_1 = \{$ 805 TASK: 806 Select Janella from the list and click Submit. 807 Action History: 808 demo action 1: {name: start, reason: "Initiating the task."} 809 demo action 2: {name: click element, arg: {type: dropdown, X: 76 [0-151], Y: 66 [56-76], text: "Storm"}, 810 reason: "The expert chose action_2, which is a click_element action, to interact with the dropdown menu 811 on the screen. The dropdown menu, identified as demo_element_1, was located at the center of the screen 812 with coordinates X: 76 [0-151] and Y: 66 [56-76], and had the text "Storm" visible on it. Clicking on 813 this dropdown was necessary to expand the list of options and select the desired item, "Janella," from the 814 list. The dropdown needed to be expanded because the task required selecting a specific item that was 815 not currently visible or selectable on the screen. After the action was performed, the dropdown menu 816 remained visible, but the "Submit" button (demo_element_2) became temporarily invisible, indicating that the dropdown list expanded over it. New elements appeared on the screen, which were the options 817 within the dropdown menu, including "Selie," "Janella," "Storm" (now highlighted to indicate it was the 818 previously selected option), "Gena," "Betti," and "Chrissie," with their respective coordinates and visibility 819 status. This expansion allowed the expert to proceed with selecting "Janella" from the list."} 820 demo action 3: {name: click element, arg: {type: tabled text, X: 76 [0-151], Y: 117 [109-125], text: 821 "Janella"}, reason: "The expert performed action_3, which is a click_element action, to select "Janella" 822 from the dropdown list. This action was necessary because the task required selecting a specific name, 823 "Janella," from the list of options that appeared after expanding the dropdown menu. The expert identified 824 the correct element to click by its type (tabled_text), its coordinates (X: 76 [0-151], Y: 117 [109-125]), 825 and its text content ("Janella"), which was visible on the screen among other options. After clicking on 826 "Janella," the dropdown list collapsed, making the "Submit" button (demo element 2) visible again, and 827 the dropdown menu now displayed "Janella" as the selected item. The other names in the list ("Selie,' 828 "Storm," "Gena," "Betti," and "Chrissie") were no longer visible, indicating that the dropdown had retracted and the task could proceed to the next step, which was to click the "Submit" button."} 829 demo_action_4: {name: click_element, arg: {type: button, X: 50 [1-98], Y: 96 [79-112], text: "Submit"}, 830 reason: "The expert chose action 4, which is a click element action, to interact with the "Submit" button 831 on the screen. This action was necessary to complete the task of selecting "Janella" from the dropdown list 832 and then submitting the choice. The "Submit" button was identified by its type (button), its coordinates (X: 833 50 [1-98], Y: 96 [79-112]), and its text content ("Submit"), which was visible and accessible on the screen. 834 Clicking the "Submit" button was the final step required to confirm the selection of "Janella" and proceed 835 with the task. After the action was performed, the screen status changed to an Empty Screen, indicating 836 that the task was completed successfully and the application or window that was previously open had 837 closed or moved to the next stage, leaving no elements visible on the screen."} 838 } 839 We are solving a similar task. 840 You are given the history of actions made correctly by the user so far, and current screen status which is 841 the result of those actions. 842 ### Task Description ### Select Helli from the list and click Submit. ### Action History ### action 1: {name: start} 18

#	### The UI Element List of the Current Screen That You Can Take Next Actions On ###
	Note: Coordinates are given in the form: center_x [left_edge_x-right_edge_x], center_y [top_edge_y-
t	1001111_edge_y]) ID: element 1 data: [tuna: drandown, X: 76 [1 152], X: 66 [55 76], taxt: "Theodora", focused: False)
_	ID: element 2 data: {type: https://doi.org/10.100/11.00/10.100
	15. element_2, dud. [type. button, A. 50 [1 70], 1. 90 [00 112], text. Submit , focused. Parse j
ŧ	## Instructions ###
ľ	Vhat should be the next actions(action_2, action_3,) that can be performed on the current screen?
I	f there is an action that can complete the task, perform it immediately. It's better if you can complete the
t	ask with fewer actions.
}	Your answer must be composed of the following five sections:
ł	first, explain in detail what has been done so far up to action_1 and analyze why these steps were needed.
I	Do not assume that the user could have made a mistake.
ç	secondly describe every single screen component that contains information that helps user solve the
t	ask or that needs to be interacted with Explain about the components step by step in a detailed manner
c	onsidering the given task.
١	Vhen the task deals with a list of items (e.g. finding the size of a group, identifying N-th item in order,
e	tc.), you must include the full iteration of each and every items, like this: (1)first_item, (2)second_item,
•	., (N)N-th_item.
_	
1	hirdly, describe what needs to be done to complete the task, detailing each action from start to finish.
1	nen, identify which steps can currently be performed based on the UI elements visible on the screen, and
I	f there is a demo available first describe the sequence of actions demonstrated then link these actions to
t	he steps in your plan before outlining the complete action plan
·	
Ι	Lastly, each action must be in the form of Action_2=(Action: functions.some_function_name, Argument:
{	property_name: property_val}).
F	Return the actions that need to be performed on the current screen.
]	The actions must be separated by new line characters.
1	n case there are three actions to be performed, your response will be in the following form: $($
ľ	Action_2=(Action: functions.some_function_name, Argument: {property_name: property_val})
ſ	Action 4=(Action: functions some function name, Argument: {property_name: property_val})
,	//
-	MAKE SURE that when you recommend actions that need to interact with UI elements, the UI elements
ľ	AUST BE in the current screen.
-	MAKE SURE that when satisfying the task completion conditions, click the 'Submit' button to finish.
-	MAKE SURE that even if you do not find a suitable action, return the most plausible one.
-	MAKE SURE that since texts in elements are extracted by recognizing with OCR, solve the given task
C	onsidering that some mis-typos could exist.
<u> </u>	
г1 С	guie of An example of the CAAP prompt. The rationales contained in the expert demonstrations and the
th	e semantic connectivity between the task description the action history and the visual state of the screen in
a (CoT manner.
1	

Γ	### Analysis of Action History
	So far, the only action taken has been the initiation of the task (action 1: {name: start}). This initial action
	is necessary to begin the process of selecting "Helli" from the dropdown list and submitting the choice. No
	specific interactions with the UI elements have been made yet.
	### Description of Screen Components
	1. **Dropdown Menu (element_1)**: This element is a dropdown menu currently displaying "Theodora"
	as the selected item. Its coordinates are given as X: $/6$ [1-152], Y: 66 [55-76], indicating its position on the
	screen. The dropdown is not focused, meaning it has it been interacted with yet. This element is crucial for the task as it contains the list of names from which "Helli" needs to be selected
	tor the task as it contains the list of names from which frem literas to be selected.
	2. **Submit Button (element 2)**: This is a button with the text "Submit" located at X: 50 [1-98]. Y: 96
	[80-112]. This button is essential for completing the task as it needs to be clicked after selecting "Helli"
	from the dropdown menu to confirm the selection.
i	### Completion Plan
	To complete the task, the following steps need to be taken:
	1. Click on the dropdown menu to expand it and reveal the list of names.
	2. Select "Helli" from the list of names in the dropdown.
	5. Click the Sublinit button to confirm the selection.
	Based on the current screen, the actions that can be performed are:
	- Clicking on the dropdown menu (element 1) to expand it and reveal the list of names, including "Helli".
	- After selecting "Helli", clicking on the "Submit" button (element 2) will be necessary, but this can only
	be done after "Helli" has been successfully selected.
	Given the current screen elements, the next action is to interact with the dropdown menu to expand it and
	reveal the list of names.
	### Actions to be Derformed
	### Actions to be Performed
	Action 2-(Action: functions click element Argument: Jelement id: 1))
	$\Delta c_1 c_1 - (\Delta c_1 c_1)$
F	Figure 7: An example of an LLM response to the CAAP prompt in Figure 8, showing how the LLM improves
	widence for action-proposal by connecting the surrounding context information.

940 E FUNCTION CALLING EXAMPLE

```
943
       [
944
          {
            "name": "click_element",
945
            "description": "This action moves the mouse pointer to a
946
           screen element and performs a left-click to activate that
947
           element.",
948
            "parameters": {
              "type": "object",
949
              "properties": {
950
                "element_id": {
951
                  "type": "integer",
952
                   "description": "The id number (ranged from 1 to N) of
953
           the screen element to act on. A list of elements will be
           provided with the corresponding id numbers."
954
                }
955
              },
956
              "required": ["element_id"]
957
            }
958
         },
959
         {
            "name": "type_text",
960
            "description": "This action makes keyboard typing actions to
961
           enter the text, for example, into the 'input_field'-type
962
           screen element. Before typing the text, the element on which
963
           enter the text must be focused.",
            "parameters": {
964
              "type": "object",
965
              "properties": {
966
                "string_to_type": {
967
                  "type": "string",
968
                   "description": "The text to type"
969
                }
              },
970
              "required": ["string_to_type"]
971
            }
972
         }
973
       ]
974
975
       Figure 8: The example provided illustrates the conversion of action type candidates into the input format
976
       supported by OpenAI API for function calling. As demonstrated in this example, information about action
977
       types can be conveyed through either of a function calling interface of an LLM service API or a prompt that
978
       includes the description of the candidates in plain text.
979
980
981
982
983
984
985
986
```



F TASK COVERAGE OF THE AGENTS THAT RELY SOLELY ON SCREEN IMAGES FOR MINIWOB++

1038

G COMPARISON OF THE CAAP AGENT WITH SOTAS IN MINIWOB++ PER-TASK RESULT

Table 7: Comparison with other works in terms of SR per task for MiniWoB++ problems.

1039									
1040	Task	CAAP	SeeClick	Pix2Act	CC-Net	WebGUM	AdaPlanner	RCI	Human
1041	bisect-angle	-	-	0.960	0.970	-	-	-	0.920
1042	book-flight	-	-	-	0.870	0.980	-	-	0.870
1043	chase-circle	-	-	-	0.930	-	-	-	0.820
1044	choose-date	-	0.020	0.790	0.970	1.000	-	-	0.970
1045	choose-date-easy	-	-	-	0.990	1.000	-	-	0.990
1046	choose-date-medium	-	-	-	0.990	1.000	-	-	0.980
1047	choose-list	1.000	-	-	0.990	1.000	1.000	1.000	0.980
1048	circle-center	1.000	-	0.960	0.970	-	-	-	0.960
1040	click-button	0.980	0.960	0.990	1.000	1.000	1.000	1.000	0.980
1049	click-button-sequence	0.920	0.860	0.990	1.000	1.000	1.000	1.000	0.940
1050	click-checkboxes	0.980	0.780	1.000	0.980	1.000	1.000	1.000	0.970
1051	click-checkboxes-large	1.000	0.020	0.990	0.710	0.990	1.000	0.940	0.870
1052	click-checkboxes-soft	1.000	0.220	0.610	0.950	1.000	0.800	0.960	0.730
1053	click-checkboxes-transfer	1.000	0.700	1.000	0.990	1.000	0.980	1.000	0.980
1054	click-collapsible	0.920	1.000	0.940	1.000	1.000	1.000	1.000	0.990
1055	click-collapsible-2	0.900	0.480	0.970	0.980	0.950	0.840	1.000	0.970
1056	click-color	-	1.000	0.990	1.000	1.000	1.000	1.000	0.970
1057	click-dialog	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
1058	click-dialog-2	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.990
1059	click-link	1.000	0.900	0.980	0.990	1.000	0.980	1.000	0.990
1060	click-menu	0.800	-	-	0.940	0.970	0.780	1.000	0.970
1061	click-menu-2	0.920	-	-	0.830	-	-	-	0.980
1062	click-option	0.960	1.000	1.000	0.990	1.000	1.000	1.000	0.990
1062	click-pie	-	0.800	0.990	0.970	0.990	-	-	0.980
1003	click-scroll-list	0.940	-	-	0.600	1.000	1.000	1.000	0.910
1004	click-shades	-	0.020	0.990	1.000	1.000	1.000	1.000	0.910
1065	click-shape	-	0.520	0.940	0.950	0.940	0.750	0.980	0.880
1066	click-tab	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.990
1067	click-tab-2	0.960	0.600	0.980	0.980	0.990	0.850	1.000	0.970
1068	click-tab-2-easy	1.000	-	0.990	0.990	-	-	-	0.990
1069	click-tab-2-hard	0.980	0.420	0.970	0.980	0.950	0.780	0.980	0.960
1070	click-tab-2-medium	1.000	-	1.000	0.990	-	-	-	0.970
1071	click-test	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
1072	click-test-z	0.920	0.940	1.000	1.000	1.000	1.000	1.000	0.990
1073	click-test-transfer	1.000	-	1.000	1.000	-	-	-	0.990
1074		1.000	0.560	1.000	0.700	1.000	1.000	0.980	0.030
1075	copy-paste	1.000	0.800	-	0.790	-	-	-	0.940
1076	coupt-shape	1.000	0.000	- 0 700	0.050	-	-	-	0.940
1077	count-snape		0.200	1.000	1.000	0.000	0.300	0.400	0.820
1077	drag-box	1 000	-	0.000	1.000	-	-	-	0.900
1070	drag-circle	0.980	_	0.990	1.000	-	-	-	0.990
1079	urag-entere	0.980	-	-	-	-	-	-	-
1080									

1082									
1083	Task	CAAP	SeeClick	Pix2Act	CC-Net	WebGUM	AdaPlanner	RCI	Human
1084	drag-cube	-	-	-	0.790	-	-	-	0.990
1085	drag-item	-	-	1.000	1.000	-	-	-	0.980
1086	drag-items	0.880	-	1.000	0.990	-	-	-	0.930
1087	drag-items-grid	0.780	-	0.890	0.980	-	-	-	0.870
1088	drag-shapes	-	-	0.980	0.990	-	-	-	0.960
1089	drag-single-shape	0.940	-	-	-	-	-	-	-
1000	drag-sort-numbers	0.720	-	0.950	0.970	-	-	-	0.920
1090	email-inbox	0.960	0.800	-	1.000	1.000	0.980	0.980	0.960
1091	email-inbox-delete	1.000	-	1.000	1.000	-	-	-	0.990
1092	email-inbox-forward	0.960	-	-	1.000	-	-	-	0.960
1093	email-inbox-forward-nl	0.960	0.740	-	1.000	1.000	1.000	1.000	0.910
1094	email-inbox-forward-nl-turk	0.820	0.560	-	1.000	1.000	1.000	0.940	0.880
1095	email-inbox-important	1.000	-	1.000	1.000	-	-	-	0.990
1096	email-inbox-nl-turk	0.900	0.680	-	1.000	1.000	0.900	0.980	0.930
1097	email-inbox-noscroll	0.980	-	-	1.000	0.000	-	-	0.960
1098	email-inbox-reply	0.960	-	-	1.000	-	-	-	0.910
1099	email-inbox-star-reply	0.980	-	-	1.000	-	-	-	0.950
1100	enter-date	-	1.000	1.000	1.000	1.000	1.000	0.960	0.970
1101	enter-password	1.000	1.000	-	1.000	1.000	0.980	1.000	0.960
1100	enter-text	1.000	1.000	-	1.000	1.000	0.980	1.000	0.980
1102	enter-text-2	1.000	-	0.970	0.980	-	-	-	0.910
1103	enter-text-dynamic	1.000	1.000	-	1.000	1.000	0.960	1.000	0.970
1104	enter-time	-	-	1.000	0.970	1.000	0.960	1.000	0.980
1105	find-greatest	1.000	-	-	-	-	-	-	-
1106	find-midpoint	-	-	0.960	0.970	-	-	-	0.940
1107	find-word	0.820	0.100	-	0.880	-	-	-	0.960
1108	focus-text	1.000	1.000	-	1.000	1.000	1.000	1.000	1.000
1109	focus-text-2	1.000	0.960	-	1.000	1.000	0.940	1.000	0.990
1110	generate-number	0.860	-	-	-	-	-	-	-
1111	grid-coordinate	-	0.520	0.920	1.000	1.000	1.000	1.000	0.870
1112	guess-number	0.980	1.000	-	1.000	0.430	0.880	0.200	0.990
1113	highlight-text	0.920	-	-	1.000	-	-	-	0.970
1113	highlight-text-2	0.460	-	-	1.000	-	-	-	0.970
1114	identify-shape	0.980	0.680	1.000	1.000	1.000	0.960	1.000	0.980
1115	login-user	1.000	1.000	-	1.000	1.000	1.000	1.000	0.960
1116	login-user-popup	1.000	0.980	-	1.000	1.000	0.980	0.680	0.940
1117	moving-items	-	-	-	0.880	-	-	-	0.180
1118	multi-layouts	0.980	0.720	-	1.000	1.000	0.840	0.960	0.950
1119	multi-orderings	1.000	0.860	-	1.000	1.000	1.000	1.000	0.960
1120	navigate-tree	-	0.820	0.990	0.990	1.000	0.820	1.000	0.980
1121	number-checkboxes	-	-	0.840	0.990	-	-	-	0.960
1122	odd-or-even	0.980	-	-	-	-	-	-	-
1123	read-table	0.940	0.720	-	0.970	-	-	-	0.970
110/	read-table-2	0.740	-	-	0.940	-	-	-	0.950
1124	resize-textarea	1.000	-	0.990	1.000	-	-	-	0.940
1125	right-angle	-	-	0.970	0.980	-	-	-	0.870
1126			-						

11	29
11	30
11	31
11	32
11	33
11	34
11	35
11	36
11	37
11	38
11	39
11	40
11	41
11	42
11	43
11	44
11	45
11	46
11	47
11	48
11	49
11	50
11	51
11	52
11	53
11	54
11	55
11	56
11	57
11	58
11	59
11	60
11	61
11	62
11	63
11	64
11	65
11	66
11	67

Task	CAAP	SeeClick	Pix2Act	CC-Net	WebGUM	AdaPlanner	RCI	Human
scroll-text	0.620	-	-	0.960	-	-	-	0.970
scroll-text-2	1.000	-	-	1.000	-	-	-	0.970
search-engine	-	0.840	-	1.000	0.960	1.000	1.000	0.970
simon-says	-	-	-	0.000	-	-	-	0.620
simple-algebra	1.000	0.380	1.000	0.750	-	0.820	1.000	0.860
simple-arithmetic	1.000	0.780	1.000	0.860	-	-	-	0.960
sign-agreement	0.960	-	-	-	-	-	-	-
social-media	-	-	-	0.900	1.000	0.820	0.980	0.960
social-media-all	-	-	-	0.750	0.520	1.000	1.000	0.890
social-media-some	-	-	-	0.850	0.730	0.900	0.960	0.910
terminal	-	-	-	0.000	-	0.980	1.000	0.880
text-editor	-	-	-	0.980	-	-	-	0.880
text-transform	-	0.460	0.920	0.600	-	-	0.800	0.860
tic-tac-toe	-	0.580	0.830	0.830	0.560	0.480	0.560	0.710
unicode-test	0.900	0.000	1.000	1.000	-	-	-	0.990
use-autocomplete	0.960	0.820	0.990	1.000	0.980	0.880	0.580	0.980
use-colorwheel	-	-	0.970	0.980	-	-	-	0.900
use-colorwheel-2	-	-	0.950	0.950	-	-	-	0.940
use-slider	-	0.320	0.920	0.910	-	-	-	0.980
use-slider-2	-	-	1.000	0.950	-	-	-	0.970
use-spinner	0.980	0.160	-	1.000	0.110	0.900	0.960	0.980
visual-addition	-	-	1.000	0.990	-	-	-	0.970
Average SR	0.945	0.694	0.962	0.936	0.925	0.929	0.940	0.935

1175 H ANNOTATION TOOL

1177					1
1178		Annotator DIR Load Prev Next click ch	neckboxes transfer 3000 20240513 093203	- 0 X	
1179		IMG: 002/002 << Prev Next >> Next with	th data >>		
1180		Save Element	Save T Load T verbose: Sort	Delete ALL	
1181		Select hj and click Submit.	Select hj and click Submit.	01: checkbox 02: checkbox	
1182				04: checkbox 05: checkbox 05: checkbox	
1183				UC: DUTTON	
1184					
1185		ov			
1186		🗹 hj	🗹 hj		
1187		□ 1Z			
1188		Submit	Submit		
1189					
1190				v	
1191		x1: 8 < > x2: 41 < > y1: 91 • •	y2: 108 Y Expand fit threshold: 50		
1192		C dropdown C input_field C text	area C resize_handle C scrollbar_v C i	con	
1193		subtype:			
1194		text			
1195		checked: focused: highlighted: kuto Fit	Load Element from Screen2Text		
1196			update		
1197					
1198	Figure 10: Our GUI-base	d annotation tool enab	les efficient annotation	n of UI ele	ments within screenshots.
1199					
1200					
1201					
1202					
1203					
1204					
1205					
1206					
1207					
1208					
1209					
1210					
1211					
1212					
1213					
1214					
1216					
1210					
1218					
1219					
1220					
1221					

I COMPARISONS TO OTHER APPROACHES

Table 8 summarizes the input types, agent architectures, training methodologies, and whether experimental results on MiniWoB++ and WebShop are reported for our method and other approaches.

Mathad	Modelity	Model		Training	Experii	nent
Wiethou	Wiodanty	Vision	Decision	Scheme	MiniWoB++	WebShop
Human	-	-	-	-	-	
WGE	DOM	-	DOM-NET	SL+RL	\checkmark	-
WebN-T5	HTML	-	WebN-T5-3B	SL	\checkmark	-
RCI	HTML	-	gpt-4	-	\checkmark	-
AdaPlanner	HTML	-	text-davinci-003	-	\checkmark	-
WebShop	HTML+Image	ResNet	BERT, BART	SL+RL	-	\checkmark
WebGUM	HTML+Image	ViT-B16	Flan-T5-XL	SL	\checkmark	\checkmark
CC Net	DOM+Image	DecNet	Transformer	SI I DI	1	
	/ Image only	Kesivet	Transformer	SLTKL		_
Pix2Act	Image	Pix2Struct	Transformer	SL+RL	\checkmark	\checkmark
SeeClick	Image	ViT	Qwen-VL 9.6B	SL		-
CAAP	Image	YOLOv8,	gnt-4-0125	SL		
Cinn	inage	Pix2Struct	SPC 1-0125	SL	V	V

Table 8: Comparison of agent configurations in input types, architectures, training methodologies.

I.1 EXPERIMENT RESULT FOR MINIWOB++

The data requirements and reported performance in literature are described in Table 9. One notable approach is CC-Net, which reports results across the broadest range of MiniWoB++ tasks. However, this required the collection of 2.4 million demonstrations, demanding approximately 6,300 hours of human labor, and faces challenges in real-world applications due to its high reliance on DOM information. RCI (Kim et al., 2023) and AdaPlanner (Sun et al., 2023) achieved average task SRs above 92% with fewer than 100 demonstrations, but their dependence on HTML inputs limits their task coverage. When recalculating SRs, including unsupported or unreported tasks, their performance drops to 50.8% and 49.2%, respectively. WebGUM (Furuta et al., 2023) expanded task coverage by using both HTML and image inputs. However, despite utilizing 401K demonstrations, its results only showed a 1.9% improvement over RCI, achieving an SR of 52.7%. Pix2Act (Shaw et al., 2023) was the first method to achieve high task SRs on MiniWoB++ using image inputs exclusively. It leveraged 1.3 million demonstrations, achieving over 96% success on 58 tasks, with an SR of 55.8% across the same set of 100 tasks. In contrast, SeeClick, which also used image inputs but on a smaller demonstration dataset, failed to sufficiently demonstrate problem-solving ability. Our approach represents the most advanced image-only agent, solving 73 tasks with a high SR of 94.5%, using fewer than 1.8K demonstrations. Furthermore, it achieved a 63.3% SR on the 100-task range, 7.5% higher than Pix2Act, with 61 tasks demonstrating an SR of over 80%—five more tasks than Pix2Act. (See Table 7 of Appendix G.) This represents a significant improvement achieved with fewer than 100 human demonstrations, resulting from the powerful reasoning capabilities of the LLM and our carefully modularized agent design.

1265 I.2 EXPERIMENT RESULT FOR WEBSHOP

Table 10 compares our approach with studies reporting performance on the WebShop benchmark. WebGUM was trained using both HTML and image inputs with 1K human demonstrations, achieving an average

1269	Table 9: Comparison with other methods that solve MiniWoB++ problems. While the primary configuration
1270	of CC-Net utilizes the DOM+Image modality, we have also included the performance of the image-only
1271	modality in this table to provide a more in-depth comparison.

1273	Method	Modality	Image datasize	Demo datasize	Reported SR	Reported tasks
1274	Human				93.5%	104
1275	WGE	DOM	0	~4.8K	66.5%	48
1276	WebN-T5	HTML	0	12K	48.4%	56
1277	RCI	HTML	0	~0.1K	94.0%	54
1278	AdaPlanner	HTML	0	~0.1K	92.9%	53
1279	WebGUM	HTML+Image	0	401K	94.2%	56
1280	CC-Net	DOM+Image	0	2.4M	93.6%	104
1281	CC-Net	Image	0	2.4M	24.0%	104
1282	Pix2Act	Image	0	1.3M	96.2%	59
1283	SeeClick	Image	0	2.8K	69.4%	55
1284	CAAP	Image	1.76K	0.1K	94.5%	73

reward of 66.6, surpassing the performance of the initial model introduced with the WebShop benchmark, which was trained using an Supervised Learning (SL)+Reinforcement Learning (RL) approach. Pix2Act, designed to operate solely on image inputs, reported an average reward of 46.7 when trained with 1K human demonstrations using a combination of SL and RL techniques. Our agent, utilizing a vision model training dataset collected through an automated process and just a single human demonstration, achieved a significantly higher average reward of 62.3, outperforming Pix2Act. This result is comparable to methods that leverage both HTML and image inputs, demonstrating that our approach can effectively function across web and non-web environments with minimal human effort in dataset collection.

Table 10: Comparison with other methods that solve WebShop problems.

Method	Modality	Image datasize	Demo datasize	Task score	SR
Human	-	-	-	82.1	59.6%
WebShop	HTML+Image	0	10.6K	62.4	28.7%
WebGUM	HTML+Image	0	1K	67.5	45.0%
Pix2Act	Image	0	1K	46.7	N/A
CAAP	Image	0.3K	1	62.3	34.7%

J PER-TASK RESULTS FOR ABLATION STUDY ON CAAP COMPONENTS

Table 11: Per-task SRs with respect to varying combinations of prompt components.

1320			-			
1321	Task	Full configurations	Demo without	CoT phrases without demo	CoT phrases with action-only demo	No CoT, no demo
1322	1 1. 4	1 000	0.080	0.040	0.020	0.000
1323	choose-list	1.000	0.980	0.940	0.920	0.980
1324	circle-center	1.000	1.000	1.000	0.980	0.980
1325	click-button	0.980	0.980	0.980	0.980	0.980
1326	click-button-sequence	0.920	0.920	0.920	0.920	0.920
1327	click-checkboxes	0.980	1.000	1.000	1.000	0.960
1328	click-checkboxes-large	1.000	0.960	1.000	1.000	0.860
1329	click-checkboxes-solt	1.000	0.980	0.980	1.000	0.960
1330	click-checkboxes-transfer	1.000	1.000	1.000	1.000	1.000
1331	click-collapsible	0.920	0.140	1.000	0.960	0.340
1332	click-collapsible-2	0.900	0.900	0.920	0.920	0.780
1222	click-dialog	1.000	1.000	1.000	1.000	1.000
1004	click-dialog-2	1.000	1.000	0.980	1.000	1.000
1334	click-lilik	1.000	1.000	1.000	0.400	0.260
1335	click-menu 2	0.800	0.880	0.560	0.400	0.500
1336	click-menu-2	0.920	1.000	0.560	0.000	0.780
1337	click-option	0.960	0.960	0.960	0.980	0.940
1338	click-scroll-list	0.940	0.940	0.860	0.920	0.700
1339	click-tab	1.000	1.000	1.000	1.000	0.080
1340	click tab 2 answ	0.900	0.980	0.980	0.900	1.000
1341	click tab 2 hard	1.000	1.000	1.000	1.000	1.000
1342	click tab 2 medium	1,000	1.000	1.000	1.000	1.000
1343	click test	1.000	1.000	1.000	1.000	1.000
1344	click test 2	0.020	0.020	0.020	0.020	0.020
13/15	click_test_transfer	0.920	0.920	0.920	0.920	0.920
1946	click-widget	1,000	0.920	0.920	0.920	0.920
1047	conv-naste	0.980	0.980	0.000	0.980	0.000
1347	copy puste	1,000	1,000	0.820	0.980	0.780
1348	drag-box	1.000	1.000	0.980	1 000	0.760
1349	drag-circle	0.980	0.980	0.980	0.980	0.780
1350	drag-items	0.880	0.860	0.940	0.940	0.580
1351	drag-items-grid	0.780	0.620	0.600	0.780	0.620
1352	drag-single-shape	0.940	0.940	0.940	0.900	0.740
1353	drag-sort-numbers	0.720	0.480	0.560	0.620	0.160
1354	email-inbox	0.960	0.960	0.860	0.900	0.720
1355	email-inbox-delete	1.000	1.000	1.000	1.000	0.980
1356	email-inbox-forward	0.960	0.980	0.640	0.960	0.620
1357	email-inbox-forward-nl	0.960	1.000	0.460	0.920	0.520
1358	email-inbox-forward-nl-turk	0.820	0.800	0.220	0.820	0.380
1250	email-inbox-important	1.000	1.000	0.980	1.000	0.960
1928	email-inbox-nl-turk	0.900	0.800	0.540	0.860	0.500
1360		0.700	2.500		2.500	2.200

1370		E-11	Dama mith mt	C-T-h	C-T -h	N. C.T
1371	Task	configurations	CoT phrases	without demo	action-only demo	no demo
1372			0.040		1.000	
1373	email-inbox-noscroll	0.980	0.940	0.920	1.000	0.860
1374	email-inbox-reply	0.960	0.980	0.940	0.900	0.960
1375	email-inbox-star-reply	0.980	0.960	0.980	0.960	0.960
1376	enter-password	1.000	1.000	1.000	1.000	0.980
1377	enter-text	1.000	1.000	1.000	1.000	0.980
1378	enter-text-2	1.000	1.000	1.000	1.000	0.980
1070	enter-text-dynamic	1.000	1.000	1.000	1.000	0.980
1379	find-greatest	1.000	1.000	0.140	0.900	0.140
1380	find-word	0.820	0.620	0.740	0.800	0.380
1381	focus-text	1.000	1.000	1.000	1.000	0.980
1382	focus-text-2	1.000	1.000	1.000	1.000	0.980
1383	generate-number	0.860	0.800	0.840	0.760	0.660
1384	guess-number	0.980	1.000	0.860	0.960	0.900
1385	highlight-text	0.920	0.960	0.680	0.900	0.940
1386	highlight-text-2	0.460	0.500	0.140	0.400	0.240
1387	identify-shape	0.980	0.980	0.940	0.960	0.960
1388	login-user	1.000	1.000	1.000	1.000	0.980
1000	login-user-popup	1.000	0.820	0.440	0.440	0.440
1309	multi-layouts	0.980	0.920	0.960	1.000	0.660
1390	multi-orderings	1.000	1.000	0.960	1.000	0.660
1391	odd-or-even	0.980	0.860	0.980	0.960	0.840
1392	read-table	0.940	0.820	0.900	0.900	0.840
1393	read-table-2	0.740	0.700	0.720	0.720	0.700
1394	resize-textarea	1.000	1.000	0.900	0.980	0.980
1395	scroll-text	0.620	0.780	0.200	0.420	0.020
1396	scroll-text-2	1.000	0.980	0.720	0.880	0.780
1397	simple-algebra	1.000	0.980	1.000	1.000	0.700
1398	simple-arithmetic	1.000	1.000	1.000	1.000	0.940
1200	sign-agreement	0.960	0.920	0.420	0.920	0.420
1/00	unicode-test	0.900	0.920	0.900	0.880	0.880
1400	use-autocomplete	0.960	0.880	0.900	0.960	0.900
1401	use-spinner	0.980	1.000	0.900	0.920	0.700
1402	Average SR	0.945	0.920	0.845	0.910	0.792
1403	0					

K COMPARISON OF THE CAAP AND RCI PROMPTINGS

Table 12: Per-task SRs for RCI agent with CAAP and original RCI agent.

	RCI	RCI		RCI	RCI
Task	agent	agent	Took	agent	agent
Task	with	(original)		with	(original)
	CAAP			CAAP	
choose-list	0.80	1.00	email-inbox	0.66	0.82
click-button	1.00	1.00	email-inbox-forward-nl	0.90	0.02
click-button-sequence	1.00	1.00	email-inbox-forward-nl-turk	1.00	0.20
click-checkboxes	1.00	1.00	email-inbox-nl-turk	0.40	0.28
click-checkboxes-large	1.00	1.00	enter-date	0.00	0.00
click-checkboxes-soft	0.98	0.84	enter-password	1.00	1.00
click-checkboxes-transfer	1.00	1.00	enter-time	0.02	0.00
click-collapsible	1.00	1.00	focus-text	0.98	1.00
click-collapsible-2	0.92	0.82	grid-coordinate	0.96	1.00
click-color	1.00	1.00	identify-shape	1.00	0.76
click-dialog	1.00	1.00	login-user-popup	0.44	0.44
click-dialog-2	1.00	1.00	multi-layouts	0.88	0.82
click-link	0.98	1.00	navigate-tree	0.92	0.86
click-menu	1.00	1.00	search-engine	0.30	0.30
click-option	1.00	1.00	simple-algebra	0.80	0.40
click-scroll-list	1.00	1.00	social-media	0.98	0.96
click-shades	1.00	1.00	social-media-all	0.98	0.98
click-shape	0.94	0.94	social-media-some	0.90	0.44
click-tab	1.00	1.00	terminal	0.00	0.00
click-tab-2	0.92	0.94	tic-tac-toe	0.30	0.14
click-tab-2-hard	0.98	0.96	use-autocomplete	0.92	0.48
click-test	1.00	1.00	use-spinner	0.88	1.00
click-test-2	0.98	1.00	Average SR	0.840	0.769
click-widget	1.00	1.00		0.0.0	009
count-shape	0.78	0.76			

L CASE STUDY OF MINIWOB++ TASKS

In this section, we closely examine the failure cases that arise during the execution of MiniWoB++ tasks by the CAAP agent. Figure 11 presents screenshots of selected exemplary failure cases. We have categorized these eight instances into three groups: incorrect directives, observation failures, and action-proposal failures.

1463									
1464	Thanks Bette, I'll "Ma	uris.".	I'd like to send the email to Ingaborg's email.	from Elka	Drag Maig	ghdiln up b	y one position.	Click button ONE, then click button TWO.	sk
1465									
1466	Primary	Q	Primary	Q î	+ Tom	al			
1467	Clarie	☆ 💼	Vinita	<>> €	• Tan				
1468	Elit ut. Lorem congue pu		Massa. Cursus. A, sem		# Mai	ghdiln			
1469	Mireielle	12 1	Ingaborg	े ं	‡ Far	rah			
1470	Est, libero lac		Id. Massa. Urna					THE	
1471	Bette	<u>ک</u>	Briny	े ह	∓ Mu	reil			
1472	Facilisis fauci		Fells. Fermentum morbi		‡ Heo	dwig			
1473	ludvo		Sucanna						
1474	(a)		(b)			(c)		(d)	
1476 1477	Find the 6th word in paragraph, type that in textbox and press "Su	the nto the ubmit".	Select Cassandry from th list and click Submit.	e scroll	Drag Hen	mione to th	e center left.	Highlight the text in the 1st paragraph and click submi	i. t.
1478 1479	Neque amet maecena cursus sed.	as neque,	Tamara Neala		Barbi	Imelda	Hermione	ld porta quis elementum praesent	sed
1481			Rayna		Jany	Sharleen	Saloma	morbi dolor urna consec id turpis eget	tetur
1482	Quitantit		Cordelia					purus ut pulvinar ultricies	s eaet
1483	Submit		Imeta		Calaatia	Manuala	Marsha		
1484			Submit		Celestia	Margalo	Nerte	Submit	
1485									
1486									
1487	(e)		(f)			(g)		(h)	
1488			F '		. C (1 C				
1489			Figure 11: F	all cases	of the C	LAAP a	gent.		
1490									

Incorrect directives In Figure 11, cases (a) and (b) correspond to failures due to incorrect directives. Case (a) involves a directive that is not a task command, exemplified by the nonsensical sentence "Thanks Bette, I'll Mauris." Thus, this case is a MiniWoB++ benchmark error. Case (b) concerns a directive with a grammatical error. In the directive "I'd like to send the email from Elka to Ingaborg's email," a human might recognize the grammatical mistake and deduce that the email should be forwarded to Ingaborg. However, in our failure case, the LLM inputs "Ingaborg's email." into the field designated for the forward recipient, instead, and attempts to send the email.

Observation failures In Figure 11, cases (c) and (d) exemplify failures due to observation errors. Case (c) involves a failure where the task required identifying an object labeled 'Maighdiln'. However, during the UI element understanding process, the text was misrecognized as 'Maighdin' and conveyed it to the LLM, which incorrectly determined that the target object was absent, leading to failure. In case (d), the button marked 'ONE' is obscured behind a button labeled 'TWO'. Although the 'ONE' text is not visible in the screenshot,

the presence of the obscured button can be visually confirmed. A human operator would likely attempt to move the 'TWO' button to reveal and verify the text on the obscured button.

Action-proposal failures In Figure 11, cases (e), (f), (g), and (h) illustrate failures resulting from action-proposal errors. Case (e) exemplifies a task requiring the identification of the sixth word, 'sed.' Although the visual observer correctly observed and reported the scene, the LLM incorrectly identified the sixth word. In case (f), the task was to find 'Cassandry', which was not visible on the screen. The LLM initiated a progressive scroll but prematurely stopped scrolling and failed to make the correct decision to continue. Case (g) involved the LLM determining the coordinates of text-written blocks and predicting their positions to move an object labeled 'Hermione' to the central left position. The LLM decided on the drag action but incorrectly positioned the endpoint, leading to failure. Case (h) required highlighting a specific paragraph within a document composed of multiple paragraphs. Our visual observer is designed to detect text on a line-by-line basis, making it a challenging task to predict paragraph structure from the coordinates of each line. In this case, our agent failed to accurately predict the paragraph configuration. These four cases represent failures due to incorrect judgments by the LLM, with particular difficulty, as seen in (g) and (h), in interpreting coordinates represented numerically.

Among the failure cases, a tiny portion of the cases result from errors in the benchmark directives. While roughly 40% arise due to observation errors inherent in the agent configuration based on visual observation data, the others are failures that occur within the action-proposal domain. However, most minor errors in directives or observations tend to be appropriately accounted for during the action-proposal process by the LLM, often leading to successful outcomes as much as failures. This tendency is expected to intensify as the reasoning capabilities of LLMs advance, thereby improving their task-solving abilities.