Multimodal LLM Agents: Exploring LLM interactions in Software, Web and Operating Systems

Allen Thomas *allent3@illinois.edu*
allent3

Kartik Ramesh *kartikr2@illinois.edu*
kartikr2

Shraddhaa Mohan *sm148@illinois.edu*
sm148

Abstract

The digital world operates through multimodal interactions, yet current LLM agents remain constrained by approaches that convert visual, auditory, and system-level data into lossy textual proxies. This approach introduces noise and limits the ability of agents to leverage holistic context to make decisions in digital environments. Although recent advancements in multimodal models like Flamingo and GPT-4 Vision demonstrate impressive capabilities in vision-language tasks, their potential as agents capable of decision-making and task execution based on multimodal inputs remains underexplored. In this survey we explore the design, evaluation, and capabilities of multimodal LLM agents, with a focus on interactions in software environments such as web browsers and operating system interfaces. We analyze recent advancements in multimodal integration within agentic systems, investigate multimodal tool orchestration frameworks and explore interactive agents that integrate human feedback to guide decision making. Our work seeks to highlight the potential of multimodal agents in developing autonomous applications that navigate the digital world like humans.

1 Introduction

The digital world is inherently multimodal—combining text, images, audio, and system-level interactions. Yet, current Large Language Model (LLM) agents predominantly operate in a text-first paradigm, translating non-textual modalities into lossy textual representations. As AI systems move toward real-world deployment—especially in domains like software development, operating systems, and the web—agents must be able to reason over and act within multimodal environments [1; 2; 3].

Recent advances in models like Flamingo [4] and GPT-4V [5] have demonstrated strong performance on perception-based vision-language tasks. However, these systems are rarely designed or evaluated as autonomous agents capable of navigating and manipulating digital environments end-to-end. Most existing work focuses on perception, with limited attention to actionable interaction, tool use, or temporal reasoning [6; 7; 8; 9].

For agents to be effective in digital ecosystems, they must integrate diverse modalities in a seamless, context-aware way—mirroring how humans naturally combine language, vision, and interaction. For instance, software engineering tasks require tight alignment between natural language and code [2; 10; 11], while operating systems and web environments demand agents that can perceive screens and take actions like clicking and typing [12; 13; 14; 15]. Advancing such capabilities will unlock robust human-AI collaboration across digital platforms [16; 17; 18].

This survey explores the development of multimodal LLM agents with a focus on digital domains—including code editors, browsers, and operating systems. We examine how these agents are built, what capabilities they possess, and how their performance is evaluated. Our goal is to surface the engineering challenges and research opportunities involved in creating truly agentic systems—those that perceive, decide, plan, and act effectively across diverse digital modalities [1; 3; 19].

2 Related Work

Recent advances in multimodal systems such as Flamingo [4] and GPT-4 Vision[5] have explored integrating diverse inputs, such as text, images, and system-level data, to enable richer task understanding and execution.

Advancements in embodied agents have demonstrated significant progress in integrating multimodal capabilities with real-world tasks. For instance, EMMA[6] focuses on visual-world alignment, enhancing how agents perceive and interact with their environment. In software-related domains, tool-augmented systems have shown particular promise. Toolformer[7] enables language models to autonomously invoke external APIs, highlighting the potential for similar multimodal integrations. Similarly, Magentic-One[8] explores multi-agent collaboration for software engineering tasks, where specialized agents work together to improve debugging and development efficiency. Another promising direction uses human feedback to enhance multimodal systems. For example, MuLan[9], uses task decomposition and iterative loops to refine text-to-image synthesis.

Research in Graphical User Interface (GUI) agents aims to interact with digital environments like humans do. Auto-GUI[12] introduces a multimodal solution that directly interacts with the GUI using vision and language inputs, bypassing the need for intermediate parsing or application-specific APIs. Windows Agent Arena [13] introduces Navi, a multimodal agent that navigates complex tasks in the Windows OS. Web browser agents such as WebVoyager[14] leverage multimodal models to navigate real-world websites through actions like clicking and typing. Similarly, Browser Use[20] provides an open-source framework that dynamically integrates AI agents with web browsers.

While prior surveys on multimodal agents[1] offer broad insights, they often overlook the challenges and requirements of software environments. Our work aims to address this gap by focusing on multimodal agents for software engineering.

3 Multi-modal Models

The landscape of artificial intelligence is undergoing a significant transformation, moving beyond systems confined to single modalities like text or vision. While Large Language Models (LLMs) have demonstrated remarkable proficiency in processing and generating text, their inherent inability to perceive or interact with non-textual data limits their applicability in complex, real-world scenarios. Concurrently, specialized models excel at tasks like image recognition or speech processing but often lack the broader reasoning and contextual understanding capabilities of LLMs.

This limitation presents a critical software engineering challenge: designing systems that can holistically understand and act upon the rich, multimodal information characteristic of human environments. The integration of vision, audio, and language processing capabilities is an engineering necessity for building more capable and versatile AI systems [1].

The motivation for developing such integrated systems stems directly from the requirements of numerous practical applications. Current AI systems often fall short, operating primarily in unimodal silos or requiring complex, brittle pipelines to connect different modalities. The emergence of Multimodal Large Language Models (MLLMs) offers a promising architectural paradigm to address this gap. These systems aim to leverage the powerful reasoning and language capabilities of LLMs as a central processing core, augmented with specialized encoders for visual and auditory perception [2; 21].

A key evolution in this domain, and the central focus of this survey, is the shift from purely perceptual multimodal models towards agentic MLLMs.

While early multimodal research successfully developed models like CLIP or DALL-E for image-text tasks, and Whisper for speech recognition, these often perform specific, predefined tasks [5]. The engineering frontier lies in creating systems that not only understand multimodal inputs but can autonomously reason, plan, and execute tasks based on this understanding.

Models like DeepMind's Flamingo [4], which demonstrated few-shot learning on vision-language tasks by integrating visual features into a frozen LLM, and OpenAI's GPT-4 with Vision (GPT-4V) [5], which accepts image inputs for complex problem-solving, represent significant milestones in MLLM development. However, engineering these models to function as autonomous agents – systems capable of making decisions and taking actions across multimodal streams – requires tackling specific architectural and integration challenges [3].

This survey focuses on the software engineering aspects of building these agentic MLLMs, specifically those integrating LLMs with vision and/or audio capabilities. We examine the core system design choices, including architectural patterns for modality fusion – how visual and auditory information is technically integrated with the language model's processing flow. Key fusion techniques range from early fusion strategies, where multimodal inputs are tokenized and concatenated at the input layer (as seen in models like Kosmos-1 or LLaVA), enabling the LLM to process modalities uniformly, to intermediate fusion, where dedicated cross-attention layers or adapter modules allow deeper interaction between modality features and the LLM's hidden states (exemplified by Flamingo [4]).

We also consider late fusion or modular approaches, where an LLM acts as an orchestrator, invoking specialized unimodal models or tools as needed—demonstrated by systems like HuggingGPT and Visual ChatGPT [8]. Understanding the trade-offs between these architectural choices—simplicity versus integration depth, monolithic design versus modularity—is crucial for engineering effective multimodal agents.

The practical implications of successfully engineering agentic MLLMs are vast. These systems are enabling a new generation of applications where AI can interact more naturally and effectively within multimodal contexts.

In vision-language domains, this includes advanced image captioning that incorporates commonsense reasoning, sophisticated Visual Question Answering (VQA) systems capable of complex dialogue about visual content, and interactive multimodal assistants that can understand and even manipulate images based on user instructions [14; 15]. Extending these capabilities to video analysis allows for automated summarization, activity recognition, and dialogue grounded in temporal visual events.

Similarly, integrating robust audio processing, particularly speech recognition like that provided by Whisper, forms the backbone of intelligent voice assistants that combine listening comprehension with the conversational prowess of LLMs, leading to richer human-computer interaction.

Perhaps the most compelling applications lie in embodied AI and complex decision-making environments. Systems like Google's PaLM-E demonstrate the potential of using MLLMs to interpret sensor data (vision, state) and generate robotic action plans, bridging the gap between high-level language instructions and low-level physical execution. This represents a significant software engineering feat, integrating perception, reasoning, and control within a single, albeit large, model architecture [16].

Similar principles apply to developing intelligent agents for autonomous driving, virtual environments (e.g., game-playing agents like Gato), and digital assistants capable of navigating complex graphical user interfaces [22; 17]. Specialized domains like healthcare also stand to benefit, with MLLMs potentially assisting clinicians by analyzing medical images in conjunction with textual patient data, although the engineering demands for safety, reliability, and accuracy in such critical applications are paramount.

Despite the rapid progress, significant software engineering challenges remain in building and deploying robust agentic MLLMs. Integrating multiple large models introduces considerable system complexity and computational cost.

Scalability is a major concern, particularly regarding the handling of long-context inputs like extended videos or large documents containing numerous images, which often exceeds the processing limits of current architectures. Ensuring robust and reliable reasoning across modalities is another hurdle; models can still struggle with complex multi-step instructions, exhibit hallucinations, or fail to properly handle ambiguity in perceptual input [18]. Addressing these issues requires innovations in model architecture, training data curation (including mitigating inherited biases), and evaluation methodologies suitable for complex, interactive tasks [23].

Furthermore, engineering these systems for real-world deployment necessitates tackling issues of latency, robustness to noisy or missing data, and safety. Ensuring that multimodal agents behave predictably, are resilient to adversarial manipulations across modalities, and align with human values is a critical ongoing research and engineering effort. The development of standardized benchmarks and evaluation protocols specifically designed for agentic multimodal tasks is also essential for measuring progress and ensuring reproducibility [24; 25].

Looking ahead, future software engineering efforts will likely focus on developing more sophisticated frameworks for multimodal reasoning and planning, potentially integrating symbolic methods or explicit memory modules. Designing more efficient and adaptive modality fusion mechanisms, extending systems to incorporate a wider range of sensors (e.g., tactile, 3D), and enabling continual learning in interactive settings are key research directions. Enhancing human-agent collaboration through better multimodal communication and explanation capabilities will also be crucial. This survey aims to provide a structured overview of the current state-of-the-art from a software engineering perspective, detailing the architectures, applications, and inherent challenges, thereby outlining the path towards more capable, reliable, and versatile multimodal AI agents.

4 Applications of Multi-modal LLM Agents

4.1 Software Agents

Large Language Models (LLMs) have shown strong capabilities in generating, editing, and reasoning about source code, especially when trained on massive code corpora alongside natural language [2]. Over the past couple of years, starting with Github Copilot [26], LLMs have shown rapid development in code generation quality. Major foundation model providers have pushed the frontier further with models like OpenAI's Codex [27] and o3 [28], which exhibit strong reasoning and multi-step coding abilities. Anthropic's latest Claude 3.7 Sonnet [29] introduces state-of-the-art performance on programming benchmarks like HumanEval and SWE-Bench, combining high reasoning depth with real-time code execution and debugging support. Similarly, Google DeepMind's Gemini 2.5 [30] extends its capabilities with long-context understanding and robust function synthesis, making it competitive on tasks like CodeContests and Natural2Code.

To extend the capabilities of LLM code-generation agents, recent research has moved beyond standalone LLMs toward agentic frameworks that equip LLMs with planning[31], tool use[7], and iterative decision-making abilities[32]. CodeCoT [10] leverages CoT to break down the given requirements into steps that are described in natural language and then convert them to code. ToolCoder [33] integrates the agent with online search and local documentation search tools that provide helpful information for both public and private APIs, alleviating the hallucination of LLMs.

However, real-world software engineering tasks often go beyond single-shot code synthesis. They require interpreting error messages, navigating large codebases, invoking tools like version control or test runners, and responding to user feedback. These tasks are inherently multimodal, involving a combination of natural language, code, execution traces, file structures, and sometimes even UI-level interactions. MMCode [11] addresses this gap by introducing a benchmark designed to evaluate the capabilities of large multimodal models in code generation tasks that require understanding both text and images. SWE-bench Multimodal [34] extends the original SWE-bench benchmark to assess the ability of AI systems to handle software engineering tasks that involve visual components.

Recently, multiagent systems have made significant progress in solving complex problems in software development by emulating development roles [35; 36]. Lin et al. [37] introduce a code generation framework inspired by established software engineering practices. This work leverages multiple LLM agents to emulate various software process models, namely LCGWaterfall, LCGTDD, and LCGScrum. [38] simplifies development of multi-agents for software development via a unified platform. This platform orchestrates multiple

AI agents, each specializing in distinct phases of the software development lifecycle, including requirements elicitation, architectural design, code generation, testing, and deployment.

VisionCoder [39] develops a multi-agent framework designed for automated software generation in the domain of image processing. It decomposes the software development lifecycle into a hierarchical structure-spanning project, module, and function levels-and assigns distinct roles to individual LLM-powered agents. These agents collaborate to iteratively plan, write, and refine code, enabling the system to handle complex, modular programming tasks more coherently.

Another notable system is CodeR [40], which targets the complex task of resolving GitHub issues using a structured multi-agent framework. CodeR models the resolution process as a predefined task graph, where each subtask-such as reproducing bugs, localizing faults, and verifying fixes-is handled by a specialized agent. These agents include roles like Manager, Reproducer, Fault Localizer, Editor, and Verifier, mimicking the collaborative workflows seen in real-world software teams. The task graph ensures that agent actions are sequenced and interdependent, promoting coherence and reducing common multi-agent pitfalls like uncoordinated execution or context loss. Though CodeR primarily operates on textual and code-based inputs, its architecture draws conceptual inspiration from multimodal LLM systems and is designed to be extensible to richer input modalities. Evaluated on the SWE-bench lite benchmark, CodeR achieved a 28.33% resolution rate with a single submission per issue, outperforming prior systems like SWE-agent and AutoCodeR.

4.2 Web Agents

Large language models (LLMs) are increasingly being used as autonomous web browsing agents, capable of mimicking human actions like searching, clicking links, filling forms, and extracting information from websites. This development promises to automate tasks ranging from open-domain question answering with live information to complex online workflows. A seminal example is OpenAI's WebGPT [41], which fine-tuned GPT-3 to use a text-based browser for researching and answering questions. WebGPT's agent could issue commands such as "Search" or "Open link", gather passages from the web, and compose an answer with cited sources, greatly improving factual accuracy.

Many recent systems use a single LLM as an autonomous agent that plans and executes web interactions step-by-step. AutoGPT [42] leverages GPT-4 to break high-level goals into actionable steps and execute them via web browsing. For instance, AutoGPT can be instructed to "collect market research on product X" and will autonomously search for information, navigate relevant sites, compile data, and even generate a report. chaining together these web actions with minimal human intervention

Another line of work focuses on browser automation via direct control. BrowserGpt [20] use an LLM to directly operate a web browser through a Chrome extension, issuing low-level commands (e.g. CLICK, ENTER TEXT, NAVIGATE) based on the page content. In this approach, the agent perceives the live DOM or visible page and behaves as a virtual user, which enables it to handle interactive tasks such as form-filling or web transactions. These agents often integrate with automation frameworks (e.g. Selenium or Playwright) and search APIs to carry out instructions.

To tackle complex web tasks, some recent frameworks adopt a multi-agent architecture, assigning different responsibilities to different agent modules. Microsoft's Magentic-One [8] exemplifies this design: it consists of a lead Orchestrator agent that plans the high-level strategy and delegates subtasks to a team of specialist agents. For web interactions, Magentic-One uses a WebSurfer agent specialized in browser-based tasks (navigating websites, clicking and reading content), alongside other specialists like FileSurfer, Coder and ComputerTerminal. Magentic-One has demonstrated strong results on benchmarks like WebArena and AssistantBench, approaching state-of-the-art performance on diverse web task scenarios.

Magma [16] is a foundation model designed for vision-and-language agent tasks, bridging screen perception with action. It extends a VL (vision-language) model by training on images of UIs and web pages annotated with the locations of clickable or input elements using Set Of Marks. As a result, Magma can interpret a webpage screenshot or GUI image and output a sequence of actions (e.g. move cursor to a button and click) in a goal-driven manner. This yields state-of-the-art results on UI navigation tasks, outperforming prior specialized models that lacked such broad visual-action grounding. By incorporating spatial reasoning and visual context, multimodal agents like Magma can handle websites that are heavy on graphics or dynamic content, where pure text-based parsing falls short.

4.3 Operating System Agents

The design of agents that interact with graphical user interfaces (GUIs) has emerged as a powerful testbed for multimodal large language models (MLLMs). These GUI agents aim to replicate human-like digital behavior by perceiving screens, interpreting instructions, and interacting with applications across platforms like web, desktop, and mobile. However, GUI environments pose unique challenges: sparse metadata, platform heterogeneity, visually dense layouts, and the need for precise pixel-level control. Recent research addresses these challenges from multiple perspectives, ranging from visual grounding to end-to-end agent frameworks.

One notable direction is the development of vision-only agents that bypass traditional HTML or accessibility (a11y) tree parsing. Early progress in this space is marked by SeeClick [17], which introduced a lightweight yet effective vision-language agent capable of grounding GUI instructions purely from screenshots. By pretraining on multimodal UI data and evaluating on the newly proposed ScreenSpot benchmark, SeeClick demonstrated that robust GUI grounding can be achieved without structured input, outperforming larger models like GPT-4V and CogAgent in both grounding accuracy and task execution across web, mobile, and desktop platforms.

Building on this paradigm [43] introduces UGround, a universal GUI visual grounding model trained on over 10 million elements from 1.3 million screenshots. Integrated into the SeeAct-V framework, UGround enables agents to perceive purely through screenshots and perform pixel-level operations. Despite being trained mostly on web data, it generalizes well to mobile and desktop tasks, outperforming prior models like SeeClick [17] and CogAgent [44] on grounding benchmarks like ScreenSpot [45]. This modular pipeline-separating planning (via MLLMs like GPT-4) and grounding (via UGround)-represents a strong case for human-like, platform-agnostic GUI agents.

Complementing this vision-first approach, Auto-GUI [22] proposes a multimodal chain-of-action agent that achieves high-accuracy GUI automation directly from screen pixels. By modeling user interaction as a chain of actions-each grounded in screen context and past interaction-Auto-GUI avoids reliance on structured UI inputs or domain-specific APIs. It sets a new state-of-the-art on the AITW benchmark [46], demonstrating that agents can effectively plan and execute tasks by perceiving only what the user sees on screen.

While both Auto-GUI and SeeAct-V advocate end-to-end visual agents, ASSISTGUI [47] highlights the importance of modular, interpretable pipelines for long-horizon tasks in complex desktop applications (e.g., After Effects, Word). Its ACE framework (Actor-Critic Embodied agent) breaks down tasks using instructional videos and GUI metadata, combining planning, grounding, and feedback through subtasks and milestones. The framework supports action validation and error recovery via visual critics, outperforming prompting-only baselines by large margins on its 100-task benchmark. ASSISTGUI shows that even with structured data (OCR, icon detection), desktop software remains a hard problem, motivating robust perception-grounding loops.

Pushing further into realism, WindowsAgentArena [13] introduces a benchmark suite for agents operating on live Windows OS environments, spanning productivity tools, browsers, system utilities, and coding IDEs. The included agent Navi demonstrates how multi-modal LLMs can operate over OCR, icon captioning, and UI Automation APIs to plan and execute full workflows using a custom computer API. The benchmark runs in dockerized VMs with automated evaluators, allowing fast, scalable assessment of GUI agents in real-world conditions. Importantly, Navi's performance remains far below human levels-highlighting the current gap in robust reasoning and grounding in highly dynamic environments.

OS-Copilot [48] proposes a modular, memory-augmented architecture for full-computer control. Its embodiment, FRIDAY, acts as a self-improving agent that not only completes complex tasks across GUIs, terminals, and filesystems, but also generates and refines tools on the fly via trial and error. Through a structured memory model (declarative, procedural, and working memory), FRIDAY maintains context, stores reusable behaviors, and self-critiques failed attempts.

Several works have extended these ideas into the smartphone domain. AppAgent [49] presents a full-stack multimodal agent that learns to operate any Android app by observing screenshots and UI metadata. It leverages a simplified action space (taps, swipes, text input) and learns app functionality through autonomous exploration or by watching human demonstrations. During deployment, the agent consults its learned documentation to plan actions, achieving strong results across 10 diverse apps. Notably, AppAgent outperforms raw GPT-4 baselines significantly (84% success rate vs. 2–49%) and matches the effectiveness of manually written UI guides.

In a similar direction, MM-Navigator [50] is a purely vision-based agent powered by GPT-4V. It operates entirely through screenshots using a "Set-of-Mark" prompting strategy, where UI elements are tagged and GPT-4V selects the next action by reasoning over the image and text. MM-Navigator achieves 91% accuracy in intended action description and 75% accuracy in localized execution on iOS screens, and further surpasses prior LLM baselines on the large-scale AITW Android dataset-all in zero-shot mode.

DroidBot-GPT [51] takes a lightweight route by pairing LLMs with existing UI testing tools like DroidBot. It converts Android screen states into natural language descriptions and queries GPT-4 to choose the next action. Despite being fully unsupervised and simple in design, it completes over 39% of tasks and demonstrates solid generalization across app categories.

5 Evaluation Frameworks and Benchmarks

Evaluation plays a pivotal role in developing multimodal LLM agents, ensuring they can robustly handle the diverse tasks encountered in software engineering, operating systems, and web environments. Benchmarks provide structured challenges that drive progress and reveal weaknesses. Given the breadth of domains – from writing and debugging code, to controlling OS-level tools, to interacting with web pages – a one-size-fits-all evaluation is impractical. Instead, researchers have proposed specialized benchmarks targeting different facets of an LLM agent's capabilities. These range from coding challenges and OS automation tasks to visual question answering and multi-agent collaboration scenarios. Benchmarking multimodal LLM agents is important not only for tracking performance improvements but also for understanding an agent's limitations before deployment in real-world settings. Recent work emphasizes that traditional NLP benchmarks are insufficient for agents acting in complex environments, prompting a shift toward more realistic, task-oriented evaluations. The following subsections survey three major categories of evaluation frameworks: Operating System Agent Benchmarks, Multi-modal Agent Benchmarks, and Multi-Agent Collaboration Benchmarks. For each, we outline their focus, compare representative benchmarks, and discuss how evaluation is conducted.

5.1 Operating System Agent Benchmarks

Operating System Agent Benchmarks test an agent's ability to fully navigate and control a simulated operating system environment. Unlike isolated task evaluations that assess simple function calls or API interactions, these benchmarks present the agent with a dynamic, often unpredictable environment where it must interact with GUI elements, manage files, operate multiple applications concurrently, and-even in some cases-handle both desktop and mobile devices. Table 1 is a comparison of several prominent OS-level evaluation suites.

OS-level benchmarks typically evaluate success by checking if the agent achieved the task's goal state within the environment. Many use programmatic logs or scripts to verify outcomes – for example, OSWorld defines a custom execution-check for each task (e.g. confirming a file was created or an email sent) to ensure reliable, reproducible scoring. cite. Some environments provide reward signals at intermediate steps (AndroidWorld) to facilitate learning and fine-grained evaluation cite. Common metrics include success rate, task completion time, and sometimes efficiency (number of actions taken vs. an optimal baseline). Because agents operate in an actual OS or high-fidelity simulator, these benchmarks emphasize robustness – minor UI changes or unexpected dialogs can cause failures, so an agent's adaptability is tested. Reproducibility is addressed by standardized initial states (e.g. reset OS to a known state before each trial) and deterministic task specifications. Overall, OS benchmarks stress an agent's ability to perform real computer-based tasks end-to-end, providing a strong measure of practical utility as an "AI assistant" on personal devices.

Table 1: Comparison of Prominent OS-level Evaluation Suites

Benchmark	**Focus Area**	**Unique Features**
OsBench	General OS control tasks (desktop)	Subset of tasks from AgentBench's [3] OS domain; first standardized OS agent eval in a multi-domain suite
Windows Agent Arena (WAA) [13]	Windows OS automation	150+ tasks across 11 applications on Windows (e.g. Office apps), enabling parallel, scalable evaluation in a real OS
CRAB (Cross-Environment Agent Benchmark) [24]	Cross-platform (PC + mobile) tasks	First benchmark to support cross-device tasks (desktop & Android simultaneously), with a graph-based fine-grained evaluation method for action sequences
OSWorld [19]	Open-ended tasks in real OS env	Multi-OS (Windows, Ubuntu, macOS) environment with 369 tasks spanning web and desktop apps, file I/O, and multi-app workflows. Uses execution-based scripts for reliable evaluation of each task's outcome
AndroidWorld [25]	Android mobile device tasks	Live Android emulator environment with 116 programmatic tasks

5.2 Multi-modal Benchmarks

This category of benchmarks is designed to assess agents that must integrate and process diverse input and output modalities, such as vision (e.g., images or GUI screenshots), language, and sometimes structured data or code. In contrast to OS benchmarks, which focus on navigating and controlling complete operating system environments, multimodal benchmarks evaluate an agent's ability to seamlessly combine and reason over visual and textual cues to accomplish specific tasks. Such tasks span activities like web browsing (where an agent interprets screenshots alongside textual instructions), interacting with applications that provide rich visual feedback, or resolving software engineering challenges that involve both code and graphical information.

Multimodal benchmarks use a mix of automated and human-in-the-loop evaluation. For tasks with well-defined end states, the environment can automatically judge success (e.g. WebArena [18] tasks might specify a target page or content that the agent must reach). Many web-based benchmarks use information-retrieval style metrics – for instance, whether the agent eventually clicks the correct link or extracts the correct answer from a page. Some tasks, especially those involving generated content like code patches in SWE-Bench [53], require qualitative evaluation: using either human experts or additional LLMs to judge if the agent's output is correct. In SWE-Bench, an agent's code fix can be validated by running test cases or comparing against the ground truth patch. For visual tasks, evaluation often checks if a final GUI state matches expectations (e.g. "button X became green"), sometimes via DOM/Screenshot comparison or specialized metrics. One challenge in multimodal evaluation is measuring partial success: an agent might get some subtasks right (navigating to the right page) but fail the final instruction. Benchmarks like MMInA [52] handle this with fine-grained, hop-by-hop scoring, crediting the agent for each correct intermediate step in a multihop task. Overall, these evaluations stress cross-modal consistency – the agent must align textual and visual information to succeed. They reveal that current agents often excel in either language or vision individually, but struggle when both are required simultaneously (e.g. correctly identifying a GUI element described in text). Multimodal benchmarks thus provide a rigorous check on an agent's integrated understanding of rich, real-world scenarios. Table 2 is a comparison of several prominent Multimodal Agent evaluation suites.

5.3 Multi-Agent Collaboration Benchmarks

Beyond single-agent scenarios, a new class of benchmarks evaluates how multiple LLM agents collaborate with each other (and with humans) to solve problems. These benchmarks simulate settings like teams of

Table 2: Comparison of Prominent Multimodal Agent Evaluation Benchmarks

Benchmark	**Focus Area**	**Unique Features**
WebVoyager [14]	Vision-enabled web browsing	An end-to-end web agent that uses browser screenshots as context. The agent must interpret visual elements (e.g., buttons, images) alongside textual cues to navigate real websites.
MMInA (Multi-hop Multimodal Internet Agents) [52]	Multi-step internet navigation (text + images)	Challenges an agent with complex, compositional web tasks built from approximately 3,000 recorded interactions ("hops") across multiple websites. It tests sequential integration of visual and textual information.
VisualWebArena [15]	Visually-grounded web tasks	Provides over 900 tasks requiring the interpretation of page images and layouts in realistic scenarios, such as online shopping or data visualization.
SWE-Bench (Text) [53]	Software engineering bug-fixing (text-only)	Evaluates code reasoning from textual bug reports and source code, without visual inputs.
SWE-Bench (Multimodal) [34]	Software engineering with UI context	Extends the text-only version by incorporating UI screenshots, enabling the assessment of code patches in response to both visual and textual bug reports.
WebLINX [54]	Conversational web navigation	Contains a large-scale dataset (100k multi-turn interactions) where the agent uses dialogue-like instructions to navigate websites via text interfaces, derived from expert demonstrations.
PixelHelp [55]	Mobile UI how-to instructions (static)	Comprises 187 high-level Android tasks based on official help guides, linking user queries with step-by-step solutions and mapping language to mobile UI actions.
OmniACT [56]	Generalist desktop & web tasks	Evaluates an agent's ability to generate executable commands that span both desktop and web environments, such as downloading data from a website and visualizing it in a desktop application.
MiniWoB++ [57]	Synthetic web UI tasks (toy domain)	A controlled testbed featuring 100+ miniature web tasks (e.g., simple forms or calculators) that assess low-level command execution (clicks/keystrokes).
Mind2Web [58]	Open-domain web tasks	Comprises 300 tasks sourced from 136 popular websites, challenging agents to generalize and interpret natural language instructions across varied and unfamiliar layouts.

agents working together or an AI assistant interacting with a human partner. They emphasize communication, coordination, and division of labor – aspects critical for autonomous agents operating in social or organizational contexts. Table 3 is a comparison of prominent Multi-agent evaluation benchmarks.

Table 3: Multi-Agent Collaboration Benchmarks

Benchmark	**Focus Area**	**Unique Features**
COMMA [59]	Multi-agent communication	Multimodal benchmark with 2+ agents collaborating via natural language under asymmetric information. Tasks test coordination and info-sharing in settings where agents see different inputs. Evaluates agent–agent and agent–human teamwork, revealing weaknesses in peer-to-peer dialog and reasoning.
TheAgentCompany [23]	Simulated workplace tasks	Extensible benchmark simulating a software company. Agents take on roles (e.g., coder, manager) and collaborate via messages to complete realistic tasks like coding, support, and research. Tests coordination on long-horizon projects involving web browsing, code execution, and information synthesis.

Collaborative benchmarks introduce subjective aspects to evaluation. A primary metric is task success – did the agent team achieve the desired outcome (solving the puzzle, completing the project)? For many tasks, success can be judged objectively (e.g. a final answer or product can be checked), but the process of collaboration is also scrutinized. COMMA [59], for instance, evaluates whether agents' communications enable them to outperform a baseline; it was found that two GPT-4 based agents often failed to leverage communication effectively. Thus, metrics like communication efficiency (how many dialogue turns, or how much irrelevant chatter) and role-specific performance (did each agent fulfill its part) are used. In TheAgentCompany's [23] simulated workplace, evaluation involves scenario-specific tests – for example, if the task is to fix a bug collaboratively, the final code is tested for correctness, and logs are analyzed to see if agents correctly delegated subtasks. Human evaluation is sometimes incorporated to rate the coherence and helpfulness of agent communications (especially in agent-human collaboration scenarios). A notable challenge is ensuring that the collaboration is genuine – i.e. the benchmark should require information exchange. These benchmarks often craft tasks such that no single agent has all the needed information or skills, forcing interaction. By analyzing transcripts and outcomes, researchers can identify failure modes like misunderstanding between agents or dominance of one agent. In summary, multi-agent benchmarks extend evaluation to the social intelligence of LLM agents, assessing how well they can engage in teamwork and communication to solve problems together.

5.4 Limitations of Existing Benchmarks

Despite the significant progress enabled by new benchmark suites, several fundamental challenges remain in evaluating multimodal LLM agents. These issues limit how well current benchmarks reflect real-world complexities and the diverse range of tasks agents must handle in practice. The following are some of the limitations of existing benchmarks.

- **Evaluation Gaming and Overfitting:** Models can sometimes exploit the structure of benchmarks in unintended ways. There is concern that an agent might detect when it is in a scripted evaluation versus a real deployment and behave differently (e.g. using hidden cues). For instance, if all tasks in a benchmark have a similar template, a model might "learn the test" rather than truly generalize. Designing evaluations that the agent cannot easily recognize (or game) is an open problem, as is detecting when a model is cheating an eval by using subtle cues.
- **Lack of Realism:** Many benchmarks simplify environments or have deterministic setups, which can diverge from messy real-world conditions. Some interactive benchmarks still rely on predefined action sequences as the success criterion [58], if an agent solves a task in an alternate valid way, it may be

unfairly marked wrong. Real users often face dynamic and unpredictable scenarios (random pop-ups, network delays, etc.) that static benchmarks don't capture. Moreover, tasks in research benchmarks may not cover the full diversity of real user goals. This gap in realism means an agent that scores high on a benchmark might still struggle on the true task in the wild. Efforts like OSWorld [19] and AndroidWorld [25] have started addressing this by using real applications and realistic initial states, but many benchmarks remain narrow or synthetic.

- **Limited Multimodal Coverage:** While a number of benchmarks highlight text+vision tasks, the multimodal spectrum is much broader. Current evaluations rarely incorporate modalities such as audio (voice commands, sound cues) or haptic/physical interactions. Even within vision-and-language, benchmarks tend to focus on either GUI screenshots or web images; few require understanding visual media (e.g. interpreting a chart image and an instruction, as a human might). Also, most benchmarks treat modalities in a fixed way – e.g. an image accompanying a text query. In realistic settings, an agent might need to dynamically decide when to use vision (e.g. whether to "look" at a screen or not) and handle continuous video or mixed-modality streams. Expanding benchmarks to cover more modalities and their combinations is needed to truly test general-purpose agency.
- **Measuring Autonomy and Long-Term Behavior:** Present evaluations mostly consist of one-off episodes or tasks with a clear end. They struggle to assess an agent's long-term autonomy – can it operate robustly over hours or days, maintaining coherence towards an open-ended goal? For example, an agent might do well in a contained task but would it eventually get stuck or behave unpredictably if left running continuously? Current benchmarks rarely test an agent's ability to learn from experience during evaluation. Agents are typically reset between tasks, so we don't measure if they improve or adapt. This limits how well we can evaluate properties like online learning, persistence of memory, or the ability to recover from mistakes – all crucial for true autonomy.

5.5 Future Directions in Benchmark Design

In pursuit of more realistic and comprehensive evaluations, researchers are exploring new ways to measure long-horizon performance, autonomy, and social interaction. By introducing persistent memory, real-world constraints, and collaborative settings, future benchmarks can more accurately reflect the conditions under which these agents will be deployed.

- **Persistent Memory and Lifelong Evaluation:** New benchmarks should test an agent's ability to remember and utilize knowledge over extended periods and across sessions. For example, an evaluation could span multiple related tasks (simulating a day's worth of assistant duties), where the agent's performance on later tasks benefits from information learned in earlier ones. This would encourage the development of agents with persistent memory and the ability to accumulate experience. Measuring memory retention and the consistency of agent behavior over time will be key – does the agent recall a user's preferences from a prior interaction, or lessons from a previous mistake? Such persistent evaluations would move beyond episodic one-shot tests towards lifelong learning assessment.
- **Social Interaction and Collaboration Trials:** Building on multi-agent benchmarks, future evaluations may involve more realistic social settings – e.g. an agent working with a human team on a project, or multiple agents with simulated personalities collaborating (or even competing). This could include benchmarks for negotiation skills, teaching/learning from other agents, or adherence to human conversational norms in long-running interactions. Social intelligence metrics (like user satisfaction, trust calibration, conflict resolution success) could complement task-success metrics. By testing nuanced interaction scenarios, we can drive the development of agents that are not only task-savvy but also adept at understanding and responding to humans and other agents in complex social environments.
- **Real-World Deployment Challenges:** Perhaps the ultimate benchmark is deploying agents in real (or extremely high-fidelity) environments and measuring their performance on truly consequential tasks. Future benchmarks might arrange controlled real-world trials – for instance, an agent managing an actual web store for a few hours, or controlling smart home devices over days. Success criteria would extend

to robustness and safety: not just accomplishing goals, but doing so without undesired side-effects (no crashing the system or violating user intent). There is growing interest in the evaluation of "consequential tasks" – those with real economic or safety impact – to understand where current agents fail. While challenging, such deployment-centric benchmarks would provide the clearest picture of an agent's readiness for practical use. They could also reveal issues (like error recovery, scalability under load, security vulnerabilities) that lab environments cannot easily simulate.

References

[1] J. Xie, Z. Chen, R. Zhang, X. Wan, and G. Li, "Large multimodal agents: A survey," 2024.

[2] K. Yang, J. Liu, J. Wu, C. Yang, Y. R. Fung, S. Li, Z. Huang, X. Cao, X. Wang, Y. Wang, H. Ji, and C. Zhai, "If llm is the wizard, then code is the wand: A survey on how code empowers large language models to serve as intelligent agents," 2024.

[3] X. Liu, H. Yu, H. Zhang, Y. Xu, X. Lei, H. Lai, Y. Gu, H. Ding, K. Men, K. Yang, S. Zhang, X. Deng, A. Zeng, Z. Du, C. Zhang, S. Shen, T. Zhang, Y. Su, H. Sun, M. Huang, Y. Dong, and J. Tang, "Agentbench: Evaluating llms as agents," 2023.

[4] J.-B. Alayrac, J. Donahue, P. Luc, A. Miech, I. Barr, Y. Hasson, K. Lenc, A. Mensch, K. Millican, M. Reynolds, R. Ring, E. Rutherford, S. Cabi, T. Han, Z. Gong, S. Samangooei, M. Monteiro, J. Menick, S. Borgeaud, A. Brock, A. Nematzadeh, S. Sharifzadeh, M. Binkowski, R. Barreira, O. Vinyals, A. Zisserman, and K. Simonyan, "Flamingo: a visual language model for few-shot learning," 2022.

[5] OpenAI, "Gpt-4v(ision) system card," *Technical report*, 2023.

[6] Y. Yang, T. Zhou, K. Li, D. Tao, L. Li, L. Shen, X. He, J. Jiang, and Y. Shi, "Embodied multi-modal agent trained by an llm from a parallel textworld," 2024.

[7] T. Schick, J. Dwivedi-Yu, R. Dessì, R. Raileanu, M. Lomeli, L. Zettlemoyer, N. Cancedda, and T. Scialom, "Toolformer: Language models can teach themselves to use tools," 2023.

[8] A. Fourney, G. Bansal, H. Mozannar, C. Tan, E. Salinas, Erkang, Zhu, F. Niedtner, G. Proebsting, G. Bassman, J. Gerrits, J. Alber, P. Chang, R. Loynd, R. West, V. Dibia, A. Awadallah, E. Kamar, R. Hosn, and S. Amershi, "Magentic-one: A generalist multi-agent system for solving complex tasks," 2024.

[9] S. Li, R. Wang, C.-J. Hsieh, M. Cheng, and T. Zhou, "Mulan: Multimodal-llm agent for progressive and interactive multi-object diffusion," 2024.

[10] D. Huang, Q. Bu, Y. Qing, and H. Cui, "Codecot: Tackling code syntax errors in cot reasoning for code generation," 2024.

[11] K. Li, Y. Tian, Q. Hu, Z. Luo, Z. Huang, and J. Ma, "Mmcode: Benchmarking multimodal large language models for code generation with visually rich programming problems," 2024.

[12] Z. Zhang and A. Zhang, "You only look at screens: Multimodal chain-of-action agents," 2024.

[13] R. Bonatti, D. Zhao, F. Bonacci, D. Dupont, S. Abdali, Y. Li, Y. Lu, J. Wagle, K. Koishida, A. Bucker, L. Jang, and Z. Hui, "Windows agent arena: Evaluating multi-modal os agents at scale," 2024.

[14] H. He, W. Yao, K. Ma, W. Yu, Y. Dai, H. Zhang, Z. Lan, and D. Yu, "Webvoyager: Building an end-to-end web agent with large multimodal models," 2024.

[15] J. Y. Koh, R. Lo, L. Jang, V. Duvvur, M. C. Lim, P.-Y. Huang, G. Neubig, S. Zhou, R. Salakhutdinov, and D. Fried, "Visualwebarena: Evaluating multimodal agents on realistic visual web tasks," 2024.

[16] J. Yang, R. Tan, Q. Wu, R. Zheng, B. Peng, Y. Liang, Y. Gu, M. Cai, S. Ye, J. Jang, Y. Deng, L. Liden, and J. Gao, "Magma: A foundation model for multimodal ai agents," *arXiv preprint arXiv:2502.13130*, 2025.

[17] K. Cheng, Q. Sun, Y. Chu, F. Xu, Y. Li, J. Zhang, and Z. Wu, "Seeclick: Harnessing gui grounding for advanced visual gui agents," *arXiv preprint arXiv:2401.10935*, 2024.

[18] S. Zhou, F. F. Xu, H. Zhu, X. Zhou, R. Lo, A. Sridhar, X. Cheng, T. Ou, Y. Bisk, D. Fried, U. Alon, and G. Neubig, "Webarena: A realistic web environment for building autonomous agents," 2024.

[19] T. Xie, D. Zhang, J. Chen, X. Li, S. Zhao, R. Cao, T. J. Hua, Z. Cheng, D. Shin, F. Lei, Y. Liu, Y. Xu, S. Zhou, S. Savarese, C. Xiong, V. Zhong, and T. Yu, "Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments," 2024.

[20] MagMueller and gregpr07, "browser-use," 2025.

[21] K. Ying, F. Meng, J. Wang, Z. Li, H. Lin, Y. Yang, H. Zhang, W. Zhang, Y. Lin, S. Liu, J. Lei, Q. Lu, R. Chen, P. Xu, R. Zhang, H. Zhang, P. Gao, Y. Wang, Y. Qiao, P. Luo, K. Zhang, and W. Shao, "Mmt-bench: A comprehensive multimodal benchmark for evaluating large vision-language models towards multitask agi," 2024.

[22] Y. Zhang, Y. Wang, Y. Li, Y. Liu, Y. Wang, Y. Li, and Y. Liu, "You only look at screens: Multimodal chain-of-action agents," *arXiv preprint arXiv:2309.11436*, 2023.

[23] F. F. Xu, Y. Song, B. Li, Y. Tang, K. Jain, M. Bao, Z. Z. Wang, X. Zhou, Z. Guo, M. Cao, M. Yang, H. Y. Lu, A. Martin, Z. Su, L. Maben, R. Mehta, W. Chi, L. Jang, Y. Xie, S. Zhou, and G. Neubig, "Theagentcompany: Benchmarking llm agents on consequential real world tasks," 2024.

[24] T. Xu, L. Chen, D.-J. Wu, Y. Chen, Z. Zhang, X. Yao, Z. Xie, Y. Chen, S. Liu, B. Qian, A. Yang, Z. Jin, J. Deng, P. Torr, B. Ghanem, and G. Li, "Crab: Cross-environment agent benchmark for multimodal language model agents," 2024.

[25] C. Rawles, S. Clinckemaillie, Y. Chang, J. Waltz, G. Lau, M. Fair, A. Li, W. Bishop, W. Li, F. Campbell-Ajala, D. Toyama, R. Berry, D. Tyamagundlu, T. Lillicrap, and O. Riva, "Androidworld: A dynamic benchmarking environment for autonomous agents," 2025.

[26] GitHub, "Github copilot." `https://github.com/features/copilot`, 2021. Accessed: 2025-04-16.

[27] OpenAI, "Openai codex," August 2021. Accessed: 2025-04-16.

[28] OpenAI, "Introducing openai o3 and o4-mini," April 2025. Accessed: 2025-04-16.

[29] Anthropic, "Claude 3.7 sonnet and claude code," February 2025. Accessed: 2025-04-16.

[30] Google, "Gemini 2.5: Our most intelligent ai model," March 2025. Accessed: 2025-04-16.

[31] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, "React: Synergizing reasoning and acting in language models," 2023.

[32] N. Shinn, F. Cassano, E. Berman, A. Gopinath, K. Narasimhan, and S. Yao, "Reflexion: Language agents with verbal reinforcement learning," 2023.

[33] H. Ding, S. Tao, L. Pang, Z. Wei, J. Gao, B. Ding, H. Shen, and X. Chen, "Toolcoder: A systematic code-empowered tool learning framework for large language models," 2025.

[34] J. Yang, C. E. Jimenez, A. L. Zhang, K. Lieret, J. Yang, X. Wu, O. Press, N. Muennighoff, G. Synnaeve, K. R. Narasimhan, D. Yang, S. I. Wang, and O. Press, "Swe-bench multimodal: Do ai systems generalize to visual software domains?," 2024.

[35] M. A. Sami, Z. Rasheed, M. Waseem, Z. Zhang, T. Herda, and P. Abrahamsson, "Prioritizing software requirements using large language models," 2024.

[36] Z. Rasheed, M. A. Sami, K.-K. Kemell, M. Waseem, M. Saari, K. Systä, and P. Abrahamsson, "Codepori: Large-scale system for autonomous software development using multi-agent technology," 2024.

[37] F. Lin, D. J. Kim, Tse-Husn, and Chen, “Soen-101: Code generation by emulating software process models using large language model agents,” 2024.

[38] M. A. Sami, M. Waseem, Z. Rasheed, M. Saari, K. Systä, and P. Abrahamsson, “Experimenting with multi-agent software development: Towards a unified platform,” 2024.

[39] Z. Zhao, J. Sun, Z. Wei, C.-H. Cai, Z. Hou, and J. S. Dong, “Visioncoder: Empowering multi-agent auto-programming for image processing with hybrid llms,” 2024.

[40] D. Chen, S. Lin, M. Zeng, D. Zan, J.-G. Wang, A. Cheshkov, J. Sun, H. Yu, G. Dong, A. Aliev, J. Wang, X. Cheng, G. Liang, Y. Ma, P. Bian, T. Xie, and Q. Wang, “Coder: Issue resolving with multi-agent and task graphs,” 2024.

[41] R. Nakano, J. Hilton, S. Balaji, J. Wu, L. Ouyang, C. Kim, C. Hesse, S. Jain, V. Kosaraju, W. Saunders, X. Jiang, K. Cobbe, T. Eloundou, G. Krueger, K. Button, M. Knight, B. Chess, and J. Schulman, “Webgpt: Browser-assisted question-answering with human feedback,” *arXiv preprint arXiv:2112.09332*, 2021.

[42] T. B. Richards, “Autogpt.” `https://github.com/Significant-Gravitas/AutoGPT`, 2023. Accessed: 2025-04-16.

[43] B. Gou, R. Wang, B. Zheng, Y. Xie, C. Chang, Y. Shu, H. Sun, and Y. Su, “Navigating the digital world as humans do: Universal visual grounding for gui agents,” *arXiv preprint arXiv:2410.05243*, 2024.

[44] Y. Hong, Y. Wang, Y. Li, Y. Liu, Y. Wang, Y. Li, and Y. Liu, “Cogagent: A visual language model for gui agents,” *arXiv preprint arXiv:2312.08914*, 2023.

[45] K. Cheng, Q. Sun, Y. Chu, F. Xu, Y. Li, J. Zhang, and Z. Wu, “Screenspot: A benchmark for gui grounding,” *arXiv preprint arXiv:2401.10935*, 2024.

[46] Y. Zhang, Y. Wang, Y. Li, Y. Liu, Y. Wang, Y. Li, and Y. Liu, “Aitw: A benchmark for gui agent evaluation,” *arXiv preprint arXiv:2309.11436*, 2023.

[47] L. Zhao, Z. Bai, M. Ouyang, P. Li, D. Mao, P. Zhang, X. Wang, H. Wang, L. Zhou, and F. Mi, “Assistgui: Task-oriented desktop graphical user interface automation,” *arXiv preprint arXiv:2312.13108*, 2023.

[48] Y. Zhang, Y. Wang, Y. Li, Y. Liu, Y. Wang, Y. Li, and Y. Liu, “Os-copilot: Towards generalist computer agents with self-improvement,” *arXiv preprint arXiv:2402.07456*, 2024.

[49] C. Zhang, Z. Yang, J. Liu, Y. Han, X. Chen, Z. Huang, B. Fu, and G. Yu, “Appagent: Multimodal agents as smartphone users,” *arXiv preprint arXiv:2408.11824*, 2024.

[50] A. Yan, Z. Yang, W. Zhu, K. Lin, L. Li, J. Wang, J. Yang, Y. Zhong, J. McAuley, J. Gao, Z. Liu, and L. Wang, “Gpt-4v in wonderland: Large multimodal models for zero-shot smartphone gui navigation,” *arXiv preprint arXiv:2311.07562*, 2023.

[51] J. Yoon, R. Feldt, and S. Yoo, “Droidbot-gpt: Gpt-powered ui automation for android,” *arXiv preprint arXiv:2304.07061*, 2023.

[52] Z. Zhang, S. Tian, L. Chen, and Z. Liu, “Mmina: Benchmarking multihop multimodal internet agents,” 2024.

[53] C. E. Jimenez, J. Yang, A. Wettig, S. Yao, K. Pei, O. Press, and K. Narasimhan, “Swe-bench: Can language models resolve real-world github issues?,” 2024.

[54] X. H. Lù, Z. Kasner, and S. Reddy, “Weblinx: Real-world website navigation with multi-turn dialogue,” 2024.

[55] Y. Li, J. He, X. Zhou, Y. Zhang, and J. Baldridge, “Mapping natural language instructions to mobile ui action sequences,” in *Annual Conference of the Association for Computational Linguistics (ACL 2020)*, 2020.

[56] R. Kapoor, Y. P. Butala, M. Russak, J. Y. Koh, K. Kamble, W. Alshikh, and R. Salakhutdinov, "Omniact: A dataset and benchmark for enabling multimodal generalist autonomous agents for desktop and web," 2024.

[57] E. Z. Liu, K. Guu, P. Pasupat, T. Shi, and P. Liang, "Reinforcement learning on web interfaces using workflow-guided exploration," in *International Conference on Learning Representations (ICLR)*, 2018.

[58] X. Deng, Y. Gu, B. Zheng, S. Chen, S. Stevens, B. Wang, H. Sun, and Y. Su, "Mind2web: Towards a generalist agent for the web," 2023.

[59] T. Ossowski, J. Chen, D. Maqbool, Z. Cai, T. Bradshaw, and J. Hu, "Comma: A communicative multimodal multi-agent benchmark," 2025.