Plan-Point-Act: o3-Molmo Powered Web Agent

Long Cheng
University of Washington
lcheng97@uw.edu

Federico Baldan
University of Washington
fbaldan@uw.edu

Abstract

This project develops a practical web automation agent that combines high-level task planning using OpenAI's o3 model, enhanced content analysis through GPT-4.1 integration, and precise spatial interaction enabled by AI2's open-source visual grounding model, Molmo-7B-D-0924. Our multi-model architecture leverages GPT-4.1 for sophisticated task understanding and page content analysis, the o3 model for strategic action generation, and Molmo for accurate visual element localization, creating a robust Chrome extension capable of performing complex web tasks based solely on natural language commands. Rather than training end-to-end models, our approach demonstrates how multiple specialized AI models can be effectively orchestrated into a user-friendly automation tool with comprehensive error handling, retry mechanisms, and real-time execution monitoring, with all implementation code released publicly to promote transparency and reproducibility within the research community.

1. Introduction

In this project, we aim to demonstrate that by leveraging a combination of state-of-the-art language models and open-source visual grounding models, we can build a functional, general-purpose web agent system. The system is capable of executing complex browsing tasks in an end-to-end fashion through a custom Chrome extension that bridges high-level planning with fine-grained spatial interaction.

The goal is to build a web automation agent that combines the reasoning capabilities of multiple large language models with the visual grounding strengths of specialized multimodal models. Our system integrates a multi-model architecture featuring OpenAI's o3 model for action generation and task orchestration, GPT-4.1 for enhanced page analysis and task understanding, and Molmo-7B-D-0924 [1] for visual grounding and spatial localization. This hybrid approach allows us to leverage the complementary strengths of each model: sophisticated reasoning for complex task decomposition, deep content understanding for contextual awareness, and precise visual understanding for accurate element identification.

Our enhanced agentic workflow proceeds as follows:

1. The user inputs a natural language query through the Chrome extension popup.
2. GPT-4.1 performs multi-layered analysis:
    - Task requirement analysis to understand the user's intent and success criteria
    - Content analysis to identify relevant page elements and context
    - Structural analysis to understand the page hierarchy and interaction patterns
3. The enhanced context is passed to OpenAI's o3 model, which uses a carefully crafted system prompt to generate a sequence of structured web actions in JSON format, informed by the GPT-4.1 analysis.
4. The system parses the generated actions and determines the appropriate execution method:
    - For elements with known CSS selectors, direct DOM manipulation is performed
    - For elements identified by natural language descriptions (e.g., "first video", "search button"), the system captures a screenshot and uses Molmo-7B-D-0924 API to obtain precise click coordinates with automatic scrolling capabilities
5. Using Chrome Extension APIs, the agent simulates interactions with the browser interface through script injection and coordinate-based clicking.
6. After action execution, the system performs comprehensive completion analysis:
    - OpenAI's o3 analyzes the updated page state to determine task completion with confidence scoring and evidence gathering
    - If o3 indicates task completion, the process terminates with a success summary

- If additional actions are needed, o3 provides suggested next steps
- If the analysis is unclear, additional actions are generated and executed recursively (with depth limiting to prevent infinite loops)
- If actions fail, the system includes retry mechanisms with exponential backoff and automatic scrolling for visual element detection

Our contribution lies in creating a practical web automation system that effectively combines multiple advanced language models with specialized visual grounding capabilities. The multi-model architecture provides robust task understanding, enhanced page comprehension, and reliable action execution. The system supports both local deployment (via SSH tunnel to Hyak-hosted Molmo service) and official API endpoints, providing flexibility for different deployment scenarios. While the system incorporates both proprietary (OpenAI o3, GPT-4.1) and open-source (Molmo) components, our focus is on demonstrating how these can be integrated into a robust, user-friendly browser automation tool that operates asynchronously with intelligent task completion detection.

The system is designed with practical considerations in mind, including comprehensive error handling, task timeout mechanisms, intelligent retry strategies, and transparent logging to help users understand and troubleshoot the automation process. The enhanced page analysis capabilities enable more accurate task completion detection and reduce false positives in automation workflows. This approach enables complex web tasks to be completed through simple natural language instructions while maintaining reliability, accuracy, and user control.

2. Related Work

Recent advancements in multimodal large language models (MLLMs) have significantly improved performance in vision-language reasoning. GPT-4V, for instance, demonstrates capabilities in complex tasks such as story generation and solving math problems without OCR reliance [5]. A common design pattern among these models is the use of encoder-decoder frameworks that fuse visual and linguistic information to strengthen cross-modal understanding. MiniGPT-4 leverages Q-Formers to align visual features extracted from ViT with language models [4]. Molmo addresses 2D spatial grounding by directly predicting normalized coordinates, achieving 92% accuracy in icon localization tasks [1]. RoboPoint [6] applies instruction tuning on synthetic datasets to enhance affordance prediction for robotics, outperforming GPT-4o and PIVOT [3] by more than 20% in spatial precision. Models like VisCPM and Qwen-VL introduce additional features such as region-specific prompting and multilingual capabilities, while NExT-GPT broadens input modalities to include 3D data, audio, and video [5]. Despite these innovations, accurately resolving fine-grained spatial references remains a core challenge, prompting further investigation into the mechanisms that enable effective space reasoning in MLLMs.

3. Methods

Instead of employing virtual machines via Linux, we developed a Chrome extension, as we believe this approach is more practical for real-world applications. This represents a key innovation in our work: users can interact directly with our agent through the Chrome extension, enhancing accessibility and ease of use.

3.1. System Architecture

We employ a hybrid multi-model approach that leverages the complementary strengths of different AI systems for comprehensive web automation. Our architecture combines OpenAI's o3 model for high-level action planning, GPT-4.1 for enhanced page analysis and task understanding, and Molmo-7B-D-0924 for fine-grained spatial decision-making. This multi-stage pipeline is operationalized through a custom Chrome extension capable of automating complex web tasks through simulated human interactions. The system architecture is shown in Figure 1, and a screenshot of the extension is provided in Figure 2.

The system implements a four-stage processing pipeline that significantly enhances task execution reliability and accuracy. Upon receiving user input, the system first employs GPT-4.1 to perform comprehensive task requirement analysis, identifying task type, constraints, success criteria, and potential challenges. Concurrently, the system extracts enhanced page content including semantic structure, interactive elements, navigation patterns, and content hierarchy. This detailed page analysis is then processed by GPT-4.1 to generate contextual insights about the current page state and its relevance to the user's task.

The enhanced context from both task analysis and page understanding is then provided to the o3 model, which generates structured action plans as JSON arrays. Each action specifies the operation type (click, type, navigate, wait, scroll, extract) and target parameters. For spatial interactions, the system supports both CSS selector-based clicking for elements with known selectors and visual grounding-based clicking using natural language descriptions processed by Molmo-7B-D-0924.

Our system supports dual Molmo API endpoints: a local implementation deployed on Hyak computing infrastructure accessed via SSH tunnel, and AI2's official hosted Molmo service. Users can dynamically configure endpoint

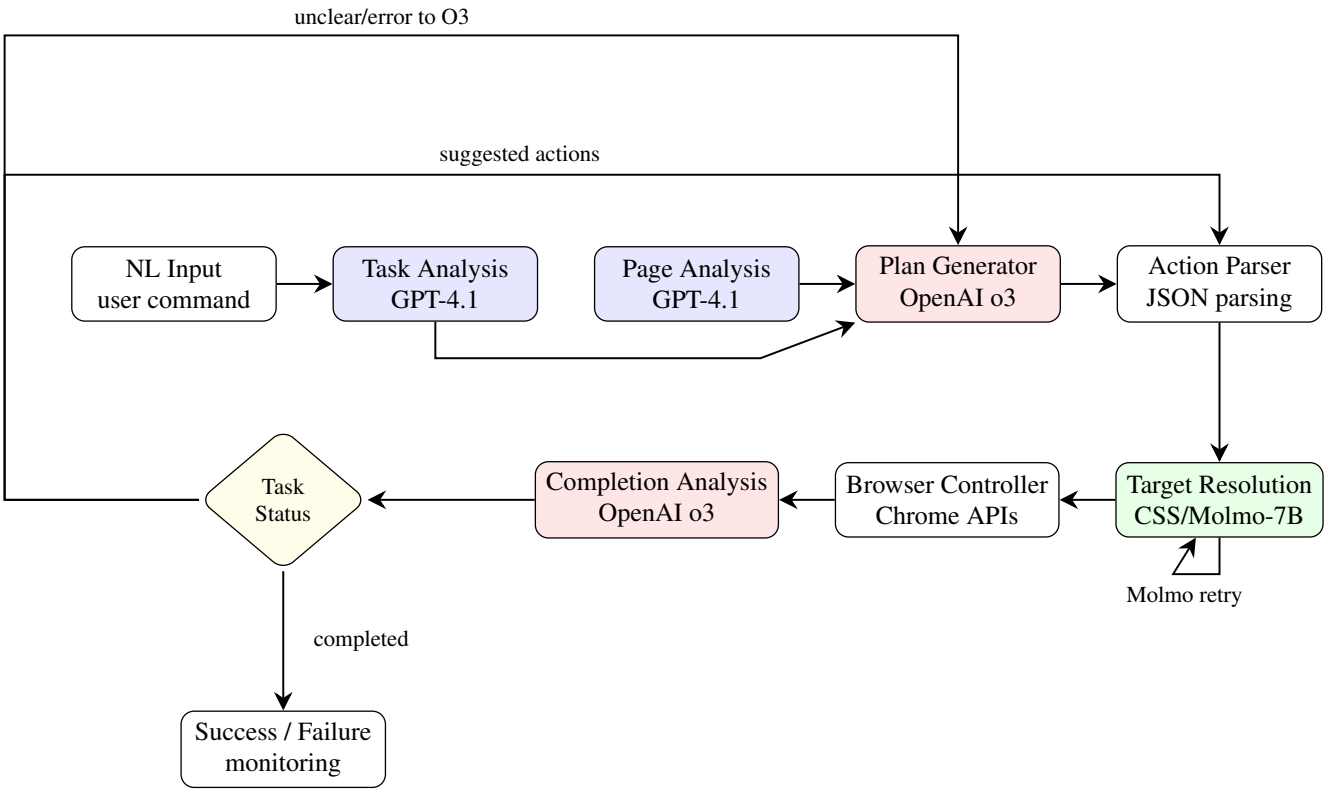

Figure 1. Execution Workflow of the Web Automation Agent

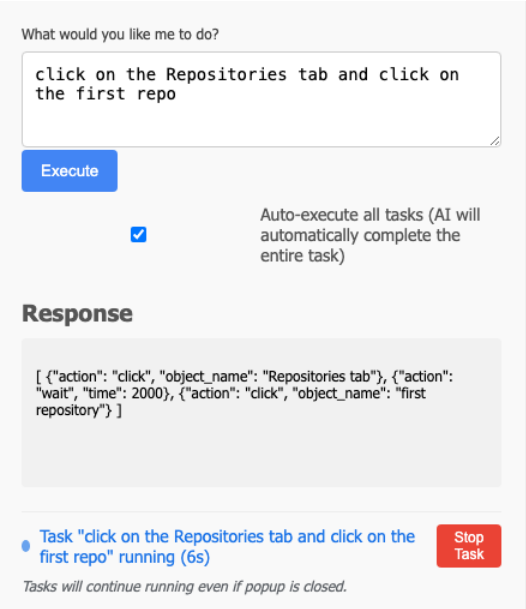

Figure 2. Screenshot of the extension executing a task on GitHub

selection through the extension interface, with configuration persistence across browser sessions.

3.2. Content Analysis Pipeline

A key innovation in our approach is the integration of GPT-4.1 for sophisticated page content analysis and task completion assessment. The system performs multi-layered content extraction that goes beyond basic text scraping to capture:

- **Semantic Structure**: HTML5 semantic elements, heading hierarchy, navigation patterns, and content organization
- **Interactive Elements**: Comprehensive cataloging of buttons, forms, input fields, and clickable elements with associated metadata
- **Content Relationships**: Understanding of content flow, data tables, lists, and structured information patterns
- **Dynamic State**: Real-time analysis of alerts, error messages, and page state changes

This enhanced content representation enables GPT-4.1 to provide both task understanding analysis and content extraction analysis.

3.3. Execution Workflow

The execution workflow implements an intelligent multi-stage process that adapts to task complexity and page dynamics. Initial command processing begins with parallel execution of task analysis and enhanced page content extraction. The GPT-4.1 task analyzer categorizes the user command, identifies key requirements, and establishes success criteria, while the content extraction system captures comprehensive page state information.

This enriched context is provided to the o3 model through an enhanced system prompt that incorporates both task insights and page analysis. The o3 model generates action sequences with improved precision due to the contextual understanding provided by the preliminary analysis stages.

For visual grounding operations, the system implements an intelligent target location mechanism. When processing `"object_name"` specifications, the system captures high-resolution screenshots and queries Molmo-7B-D-0924 with formatted pointing instructions. If initial target detection fails, the system automatically implements a multi-step scrolling search strategy, scrolling down in 300-pixel increments up to 20 times, then scrolling back up to check previously viewed areas. This approach significantly improves element detection rates for complex page layouts.

The system incorporates advanced JSON processing capabilities to handle model response variability. A comprehensive JSON validation and repair system automatically corrects common formatting issues including incomplete URLs, malformed strings, trailing commas, and incomplete object structures. This robust parsing augmented by GPT-4.1 ensures reliable action execution even when model outputs contain minor formatting inconsistencies.

3.4. Intelligent Task Continuation

After executing each action sequence, the system performs a comprehensive completion analysis by combining insights from the o3 model with a secondary planning phase. This analysis evaluates the current page state against the original task requirements, producing structured outputs such as task completion confidence, supporting evidence, and recommended next steps.

If the analysis indicates that the task is incomplete but provides clear instructions, those follow-up actions are executed directly. If the feedback is ambiguous or contains errors, the system re-engages the o3 model for a fresh round of action planning—mirroring the initial planning phase. To balance task thoroughness with efficiency, this continuation process includes a depth limit of 10 iterations and intelligent stopping criteria to prevent infinite loops while supporting complex, multi-step task execution.

The system maintains comprehensive execution state across all stages, including conversation history, task analysis results, page content snapshots, and execution logs. This persistent state enables robust error recovery and provides detailed execution transparency for debugging and optimization purposes.

Task execution operates asynchronously with a 5-minute global timeout, allowing users to close the interface while maintaining background operation. Real-time status monitoring and user-initiated task termination provide flexible control over execution flow, while comprehensive error handling addresses API timeouts, navigation restrictions, and element interaction failures.

4. Experiments

Our web browsing tasks vary significantly in nature—ranging from navigation to description or computation, and often involve a combination of these elements. As a result, many tasks do not yield binary outcomes (i.e., simple yes or no answers). Instead, the agent may only partially complete a given task. Based on this observation, we categorize outcomes into three classes: success, partial success, and failure. For the same reason, establishing definitive ground truths is challenging. Therefore, we currently rely on human evaluation to assess performance.

4.1. Experiment Methods

We evaluate our agent using the WebVoyager dataset [2], which comprises 643 task queries across 15 websites, each containing over 40 queries. For our experiments, we selected 300 tasks from 11 of these websites. Given that the tasks are significantly more complex than binary (yes/no) questions, and that reference answers can be time-sensitive—even after excluding two websites known for their inherent time sensitivity—we opted for human evaluation. This evaluation was based on the agent's automatic navigation actions and log messages, as some tasks involve computation or textual interpretation rather than direct navigation. We report both the overall accuracy and the per-website accuracy across the 11 selected websites.

In most cases, the system effectively decomposes complex tasks into logical sequences of actions and executes them accurately. For navigation tasks, the agent can determine whether visual grounding is necessary or whether using the search bar is sufficient, subsequently navigating to the correct page. For calculation or description tasks, the agent outputs the result directly to the console. However, in scenarios requiring special permissions, the agent may struggle or fail to complete the task due to limitations such as lack of access rights or missing authorization.

4.2. Experiment Results

As shown in Table 1, there is a marked contrast in performance across different websites. Platforms such as Apple and Wolfram Alpha attain high validation accuracy due to

Website	Total	Success	Acc. (%)	Partial Successes Rate (%)
Overall	300	154	51.3%	24.0%
Allrecipes	13	7	53.8%	15.4%
Apple	42	39	92.0%	0.0%
ArXiv	26	14	53.8%	26.9%
BBC News	21	2	9.5%	76.2%
Cambridge Dictionary	42	22	52.4%	0.0%
GitHub	32	10	31.3%	34.4%
ESPN	31	8	25.8%	32.3%
Google Maps	23	13	56.5%	13.0%
Google Search	15	2	13.3%	80.0%
Hugging Face	26	15	57.7%	34.6%
Wolfram Alpha	29	22	75.9%	6.9%

Table 1. Agent Evaluation Results

their use of stable, semantically meaningful CSS selectors that expose every interactive element. Our agent is able to interact with these elements and extract relevant data exclusively through direct DOM APIs, thereby avoiding the need for extensive visual grounding procedures. In such cases, the console interface reports latencies below 100 ms, coupled with high reliability.

In contrast, highly dynamic websites such as BBC News, ESPN, and Google Search exhibit significantly lower accuracy, often accompanied by a higher rate of partial successes, indicating that tasks were only partially completed. This degradation is primarily due to these platforms concealing their user interfaces within client-side frameworks, employing scroll feeds, and integrating custom widgets. In the absence of stable selectors, our agent defaults to the Molmo visual grounding pipeline, which involves screenshot capture, remote inference, coordinate mapping, and multiple scroll-and-retry loops. This process introduces an additional overhead of 200–800 ms per step and creates multiple points of failure, thereby reducing overall task accuracy.

A secondary yet substantial source of partial successes (accounting for approximately 24% of total failures) stems from permission and context-related errors inherent to the current system architecture. The agent operates through a multi-stage execution loop involving navigation, DOM-based interaction, and visual grounding. For visual clicks, the agent initiates a screenshot capture followed by coordinate inference using the Molmo pipeline. However, this step is aborted if the target domain lacks the required `activeTab` or `<all_urls>` permissions, resulting in errors such as:

```
Error with visual click action:
Screenshot capture failed:
Either the '<all_urls>' or
'activeTab' permission is
required.
```

This error typically arises when the task plan generated by the o3 model includes navigation across different domains (e.g., a task involving ArXiv that redirects to Cornell.edu) without proper permissions. Under these conditions, the agent is unable to load dynamic scripts or capture screenshots, leading to unanticipated execution failures. Furthermore, task outcomes may vary depending on the type of user account employed. Switching between personal and work accounts yielded inconsistent results, influenced by account-specific settings and permission grants.

Another complication arises from websites that employ client-side routing to dynamically update content and modify the URL without performing a full page reload. In such instances, the `url` variable may become stale or undefined, leading to runtime errors such as:

```
Error analyzing page:
TypeError: Cannot read
properties of undefined (reading
'url')
```

This frequently results in task failure or ambiguous classification.

To mitigate this, we subscribe to Chrome's `webNavigation.onHistoryStateUpdated` event and re-sample `document.location.href` at the start of each iteration phase that includes content analysis. Nonetheless, this strategy fails when web applications utilize `history.replaceState`—which does not trigger the event—or when navigation occurs within a cross-origin `<iframe>`. In such scenarios, the iframe's internal URL changes without modifying the browser's address bar, leaving our extension unaware of the update. Consequently, the agent's contextual understanding of the task becomes outdated, and the task is ultimately marked as partially successful.

Another critical failure mode involves malformed JSON responses generated by the o3 model. Our agent employs a two-stage repair pipeline: the first stage normalizes superficial issues such as trailing commas, while the second stage invokes GPT-4.1 to resolve deeper syntax errors. Nevertheless, certain responses continue to produce failures, as illustrated by the following error:

```
Advanced JSON repair also
failed: SyntaxError:
Unterminated string in JSON at
position 128
```

Such cases prevent the JSON parser from generating a valid action sequence, thereby halting the execution loop entirely. These errors are often linked to complex, recursive plans where small formatting inconsistencies accumulate and propagate.

The agent employs a dual-strategy mechanism for interface interaction, combining DOM-based extraction and Molmo-7B visual grounding. When a reliable selector is

available, the agent utilizes direct DOM API interactions with high precision. In the absence of such selectors, the agent captures a screenshot and uses Molmo-7B to predict the pixel coordinates of the target element. These coordinates are then denormalized to viewport space, followed by a scroll-and-retry mechanism to bring the element into view. However, many modern websites load content incrementally as the user scrolls, encapsulate UI elements within custom Web Components (thus obfuscating their internal HTML), or dynamically reposition elements. In these instances, the visual grounding strategy often fails, leading Molmo to generate incorrect coordinates and causing the task to be classified as partially successful.

It is important to emphasize that all 300 tasks were manually evaluated by reviewing console outputs and the corresponding page states, assessed on an iteration-by-iteration basis. This rigorous evaluation methodology led to the introduction of a "partial success" category, which captures ambiguous cases arising from subjective thresholds. As discussed, discrepancies between UI behavior and backend logs—due to permission errors, contextual misalignment, or JSON parsing issues—may result in tasks appearing complete on the frontend while being internally marked as partially successful.

5. Discussion

Our web browser agent is capable of handling multi-step tasks, such as "navigate to the repositories page and open the first repository" in the context of GitHub. However, it encounters significant limitations when attempting more complex actions, such as shopping cart operations or accessing websites that require user authentication. These constraints highlight a broader challenge: current web browser architectures are not designed with AI agents in mind. To unlock a wider range of use cases, we believe it is necessary to fundamentally rethink the design of web browsers—specifically, to grant agents greater permissions and integration capabilities. Without such systemic changes, AI agents will remain restricted to a narrow set of tasks.

Moreover, from a user perspective, there is understandable hesitation around delegating high-stakes actions—such as changing account passwords or making purchases on platforms like Amazon—to autonomous agents. Building trust will require substantial improvements in model reliability and a dramatic reduction in hallucinations. Ensuring consistent, verifiable task execution is essential for users to feel confident in allowing agents to act on their behalf.

Future improvements to our agent will focus primarily on enhancing robustness, particularly by addressing the frequent parsing errors stemming from malformed JSON generated by large language models (LLMs). To further refine performance, we plan to explore integrating additional vision and reasoning models optimized for specific web tasks, potentially involving custom-trained vision models based on collected website screenshot datasets. Moreover, combining advanced visual grounding techniques with traditional XML parsing could significantly boost execution accuracy. Lastly, extending the agent's capabilities to handle complex real-world scenarios—such as reservations on Booking.com or flight bookings on airline websites, to make more practical the usage of the web agent.

6. Acknowledgments

This project is completed as a requirement for Ranjay Krishna's course *Deep Learning for Computer Vision*, with mentorship from Zixian Ma and Ranjay Krishna. The project is conducted in collaboration with the Allen Institute for Artificial Intelligence (AI2), with support in model access and infrastructure.

References

[1] Matt Deitke, Christopher Clark, Sangho Lee, Rohun Tripathi, Yue Yang, Jae Sung Park, Mohammadreza Salehi, Niklas Muennighoff, Kyle Lo, Luca Soldaini, Jiasen Lu, Taira Anderson, Erin Bransom, Kiana Ehsani, Huong Ngo, Yen-Sung Chen, Ajay Patel, Mark Yatskar, Chris Callison-Burch, Andrew Head, Rose Hendrix, Favyen Bastani, Eli VanderBilt, Nathan Lambert, Yvonne Chou, Arnavi Chheda, Jenna Sparks, Sam Skjonsberg, Michael Schmitz, Aaron Sarnat, Byron Bischoff, Pete Walsh, Chris Newell, Piper Wolters, Tanmay Gupta, Kuo-Hao Zeng, Jon Borchardt, Dirk Groeneveld, Crystal Nam, Sophie Lebrecht, Caitlin Wittlif, Carissa Schoenick, Oscar Michel, Ranjay Krishna, Luca Weihs, Noah A. Smith, Hannaneh Hajishirzi, Ross Girshick, Ali Farhadi, and Aniruddha Kembhavi. Molmo and pixmo: Open weights and open data for state-of-the-art vision-language models, 2024. 1, 2

[2] Hongliang He, Wenlin Yao, Kaixin Ma, Wenhao Yu, Yong Dai, Hongming Zhang, Zhenzhong Lan, and Dong Yu. Webvoyager: Building an end-to-end web agent with large multimodal models. *arXiv preprint arXiv:2401.13919*, 2024. 4

[3] Soroush Nasiriany, Fei Xia, Wenhao Yu, Ted Xiao, Jacky Liang, Ishita Dasgupta, Annie Xie, Danny Driess, Ayzaan Wahid, Zhuo Xu, Quan Vuong, Tingnan Zhang, Tsang-Wei Edward Lee, Kuang-Huei Lee, Peng Xu, Sean Kirmani, Yuke Zhu, Andy Zeng, Karol Hausman, Nicolas Heess, Chelsea Finn, Sergey Levine, and Brian Ichter. Pivot: Iterative visual prompting elicits actionable knowledge for vlms, 2024. 2

[4] Jiaqi Wang, Hanqi Jiang, Yiheng Liu, Chong Ma, Xu Zhang, Yi Pan, Mengyuan Liu, Peiran Gu, Sichen Xia, Wenjun Li, Yutong Zhang, Zihao Wu, Zhengliang Liu, Tianyang Zhong, Bao Ge, Tuo Zhang, Ning Qiang, Xintao Hu, Xi Jiang, Xin Zhang, Wei Zhang, Dinggang Shen, Tianming Liu, and Shu Zhang. A comprehensive review of multimodal large language models: Performance and challenges across different tasks, 2024. 2

[5] Shukang Yin, Chaoyou Fu, Sirui Zhao, Ke Li, Xing Sun, Tong Xu, and Enhong Chen. A survey on multimodal large language models. *National Science Review*, 11(12):nwae403, 11 2024. 2

[6] Wentao Yuan, Jiafei Duan, Valts Blukis, Wilbert Pumacay, Ranjay Krishna, Adithyavairavan Murali, Arsalan Mousavian, and Dieter Fox. Robopoint: A vision-language model for spatial affordance prediction for robotics. *arXiv preprint arXiv:2406.10721*, 2024. 2