

OmniACT: A Dataset and Benchmark for Enabling Multimodal Generalist Autonomous Agents for Desktop and Web

Raghav Kapoor^{1*}, Yash Parag Butala^{1*}, Melisa Russak², Jing Yu Koh¹, Kiran Kamble², Waseem AlShikh², and Ruslan Salakhutdinov¹

¹ Carnegie Mellon University

² Writer.com

{raghavka, ypb}@cs.cmu.edu

<https://huggingface.co/datasets/Writer/omniact>

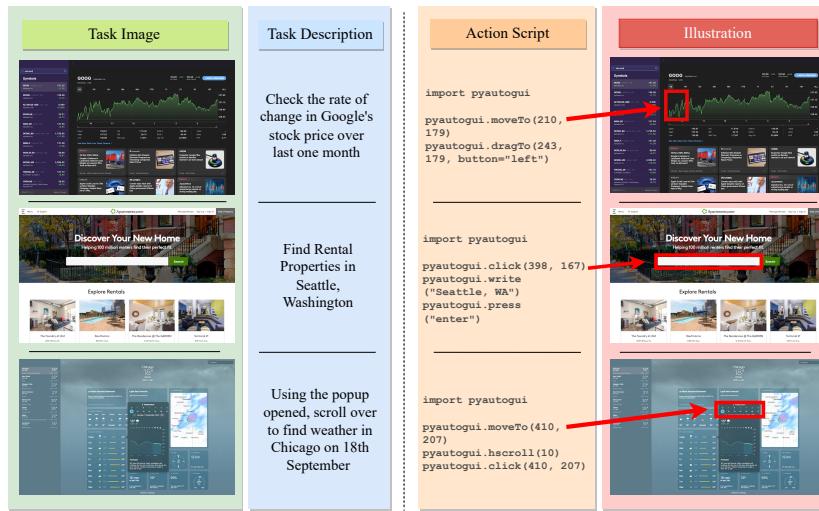


Fig. 1: **OmniACT** dataset and benchmark for enabling autonomous human-computer interaction agents. The left shows an image paired with a natural language task description as the input, the right shows the resulting action script to be executed on the screen. Examples are presented from Stocks, Apartments, and Weather application.

Abstract. For decades, human-computer interaction has fundamentally been manual. Even today, almost all productive work done on the computer necessitates human input at every step. Autonomous virtual agents represent an exciting step in automating many of these menial tasks. Virtual agents would empower users with limited technical proficiency to harness the full possibilities of computer systems. They could also enable the efficient streamlining of numerous computer tasks, ranging from calendar management to complex travel bookings, with minimal human intervention. In this paper, we introduce **OmniACT**, the first-of-a-kind

* These authors contributed equally. The order is determined by dice rolling.

dataset and benchmark for assessing an agent’s capability to generate executable programs to accomplish computer tasks. Our scope extends beyond traditional web automation, covering a diverse range of desktop applications. The dataset consists of fundamental tasks such as “Play the next song”, as well as longer horizon tasks such as “Send an email to John Doe mentioning the time and place to meet”. Specifically, given a pair of screen image and a visually-grounded natural language task, the goal is to generate a script capable of fully executing the task. We run several strong baseline language model agents on our benchmark. The strongest baseline, GPT-4, performs the best on our benchmark However, its performance level still reaches only 15% of the human proficiency in generating executable scripts capable of completing the task, demonstrating the challenge of our task for conventional web agents. Our benchmark provides a platform to measure and evaluate the progress of language model agents in automating computer tasks and motivates future work towards building multimodal models that bridge large language models and the visual grounding of computer screens.

Keywords: Generalist Agent · Multimodal Machine Learning · Dataset and Benchmark · Vision Language Understanding · UI grounding · Human-computer interaction

1 Introduction

Performing computer tasks based on natural language instructions has been a long-standing goal of artificial intelligence [49]. One concrete objective in the line of research is to develop generalist agents that can assist humans in doing computer tasks [21], such as “*Order a pizza from Domino’s*” or “*Write a message to John.*” The agent should be able to open the application and perform the task. Executing these actions on a personal computer involves a sequence of interactions with a mouse and keyboard. For example, the simple task of writing an email involves hovering over the application icon, clicking it, clicking the ‘*New Email*’ button, writing the content of the email, and clicking send. Successfully sending an email requires accurately predicting the correct action at each step and accurately executing it, which is a herculean task even for the best agents today [14].

A generalist agent for computer tasks must understand natural language instructions, process visual screenshots, and produce the correct sequence of actions to be performed to achieve the intended task. Several existing approaches focus on building agents based on the HTML model [9, 40, 62]. However, this approach introduces several challenges and constraints. These agents are limited to web applications and often struggle with complex or long-context HTML code. They cannot interact with native desktop applications or perform tasks that span multiple applications, like drafting an email using text from a code editor, without significant alterations. Furthermore, HTML-based agents, which are inherently powered by text-only language models, typically underperform in tasks requiring visual cues, such as identifying and clicking a blue button

on a desktop’s top-right corner. In contrast, humans can easily understand UI elements like dropdown menus, typable areas, redirections, and options with just a glance.

Towards the goal of developing a generalist autonomous agent with robust visual and user interface (UI) understanding capabilities, we introduce a new task and dataset, **OmniACT**, containing over 9.8K pairs of images and instructions (Figure 1) across different operating systems and the web. This dataset includes screenshots of various UI screens and corresponding natural language instructions. The objective of these instructions is to generate executable commands using the *PyAutoGUI* Python library [1]. *PyAutoGUI* enables the automation of the mouse and keyboard operations, which helps to facilitate interactions with various native applications across macOS, Windows, and Linux. This simplifies completing specified tasks across different web domains and native desktop applications.

We evaluate several language model-based agent baselines on this dataset, including LLaMA [47], Vicuna [7], Palmyra-X (43B) [2], InstructPalmyra-30B [45], GPT 3.5, and GPT-4 [32]. We experiment with fine-tuning Vicuna-13B and LLaMA-13B models using QLoRA [10]. We also benchmark multimodal baseline LLaVa-v1.5-7B, LLaVa-v1.5-13B [47], Gemini-Pro [44] and GPT-4-vision-preview [55] for the task. Our findings highlight the necessity for a multimodal model capable of executing these tasks, and our analysis provides insights into promising future work in the space. Our key contributions are outlined as follows:

1. We release a novel dataset of desktop and website applications consisting of over 9.8K natural language tasks, UI screens, and corresponding code snippets collected through human annotation. We introduce custom performance metrics tailored for computer tasks.
2. We propose DetACT, a module for creating textual representations of the screen using signals from OCR, color, and icon-template matching.
3. We conduct a comprehensive benchmark and analysis of state-of-the-art LLMs and multimodal models on our benchmark. Our results show that **OmniACT** is a challenging task for even the best LLM agents today, and existing models are far below human performance.

2 Related Work

2.1 UI Understanding

User interface (UI) understanding has garnered interest from researchers in the machine learning and human-computer interaction communities, evolving with various models focusing on understanding the semantics of mobile and web user interfaces. UIBert [3], PixelBERT [16], ActionBert [15], VUT [25], Screen2Words [48], WidgetCaptioning [24] and Pix2Act [39] are notable models in this area. They propose approaches for learning the user-interface semantics of the mobile screen using the image and view hierarchy. These models have demonstrated effectiveness in tasks like capability prediction, screen segmentation and

Table 1: Comparison of OmniACT with other related benchmarks.

| Datasets | Size | Env Type | Task | Real-World | Executiona | Supports | Continuous | Scale | Task |
|-----------------------|----------------|------------------------|---------------|------------|-------------|--------------|---------------------|------------|------------------------|
| | | | Heterogeneity | Portayal | Correctness | Desktop Apps | Adaptive Evaluation | | |
| VisualWebArena [20] | 910 | Web | Yes | Yes | Yes | No | No | | Web Navigation |
| WebArena [62] | 812 | Web | Yes | Yes | Yes | No | No | | Web Navigation |
| Mind2Web [9] | 2350 | Web | Yes | Yes | No | No | No | | Web Navigation |
| WebShop [56] | 12000 Products | Web | No | No | Yes | No | No | | Web Navigation |
| RUSS [53] | | Web | Yes | Yes | No | No | No | | Web Navigation |
| WebSRC [6] | | Web | Yes | Yes | - | No | No | | QA |
| MiniWoB++ [17] | | Mobile Websites | No | No | Yes | No | No | | Web Navigation |
| PixelHelp [23] | | Mobile | Yes | Yes | No | No | No | | UI Grounding |
| MetaGUI [42] | | Mobile | Yes | Yes | Yes | No | No | | Mobile Navigation |
| MoTIF [5] | 756 | Mobile | Yes | Yes | Yes | No | No | | Mobile Navigation |
| AITW [35] | 715142 | Mobile and Web | Yes | Yes | Yes | No | No | | Mobile/Web Navigation |
| OmniACT (Ours) | 9802 | Desktop and Web | Yes | Yes | Yes | Yes | Yes | Yes | Code Generation |

understanding, and screen caption generation. Lexi [4] and Spotlight [22] propose models that use vision-only inputs to minimize the reliance on metadata such as view hierarchy. Furata et al. [11] demonstrates the use of fine-tuning for multimodal web navigation. The majority of machine learning models trained for UI understanding leverage the Rico dataset [8] and its extensions, which contain 64,462 unique Android screens and metadata. In addition, [4] released the UICaptions dataset, which consists of diverse image-captions pairs across a wide range of applications. PixelHelp [23] also released a corpus to train models that can interpret natural language instructions and map them to mobile UI actions.

2.2 Autonomous Computer Agents

The advent of large language models (LLMs) has been pivotal in the rapid advancement of agents that operate on web pages. Recent research such as ViperGPT [43] Chameleon [29], RCI Agent [18], VisProg [12], and [31] employ LLMs for planning or action prediction in developing autonomous agents. Benchmark datasets, such as MiniWoB [40], WebShop [56], Macaw-LLM [30], ASH-Prompting [41] Mind2Web [9], WebArena [62], AgentBench [28] and VisualWebArena [20] have also been proposed to measure the ability of LLM-based agents to automate web tasks. These methods mainly involve agents that operate on a text-based Document Object Model (DOM) of HTML scripts. This limits their understanding of screen context, which is crucial for the model’s decision-making and action-taking processes. To address this limitation, [35] released Android in the Wild, a dataset comprising screens, natural language instructions, and corresponding actions. Following this, [59] proposed a multi-modal model, AutoUI, which is designed to build an agent on the Android in the Wild dataset confined to the Android ecosystem. WebAgent [13] utilized Flan-U-PaLM, for grounded code generation, and HTML-T5 and showed improvement on real-world websites.

Current benchmarks for autonomous agents focus mainly on the Web or Android environments, posing challenges for tasks involving desktop applications or spanning multiple applications beyond the web domain. The absence of established benchmarks and datasets in this area, coupled with basic methods

for extracting user interface (UI) elements, underscores the need for significant progress in developing more versatile autonomous agents capable of handling diverse tasks beyond the current scope. To highlight the unique features that **OmniACT** introduces in the assessment of capable autonomous agents, we provide a comparison between the existing benchmarks and our proposed benchmark, **OmniACT**, in Table 1.

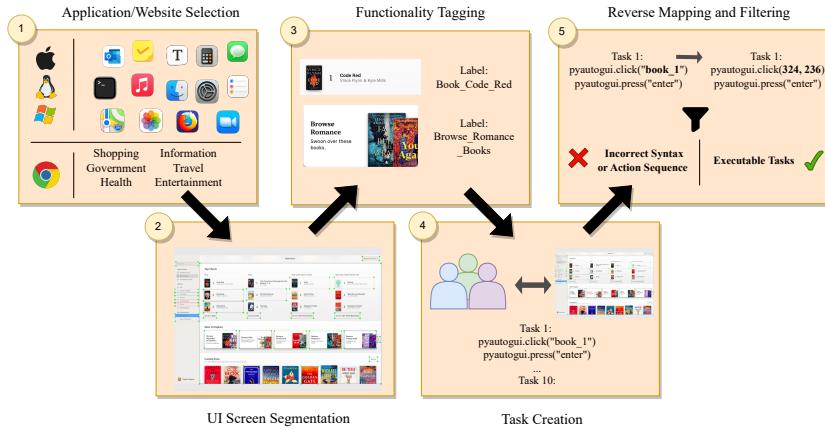


Fig. 2: Data Collection Pipeline. (1) We select over 60 applications and websites to ensure diversity, (2) segment the screen through human-annotated bounding boxes, (3) label the bounding boxes based on functionality, (4) ask student volunteers to come up with tasks, given a screen image, and (5) reverse map the textual labels to coordinates and filter the scripts based on execution and syntax.

3 OmniACT

We introduce a novel dataset and benchmark, **OmniACT**, which measures the performance of autonomous agents on both web and desktop applications. Compared to previous benchmarks which focus on text-based reasoning [9, 17, 40, 56, 62], our benchmark aims to measure multimodal agents that bridge large language model planners and UI understanding vision models. **OmniACT** can be accomplished as a standalone task as it is not under a mock environment.

All actions that a human can execute on the computer can be encoded in the *PyAutoGUI* [1] Python framework. This framework allows a user to execute keyboard and mouse operations by running Python code. The *PyAutoGUI* code to execute these tasks is shown in the third column of Figure 1. For other computer tasks, the *PyAutoGUI* library provides functions such as ‘press’, ‘write’, and ‘scroll’ which can be used to execute the task. Our dataset consists of parallel data of natural language tasks, UI screenshots, and ground truth *PyAutoGUI* scripts that achieve successful execution.

3.1 Task Formulation

Given an input state of a computer defined by the screen S and the task description T in natural language, the goal of the task is to output a sequence of actions A that can successfully accomplish the task T within a screenshot $S \in \{\text{Linux, Windows, MacOS, Webpage}\}$. Formally, the task can be defined as learning the transition function $f : T \times S \rightarrow A$. During dataset collection, we ensure that all task descriptions T are feasible and can be accomplished in the current screenshot S . To reduce ambiguity and facilitate better evaluation, we ensure that task descriptions are detailed and unambiguous. Tasks can also be visually grounded (e.g., ‘Click the red button to start recording’) or natural language based (e.g., ‘Click the My Account button’). We define the action space using the functionalities in the *PyAutoGUI* library: $A \in \{\text{‘click’, ‘dragTo’, ‘scroll’, ‘write’, …}\}$. The exhaustive list of actions is provided in Table 2. Our action space is much larger than other benchmarks [9, 40, 62] that resort to two or three interaction options. Mouse actions such as ‘moveTo’, ‘click’, ‘rightClick’, ‘doubleClick’, and ‘dragTo’, additionally require screen coordinates as arguments, which indicate the pixel location of the action.

Figure 1 illustrates sample tasks and corresponding outputs for three applications within **OmniACT**: (1) Stocks (MacOS), (2) Apartments.com (web page), and (3) Weather (MacOS). The first column depicts the input image, and the second column shows the natural language task that is to be executed on the current screen. To execute these tasks, a user must accurately perform a series of operations using the mouse and keyboard. Eg: to check the rate of change in Google’s stock price over the last month, the mouse has to be moved to the last month and dragged while holding the left-click button to the current month.

3.2 Dataset Preparation

To prepare our dataset, we followed a pipelined approach, as summarized in Figure 2. We first selected a variety of applications and websites. For each application or website, we created bounding boxes around key UI elements and labeled them according to their functionality, which is crucial for assisting human annotators in writing accurate *PyAutoGUI* scripts. After each script is written, we converted the labels back into numeric coordinates, allowing us to align the scripts precisely with the locations of the UI elements. Finally, we thoroughly reviewed each script, focusing on its executability and adherence to syntax standards. This ensured the high quality and functionality of our dataset, making it a valuable resource for training and evaluating autonomous agents.

Application/Website Selection To test the computer agents’ generalization ability across different tasks, we collect tasks across multiple domains on both desktop and web applications. In total, we collect and annotate 9802 data points (Table 3), with the split between desktop and web applications approximately 3:1. The emphasis on desktop applications, which do not contain Document Object Model (DOM) hierarchies unlike HTML-based web pages, presents a more

complex multimodal challenge where visual cues are crucial. We collect tasks from applications within the three most popular operating systems. We select 22 native applications from MacOS, and 8 each from Linux and Windows. We annotate roughly 3 to 4 screens for every application. The full list of applications is provided in the Appendix.

Many common computer tasks today are still performed through web applications, so we also collect 3-4 screenshots from 27 different web applications. To ensure diversity in task intents, we categorize these tasks into one of the following 6 categories: (1) Shopping, (2) Entertainment, (3) Service, (4) Government, (5) Travel, (6) Health. Inspired by the methodology of [9], these categories were selected to cover a wide range of user intents and functionalities.

UI Screen Segmentation To collect gold-standard data, we first annotate and segment the screen by identifying the bounding boxes present on the screen. We employ slightly different techniques for web and desktop applications to create the bounding boxes:

1. **Desktop Applications:** We build a custom annotation interface based on PyQt5³ to create bounding boxes manually over a screen image using a simple drag-and-click mechanism. This custom interface expedites the process and allows us to get highly accurate gold-label data for desktop images.
2. **Websites:** For webpages, we write JavaScript code to extract all interactable (click, type, etc.) regions from HTML source code. We also extract banners, dropdowns, submit, and radio buttons from the screen. We filter the elements to retain only those that are visible and interactable within the screen.

Functionality Tagging To map each bounding box to its correct functional description, we leverage Amazon MTurk workers (see details in Appendix), who are given an image with a bounding box and are required to write the correct description or label of the bounding box’s function. For example, given an image of an Amazon webpage with a *search bar*, the annotator labels it as “*find-product-search-bar*”. The logical descriptions are used to create tasks in a structured manner without the need to identify individual bounding box coordinates.

Task Creation Our approach for each screen involves utilizing all human-annotated bounding boxes and their labels to create tasks that can be executed

³ <https://pypi.org/project/PyQt5/>

Table 2: Action types supported by OmniACT and the number of instances for each action in the dataset.

| Type | Action | % |
|----------|-------------------|-------|
| Mouse | Click | 63.73 |
| | Double Click | 0.58 |
| | Right Click | 0.77 |
| | Move/Hover | 1.85 |
| | Drag | 0.29 |
| | Scroll | 1.68 |
| Keyboard | Horizontal Scroll | 0.17 |
| | Press | 16.28 |
| | Hotkey | 3.00 |
| | Write | 11.65 |

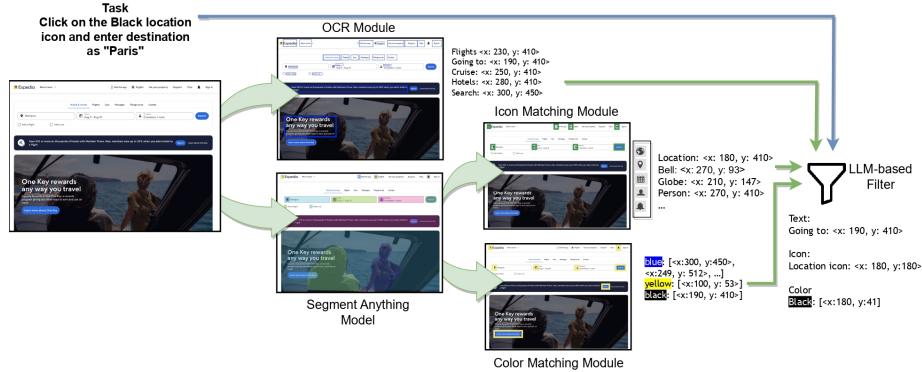


Fig. 3: DetACT Module. Given an initial image and a natural language task description, we use a pipelined approach to run OCR and SAM on the screen. The outputs from SAM are then used by icon and color-matching modules to obtain an exhaustive set of useful UI elements. The list of elements is passed through LLM based filter to select only the elements related to the given task.

within the confines of a single screen. These tasks are designed to be visually grounded in order to measure the capabilities of multimodal agents. We plan to release the bounding box and their corresponding labels as the metadata for evaluation purposes.

For dataset compilation, college students with basic Python programming skills served as annotators, accessing API references for *PyAutoGUI* and examples of potential tasks. Each student generated multiple tasks, each accompanied by three alternative natural language reformulations. For instance, “*What is 3+2?*” might be reformulated as “*Calculate the sum of 2 and 3*” or “*Add two to three*”. To avoid train-test leakage, rephrased tasks were consistently placed in the same dataset split. Further details on the annotation process are available in the Appendix.

Reverse Mapping and Filtering To ensure high-quality data, we incorporate an additional step into the data collection pipeline. We build scripts to map the text-based labels of each bounding box back to their numeric coordinates, and then match the syntax and verify if the task will be executed on the screen. Using this filter, we remove all the non-working or syntactically incorrect data points and finally manually review the set of tasks.

After filtering, we obtain 9802 human-annotated, gold-label data points across more than 200 desktop and web screens

Table 3: Dataset distribution across splits and platforms.

| | Domain | Train | Validation | Test | Total |
|--------------|---------|-------|------------|-------|-------|
| Desktop | Mac OS | 3028 | 444 | 786 | 4258 |
| | Linux | 761 | 126 | 247 | 1134 |
| | Windows | 1573 | 216 | 458 | 2247 |
| Web | - | 1427 | 206 | 530 | 2163 |
| Total | | 6789 | 992 | 2,021 | 9802 |

(Table 3), split into train, validation, and test sets in a 7:1:2 ratio. All collected data will be publicly released to encourage future work on multimodal agents.

4 Evaluation Metrics

In this section, we detail various evaluation metrics for benchmarking model performance on the **OmniACT** dataset. UI screens have additional constraints such as spatial relevance which are not factored in most conventional similarity-based metrics such as BLEU [34], CodeBLEU [36], BERTScore [58] and Code-BERTScore [61]. For example, a valid click action is usually not constrained to a single coordinate but can be any coordinate within a specified region. In the event of invalid coordinate predictions, an agent that predicts coordinates further away from the valid region should invoke a higher penalty compared to an agent that predicted coordinates close to the region. We propose two new metrics adapted: Sequence Score (Section 4.1) and Action Score (Section 4.2) aimed at utilizing UI information.

4.1 Sequence Score

The sequence score measures whether the predicted action sequence (e.g., ‘click’, ‘write’, ‘press’) exactly matches the gold sequence. Since predicting the first action in the sequence is relatively straightforward and later actions are more difficult, we define sequence score as follows:

$$SeqScore_i = \begin{cases} \beta_1 + \beta_2 * (s - 1) & \text{if all actions match} \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

where s is the action sequence length, β_1 is set to 0.1 and β_2 is set to 1.

4.2 Action Score

The action score measures how well a code snippet containing the correct action sequence can perform the task. Specifically, for a script with a correct action sequence, we introduce penalties for inaccurate behavior. The penalties are described below:

1. **Click penalty (M):** For actions ‘click’, ‘rightClick’, ‘doubleClick’, ‘moveTo’, and ‘dragTo’, we penalize code snippets where predicted coordinates lie outside of the bounding box of the UI element. The click penalty for the j^{th} action of the i^{th} example is defined as:

$$M_i^j = \alpha_i \times \begin{cases} 1 - \frac{\mu}{\mu + L_2} & \text{if } SeqScore_i > 0 \\ 1 & \text{otherwise} \end{cases} \quad (2)$$

Here L_2 corresponds to the smallest Euclidean distance between the predicted coordinate and bounding box. L_2 is zero when the predicted coordinate lies within the target bounding box. μ is the Dirichlet smoothing

coefficient which we dynamically set to the inverse of the length of the diagonal of the bounding box. This ensures that the penalty for points outside the bounding box varies based on the size of the bounding box. For two predicted points with the same L_2 , the metric penalizes more heavily if the box is larger. This is sound with the intuition that the chances of clicking on a larger box are higher and should be penalized more in case of a mistake.

2. **Key penalty (K):** For actions ‘press’ and ‘hotkey’, we check whether the set of keys in the target code (represented as GK_i^j) and predicted code (represented as PK_i^j) are the same. It is formally defined as:

$$K_i^j = \alpha_i \times \begin{cases} 0 & \text{if } GK_i^j = PK_i^j \text{ and } SeqScore_i > 0 \\ 1 & \text{otherwise} \end{cases} \quad (3)$$

3. **Write penalty (W_p):** For action type ‘write’, we penalize the output for the sentence to be typed. Specifically, we employ BLEU score [34], and compute:

$$W_i^j = \alpha_i \times \begin{cases} 1 - BLEU(GS_i^j, PS_i^j) & \text{if } SeqScore_i > 1 \\ 1 & \text{otherwise} \end{cases} \quad (4)$$

Here, GS_i^j represents the actual sentence to be typed, and PS_i^j represents the sentence predicted by the model in the j^{th} action of example i .

In the above equations, (α_i) is the weighting factor:

$$\alpha_i = SeqScore_i / \text{length of sequence } i \quad (5)$$

This ensures that the action score $\in [0, 1]$. The mean action score is calculated as follows:

$$\text{Action Score} = \frac{\sum_i \max \left(SeqScore_i - \sum_j (M_i^j + K_i^j + W_i^j), 0 \right)}{\sum_i SeqScore_i} \quad (6)$$

5 DetACT: DETecting ACTions from UI

Understanding UI screens is crucial for multimodal computer tasks. Web-based agents typically use language-only inputs from the HTML DOM. This is insufficient for comprehending the full extent of an application UI, as many components may not be easily described with HTML code. To address this, we propose DetACT, which allows us to convert images of UI layouts into structured code and text outputs for a downstream LLM. DetACT is a system comprised of three distinct modules: the text module, the icon module, and the color module.

1. **Text Extraction:** We use the EasyOCR model⁴ to parse over the UI screens and collect all text-based elements. Along with the text, we also note the locations of each of these elements. This is depicted in Figure 3, along with a list of text elements found on the screen using the OCR Module. We segment and classify the different regions within the screenshot using the

⁴ <https://github.com/JaideedAI/EasyOCR>

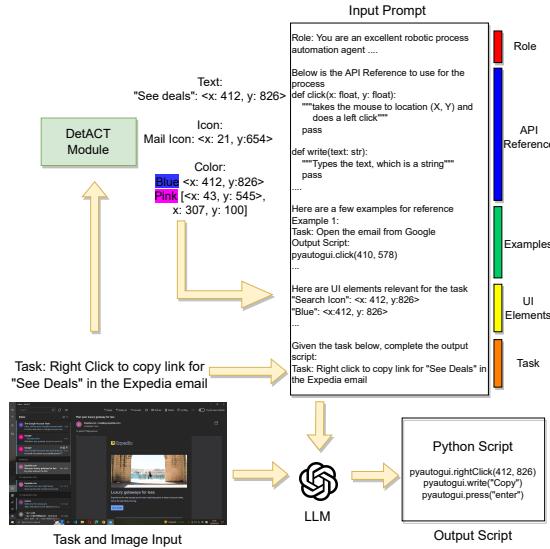


Fig. 4: Baseline Model Architecture. Image and task descriptions are sent to DetACT module, which gives a filtered list of UI elements relevant to feed into the prompt along with the task. We also show the prompt structure used for action script generation. This structure is passed through the LLM (along with the image for multimodal LLM) to generate the automation script.

Segment Anything Model (SAM) [19]. From the outputs, we filter out the non-textual segments for our icon and color detection.

2. **Icon Module:** For matching with the appropriate icon, we use a pack of 1600 icons⁵ as templates. Each of these icons is labeled with their appropriate functionality and is matched with the filtered outputs SAM [19]. For the similarity of the two images, we resize the reference icons and segmented region of interest (ROI) to the same size, and convert both images to grayscale. After this, we use the Structural Similarity Index (SSIM) [52], to find the closest match of the ROI to the icons in our set, and select the ones above the SSIM threshold of 0.95. As seen in Figure 3, a few icons matched on the screen are *Globe* icon, *Calendar* icon, *Person* icon, and *Location* icon; each depicting a different use case.
3. **Color Module:** Finally, to place all segments of interest into appropriate buckets of colors, we average the RGB pixel values over the ROI and, based on that value, bucket them into different color categories. We categorize colors differently based on the human perspective of the ranges of each color. To avoid ambiguity, we consider eleven major colors, namely yellow, blue, green, red, pink, violet, white, black, orange, brown, and grey. We record the center of the element along with the color.

⁵ <https://icomoon.io/>

Once all the elements of each category are extracted with their coordinates, we then filter these UI elements by prompting GPT-4 [32]. We ensure that the elements selected are suited only for our task, for which we also provide the task description in our prompts along with the list of elements. Full details of the prompt are provided in the appendix section of the paper. As we observe in Figure 3, given an image from the Expedia application, and a task (“*Click on the Black Location icon and enter the destination as Paris.*”), the LLM filters out the elements to retain only “*Going To*”, “*Location Icon*”, and the *Black* colored elements from the screen. This is passed as input to the LLM or VLM backbone.

6 Baselines

To evaluate the performance of existing language model-based agents, we conduct experiments with both language-based and multimodal baselines. The DetACT module takes in image and text descriptions of the task and outputs the color, icon, and text-based signals. This is concatenated to the prompt for the LLM prompt-based baselines (see Figure 4). Every prompt starts with a role assignment [60], followed by the detailed API reference of the *PyAutoGUI* function set, along with a textual description of their function. We then add five in-context examples from the training set that most closely match the task (based on the cosine similarity of the MiniLM [50] embeddings of the reference task and the train examples). We add a list of UI elements filtered by the DetACT module to the prompt. Finally, we provide the rules with the task description. For multimodal baselines, we also pass the image pixels to the vision encoder. We choose coordinate-based UI elements in the prompt as recent techniques like the Set-of-Mark (SOM) [54] prompting does not work for desktop settings since it is difficult to obtain interactive elements from the desktop screen images. We report the results of several baselines:

- **Few-shot Generative LLM:** We experiment with models from LLaMA-2 [47], Vicuna-1.5 [7], CodeLLaMA-34B [37], Palmyra [46], and GPT [32] series. We use the prompts structure as shown in Figure 4 to prompt the model. For LLaMA and CodeLLaMa, we reduce the prompt length to 2000 tokens by removing outputs from the DetACT module with lower confidence, as we observed poor performance on longer prompts. For the other models, we allow prompts with up to 4000 token sizes.
- **Finetuned Generative LLM:** We fine-tuned the LLaMA-13B model and Vicuna-13B using QLoRa [10] with rank 64 and scaling factor 16 for 300 steps to generate the code given screen description from the DetACT module and the instruction.
- **Few-shot Generative Multimodal Models:** As *OmniACT* is predominantly multimodal, with a majority of tasks being visually grounded, we conduct experiments with large multimodal models. Given the limited research in this domain [51, 57], there is a scarcity of available multimodal models with significant size adept for this task. Here, we experiment with [26, 27], providing a similar prompt as well as the screen image.

7 Results and Analysis

As shown in Table 4, we experiment with three different categories of models, namely Prompt-based LLMs, Fine-tuned LLMs, and Prompt-based Multimodal Models. GPT-4 is the best-performing approach, scoring higher on the sequence score and invoking lower penalties on coordinate predicting and text input. For prompt-only LLMs, the GPT-3.5-turbo and GPT-4 models outperform the other LLM baselines, including the LLaMA [47] and Vicuna [7] models. We observe that CodeLLaMA-34B [38], which is trained for code generation, also achieves a higher performance than other models of the same size at predicting the action sequences.

Fine-tuned models also perform much better than their few-shot prompt-only counterparts. Fine-tuning substantially improves LLaMA-13B’s sequence score (4.80 to 8.92) and action score (1.62 to 2.14), as well as the other metrics. Despite this, we observed that both, prompt-based LLMs and finetuned LLMs face severe mouse penalties, especially on click coordinates. This is because they rely solely on text-based signals.

To address this, we experiment with multimodal language models (Table 4). We observe that the coordinate prediction improves significantly when we provide the entire image as input to the multimodal LLM, as this enables it to fully utilize the screen representation. In addition to open sourced models, we also experiment with the GPT-4-vision API [55] which shows that GPT-4 Vision [55] outperforms GPT-4 significantly on the Action Score along with improving the sequence score, which we attribute to the strong reasoning abilities of GPT-4 coupled with the improved visual understanding capabilities of the GPT-4-vision model [55]. These findings pave the way towards exciting new research directions on building multimodal models for long-horizon planning and code generation.

Human performance over the task: OmniACT consists of visually complicated tasks, and tests various types of computer skills. In order to get a gauge of how well humans perform, we collect evaluation data from human evaluators. We split the test set uniformly amongst 10 human evaluators, and provided them with the screenshot and task instruction. We record the actions taken by the an-

Table 4: Baseline Performance. (A) Prompt-only LLMs, (B) Fine Tuned LLMs, (C) Prompt-only Multimodal Models. The table represents the Sequence score (SS), click penalty (M_p), Key penalty (K_p), Write Penalty (W_p), and Action Score (AS). The best results for the (SS) and (AS) are highlighted.

| Model | SS(\uparrow) | M_p | K_p | W_p | AS(\uparrow) |
|---------------------------|------------------|-------|-------|-------|------------------|
| Prompt based LLMs | | | | | |
| LLaMA-7B [47] | 4.12 | 1.24 | 1.83 | 0.57 | 0.48 |
| Vicuna-7B [7] | 3.88 | 1.17 | 1.51 | 0.43 | 0.77 |
| LLaMA-13B [47] | 4.80 | 1.32 | 0.93 | 0.93 | 1.62 |
| Vicuna-13B [7] | 5.44 | 1.65 | 0.94 | 1.06 | 1.78 |
| Palmyra-Instruct-30B [45] | 7.51 | 5.68 | 0.12 | 0.40 | 1.31 |
| CodeLLaMA-34B [38] | 10.09 | 2.99 | 2.71 | 0.66 | 3.72 |
| Palmyra-X 43B [2] | 11.20 | 3.12 | 3.02 | 2.12 | 2.94 |
| GPT-3.5-turbo-0613 [33] | 22.85 | 8.13 | 4.51 | 2.31 | 7.89 |
| GPT-4 [32] | 32.75 | 10.27 | 6.99 | 3.89 | 11.60 |
| Finetuned LLMs | | | | | |
| LLaMA-13B FT | 8.92 | 4.61 | 1.43 | 0.74 | 2.14 |
| Vicuna-13B FT | 8.78 | 4.12 | 1.31 | 0.63 | 2.72 |
| Multimodal LLMs | | | | | |
| LLaVA-v1.5-7B [27] | 13.23 | 4.73 | 1.24 | 1.44 | 5.82 |
| LLaVA-v1.5-13B [26] | 20.56 | 6.07 | 3.44 | 2.85 | 8.19 |
| Gemini-Pro [44] | 30.98 | 9.05 | 6.81 | 3.66 | 11.46 |
| GPT-4V [26] | 38.72 | 10.53 | 7.14 | 4.03 | 17.02 |
| Human Performance | 82.23 | 0.12 | 0.36 | 1.61 | 80.14 |

notators, and measure their performance on our predefined metrics (Table 4). We find that users generally exhibit a high level of proficiency when attempting most tasks for the first time. However, there are instances where users face difficulties in successfully completing certain tasks. These are due to factors including the user’s inability to fully comprehend the task, difficulties in grounding the task to the provided screenshot, or a lack of familiarity with the UI.

8 Conclusion and Future Work

Autonomous virtual agents offer the potential to automate routine tasks, benefiting users with limited technical expertise. To solve this task, we introduce **OmniACT**, a unique dataset of 9.8K human-labeled data points. **OmniACT** benchmarks autonomous agents across a range of tasks on web and desktop applications. LLM-based agents, like GPT-4, achieve a respectable action score of 11.6 on our dataset. However, **OmniACT** presents a challenge for the current state-of-the-art language and multimodal models. It provides a direction for future research on foundational multimodal models that seamlessly integrate language and visual understanding of computer screens and stands poised to drive the next wave of advancements in generalist autonomous agents offering omnipotent assistance to humans.

9 Limitations

This work introduces a valuable dataset, yet we recognize a few limitations that exist. State-of-the-art models like GPT-4, may exhibit susceptibility to hallucinations and bias towards specific data types, hindering broad applicability. Reliance on closed models like GPT-4V poses integration challenges due to high costs and time constraints. Despite efforts for equal representation and data collection without personal information, biases may be introduced as the dataset is exclusively in English, and human-curated content may have temporal biases.

10 Ethics Statement

As a part of the dataset creation process, we carefully review the pipeline at every stage, ensuring there is no personally identifiable information or offensive content, either during data collection or through the use of LLMs. For all purposes, we create dummy accounts that mimic real-like user content. To get the gold labels scripts we seek help from well-qualified student workers, approved through the institution, and get the bounding box data annotated through MTurk workers, both of whom are paid \$25 per hour, which is greater than the minimum wage rate (We detail this process in the supplementary material). Human studies are also done with the help of student workers approved by the institution at the above-mentioned payscale. We also ensure that all groups have equitable representation and that no personal opinions are reflected in the dataset, avoiding bias during the collection as well as the annotation process.

Acknowledgements

We extend our heartfelt gratitude to Writer.com for their generous and unwavering support throughout this project. Their dedicated team's expertise and collaboration were invaluable in achieving our goals.

References

1. Pyautogui: A cross-platform gui automation python module for human beings. <https://github.com/asweigart/pyautogui> (2023)
2. AlShikh, W., Daaboul, M., Goddard, K., Imel, B., Kamble, K., Kulkarni, P., Russak, M.: Becoming self-instruct: introducing early stopping criteria for minimal instruct tuning (2023)
3. Bai, C., Zang, X., Xu, Y., Sunkara, S., Rastogi, A., Chen, J., y Arcas, B.A.: Uibert: Learning generic multimodal representations for ui understanding (2021)
4. Banerjee, P., Mahajan, S., Arora, K., Baral, C., Riva, O.: Lexi: Self-supervised learning of the ui language (2023)
5. Burns, A., Arsan, D., Agrawal, S., Kumar, R., Saenko, K., Plummer, B.A.: Mobile app tasks with iterative feedback (motif): Addressing task feasibility in interactive visual environments
6. Chen, X., Zhao, Z., Chen, L., Ji, J., Zhang, D., Luo, A., Xiong, Y., Yu, K.: Websrc: A dataset for web-based structural reading comprehension. In: Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing. pp. 4173–4185 (2021)
7. Chiang, W.L., Li, Z., Lin, Z., Sheng, Y., Wu, Z., Zhang, H., Zheng, L., Zhuang, S., Zhuang, Y., Gonzalez, J.E., Stoica, I., Xing, E.P.: Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality (March 2023), <https://lmsys.org/blog/2023-03-30-vicuna/>
8. Deka, B., Huang, Z., Franzen, C., Hirschman, J., Afergan, D., Li, Y., Nichols, J., Kumar, R.: Rico: A mobile app dataset for building data-driven design applications. In: Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology. p. 845–854. UIST '17, Association for Computing Machinery, New York, NY, USA (2017). <https://doi.org/10.1145/3126594.3126651>, <https://doi.org/10.1145/3126594.3126651>
9. Deng, X., Gu, Y., Zheng, B., Chen, S., Stevens, S., Wang, B., Sun, H., Su, Y.: Mind2web: Towards a generalist agent for the web. arXiv preprint arXiv:2306.06070 (2023)
10. Dettmers, T., Pagnoni, A., Holtzman, A., Zettlemoyer, L.: Qlora: Efficient finetuning of quantized llms (2023)
11. Furuta, H., Nachum, O., Lee, K.H., Matsuo, Y., Gu, S.S., Gur, I.: Multimodal web navigation with instruction-finetuned foundation models. arXiv preprint arXiv:2305.11854 (2023)
12. Gupta, T., Kembhavi, A.: Visual programming: Compositional visual reasoning without training. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 14953–14962 (2023)
13. Gur, I., Furuta, H., Huang, A., Safdari, M., Matsuo, Y., Eck, D., Faust, A.: A real-world webagent with planning, long context understanding, and program synthesis (2024)

14. Gur, I., Rueckert, U., Faust, A., Hakkani-Tur, D.: Learning to navigate the web. In: International Conference on Learning Representations (2018)
15. He, Z., Sunkara, S., Zang, X., Xu, Y., Liu, L., Wichters, N., Schubiner, G., Lee, R., Chen, J., y Arcas, B.A.: Actionbert: Leveraging user actions for semantic understanding of user interfaces (2021)
16. Huang, Z., Zeng, Z., Liu, B., Fu, D., Fu, J.: Pixel-bert: Aligning image pixels with text by deep multi-modal transformers (2020)
17. Humphreys, P.C., Raposo, D., Pohlen, T., Thornton, G., Chhaparia, R., Muldal, A., Abramson, J., Georgiev, P., Santoro, A., Lillicrap, T.: A data-driven approach for learning to control computers. In: International Conference on Machine Learning. pp. 9466–9482. PMLR (2022)
18. Kim, G., Baldi, P., McAleer, S.: Language models can solve computer tasks. arXiv preprint arXiv:2303.17491 (2023)
19. Kirillov, A., Mintun, E., Ravi, N., Mao, H., Rolland, C., Gustafson, L., Xiao, T., Whitehead, S., Berg, A.C., Lo, W.Y., Dollár, P., Girshick, R.: Segment anything (2023)
20. Koh, J.Y., Lo, R., Jang, L., Duvvuri, V., Lim, M.C., Huang, P.Y., Neubig, G., Zhou, S., Salakhutdinov, R., Fried, D.: Visualwebarena: Evaluating multimodal agents on realistic visual web tasks (2024)
21. LeCun, Y.: A path towards autonomous machine intelligence version 0.9. 2, 2022-06-27 (2022)
22. Li, G., Li, Y.: Spotlight: Mobile ui understanding using vision-language models with a focus (2023)
23. Li, Y., He, J., Zhou, X., Zhang, Y., Baldridge, J.: Mapping natural language instructions to mobile ui action sequences (2020)
24. Li, Y., Li, G., He, L., Zheng, J., Li, H., Guan, Z.: Widget captioning: Generating natural language description for mobile user interface elements (2020)
25. Li, Y., Li, G., Zhou, X., Dehghani, M., Gritsenko, A.: Vut: Versatile ui transformer for multi-modal multi-task user interface modeling (2021)
26. Liu, H., Li, C., Li, Y., Lee, Y.J.: Improved baselines with visual instruction tuning (2023)
27. Liu, H., Li, C., Wu, Q., Lee, Y.J.: Visual instruction tuning (2023)
28. Liu, X., Yu, H., Zhang, H., Xu, Y., Lei, X., Lai, H., Gu, Y., Ding, H., Men, K., Yang, K., Zhang, S., Deng, X., Zeng, A., Du, Z., Zhang, C., Shen, S., Zhang, T., Su, Y., Sun, H., Huang, M., Dong, Y., Tang, J.: Agentbench: Evaluating llms as agents (2023)
29. Lu, P., Peng, B., Cheng, H., Galley, M., Chang, K.W., Wu, Y.N., Zhu, S.C., Gao, J.: Chameleon: Plug-and-play compositional reasoning with large language models (2023)
30. Lyu, C., Wu, M., Wang, L., Huang, X., Liu, B., Du, Z., Shi, S., Tu, Z.: Macaw-llm: Multi-modal language modeling with image, audio, video, and text integration. arXiv preprint arXiv:2306.09093 (2023)
31. Nakano, R., Hilton, J., Balaji, S., Wu, J., Ouyang, L., Kim, C., Hesse, C., Jain, S., Kosaraju, V., Saunders, W., et al.: Webgpt: Browser-assisted question-answering with human feedback
32. OpenAI: Gpt-4 technical report (2023)
33. OpenAI: Introducing chatgpt (2023), <https://openai.com/blog/chatgpt>
34. Papineni, K., Roukos, S., Ward, T., Zhu, W.J.: Bleu: a method for automatic evaluation of machine translation. In: Proceedings of the 40th annual meeting of the Association for Computational Linguistics. pp. 311–318 (2002)

35. Rawles, C., Li, A., Rodriguez, D., Riva, O., Lillicrap, T.: Android in the wild: A large-scale dataset for android device control (2023)
36. Ren, S., Guo, D., Lu, S., Zhou, L., Liu, S., Tang, D., Sundaresan, N., Zhou, M., Blanco, A., Ma, S.: Codebleu: a method for automatic evaluation of code synthesis. arXiv preprint arXiv:2009.10297 (2020)
37. Rozière, B., Gehring, J., Gloeckle, F., Sootla, S., Gat, I., Tan, X.E., Adi, Y., Liu, J., Remez, T., Rapin, J., Kozhevnikov, A., Evtimov, I., Bitton, J., Bhatt, M., Ferrer, C.C., Grattafiori, A., Xiong, W., Défossez, A., Copet, J., Azhar, F., Touvron, H., Martin, L., Usunier, N., Scialom, T., Synnaeve, G.: Code llama: Open foundation models for code (2023)
38. Rozière, B., Gehring, J., Gloeckle, F., Sootla, S., Gat, I., Tan, X.E., Adi, Y., Liu, J., Remez, T., Rapin, J., Kozhevnikov, A., Evtimov, I., Bitton, J., Bhatt, M., Ferrer, C.C., Grattafiori, A., Xiong, W., Défossez, A., Copet, J., Azhar, F., Touvron, H., Martin, L., Usunier, N., Scialom, T., Synnaeve, G.: Code llama: Open foundation models for code (2023)
39. Shaw, P., Joshi, M., Cohan, J., Berant, J., Pasupat, P., Hu, H., Khandelwal, U., Lee, K., Toutanova, K.: From pixels to ui actions: Learning to follow instructions via graphical user interfaces. arXiv preprint arXiv:2306.00245 (2023)
40. Shi, T., Karpathy, A., Fan, L., Hernandez, J., Liang, P.: World of bits: An open-domain platform for web-based agents. In: International Conference on Machine Learning. pp. 3135–3144. PMLR (2017)
41. Sridhar, A., Lo, R., Xu, F.F., Zhu, H., Zhou, S.: Hierarchical prompting assists large language model on web navigation. arXiv preprint arXiv:2305.14257 (2023)
42. Sun, L., Chen, X., Chen, L., Dai, T., Zhu, Z., Yu, K.: Meta-gui: Towards multi-modal conversational agents on mobile gui. In: Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing. pp. 6699–6712 (2022)
43. Surís, D., Menon, S., Vondrick, C.: Vipergpt: Visual inference via python execution for reasoning (2023)
44. Team, G., Anil, R., Borgeaud, S., Wu, Y., Alayrac, J.B., Yu, J., Soricut, R., Schalkwyk, J., Dai, A.M., Hauth, A., et al.: Gemini: a family of highly capable multimodal models. arXiv preprint arXiv:2312.11805 (2023)
45. team, W.E.: InstructPalmyra-30b : Instruct tuned Palmyra-Large model. <https://dev.writer.com> (2023)
46. team, W.E.: Palmyra-base Parameter Autoregressive Language Model. <https://dev.writer.com> (2023)
47. Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., Rodriguez, A., Joulin, A., Grave, E., Lample, G.: Llama: Open and efficient foundation language models (2023)
48. Wang, B., Li, G., Zhou, X., Chen, Z., Grossman, T., Li, Y.: Screen2words: Automatic mobile ui summarization with multimodal learning. In: The 34th Annual ACM Symposium on User Interface Software and Technology. pp. 498–510 (2021)
49. Wang, L., Ma, C., Feng, X., Zhang, Z., Yang, H., Zhang, J., Chen, Z., Tang, J., Chen, X., Lin, Y., et al.: A survey on large language model based autonomous agents. arXiv preprint arXiv:2308.11432 (2023)
50. Wang, W., Wei, F., Dong, L., Bao, H., Yang, N., Zhou, M.: Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. Advances in Neural Information Processing Systems **33**, 5776–5788 (2020)
51. Wang, X., Chen, G., Qian, G., Gao, P., Wei, X.Y., Wang, Y., Tian, Y., Gao, W.: Large-scale multi-modal pre-trained models: A comprehensive survey (2023)

52. Wang, Z., Bovik, A.C., Sheikh, H.R., Simoncelli, E.P.: Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing* **13**(4), 600–612 (2004)
53. Xu, N., Masling, S., Du, M., Campagna, G., Heck, L., Landay, J., Lam, M.: Grounding open-domain instructions to automate web support tasks. In: Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. pp. 1022–1032 (2021)
54. Yang, J., Zhang, H., Li, F., Zou, X., Li, C., Gao, J.: Set-of-mark prompting unleashes extraordinary visual grounding in gpt-4v (2023)
55. Yang, Z., Li, L., Lin, K., Wang, J., Lin, C.C., Liu, Z., Wang, L.: The dawn of lmms: Preliminary explorations with gpt-4v (ision). *arXiv preprint arXiv:2309.17421* **9** (2023)
56. Yao, S., Chen, H., Yang, J., Narasimhan, K.: Webshop: Towards scalable real-world web interaction with grounded language agents. *Advances in Neural Information Processing Systems* **35**, 20744–20757 (2022)
57. Yin, S., Fu, C., Zhao, S., Li, K., Sun, X., Xu, T., Chen, E.: A survey on multimodal large language models (2023)
58. Zhang, T., Kishore, V., Wu, F., Weinberger, K.Q., Artzi, Y.: Bertscore: Evaluating text generation with bert (2020)
59. Zhang, Z., Zhang, A.: You only look at screens: Multimodal chain-of-action agents (2023)
60. Zhao, W.X., Zhou, K., Li, J., Tang, T., Wang, X., Hou, Y., Min, Y., Zhang, B., Zhang, J., Dong, Z., et al.: A survey of large language models. *arXiv preprint arXiv:2303.18223* (2023)
61. Zhou, S., Alon, U., Agarwal, S., Neubig, G.: Codebertscore: Evaluating code generation with pretrained models of code. *arXiv preprint arXiv:2302.05527* (2023)
62. Zhou, S., Xu, F.F., Zhu, H., Zhou, X., Lo, R., Sridhar, A., Cheng, X., Bisk, Y., Fried, D., Alon, U., et al.: Webarena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854* (2023)