# AssistGUI: Task-Oriented PC Graphical User Interface Automation

Difei Gao, Lei Ji, Zechen Bai, Mingyu Ouyang, Peiran Li, Dongxing Mao, Qinchen Wu,
Weichen Zhang, Peiyi Wang, Xiangwu Guo, Hengxu Wang, Luowei Zhou, Mike Zheng Shou†
†Corresponding author
Show Lab, National University of Singapore
https://showlab.github.io/assistgui/

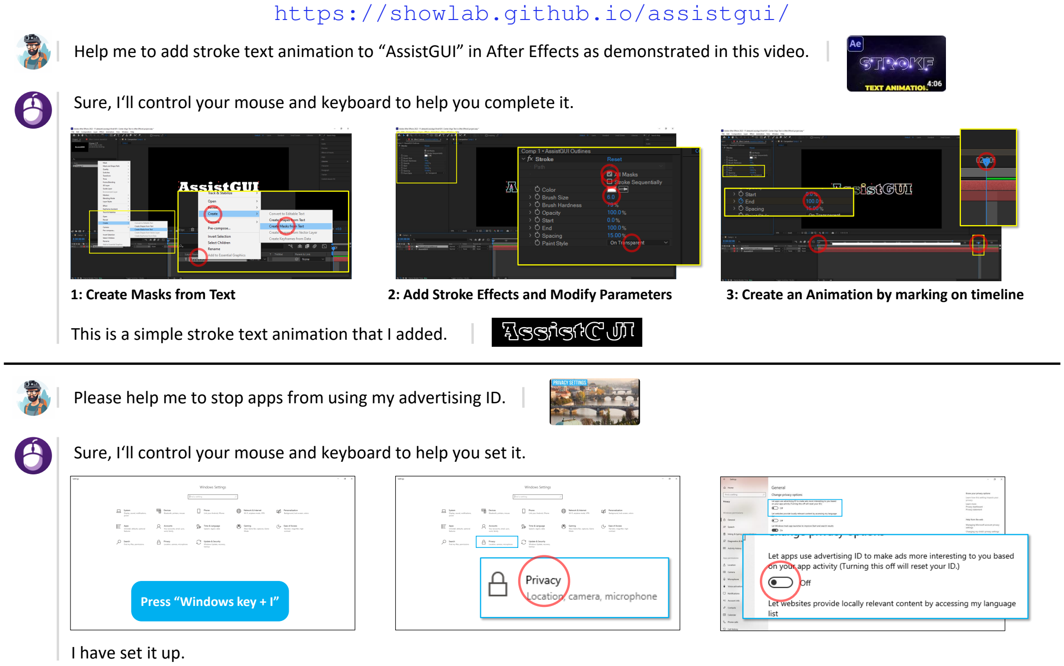


Figure 1. **Illustration of GUI Task Automation in AssistGUI.** Given a user query and an instructional video for reference, an agent is required to manipulate the keyboard and mouse on the desktop to complete the task as requested by the user.

## Abstract

*Graphical User Interface (GUI) automation holds significant promise for assisting users with complex tasks, thereby boosting human productivity. Existing works leveraging Large Language Model (LLM) or LLM-based AI agents have shown capabilities in automating tasks on Android and Web platforms. However, these tasks are primarily aimed at simple device usage and entertainment operations. This paper presents a novel benchmark,* AssistGUI, *to evaluate whether models are capable of manipulating the mouse and keyboard on the Windows platform in response to user-requested tasks. We carefully collected a set of 100 tasks from nine widely-used software applications, such as, After Effects and MS Word, each accompanied by the necessary project files for better evaluation. Moreover, we propose a multi-agent collaboration framework, which incorporates four agents to perform task decomposition, GUI parsing, action generation, and reflection. Our experimental results reveal that our multi-agent collaboration mechanism outshines existing methods in performance. Nevertheless, the potential remains substantial, with the best model attaining only a 46% success rate on our benchmark. We conclude with a thorough analysis of the current methods' limitations, setting the stage for future breakthroughs in this domain.*

## 1. Introduction

Novices often face a steep learning curve when acquainting themselves with complex PC applications. For instance, software like After Effects and Premiere Pro offer a suite of advanced functions for video editing, yet its richness in features sets a high entry barrier for new users. An AI Assistant with the capacity to comprehend GUI interfaces, grasp software usage methodologies, and manipulate applications would significantly expedite the learning and operating processes. As such an Assistant evolves, it will liberate humans from the tedious complexities that currently impede their creativity and productivity.

Early software automation methods, exemplified by voice assistants such as Siri or Alexa, rely on predefined intents and the extraction of parameters from user queries to execute functions, lacking the flexibility required for complex operations. With the advent of generative models, e.g., GPT [20], there has been a shift towards using Large Language Models (LLMs) [36–38] or LLM agents [48, 50] to formulate interactive tasks as a text-to-text generation. Several benchmarks [18] are proposed to evaluate their performances on using an Ubuntu bash terminal, using a database, or engaging in card games, with some recent works [20, 21] demonstrating impressive results. Moreover, some benchmarks [25, 32, 41, 49] are proposed to evaluate Web navigation and Smartphone manipulation. Some work has proposed methods based on HTML [25, 41] and pure vision [30, 46]. [46] utilized GPT-4V-SoM [47] for Smartphone GUI Navigation, which has achieved promising results. While these studies are indeed exciting, these tasks are primarily centered around entertainment scenarios. Consequently, an agent's proficiency in these tasks may not necessarily lead to a substantial increase in human productivity.

Therefore, this paper aims to evaluate the model on task-oriented PC Graphical User Interface Automation, aimed at assessing model performance in utilizing productivity software. This task poses unique challenges compared to previous Web and Android Automation:

- *Dense GUI Understanding*: This involves interpreting various forms of information, not only salient texts on the screen but also various visual elements like icons and footage in the office or design software.
- *Complex Operations*: PC operations demand more sophisticated actions than those on the Web or Smartphone, extending beyond basic tapping, typing, etc. to include operations like dragging files or drawing masks on footage.
- *Long Procedure*: Executing a task in productivity software can involve a sequence of complex steps. For example, creating a single effect in AE will include layer creation, media import, effect adding, animation creation, etc.

In order to better research this important but still largely unexplored domain, we introduce ASSISTGUI, a benchmark designed for PC GUI Automation. As illustrated in Figure 1, the model receives an instructional video demonstrating a specific function of an application, along with a user query pertinent to the video's content. The model's objective is to interact with the software to fulfill the task specified in the query. The inclusion of instructional videos is crucial, particularly for tools like After Effects, which have a vast array of user-developed customized features. This design aims to make the model adaptable and efficient at acquiring new usage techniques.

Correspondingly, we constructed a benchmark that spans 5 major categories of PC tasks: office work, design, widget usage, system setting, and file manipulation, covering 9 popular applications, such as Premiere Pro, After Effect, PowerPoint, etc. In total of 100 specific tasks are provided, each accompanied by a textual query, an instructional video, and carefully created project files. In addition to the data, we have developed a system that enables a local Windows environment to be presented as an interactive platform to a remote server, facilitating model development and testing.

In addition, we introduce a multi-agent collaboration framework, named AutoPC, in which four agents work together to achieve software automation, as depicted in Figure 3. Specifically, we propose a novel planner that facilitates the hierarchical decomposition of tasks. An advanced GUI parser is designed to identify a variety of UI elements. Furthermore, an actor can generate specific actions under the guidance of a critic, with the ability to adjust future steps based on the critic's feedback. Our experiments on the ASSISTGUI benchmark revealed that while the proposed model demonstrates promising potential, it also underscores the task's inherent complexity. Subsequent ablation analysis of different components within our agent framework revealed limitations of current methods when it comes to intricate GUI automation tasks. These insights lead us to suggest future directions for improvement in GUI understanding and action generation for PC GUI applications.

In summary, our work makes the following contributions:

- We introduce, to the best of our knowledge, the first task specifically designed for PC software automation.
- We have created a comprehensive benchmark featuring a carefully selected collection of samples and developed environments that aid in evaluation.
- We present a strong baseline equipped with advanced GUI perception capability and a new planning mechanism.
- Extensive experimentation assesses our approach's effectiveness and highlights the challenges in PC GUI automation for existing models.

## 2. Related Work

**UI Task Automation Benchmark.** UI automation tasks mainly focus on mobile or web applications with both environment development and benchmark construction. The *mobile* scenarios are widely studied with open-source environments built on top of the Android ecosystem. The environments [28, 39] provide an interactive way for reinforcement learning for relatively simple tasks. The benchmarks [5, 15, 25, 41] further extend to more diverse and complex low-level or high-level tasks. Additionally, there are several simulated *web* environments developed for agents to learn in an interactive way[6, 24, 32, 49, 53]. Regarding further computer tasks, NL2Bash[16] and agentbench[18] provide interaction with the *terminal* systems taking language as inputs and outputs. Different from them, ours is more challenging to handle graphical interaction within a real-world PC environment for complex UI and diverse tasks.

**LLM-as-Agent.** Recent studies present promising research directions prompting LLM for multi-step reasoning and invoking application-specific APIs, external tools, or domain-specific models. Some works [23, 27, 27, 33, 40, 50, 50], such as CoT, and ReAct, enhance the model's capability for better conversation by logical reasoning. There is also a growing body of work [7, 19, 31, 35, 44, 48] focusing on using LLMs in conjunction with visual tools to perform multimodal tasks, such as visual question answering and video summarization. Some research [44] even proposes LLM-based agents for image editing. We introduce a specialized LMM-based agent tailored for PC GUI Automation, aiming to provide a powerful baseline for this task.

**Embodied AI for UI Task Automation.** The significant challenges of GUI task automation are the understanding of the complex graphical UI observation and the planning to achieve various tasks, leading to *end-to-end* supervised approaches or LLM-based zero-shot *two-stage* solutions. Previous end-to-end methods adopt reinforcement learning[8] or imitation learning[10]. [4, 30, 34, 51, 52] rely on vision-language-action pretraining to learn to directly map visual observation to actions. However, these methods usually require a significant number of human expert demonstrations, which are still hard to generalize to the general applications. With the advent of LLM, there are some LLM-based two-stage methods. The first stage is to semantically understand the elements of the observed UI by either off-the-shelf models like OCR or learnable vision-language models [1, 3, 9, 13]. For example, [41, 42, 46] propose to convert GUI into HTML representation or natural language sentence. Consequently, the second stage is to generate executable steps given the UI elements [11, 14, 49, 53] usually with LLM. However, single OCR and vision-language models are limited to simple GUIs and fail to capture the full complexity of PC GUIs. They also struggle with long processes due to their single-step generation approach. To address these limitations, we've developed an LLM-based agent equipped with diverse tools for parsing various UI elements and a new hierarchical planning and critic mechanism for handling extended procedures.

## 3. AssistGUI Benchmark

AssistGUI benchmark provides an interactive environment, dataset across broad tasks, and goal-oriented evaluation.

### 3.1. Task Formulation

PC task automation in AssistGUI can be formulated as follows: given a natural language query that briefly describes a specific task, along with an instructional video as a supplement that more detailed illustrates how to complete it,

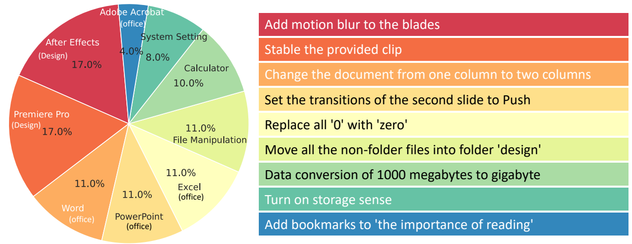


Figure 2. **Distribution of collect tasks and one example query for each task.** We have gathered tasks across 9 applications, focusing on the use of productivity software as well as fundamental computer operations and settings.

and the relevant application, the output is a sequence of UI actions to fulfill the user's query.

**Task description.** To describe the task, a textual request $\boldsymbol{q}$ is provided by the user, which describes the functionality of an application to be accomplished, e.g., *Center align the text "AssistGUI" in my opened After Effect project*. For some functions of productivity tools, there might be multiple user-developed implementations. We aim for the model to generate actions based on the given references. Thus, an instructional video, denoted as $\boldsymbol{v}$, is also provided.

**State observation.** The state of the environment is composed of two types of information. The first type stems from the operating system's textual metadata about the software being used. In contrast to web pages, where HTML offers comprehensive information, much of this metadata in PC applications is internal and thus not readily accessible. As a result, the metadata mainly includes the layout of panels and pop-up windows. The second type of information consists of screen captures, which offer a more holistic view by providing visual context.

**Action space.** Our action space consists of all the raw mouse and keyboard actions, including left-click, right-click, double-click, drag, keystrokes, and combinations of keys for shortcuts, among others. Mouse-related operations also include the target position at the pixel space of the observed screenshot. To construct a universal and complete representation of actions, we exactly followed a widely utilized Python library for controlling the mouse and keyboard, PyAutoGUI. One action is denoted by the syntax `action_type(arguments)`, e.g., `dragTo(100, 100)`, which indicates the execution of a drag action from the current position to the coordinate (100, 100).

**Environment Implementation.** Recognizing that productivity tools usually only support Windows or Mac systems, while AI models are often deployed on Linux, we've created a Python library to expose a local Windows environment as an interactive platform to a remote server. This is done using PyWinAuto API to collect metadata and screenshots from Windows. A communication system sends data to the server, and let server then sends predicted actions back to the local client for execution on the productivity

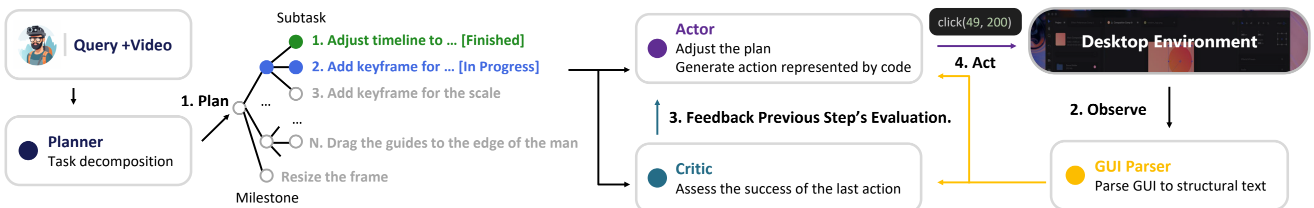


Figure 3. **Diagram Illustration of AutoPC.** It first outlines key milestones and subtasks, then iteratively employs a GUI Parser, a Critic module for action assessment, and an Actor module for adjusting the plan and generating code for controlling the PC, sequentially completing subtasks until the task is finished.

tools. This setup allows remote control of the software by the server-based model through specific action commands.

### 3.2. Data Collection

Our benchmark is designed to include a broad spectrum of PC tasks, systematically segmented into five major categories that are indicative of routine computer-based work. These categories include design, Office work (Office), system settings (Sys. Set.), widget usage (Widget), and file management (File Mani.). The collection of task data within ASSISTGUI is achieved by the following steps:

**Task Collection.** Due to the complexity of GUI operations, at this early stage, we are primarily focusing on relatively basic tasks. We carefully select some popular instructional videos and those duration do not exceed five minutes from official software websites and video-sharing platforms. We also manually crafted one query for each instructional video. These queries illustrate the tasks that the model is expected to complete. It is important to note that the task indicated by the query may not always align exactly with the operations shown in the video; it could include some user-customized requirements. Therefore, the model needs to modify the steps based on the instructional video, e.g., type in a different text.

**Project File Preparation.** To make the results in the environment to be reproducible, we provide project files for all editing-related tasks. This ensures that all models initiate their tasks from an identical starting state. The project files included in our benchmark stem from two primary sources: A portion of the project files is directly sourced from the official tutorials available on the software's website. These files are typically crafted by the software providers to accompany their instructional materials. The remaining project files are meticulously prepared by annotators. We have also documented the version of each project file. The tested models are expected to modify this file using applications of the same version for fair comparison.

**Quality Checking.** To guarantee the correctness of our benchmark, each task has undergone a quality check by letting our annotators complete the tasks within the software to verify if they yield accurate outputs. The quality check focuses on two main aspects: Firstly, it verifies the correctness of the content in the instructional video, ensuring that the demonstrated steps are accurate and lead to the anticipated outcome. Secondly, it confirms that the project files are correct and fully functional.

ASSISTGUI finally collects 100 specific tasks from 9 commonly used applications like Premiere Pro, After Effects, and PowerPoint. We present the distribution of collected over software and show one example query for each software ASSISTGUI task in Figure 2.

### 3.3. Evaluation

ASSISTGUI adopts an outcome-oriented evaluation approach to determine the success rate of models. Since ASSISTGUI yields several types of outputs: video output (Design), document output (Office), the final state of the software (Widget), system settings (Sys. Set.), and folder structure (File Mani.), it is hard to construct one general metric to fit all tasks, thus, we design specific metrics to calculate the success rate tailored to each type of task.

For the Design and Office tasks, we compare the similarity of the model's results with the ground truth at a pixel granularity. If it exceeds a certain threshold, it is considered successful and scores 1 point; otherwise, it scores 0. The threshold varies slightly for different tasks, depending on whether the task inherently includes a certain level of randomness. We did not adopt CLIP-Sim [43], commonly used in video generation, because video editing often involves animation changes rather than semantic changes, making it difficult for CLIP to discern subtle differences. For Widget tasks, we compare the final screenshot with the ground truth, if the same in the display region (obtained by metadata), then consider it a success. For the Sys. Set. and File Mani., we write scripts to automatically determine whether the system settings and folder structure meet the expected criteria.

## 4. Method

**Overview.** We introduce a multi-agent collaboration framework for PC software automation, AutoPC, that possesses the capabilities to perceive the software environment, plan actions, and execute them, as shown in Figure 3. Specifically, the agent works in two stages: In the first stage, given a query and a video, the Planner creates a high-level plan outlining

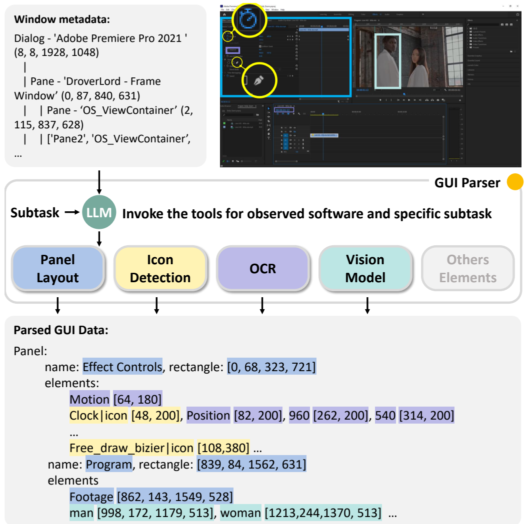


Figure 4. **Diagram Illustration of GUI Parser.** An LLM invokes different vision tools to parse various UI elements.

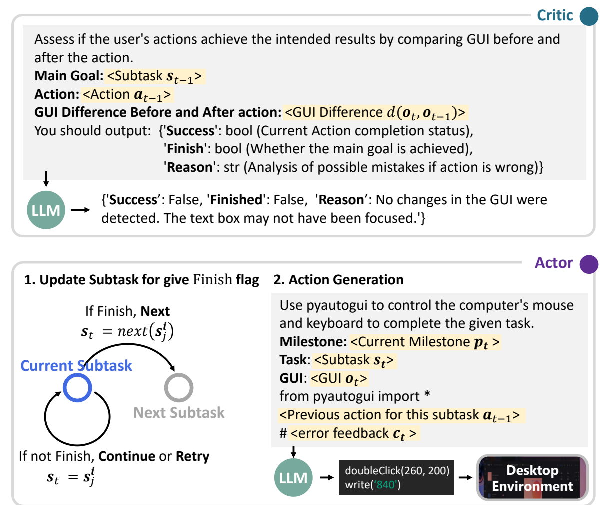


Figure 5. **Top**: The **Critic** assesses the effectiveness of the previous action by analyzing the screenshots taken before and after its execution. **Bottom**: The **Actor** first updates the current subtask, then generates the subsequent action, considering the current observation, current subtask, historical actions, and Critic's feedback.

the key milestones and subtasks of the task. The second stage involves the collaborative work of three modules to sequentially accomplish these subtasks. The GUI Parser observes the GUI environment, the Critic module assesses the quality of the previous action, and the Actor then adjusts the plan based on this assessment and generates code to control the PC.

### 4.1. Planner

For a given query $\boldsymbol{q}$ and instructional video $\boldsymbol{v}$, the Planner aims to output a hierarchical task tree $\boldsymbol{p} = [\boldsymbol{p}_1, \boldsymbol{p}_2, ..., \boldsymbol{p}_N]$, where $\boldsymbol{p}_i$ is a text string describe the $i$-th milestone of the task. And each $\boldsymbol{p}_i$ corresponds to a list of subtasks $[\boldsymbol{s}_1^i, ..., \boldsymbol{s}_{N_i}^i]$, $\boldsymbol{s}_j^i$ is also a text string, indicating the $j$-th subtask for $i$-th milestone. This is achieved in the following steps. First, the LLM is prompted to extract hierarchical steps based on the subtitles of the video. Subsequently, the LLM is requested to modify the extracted steps in accordance with the user's query. Finally, we design a specific traversal algorithm that will only traverse the leaf nodes in order and send the corresponding subtask to the following modules. We show more details of prompts in the Supp.

### 4.2. GUI Parser

The goal of the GUI Parser is to convert an observed screenshot into a structured textual representation $\boldsymbol{o}_t$ like the Document Object Model (DOM). Given that PC software typically comprises a wide variety of UI elements, it is hard for one model to extract all information, thus, we adopt approaches similar to MMReAct [48] and VisualClues [45], invoking multiple tools to extract information, as shown in Figure 4. Specifically, we utilize metadata from the system for panel segmentation, employ the OCR model to extract text from images and develop a pattern-matching method to identify icons. Additionally, some vision models, including a detector, and a segmentation model, are used to localize the objects in footage, and we have designed simple algorithms to extract specific elements such as scrolls and reference lines, etc. The GUI information is represented panel by panel, including the meanings of UI elements and their spatial position coordinates. Figure 6 shows an example.

### 4.3. Critic

The Critic utilizes an LLM to evaluate the success of the executed action by analyzing the screenshots taken before and after the execution of the action $d(\boldsymbol{o}_t, \boldsymbol{o}_{t-1})$, where $d(.)$ is a function for identifying differences. It outputs four kinds of information: whether the previous action was executed correctly (a Boolean Success Flag), and if not, it provides an explanation; whether the current subtask is completed (Boolean Finish Flag), and if not, it offers an explanation, as shown in the Top of Figure 5. The two flags and explanations, denoted as $\boldsymbol{c}_t$ will feed to the Actor.

### 4.4. Actor

The Actor is built upon an LLM, aiming to generate actions within the action space of the ASSISTGUI benchmark. Specifically, given the Finish Flag provided by the Critic, the mode first plans what should be done next, as shown in Figure 5. If the Finished Flag is False, the subtask $\boldsymbol{s}_t$ at time $t$ will still be $\boldsymbol{s}_j^i$, otherwise, $\boldsymbol{s}_t = next(\boldsymbol{s}_j^i)$, where $next(.)$ indicates moving to next subtask by using our designed traverse method illustrated in Sec. 4.1.

Then, the Actor generates an output action by considering

Table 1. Success rate (%) of agents with different planning methods on ASSISTGUI. Human* represents the average performance of three non-expert humans who have viewed the instructional video only once, like how the model does. These results are a reference to better sense the extent of the model's capabilities.

| Method | Design | Office | Widget | Sys. Set | File Mani. | Overall |
|---|---|---|---|---|---|---|
| CoT | 5.9 | 10.8 | 20.0 | 0.00 | 36.7 | 12.0 |
| ReAct | 14.7 | 27.0 | 50.0 | 62.5 | 63.6 | 32.0 |
| Ours | 32.4 | 40.5 | 60.0 | 75.0 | 72.7 | 46.0 |
| Human* | 73.5 | 83.7 | 100.0 | 100.0 | 100.0 | 85.0 |

various factors: the current state of the observed software $\boldsymbol{o}_t$, the previous action $\boldsymbol{a}_t$, the current subtask $\boldsymbol{s}_t$, and its corresponding milestone $\boldsymbol{p}_t = parent(\boldsymbol{s}_t)$ (which indicates the milestone associated with the current subtask). Additionally, the Actor takes into account the Critic's feedback $\boldsymbol{c}_t$ on the performance of the previous action. Formally,

$$\boldsymbol{a}_t = \arg\max_{\boldsymbol{a} \in \mathcal{A}} p(\boldsymbol{a}|\boldsymbol{a}_{t-1}, \boldsymbol{o}_t, \boldsymbol{s}_t, \boldsymbol{p}_t, \boldsymbol{c}_t), \quad (1)$$

where $\mathcal{A}$ denotes the action space, comprised of Python code. It's important to note that the output action $\boldsymbol{a}_t$ can either be a single action or a sequence of actions. This is implemented by prompting an LLM to process all the aforementioned information as input and subsequently generate the code for the next step, as illustrated at the bottom of Figure 5.

## 5. Experiments

**Implementation Details.** In the following experiments, we use gpt-4-0613 [21] provided by OpenAI as the default LLM. In the GUI parser, we use Google OCR for extracting text, Yolo-v8 [26] to coarsely localize objects, and LangSAM [12, 17] to obtain the precise object contours. The difference module $d(.)$ is implemented by using DeepDiff [29].

### 5.1. Quantitative Results

As ASSISTGUI is a novel Task that requires planning with an instructional video and processing PC environments (previous works mostly focus on Web or Android), there are no ready-made state-of-the-art models available for evaluation. Thus, we construct various variants based on our approach, to contrast some core concepts from previous works, thereby showing the effectiveness of our method and the challenges of ASSISTGUI.

**Comparision with SOTA Planning Method.** In Table 1, we compare the planning approaches that have recently demonstrated exceptional performance in other environments. Specifically, we retained the GUI Parser and removed both the Planning and Actor-Critic modules. The subtitle of the instructional video is simply put into the prompt. Then, the model plans the steps in the following methods:

- CoT [40]: The CoT generates all the steps at once, which cannot obtain information from the environment.

Table 2. Success rate (%) of agents with ablation of reasoning module.

| Method | Design | Office | Widget | Sys. Set. | File Mani. | Overall |
|---|---|---|---|---|---|---|
| Full Model | 32.4 | 40.5 | 60.0 | 75.0 | 72.7 | 46.0 |
| w/o Planning | 20.6 | 27.0 | 50.0 | 75.0 | 63.6 | 35.0 |
| w/o Critic | 26.5 | 32.4 | 60.0 | 75.0 | 72.7 | 41.0 |
| w/o Ins. Video | 11.8 | 37.8 | 60.0 | 62.5 | 72.7 | 37.0 |

- ReAct [50]: It iteratively interacts with the environment through a cycle of thought, action, and observation.

The experimental results demonstrate that our model significantly surpasses previous planning methods. The result of CoT reveals that PC GUI Automation tasks often entail screen chanthus, thus, it is unable to cope effectively. Regarding ReAct, since it does not convert lengthy videos into discrete steps, it can operate on the finest granularity of action plans. Additionally, ReAct's absence of a dedicated module for evaluating and adjusting the planning path becomes a shortcoming, especially for complex tasks in office and design environments. The overall results indicate that ASSISTGUI poses significant challenges, especially for complex productivity tools. This difficulty arises from the intricacies involved in understanding and navigating sophisticated software interfaces, which require nuanced interpretation of visual elements and context-aware decision-making.

**Ablation on Planner, Actor and Critic.** We also conducted ablation studies on our Critic and Planner, as shown in Table 2, where the w/o Planner method directly feeds the whole subtitle into Actor, instead of the parsed subtask. For simple tasks, the impact of these components was not particularly significant. However, their influence becomes much more apparent in complex Office and Design tasks. On another note, while the Critic appears to be a very important module, its performance enhancement was not as large as we initially expected. This is primarily because the Critic's judgments in complex tasks are not always accurate. It requires a high level of action-vision alignment, which still remains a relatively underexplored area, but we believe it is a direction worth exploring. Additionally, we constructed a variant that does not take into account the subtitles of videos. Instead, it utilizes GPT-4 to plan milestones and subtasks, denoted as w/o Ins. Video. This approach showed almost no significant performance loss in simple tasks because there weren't many alternative solutions. However, for the use of complex software like After Effects and Premiere Pro, instructional Videos proved to be very helpful.

**Ablation on GUI Parser.**

Correctly parsing UI Elements is essential for generating actions. Here, we eliminate different UI elements in parsed GUI data to observe their impact. Table 3 shows that removing OCR had the most significant impact since text often contains crucial information in a GUI. Icons also led to notable performance loss, especially in Design and Office software, where many icons lack corresponding textual

Table 3. Success rate (%) of agents with ablation of GUI Parser.

| UI Elements | | | | |
|---|---|---|---|---|
| Panel Layout | Icon | OCR | Others | Overall |
| ✓ | ✓ | ✓ | ✓ | 46.0 |
| ✗ | ✓ | ✓ | ✓ | 44.0 |
| ✓ | ✗ | ✓ | ✓ | 13.0 |
| ✓ | ✓ | ✗ | ✓ | 4.0 |
| ✓ | ✓ | ✓ | ✗ | 43.0 |
| Qwen-VL-Chat [2] | | | | 5.0 |

descriptions and are essential for specific functions. Interestingly, Panel Layout had minimal impact on performance, indicating GPT-4 can recognize the button without panel information, though it's still necessary for operations like clicking in a blank or margin of the panel area. The Others category, including footage content, scrolls, and similar elements, also had little effect. This is due to the model's current limitations in handling complex footage operations, even though they correctly recognize but still fail to complete the task. We also try to replace Qwen-VL-Chat [2] to replace the GUI Parser, allowing GPT-4 to plan button interactions and Qwen-VL-Chat to determine their positions. However, the results were not very satisfactory, as there may not have particular training for GUI button grounding.

**Impact of Large Language Model.** We also experimented with different language models, gpt-3.5-turbo, and Llama2-7B [38], in various modules, but found the results to be generally unsatisfactory, as shown in Table 4. There are two main reasons for this: 1) The requirement for specific output formats. For instance, an action must be in the form of current step code and can only output code; any other content would render it non-executable. Similarly, the results from planning need to adhere to a certain format, which other language models sometimes fail to follow. 2) The issue of model hallucination. For the generation of actions, the model needs to stop at appropriate times, using updated GUI information to continue generating actions. However, non-GPT-4 models often hallucinate or invent too much information, leading to an incorrect code. For these relatively lightweight models to perform such customized functions effectively, they may require fine-tuning with specific datasets.

### 5.2. Qualitative Results

In Fig 6, we showcase some visualized results. Firstly, we present a successful prediction example, demonstrating that the model can effectively plan each step for relatively long processes, accurately perceive specific elements in the GUI, and convert them into the correct action code. Additionally, we display the performance of our designed Multi-modal LLM Agent, which can accurately identify most content, including small icons such as a clock-shaped keyframe button, checkboxes, and expand buttons. In contrast, although GPT-4V [22] possesses robust OCR capabilities, it fails to output button positions, rendering it unable to execute operations. The current best method to modify GPT-4V for button grounding is GPT-4V-SoM [47], which uses semantic-SAM to segment the image first, then label it, and finally input it to GPT-4V. This approach achieves remarkable results in Web and Android Navigation tasks. However, as seen, for PC GUI understanding, the performance of GPT-4V-SoM is almost nullified due to the limitations of Semantic-SAM's segmentation capabilities in productivity software.

Table 4. Success rate (%) of agents with ablation on LLM.

| Planner | Actor & Critic | Overall Score |
|---|---|---|
| GPT-4 | GPT-4 | 46.0 |
| GPT-4 | GPT-3.5 | 12.0 |
| GPT-4 | Llama2 | 1.0 |
| GPT-3.5 | GPT-4 | 19.0 |
| Llama2 | GPT-4 | 5.0 |

Finally, we also highlight some common errors encountered. 1) The model struggles with complex operations on footage, which can be highly intricate. For instance, Query 1 requires using a roto brush to select an object, necessitating continuous adjustments based on the generated edges, a capability our model currently lacks. Achieving this function might require training with specific samples or a more powerful Agent framework. 2) The model has difficulty understanding blurred areas, such as the edges of documents, blank spaces in Panels, or determining which area to select when multiple files are involved. 3) The spatial relationship in dense text. The granularity of OCR output bounding boxes is uncontrollable. Selecting a specific word or character in a text segment is not straightforward with the current OCR predictions. This may require a highly versatile text grounding model to address effectively.

## 6. Conclusion

This paper introduced ASSISTGUI, a novel benchmark for assessing the capability of models to manipulate the mouse and keyboard on the Windows platform in response to user requests. To this end, we collected a diverse set of 100 tasks across 9 widely-used applications, ensuring each task was supplemented with the necessary project files for a fair evaluation. We also presented our multi-agent collaboration framework. This framework is anchored by an enhanced reasoning mechanism to coordinate four GUI-related agents for software automation. Our design is particularly adept at handling complex, lengthy procedural tasks that are commonplace in professional software environments. Our experimental results were promising, demonstrating that our approach notably outperforms existing methods in GUI automation. However, despite these advancements, our findings also highlight the considerable challenges that remain in this field.

**Acknowledgements.** This project is supported by Mike Zheng Shou's Start-Up Grant from National University of Singapore.


Figure 6. **Qualititave Results.** Top: We show one successful prediction. Middle: We compare our GUI Parser results with Semantic-SAM which is the core component for supporting GUI-4V to ground in Web or Smartphone Platform (i.e., GPT-4V-SoM). Bottom: We display some common errors with explanation.

## References


[1] Chongyang Bai, Xiaoxue Zang, Ying Xu, Srinivas Sunkara, Abhinav Rastogi, Jindong Chen, et al. Uibert: Learning generic multimodal representations for ui understanding. *arXiv preprint arXiv:2107.13731*, 2021.

[2] Jinze Bai, Shuai Bai, Shusheng Yang, Shijie Wang, Sinan Tan, Peng Wang, Junyang Lin, Chang Zhou, and Jingren Zhou. Qwen-vl: A frontier large vision-language model with versatile abilities. *arXiv preprint arXiv:2308.12966*, 2023.

[3] Pratyay Banerjee, Shweti Mahajan, Kushal Arora, Chitta Baral, and Oriana Riva. Lexi: Self-supervised learning of the ui language. *arXiv preprint arXiv:2301.10165*, 2023.

[4] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Chen, Krzysztof Choromanski, Tianli Ding, Danny Driess, Avinava Dubey, Chelsea Finn, et al. Rt-2: Vision-language-action models transfer web knowledge to robotic control. *arXiv preprint arXiv:2307.15818*, 2023.

[5] Andrea Burns, Deniz Arsan, Sanjna Agrawal, Ranjitha Kumar, Kate Saenko, and Bryan A Plummer. A dataset for interactive vision-language navigation with unknown command feasibility. In *European Conference on Computer Vision*, pages 312–328. Springer, 2022.

[6] Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Samuel Stevens, Boshi Wang, Huan Sun, and Yu Su. Mind2web: Towards a generalist agent for the web. *arXiv preprint arXiv:2306.06070*, 2023.

[7] Difei Gao, Lei Ji, Luowei Zhou, Kevin Qinghong Lin, Joya Chen, Zihan Fan, and Mike Zheng Shou. Assistgpt: A general multi-modal assistant that can plan, execute, inspect, and learn. *arXiv preprint arXiv:2306.08640*, 2023.

[8] Izzeddin Gur, Ulrich Rueckert, Aleksandra Faust, and Dilek Hakkani-Tur. Learning to navigate the web. *arXiv preprint arXiv:1812.09195*, 2018.

[9] Zecheng He, Srinivas Sunkara, Xiaoxue Zang, Ying Xu, Lijuan Liu, Nevan Wichers, Gabriel Schubiner, Ruby Lee, and Jindong Chen. Actionbert: Leveraging user actions for semantic understanding of user interfaces. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 5931–5938, 2021.

[10] 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.

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

[12] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C Berg, Wan-Yen Lo, et al. Segment anything. *arXiv preprint arXiv:2304.02643*, 2023.

[13] 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.

[14] Tao Li, Gang Li, Zhiwei Deng, Bryan Wang, and Yang Li. A zero-shot language agent for computer control with structured reflection. *arXiv preprint arXiv:2310.08740*, 2023.

[15] Yang Li, Jiacong He, Xin Zhou, Yuan Zhang, and Jason Baldridge. Mapping natural language instructions to mobile ui action sequences. *arXiv preprint arXiv:2005.03776*, 2020.

[16] Xi Victoria Lin, Chenglong Wang, Luke Zettlemoyer, and Michael D Ernst. Nl2bash: A corpus and semantic parser for natural language interface to the linux operating system. *arXiv preprint arXiv:1802.08979*, 2018.

[17] Shilong Liu, Zhaoyang Zeng, Tianhe Ren, Feng Li, Hao Zhang, Jie Yang, Chunyuan Li, Jianwei Yang, Hang Su, Jun Zhu, et al. Grounding dino: Marrying dino with grounded pre-training for open-set object detection. *arXiv preprint arXiv:2303.05499*, 2023.

[18] Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, et al. Agentbench: Evaluating llms as agents. *arXiv preprint arXiv:2308.03688*, 2023.

[19] Pan Lu, Baolin Peng, Hao Cheng, Michel Galley, Kai-Wei Chang, Ying Nian Wu, Song-Chun Zhu, and Jianfeng Gao. Chameleon: Plug-and-play compositional reasoning with large language models. *arXiv preprint arXiv:2304.09842*, 2023.

[20] OpenAI. Introducing chatgpt. OpenAI Blog, 2021.

[21] OpenAI. Gpt-4 technical report, 2023.

[22] OpenAI. Gpt-4v(ision) system card., 2023.

[23] Bhargavi Paranjape, Scott M. Lundberg, Sameer Singh, Hannaneh Hajishirzi, Luke Zettlemoyer, and Marco Túlio Ribeiro. ART: automatic multi-step reasoning and tool-use for large language models. *arXiv preprint arXiv:2303.09014*, 2023.

[24] Panupong Pasupat, Tian-Shun Jiang, Evan Zheran Liu, Kelvin Guu, and Percy Liang. Mapping natural language commands to web elements. *arXiv preprint arXiv:1808.09132*, 2018.

[25] Christopher Rawles, Alice Li, Daniel Rodriguez, Oriana Riva, and Timothy Lillicrap. Android in the wild: A large-scale dataset for android device control. *arXiv preprint arXiv:2307.10088*, 2023.

[26] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[27] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *arXiv preprint arXiv:2302.04761*, 2023.

[28] Stefan Schneider, Stefan Werner, Ramin Khalili, Artur Hecker, and Holger Karl. mobile-env: An open platform for reinforcement learning in wireless mobile networks. In *NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium*, pages 1–3. IEEE, 2022.

[29] Dehpour Sep. Deepdiff (version 6.7.1). *https://github.com/seperman/deepdiff*, 2023.

[30] 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.

[31] Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. Hugginggpt: Solving ai tasks with chatgpt and its friends in huggingface. *arXiv preprint arXiv:2303.17580*, 2023.

[32] 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.

[33] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik R Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

[34] Liangtai Sun, Xingyu Chen, Lu Chen, Tianle Dai, Zichen Zhu, and Kai Yu. Meta-gui: Towards multi-modal conversational agents on mobile gui. *arXiv preprint arXiv:2205.11029*, 2022.

[35] Dídac Surís, Sachit Menon, and Carl Vondrick. Vipergpt: Visual inference via python execution for reasoning. *arXiv preprint arXiv:2303.08128*, 2023.

[36] Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. Stanford alpaca: An instruction-following llama model. https://github.com/tatsu-lab/stanford_alpaca, 2023.

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

[38] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

[39] Daniel Toyama, Philippe Hamel, Anita Gergely, Gheorghe Comanici, Amelia Glaese, Zafarali Ahmed, Tyler Jackson, Shibl Mourad, and Doina Precup. Androidenv: A reinforcement learning platform for android. *arXiv preprint arXiv:2105.13231*, 2021.

[40] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed Chi, Quoc Le, and Denny Zhou. Chain of thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903*, 2022.

[41] Hao Wen, Yuanchun Li, Guohong Liu, Shanhui Zhao, Tao Yu, Toby Jia-Jun Li, Shiqi Jiang, Yunhao Liu, Yaqin Zhang, and Yunxin Liu. Empowering llm to use smartphone for intelligent task automation. *arXiv preprint arXiv:2308.15272*, 2023.

[42] Hao Wen, Hongming Wang, Jiaxuan Liu, and Yuanchun Li. Droidbot-gpt: Gpt-powered ui automation for android. *arXiv preprint arXiv:2304.07061*, 2023.

[43] Chenfei Wu, Lun Huang, Qianxi Zhang, Binyang Li, Lei Ji, Fan Yang, Guillermo Sapiro, and Nan Duan. Godiva: Generating open-domain videos from natural descriptions. *arXiv preprint arXiv:2104.14806*, 2021.

[44] Chenfei Wu, Shengming Yin, Weizhen Qi, Xiaodong Wang, Zecheng Tang, and Nan Duan. Visual chatgpt: Talking, drawing and editing with visual foundation models. *arXiv preprint arXiv:2303.04671*, 2023.

[45] Yujia Xie, Luowei Zhou, Xiyang Dai, Lu Yuan, Nguyen Bach, Ce Liu, and Michael Zeng. Visual clues: Bridging vision and language foundations for image paragraph captioning. *Advances in Neural Information Processing Systems*, 35: 17287–17300, 2022.

[46] An Yan, Zhengyuan Yang, Wanrong Zhu, Kevin Lin, Linjie Li, Jianfeng Wang, Jianwei Yang, Yiwu Zhong, Julian McAuley, Jianfeng Gao, et al. Gpt-4v in wonderland: Large multimodal models for zero-shot smartphone gui navigation. *arXiv preprint arXiv:2311.07562*, 2023.

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

[48] Zhengyuan Yang, Linjie Li, Jianfeng Wang, Kevin Lin, Ehsan Azarnasab, Faisal Ahmed, Zicheng Liu, Ce Liu, Michael Zeng, and Lijuan Wang. Mm-react: Prompting chatgpt for multimodal reasoning and action. *arXiv preprint arXiv:2303.11381*, 2023.

[49] 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, 2022.

[50] 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*, 2022.

[51] Zhuosheng Zhan and Aston Zhang. You only look at screens: Multimodal chain-of-action agents. *arXiv preprint arXiv:2309.11436*, 2023.

[52] Zhizheng Zhang, Wenxuan Xie, Xiaoyi Zhang, and Yan Lu. Reinforced ui instruction grounding: Towards a generic ui task automation api. *arXiv preprint arXiv:2310.04716*, 2023.

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