What Limits Virtual Agent Application? OmniBench: A Scalable Multi-Dimensional Benchmark for Essential Virtual Agent Capabilities

Anonymous Authors[1]

Abstract

As multimodal large language models (MLLMs) advance, MLLM-based virtual agents have demonstrated remarkable performance. However, existing benchmarks face significant limitations, including uncontrollable task complexity, extensive manual annotation with limited scenarios, and a lack of multidimensional evaluation. In response to these challenges, we introduce **OmniBench**, a self-generating, graph-based benchmark with an automated pipeline for synthesizing tasks of controllable complexity through subtask composition. To evaluate the diverse capabilities of virtual agents on the graph, we further present **OmniEval**, a multidimensional evaluation framework that includes subtask-level evaluation, graph-based metrics, and comprehensive tests across 10 capabilities. Our synthesized dataset contains 36k graph-structured tasks across 20 scenarios, achieving a 91% human acceptance rate. Training on our graph-structured data shows that it can more efficiently guide agents compared to manually annotated data. We conduct multi-dimensional evaluations for various open-source and closed-source models, revealing their performance across various capabilities and paving the way for future advancements.

1 Introduction

With the development of MLLMs, recent MLLM-based virtual agents (Xu et al., 2024b; Gao et al., 2024; Wu et al., 2024) have demonstrated promising performance in web navigation (Shen et al., 2024), mobile device control (Ge et al., 2024), and computer interaction (Hu et al., 2024). To explore real-world values of visual agents, current research mainly evaluates their performance based on offline trajectory similarity with human demonstrations (Rawles et al., 2024; Lu et al., 2024; Deng et al., 2024) or by using expert-crafted functions in interactive online environments (Xie et al., 2024; Zhou et al., 2023; Yao et al., 2022).

However, these two types of benchmarks mentioned above still have notable limitations: **1) Uncontrollable and fixed task complexity.** Existing benchmarks typically propose tasks entirely rather than progressively with fine-grained guidance, which results in uncontrollable and fixed task complexity. Uncontrollable task complexity makes it hard to design fine-grained test data for various capabilities, while fixed complexity makes it challenging for benchmarks to keep up with agents' growing capabilities. **2) Extensive manual labor and limited task scenarios.** The existing benchmarks rely on manual annotations to synthesize demonstration trajectories or evaluation functions, making the cost of designing benchmarks unaffordable and hindering the expansion of scale. Moreover, the annotated data with a limited amount is influenced by human prior experience, making it difficult to cover comprehensive scenarios. **3) Absence of multidimensional evaluation**: Existing benchmarks commonly evaluate agents based on the final state of tasks, lacking an evaluation of the intermediate steps in task execution. Additionally, the various capabilities required by virtual agents to complete tasks (*e.g.*, planning, instruction understanding, etc.) cannot be quantified by coarse task success rates, failing to provide sufficient feedback for potential future improvements. **In summary**, an ideal benchmark should include not only diverse task scenarios with controllably complexity, but also a comprehensive evaluation across multiple dimensions.

To cost-effectively construct diverse task scenarios with multi-granularity complexity for comprehensive agent evaluation, we propose a novel self-generating graph-based benchmark, **OmniBench**, dynamically synthesizing tasks with controllable complexity based on a bottom-up pipeline. OmniBench spans five fundamental task complexities to construct 10 evaluation dimensions (c.f. Figure 1), with test tasks in different dimensions categorized according to combined complexities, *e.g.*, long-range planning task is combined from dependency complexity and hierarchical complexity. OmniBench consists of 36k high-quality

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

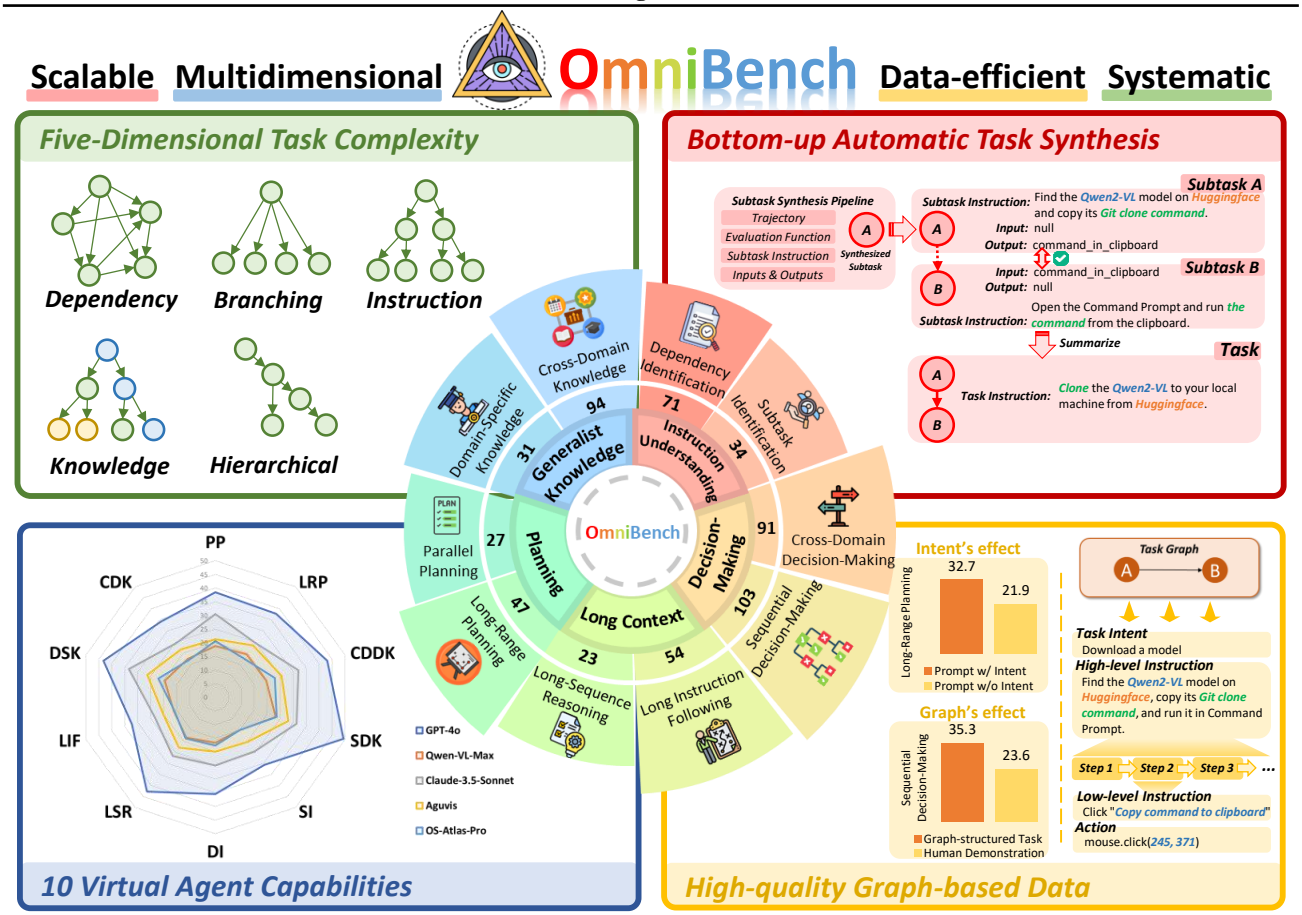

Figure 1. Overview of OmniBench, a systematic benchmark with five-dimensional task complexity and bottom-up automatic task synthesis for generating structured task graphs. It evaluates ten virtual agent capabilities using high-quality graph-based data, ensuring scalable and realistic task assessments.

graph-structured tasks across 20 distinct scenarios (*e.g.* image editing, video editing) derived from its self-generating framework, with the task scale being 40x larger than most environment-based benchmarks, as shown in Tab. 1. This automated process opens up the possibility of scaling up virtual agent evaluation in a low-resource manner. For multi-dimensional task synthesis, we take motivations from the DAG topology (Xu et al., 2024a) to design a bottom-up pipeline. Specifically, we define five fundamental task complexities and **synthesize tasks with controllable complexity by constraining the DAG composition process**. Notably, we extracted task intents to guide this process to avoid composing meaningless tasks (*e.g.*, playing music while playing video). We further incorporate a consistency validation mechanism to optimize subtask trajectories and ensure task alignment. In this way, we cost-effectively synthesize high-quality graph-structured tasks, significantly **broadening task scenarios without requiring human annotations.**.

Moreover, as virtual agents exhibit combined complexities and varying capability dimensions across diverse tasks, we propose a graph-based multi-dimensional evaluation framework, **OmniEval**. Differing from prior rough evaluations, we introduce a graph-based evaluator using subtask-level evaluation functions in OmniBench. Specifically, we design two novel fine-grained metrics to assess agents' performance on graph-structured tasks and their alignment with human logic. Based on OmniBench, we comprehensively evaluate 12 virtual agents, including both open-source and proprietary models, across all 10 capability dimensions as shown in Figure 1, fully reflecting the multi-dimensional capability of the rapidly evolving agent for application.

The performance comparison between various models provides valuable findings for future virtual agent application. Specifically: **1) Existing agents struggle to handle graph-structured tasks**. Compared to tasks with linear structures, the agents fall significantly short when facing graph-structured tasks, with even GPT-4o achieving only 20.5% performance, while humans can reach 80.1%. **2) Task intents are crucial for task planning.** Incorporating task intents provided by OmniBench into the prompt improves agents' long-range planning performance, on average, from 23.5% to 34.2% for two open-source agents. **3) Curriculum learning can better optimize agents**. We find that agents gradually trained on simple tasks before complex ones outperform those trained with naive methods. This

Table 1. Comparison of virtual agent benchmarks across environment, task, and evaluation dimensions. Unlike previous benchmarks, OmniBench features automatic task composition, five-dimensional task complexity, and a 10-capability evaluation framework.

	Environment		Task								Evaluation			
	Interactive	Real-World	# Instance	# Compl. Dimen.	Dyna. Scale	Intent	# Scenario	Demo. Traj.	Construction	Instruction Level	# Cap. Dimen.	Eval. Level	# Eval. Func.	Evaluation Strategy
AitW (Rawles et al., 2024)	✗	✗	30378	-	✗	✗	5	✓	Manual Annotation	High & Low	1	Task	-	Trajectory-based
Mind2Web (Deng et al., 2024)	✗	✗	2350	-	✗	✗	5	✓	Manual Annotation	High	1	Task	-	Trajectory-based
MoTIF (Burns et al., 2022)	✗	✗	756	-	✗	✗	-	✓	Manual Annotation	High & Low	1	Task	-	Trajectory-based
OmniACT (Kapoor et al., 2025)	✗	✗	9802	-	✗	✗	6	✓	Manual Annotation	Low	1	Task	-	Trajectory-based
GUI Odyssey (Lu et al., 2024)	✗	✗	7735	-	✗	✗	6	✓	Manual Annotation	Low	1	Task	-	Trajectory-based
WebArena (Zhou et al., 2023)	✓	✗	812	-	✗	✗	4	✗	Manual Annotation	Low	1	Task	5	Result-based
VisualWebArena (Koh et al., 2024)	✓	✗	910	2	✗	✗	3	✗	Manual Annotation	Low	1	Task	6	Result-based
OSWorld (Xie et al., 2024)	✓	✓	369	-	✗	✗	5	✓	Manual Annotation	Low	1	Task	134	Result-based
Spider2-V (Cao et al., 2024)	✓	✓	494	1	✗	✗	7	✗	Manual Annotation	High & Low	1	Task	151	Result-based
CRAB (Xu et al., 2024a)	✓	✓	100	-	✓	✗	-	✗	Manual Composition	Low	1	Subtask	59	Graph-based
OmniBench (Ours)	✓	✓	36076	5	✓	✓	20	✓	Automatic Composition & Human Verification	High & Low	10	Subtask	255	Graph-based

highlights the systematic definition of tasks in OmniBench.

Furthermore, we fine-tune two open-source agents with different architectures on our synthesized graph-structured task trajectories and find they are more effective in **guiding virtual agents to improve their overall capabilities**, especially with an average 10.35% improvement on tasks requiring sequential decision-making. Additionally, compared to agents trained on manually annotated datasets, our trained agents achieve enhanced performance across diverse benchmarks, with reasoning-rich trajectories, which highlight their wide-ranging applicability and substantial potential.

2 Related Work

Benchmarks for Virtual Agents. Mainstream benchmarks for virtual agents are generally categorized into two types: trajectory-based and result-based. Trajectory-based benchmarks (Rawles et al., 2024; Cheng et al., 2024; Deng et al., 2024) compare agent trajectories to human demonstrations but can be inaccurate due to the existence of multiple valid trajectories. Result-based benchmarks (Xie et al., 2024; Zhou et al., 2023; Cao et al., 2024) focus on the final state of the environment, overlooking the fine-grained evaluation of intermediate processes. More recently, some studies (Shen et al., 2023; Xu et al., 2024a) have introduced graph-based evaluations, which support both multiple feasible trajectories and the assessment of intermediate processes. TASKBENCH (Shen et al., 2023) evaluates agents for task automation, but its simplistic metrics fail to fully utilize the potential of graph structures. CRAB (Xu et al., 2024a) evaluates agents using handcrafted graphs, but it lacks a systematic task analysis, limiting fine-grained capability assessment. Notably, to the best of our knowledge, **OmniBench** is the only scalable benchmark for virtual agents that defines composable task complexity using graphs to evaluate multiple essential capabilities.

3 OmniBench

OmniBench consists of 36k high-quality graph-structured tasks across 10 evaluation dimensions to simulate the way humans perceive the digital world, including planning, decision-making, etc. In this section, we first introduce task graphs of OmniBench and systematically define corresponding task complexities (Section 3.1). Then, we present the bottom-up data collection pipeline for controllably synthesizing tasks (Section 3.2). Additionally, we explain how to control the quality of the synthesized data (Section 3.3). Finally, we showcase the statistics of OmniBench (Section 3.4).

Table 2. Complexity Dimensions and Their Corresponding Levels

Complexity Dimension	Calculation	Easy	Medium	Hard
Dependency Complexity	Number of Edges	≤ 1	$2 \sim 3$	≥ 4
Instruction Complexity	Number of Nodes	≤ 2	$3 \sim 4$	≥ 5
Knowledge Complexity	Number of Application Categories	≤ 1	$2 \sim 3$	≥ 4
Hierarchical Complexity	Depth	≤ 2	$3 \sim 4$	≥ 5
Branching Complexity	Width	≤ 2	$3 \sim 4$	≥ 5

3.1 Task Complexity on Task Graph

In this section, we define task complexity on graphs, which is later constrained in Section 4.2 to construct test tasks for multidimensional capability. We propose a new complexity definition because existing benchmarks (Wang et al., 2024a; Cao et al., 2024; Koh et al., 2024) typically define task complexity based on the number of steps in human demonstrations. However, this approach has two limitations: 1) The inherent subjectivity of human demonstrations makes this definition unreliable; 2) It is one-dimensional and fails to capture the multifaceted complexity of real-world tasks. Inspired by previous works (Xu et al., 2024a; Shen et al., 2023) that represent tasks as graph structures, we introduced the concept of the task graph and systematically defined five fundamental task complexities on the task graph.

Specifically, we define a subtask as a smaller, independent task that contributes to completing a more complex task. Each subtask has input and output resources to constrain the dependencies between them. To formalize this, we assume each subtask as s and define a task graph as $\mathcal{G} = \{S, R\}$, where $S = \{s_1, s_2, \ldots, s_n\}$ is the collection of subtasks, and R is a set of relations $\{(s_a, s_b)\}$ indicating that subtask s_b depends on subtask s_a when the output resources of s_a match the input resources of s_b.

After representing the task as a graph, a natural idea is to define task complexity based on the topology of the task graph. We systematically analyzed five characteristics of the graph and designed a corresponding five-dimensional task complexity. Specifically: **1) Dependency Complexity.** Since each edge in the task graph represents a dependency between subtasks, we define dependency complexity based on the number of edges. **2) Instruction Complexity.** The semantics of a task instruction are composed of all subtasks, with more subtasks leading to more complex instruction se-

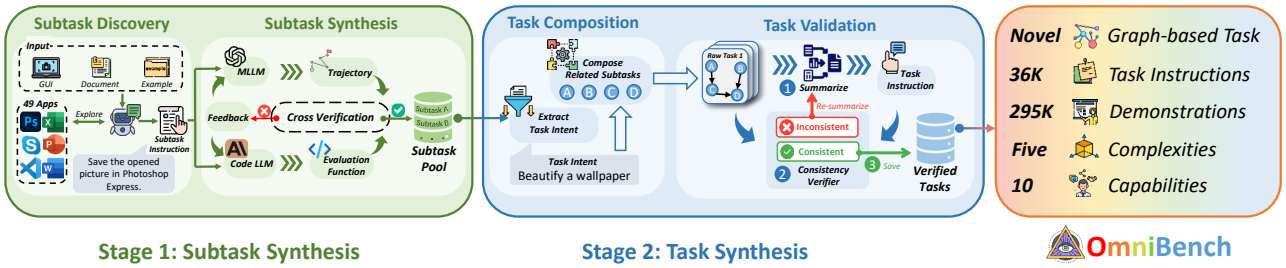

Figure 2. Bottom-up task synthesis pipeline

mantics. Therefore, we define instruction complexity based on the number of nodes. **3) Hierarchical Complexity.** The depth of the task graph represents the number of hierarchical levels in the task structure. Thus, we define hierarchical complexity based on the depth. **4) Branching Complexity.** A wider task graph indicates more branches that can be executed concurrently. Therefore, we define branching complexity based on the width. **5) Knowledge Complexity.** We categorize all 49 applications and define knowledge complexity based on the number of applications from different categories (*e.g.*, multimedia playback, productivity) in the task graph. The detailed categorization is provided in Appendix A.4. The classification criteria for the complexity levels are shown in Table 2.

3.2 Controllable Task Synthesis

Although we define five fundamental task complexities on the task graph, converting task instructions into a graph remains challenging. A straightforward idea is to directly top-down decompose tasks into task graphs. However, this process is typically based on uncontrollable MLLMs or expensive manual efforts. Therefore, to effectively synthesize tasks with controllable complexity, we designed a bottom-up automated task synthesis pipeline, as shown Figure 2.

Overview. The task synthesis pipeline we propose consists of four processes. First, we synthesize a series of simple subtask instructions from the explorable environment. Then, we iteratively synthesize subtask trajectories and evaluation functions. Next, the subtasks are combined into a task bottom-up. Finally, we validate the semantics of the tasks.

Subtask Exploration. We designed an environment containing 49 diverse applications, inspired by OSWorld (Xie et al., 2024), allowing advanced MLLMs to thoroughly explore each application to propose diverse and achievable subtasks. During exploration, documentation and example subtasks for each application are provided to help synthesis. To accurately synthesize the dependencies between subtasks, we provide a predefined resource list, which MLLMs use to determine the input and output resources for each subtask. The implementation details can be found in Appendix B.1.

Subtask Synthesis. We leverage advanced MLLMs to synthesize high-quality subtask samples. To enhance trajectory accuracy, we employ GPT-4o to synthesize demonstration trajectories based on subtask instructions. Our prompts guide the MLLM to describe screenshots for environment understanding, reason explicitly to improve inference, and select actions from the action space. For evaluation functions, we predefine 11 basic APIs to retrieve information such as clicked text, keyboard inputs, and file directory existence. Subsequently, we use Claude-3.5-Sonnet, which excels in the code domain, to compose these basic APIs into evaluation functions for subtasks. To improve the quality of subtask examples, we propose a novel cross-verification algorithm that iteratively refines the synthesized data. The implementation details can be found in Appendix B.2.

Task Composition with Consistent Intent. For high-quality subtask samples that pass cross-verification, we add them to the subtask pool. Directly bottom-up composing these subtasks into a task graph using input and output resources may result in tasks that lack a coherent core goal, such as "open Spotify to play music and use Media Player to play local videos". To avoid synthesizing such low-quality tasks, we extract task intents for the composition scenarios from the subtask pool, such as "deploy a model," as shown in Figure 1. Each task intent involves a group of subtasks, which are then combined into a task graph using input and output resources. Since the bottom-up composition process is rule-based, The synthesis of the task graph is controllable, ensuring that tasks with controllable complexity can be synthesized. Implementation details are in Appendix B.3.

Task Validation. For the task graphs construct through composition, we employ GPT-4o to summarize the task instruction based on the subtask instructions and the graph structure. However, such synthesized instructions may deviate from the original semantics of the task graph, such as losing the graph's nonlinear semantics and degenerating into a simple sequential task. To ensure the synthesis of high-quality graph-structured tasks, we designed a consistency validator to verify the semantic alignment between the task graph and the summarized task instruction. Specifically, GPT-4o determine the dependency for subtasks solely based on the task instruction. If the inferred dependencies match those in the task graph, the instruction passes validation; otherwise, the instruction needs to be re-summarized. Implementation details are in Appendix B.4.

Table 3. Ablation study evaluating the impact of each quality control module on acceptability of the synthesized tasks.

Cross-Verification	Intent Extraction	Consistency Validator	Human Acceptance
✗	✗	✗	41.2%
✗	✓	✓	61.2%
✓	✗	✓	82.7%
✓	✓	✗	86.5%
✓	✓	✓	90.7%

3.3 Quality Control

Since the quality of graph-structured tasks is critical to the accurate evaluation of the virtual agents, we further introduce three designs to enhance the quality of synthesized data: a cross-verification mechanism, an intent extraction module, and a consistency validator. The cross-verification mechanism iteratively optimizes the demonstration trajectories and evaluation functions of subtasks, the intent extraction module ensures that the tasks have coherent goals, and the consistency validator aligns the semantics of the task graph and task instructions. We perform an ablation analysis of these three quality control methods, validating their effectiveness, as shown in Table 3. For each ablation shown in the table, we sampled 400 task graphs and calculated the average acceptance by three specially trained annotators. The experiment shows that removing any quality control module decreases human acceptance, with the removal of the cross-verification algorithm resulting in the largest drop to 61.2%.

3.4 OmniBench Statistics

OmniBench comprises a total of 36,076 task instances, spanning across 20 common interactive scenarios and involving 49 diverse applications, as illustrated in Appendix A.4. In Section 3.1, we introduced the five-dimensional complexity metrics for each task, with the distribution of different complexity levels for each dimension shown in Figure 4.

Statistics	Value
Total Tasks	36076 (100%)
- Network-dependent Real-world Tasks	16614 (46.05%)
- Network-independent Local Tasks	19462 (53.95%)
- Avg. Number of Used App Per Task	2.21
Task Instruction	
- Avg. Words of High-level Instruction	51.7
- Avg. Words of Low-level Instruction	237.9
Total Task Scenarios	20
Total Subtasks	255

Figure 3. Statistics of OmniBench

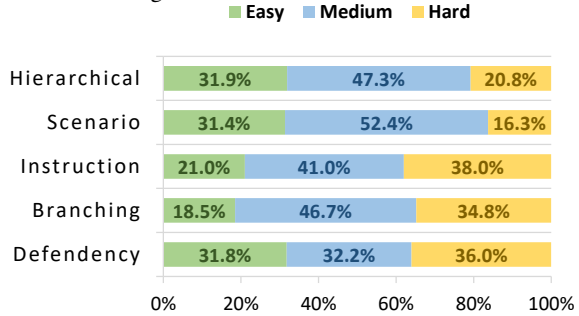

Figure 4. Distribution of Five-Dimensional Complexity

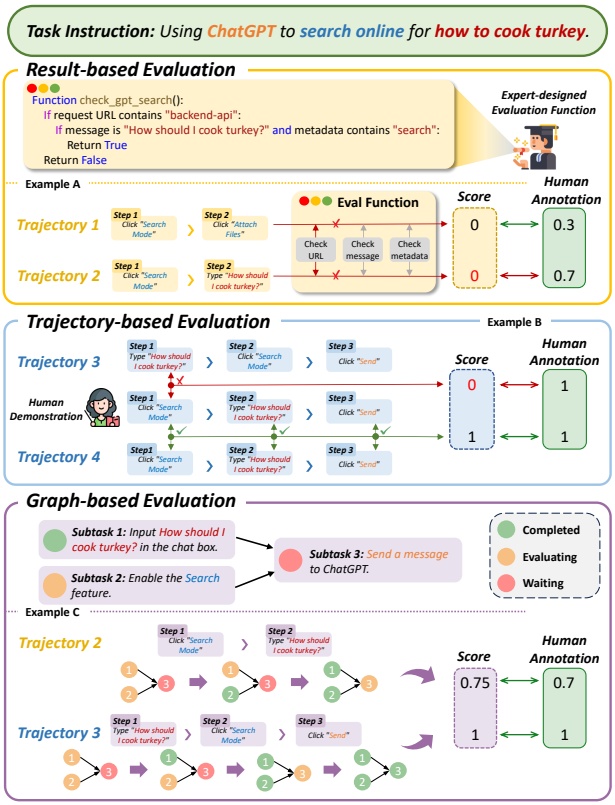

Figure 5. Comparison of mainstream virtual agent evaluation strategies with the evaluation strategy we propose.

4 OmniEval

To comprehensively analyze the limited capabilities of existing agents, we further propose OmniEval, a graph-based multi-dimensional evaluation framework. In this section, we first introduce a graph-based evaluator with two novel metrics for fine-grained and diverse evaluation (Section 4.1). Then, we describe the construction of test tasks designed to evaluate 10 distinct capabilities by constraining task complexity (Section 4.2).

4.1 Graph-based Evaluator

Currently, most benchmarks still evaluate agents in a coarse-grained and unreasonable paradigm. Specifically, **result-based** evaluations (Xie et al., 2024; Zhou et al., 2023) consider whether the final environment state aligns with expectations, lacking fine-grained intermediate evaluation. As shown in Figure 5, although both trajectories ultimately fail to complete the task, the progress of trajectory 2 is superior to trajectory 1, and they should not simply be categorized as failures. While **trajectory-based** evaluations (Rawles et al., 2024; Lu et al., 2024) compare the agents' predicted actions to human demonstrations at each step, they overlook multiple feasible trajectories. As shown in Figure 5, both trajectories can accomplish the task, but trajectory 3 is deemed a failure because it does not align with the demonstration trajectory, which is unreasonable.

Considering the limitations of these two evaluation

paradigms, we introduce a **graph-based** multi-metric evaluator inspired by previous research (Xu et al., 2024a), as shown in Figure 5. Specifically, we define three evaluation states for each node on the task graph: Completed, Evaluating, and Waiting. Initially, when the evaluator is set up, nodes with an in-degree of 0 are marked as Evaluating, while the remaining nodes are marked as Waiting. After the agent executes each action, it checks whether all Evaluating nodes have been completed. Once a node is completed, it is marked as Completed, and new Evaluating nodes are added according to the topological order. We set a maximum number of steps N, and if the agent does not complete any subtasks within N steps, the entire task is considered a failure.

Additionally, to fully leverage the potential of the graph-based evaluator, we have designed two novel graph-based metrics. Traditional metrics fail to evaluate intermediate processes and alignment with human operational logic. For example, common metrics such as **Success Rate (SR)** (Xie et al., 2024) focus on task outcomes rather than the process, while **Action Match Score (AMS)** (Li et al., 2020) treats action sequences as strings and compares the similarity between human demonstrations and agent predictions, rather than their logical similarity. To comprehensively quantify agent performance on the task graph, we propose two novel metrics inspired by its topology. The **Coverage Rate (CR)** assesses agent progress on the task graph, while the **Logical Consistency (LC)** reflects the similarity in operational logic between agents and humans.

Coverage Rate (CR). It evaluates an agent's progress on a task graph by weighting subtasks based on their depth, where deeper subtasks are assigned higher weights due to their increased number of prerequisite subtasks.

Referring to the relevant definitions in Section 3.1, let $d(s_i)$ denote the depth of subtask s_i. The weight $w(s_i)$ is:

$$w(s_i) = \frac{d(s_i)}{\sum_{j=1}^{n} d(s_j)}.$$

The Coverage Rate is then:

$$CR = \frac{\sum_{i=1}^{n} w(s_i) \cdot \mathbb{I}(s_i)}{\sum_{i=1}^{n} w(s_i)},$$

where $\mathbb{I}(s_i) = 1$ if subtask s_i is completed, and 0 otherwise. This metric emphasizes deeper, more complex subtasks, providing a refined measure of agent performance.

Logical Consistency (LC). It quantifies the operational logic similarity between agents and humans. This metric is motivated by the observation that humans prefer to complete all possible subtasks within an application before switching to another, unless necessary. It is computed as the ratio of the agent's Coherency Score (CS) to the maximum possible CS:

$$LC = \frac{CS_{agent}}{CS_{max}},$$

where CS quantifies the coherence of the subtask sequence. For each pair of adjacent subtasks (s_i, s_{i+1}) in sequence, CS increases by 1 if both subtasks belong to the same application. CS_{agent} is the coherence score calculated from the executing subtask sequence, and CS_{max} is the maximum possible coherence score calculated among all topological sequences.

4.2 Evaluation Strategy

Currently, there is limited discussion on the categorization of capabilities in the virtual agent field. The previous mainstream classifications (Lin et al., 2024; Gou et al., 2024) typically divided the capabilities into two simple categories: grounding and planning. However, this simple classification is quite coarse and does not take into account other essential and fine-grained capabilities for agents, such as decision-making and instruction understanding. To address this, we propose 10 fine-grained capabilities, derived from five categories that we consider essential, with each category contributing two capabilities for agents. Specific test tasks are constructed for each capability based on the combination of five complexity dimensions, as shown in Figure 1.

Taking long-range planning capability as an example, we categorize tasks with higher dependency complexity and hierarchical complexity as test tasks for this capability. This is because higher dependency complexity means the task involves more dependencies, requiring stronger planning capability. Meanwhile, higher hierarchical complexity indicates the task has deeper levels, which places higher demands on long-sequence processing capability. Therefore, we select tasks with dependency complexity and hierarchical complexity at the hard level as test tasks for long-sequence reasoning capability. For the test tasks of these 10 capabilities, we engaged professionally trained annotators to filter and construct high-quality test data. The specific definitions and corresponding explanations for the other 9 capabilities can be found in Appendix D.1.

5 Experiments

In this section, we first introduce the experimental setup (Section 5.1). Then, we comprehensively compare the differences in capabilities across various models on OmniBench, along with several key findings (Section 5.2). Finally, we provide an in-depth analysis of the reasons behind the poor performance and methods for performance improvement (Section 5.3).

5.1 Experimental Setup

Settings. We evaluate various models including MLLMs and Virtual Agents on OmniBench. For all virtual agents, we use the default prompt provided by each agent for inference,

Table 4. Performance of models on OmniBench. For each capability, we use the CR metric on test tasks for quantification. Abbreviations adopted: PP for Parallel Planning; LRP for Long Range Planning; CDDK for Cross-Domain Decision-Making; SDK for Sequential Decision-Making; SI for Subtask Identification; DI for Dependency Identification; LSR for Long Sequence Reasoning; LIF for Long Instruction Following; DSK for Domain-Specific Knowledge; CDK for Cross-Domain Knowledge. * denotes using GPT-4o as the planner.

	Overall		**Planning**		**Decision-making**		**Instruction Understanding**		**Long Context**		**Generalist Knowledge**	
	CR	*LC*	*PP*	*LRP*	*CDDK*	*SDK*	*SI*	*DI*	*LSR*	*LIF*	*DSK*	*CDK*
Human	80.1	92.8	80.1	76.9	91.9	93.0	69.1	72.1	79.5	66.1	89.4	71.5
Open-source Multimodal Large Language Models (A11Y+Screenshot)												
Qwen2-VL-7B-Instruct (Wang et al., 2024b)	14.8	9.0	15.5	13.5	16.5	17.8	14.1	13.8	14.7	12.4	15.8	13.8
InternVL2-8B (Chen et al., 2024)	14.2	13.0	15.0	13.5	16.2	16.9	12.1	12.9	15.8	11.7	15.8	12.4
InternVL2.5-8B (Chen et al., 2024)	17.4	18.8	18.2	16.7	19.6	21.5	16.4	15.3	15.8	15.4	19.0	16.3
Closed-source Multimodal Large Language Models (A11Y+Screenshot)												
Qwen-VL-Max (Bai et al., 2023)	18.4	23.3	18.7	19.4	19.6	23.3	15.0	16.7	18.3	16.1	19.6	17.3
Gemini-2.0-Flash	25.9	38.0	24.8	24.6	31.5	33.2	22.5	22.5	25.7	21.9	27.8	24.8
Claude-3.5-Sonnet	27.6	35.0	30.5	24.7	32.0	31.3	24.5	25.0	26.6	23.5	33.4	24.5
GPT-4o (Hurst et al., 2024)	**38.7**	**49.0**	**38.4**	**37.8**	**43.2**	**49.4**	**30.6**	**35.5**	**42.7**	**32.2**	**43.2**	**34.2**
Visual Digital Agents (Screenshot)												
Aguvis-7B (Xu et al., 2024b)	22.9	27.1	21.2	23.5	25.5	28.1	20.2	20.0	22.8	20.1	26.3	21.6
OS-Atlas-Pro-4B (Wu et al., 2024)	19.1	23.9	20.6	17.6	22.9	23.6	15.0	17.7	18.7	15.9	22.0	16.8
ShowUI-2B* (Lin et al., 2024)	23.2	24.6	23.2	23.1	26.3	26.6	21.5	20.3	24.7	20.4	24.8	20.7
OS-Atlas-Base-4B* (Wu et al., 2024)	22.2	23.8	23.2	21.9	26.2	25.6	19.4	19.5	23.5	20.0	23.4	19.3
UGround-V1-7B* (Gou et al., 2024)	25.0	26.3	25.7	25.1	30.6	31.4	21.5	21.3	24.8	21.3	27.2	21.5
Supervised Fine-Tuning Agents (Screenshot)												
Omni-OS-Atlas-Base-4B (Ours)	29.7	30.1	24.2	**33.0**	34.9	35.3	**28.7**	24.2	27.9	26.5	33.8	**28.2**
Omni-UGround-V1-7B (Ours)	**34.4**	**37.4**	**33.2**	31.3	**43.1**	**42.4**	21.9	**35.0**	**40.3**	**31.7**	**36.7**	27.6

if available. If models do not provide prompts for agent tasks, we use a unified prompt designed by us. We also report results trained on OmniBench data for some selected models. All experiments are conducted with NVIDIA A100 80G GPUs.

Baselines. We conducted a comprehensive evaluation of the following four types of models. The specific details about the baselines can be found in Appendix D.2.2.

5.2 Main Results

OmniEval. Before delving deeper into the concrete evaluation results, we first compare the alignment between OmniEval and human evaluation for agent tasks. Specifically, we randomly sampled 50 trajectories from all models to calculate the correlation between OmniEval and human evaluation. Each trajectory was scored by ten specially trained annotators, who referenced the task instructions to assign task completion scores and logical alignment scores from {0%, 10%, ..., 90%, 100%}. The final human evaluation score was determined by averaging the scores given by the three annotators. In Figure 6, we present the Pearson correlation between OmniEval and human evaluation. The results indicate a strong correlation between OmniEval and human evaluation.

Challenges in Handling Graph-structured Tasks. We compared the performance differences of each model on chain-structured tasks and graph-structured tasks, as shown in Figure 7. To eliminate the influence of other factors, we utilized OmniBench's controllable task synthesis mechanism to construct a set of chain-structured and graph-structured tasks, each with the same number of nodes and edges. All tasks belong to the same knowledge domain and share the same level of knowledge complexity. For detailed experimental settings, refer to Appendix.

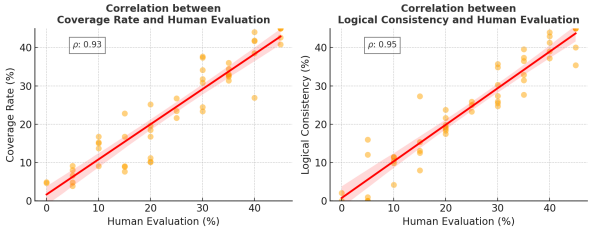

Figure 6. Correlation between Coverage Rate and Logical Consistency with Human Evaluation.

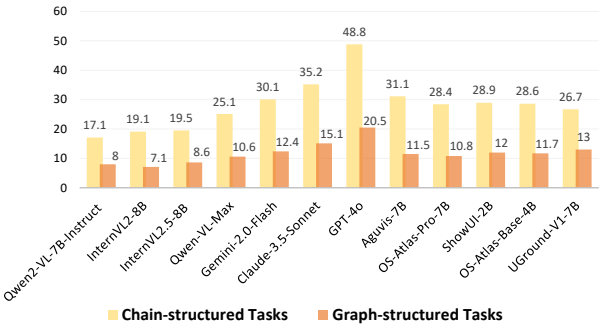

Figure 7. Performance comparison of various models on chain-structured tasks and graph-structured tasks.

The experiment shows that, even the most advanced agent , GPT-4o (with A11Y), achieves only 20.5% performance on graph-structured tasks, which is far below the human performance benchmark of 80.1%. This phenomenon can be attributed to the fact that most existing agents are predominantly fine-tuned on chain-structured tasks, which may

result in their tendency to interpret graph-structured tasks as linear. Such misinterpretation can significantly impair the agents' capability to accurately identify the dependency relationships between subtasks, ultimately leading to task execution failures.

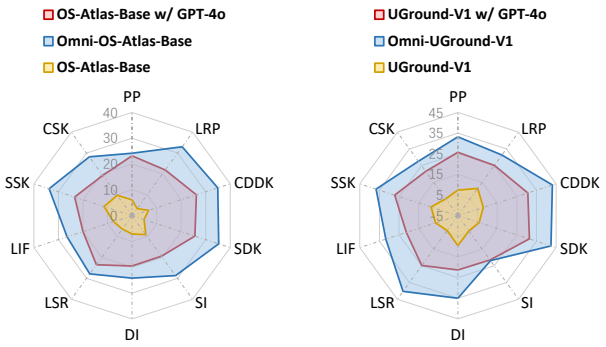

Figure 8. The fine-tuning performance on two different backbones.

Effectiveness of Graph-Structured Task Trajectories. In our fine-tuning experiments with two open-source agents of different architectures, we observed that the graph-structured task trajectories synthesized by our approach were particularly effective in guiding the models to enhance their various capabilities, as shown in Figure 8. Notably, the Sequential Decision-Making capability showed significant improvement, which we attribute to the multi-branch and long-sequence characteristics inherent in the task graph. The specific fine-tuning details are provided in the Appendix.

5.3 In-Depth Analysis

In this section, we present a comprehensive analysis of the experimental results obtained from evaluating mainstream agents on our test set and fine-tuning agents with various model architectures on our training set. Our analysis reveals several critical insights into the strengths and limitations of current agents, particularly in handling graph-structured tasks and compositional reasoning. Below, we discuss the key findings in detail.

Importance of Task Intents for Planning. Our experiments also highlight the critical role of task intents in enhancing agents' planning capabilities. By incorporating intents into the prompt, we observed a marked improvement in the agents' performance on tasks designed to evaluate planning (i.e., Long-Range Planning and Parallel Planning), as shown in Figure 9. The results suggest that explicit high-level guidance can significantly enhance the agents' capability to perform complex long-term planning.

Error Analyse. In this section, we delve into the analysis of errors encountered during the OmniBench evaluation, a key aspect for understanding the capabilities and limitations of virtual agents. This analysis aims not only to identify the current shortcomings of the agents but also to inform future improvements in their design and training. We carefully examine 100 randomly sampled error instances from the

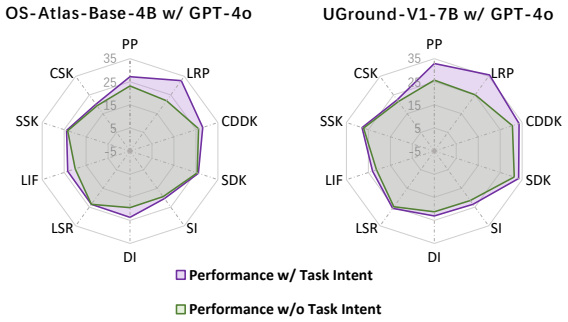

Figure 9. Comparison of the impact of task intent on various capabilities of the two agents.

OmniBench evaluations. These instances are analyzed by expert annotators who identify the root causes of mispredictions based on their knowledge. Specifically, there are five types of errors: **1) Instruction Understanding Errors.** We observed that 23 failures were due to the agent's misunderstanding of the instructions. For example, it overlooked the final save file operation requested in the image editing instruction. **2) Lack of Knowledge.** We found that 21 failures were due to agents lacking knowledge of the target application, such as being unfamiliar with how to create a reference list in Zotero. **3) Environment Error.** We observed that a small number of errors were caused by environmental interference, such as network delays. **4) Grounding Error.** We found that task failures may also occur due to the model's lack of grounding ability, meaning the agent knows the target to click next but locates it in the wrong position. **5) Falsely Completed.** Finally, the agent may fail the task by incorrectly assuming it is complete, which could be due to its weak contextual memory capabilities. The distribution of these errors is shown in Figure 10, and a selection of five notable cases, along with detailed analyses, can be found in Appendix E.

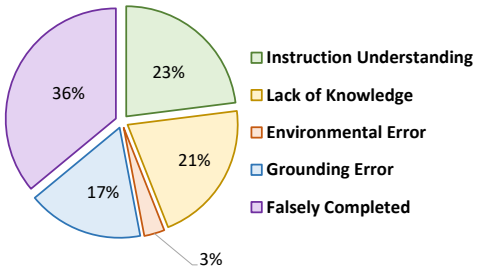

Figure 10. Distribution of Five Major Errors in 100 Failure Examples.

6 Conclusion

In conclusion, we introduced **OmniBench**, a graph-based benchmark that addresses the limitations of existing evaluation frameworks by enabling controllable task complexity through automated subtask composition. Along with **OmniEval**, a multidimensional evaluation framework, we evaluate virtual agents across 10 capabilities. Our results show that training on this data improves agent generalization, and our evaluations provide valuable insights into the

strengths and areas for improvement in MLLM-based virtual agents.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

References

Bai, J., Bai, S., Yang, S., Wang, S., Tan, S., Wang, P., Lin, J., Zhou, C., and Zhou, J. Qwen-vl: A versatile vision-language model for understanding, localization, text reading, and beyond. *arXiv preprint arXiv:2308.12966*, 2023.

Burns, A., Arsan, D., Agrawal, S., Kumar, R., Saenko, K., and Plummer, B. A. A dataset for interactive vision-language navigation with unknown command feasibility. In *European Conference on Computer Vision*, pp. 312–328. Springer, 2022.

Cao, R., Lei, F., Wu, H., Chen, J., Fu, Y., Gao, H., Xiong, X., Zhang, H., Mao, Y., Hu, W., et al. Spider2-v: How far are multimodal agents from automating data science and engineering workflows? *arXiv preprint arXiv:2407.10956*, 2024.

Chen, Z., Wu, J., Wang, W., Su, W., Chen, G., Xing, S., Zhong, M., Zhang, Q., Zhu, X., Lu, L., et al. Internvl: Scaling up vision foundation models and aligning for generic visual-linguistic tasks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 24185–24198, 2024.

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

Deng, X., Gu, Y., Zheng, B., Chen, S., Stevens, S., Wang, B., Sun, H., and Su, Y. Mind2web: Towards a generalist agent for the web. *Advances in Neural Information Processing Systems*, 36, 2024.

Gao, M., Bu, W., Miao, B., Wu, Y., Li, Y., Li, J., Tang, S., Wu, Q., Zhuang, Y., and Wang, M. Generalist virtual agents: A survey on autonomous agents across digital platforms. *arXiv preprint arXiv:2411.10943*, 2024.

Ge, Z., Li, J., Pang, X., Gao, M., Pan, K., Lin, W., Fei, H., Zhang, W., Tang, S., and Zhuang, Y. Iris: Breaking gui complexity with adaptive focus and self-refining. *arXiv preprint arXiv:2412.10342*, 2024.

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

Hu, S., Ouyang, M., Gao, D., and Shou, M. Z. The dawn of gui agent: A preliminary case study with claude 3.5 computer use. *arXiv preprint arXiv:2411.10323*, 2024.

Hurst, A., Lerer, A., Goucher, A. P., Perelman, A., Ramesh, A., Clark, A., Ostrow, A., Welihinda, A., Hayes, A., Radford, A., et al. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*, 2024.

Kapoor, R., Butala, Y. P., Russak, M., Koh, J. Y., Kamble, K., AlShikh, W., and Salakhutdinov, R. Omniact: A dataset and benchmark for enabling multimodal generalist autonomous agents for desktop and web. In *European Conference on Computer Vision*, pp. 161–178. Springer, 2025.

Koh, J. Y., Lo, R., Jang, L., Duvvur, V., Lim, M. C., Huang, P.-Y., Neubig, G., Zhou, S., Salakhutdinov, R., and Fried, D. Visualwebarena: Evaluating multimodal agents on realistic visual web tasks. *arXiv preprint arXiv:2401.13649*, 2024.

Li, Y., He, J., Zhou, X., Zhang, Y., and Baldridge, J. Mapping natural language instructions to mobile ui action sequences. *arXiv preprint arXiv:2005.03776*, 2020.

Lin, K. Q., Li, L., Gao, D., Yang, Z., Wu, S., Bai, Z., Lei, W., Wang, L., and Shou, M. Z. Showui: One vision-language-action model for gui visual agent. *arXiv preprint arXiv:2411.17465*, 2024.

Lu, Q., Shao, W., Liu, Z., Meng, F., Li, B., Chen, B., Huang, S., Zhang, K., Qiao, Y., and Luo, P. Gui odyssey: A comprehensive dataset for cross-app gui navigation on mobile devices. *arXiv preprint arXiv:2406.08451*, 2024.

Rawles, C., Li, A., Rodriguez, D., Riva, O., and Lillicrap, T. Androidinthewild: A large-scale dataset for android device control. *Advances in Neural Information Processing Systems*, 36, 2024.

Shen, J., Jain, A., Xiao, Z., Amlekar, I., Hadji, M., Podolny, A., and Talwalkar, A. Scribeagent: Towards specialized web agents using production-scale workflow data. *arXiv preprint arXiv:2411.15004*, 2024.

Shen, Y., Song, K., Tan, X., Zhang, W., Ren, K., Yuan, S., Lu, W., Li, D., and Zhuang, Y. Taskbench: Benchmarking large language models for task automation. *arXiv preprint arXiv:2311.18760*, 2023.

Wang, L., Deng, Y., Zha, Y., Mao, G., Wang, Q., Min, T., Chen, W., and Chen, S. Mobileagentbench: An efficient and user-friendly benchmark for mobile llm agents. *arXiv preprint arXiv:2406.08184*, 2024a.

Wang, P., Bai, S., Tan, S., Wang, S., Fan, Z., Bai, J., Chen, K., Liu, X., Wang, J., Ge, W., Fan, Y., Dang, K., Du, M., Ren, X., Men, R., Liu, D., Zhou, C., Zhou, J., and Lin, J. Qwen2-vl: Enhancing vision-language model's perception of the world at any resolution. *arXiv preprint arXiv:2409.12191*, 2024b.

Wu, Z., Wu, Z., Xu, F., Wang, Y., Sun, Q., Jia, C., Cheng, K., Ding, Z., Chen, L., Liang, P. P., et al. Os-atlas: A foundation action model for generalist gui agents. *arXiv preprint arXiv:2410.23218*, 2024.

Xie, T., Zhang, D., Chen, J., Li, X., Zhao, S., Cao, R., Hua, T. J., Cheng, Z., Shin, D., Lei, F., et al. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *arXiv preprint arXiv:2404.07972*, 2024.

Xu, T., Chen, L., Wu, D.-J., Chen, Y., Zhang, Z., Yao, X., Xie, Z., Chen, Y., Liu, S., Qian, B., et al. Crab: Cross-environment agent benchmark for multimodal language model agents. *arXiv preprint arXiv:2407.01511*, 2024a.

Xu, Y., Wang, Z., Wang, J., Lu, D., Xie, T., Saha, A., Sahoo, D., Yu, T., and Xiong, C. Aguvis: Unified pure vision agents for autonomous gui interaction. *arXiv preprint arXiv:2412.04454*, 2024b.

Yao, S., Chen, H., Yang, J., and Narasimhan, K. Webshop: Towards scalable real-world web interaction with grounded language agents. *Advances in Neural Information Processing Systems*, 35:20744–20757, 2022.

Zhou, S., Xu, F. F., Zhu, H., Zhou, X., Lo, R., Sridhar, A., Cheng, X., Ou, T., Bisk, Y., Fried, D., et al. Webarena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*, 2023.

Table of Contents in Appendix

A Environment Setup

Inspired by OSWorld (Xie et al., 2024), we designed an interactive Windows environment based on a virtual machine. We utilized Windows 11 as the system image and employed VMware Workstation 17 Pro (version 17.5.1) as the virtualization platform.

A.1 Environment Infrastructure

A.2 Observation Space

A.3 Action Space

A.4 Categorization of Applications

The 49 applications in the environment can be categorized into the following seven types based on domain knowledge: Social Communication, Creative Design, Multimedia Playback, Multimedia Editing, Office, Utility Tools, and Programming. The applications corresponding to each category are shown in Table 5.

Category	Applications
Social Communication (4)	Zoom Workplace, Skype, People, Mail
Multimedia Playback (4)	Media Player, Spotify, Photos, TuneIn
Multimedia Editing (6)	Adobe Photoshop Express, Microsoft Clipchamp, paint.net, Openshot, Handbrake, Paint
Office (3)	Word, PowerPoint, Excel
Utility Tools (10)	Calculator, 7-Zip, PDF24, Power Automate, Wikipedia, BreeZip, Maps, Calendar, Zotero, DeepL
Programming (3)	Visual Studio Code, Cursor, Windows PowerShell ISE
System Management (4)	File Explorer, Settings, Control Panel, Microsoft Store
Web Browsing (2)	Google Chrome, Microsoft Edge
Screen Capture (4)	Record Screen, Snipping Tool, OBS Studio, ShareX
Task Management (3)	Microsoft To Do, Todoist, Notion
Note management (4)	Evernote, OneNote, Sticky Notes, Sticky Notes (New)
Lifestyle (2)	Recipe Keeper, paisa

Table 5. Categorization of applications in the environment.

A.5 Definition of Scenarios

B Data Collection

B.1 Subtask Discovery

To facilitate comprehensive subtask generation, we constructed a dynamic environment incorporating 49 heterogeneous applications, drawing inspiration from OSWorld (Xie et al., 2024). This environment allows MLLMs to interact with diverse functionalities, systematically analyze operational constraints, and propose well-structured subtasks tailored to each application's context. To support this process, we provide detailed API documentation, user manuals, and curated example subtasks, ensuring that MLLMs can infer practical usage scenarios. Additionally, our approach emphasizes dependency modeling by introducing a structured resource framework, where each subtask must explicitly define its required inputs and expected outputs. This predefined resource list serves as a guiding constraint, enabling MLLMs to reason about inter-subtask dependencies, avoid conflicts, and ensure smooth task execution. By leveraging this controlled exploration, MLLMs can generate coherent and executable subtask sequences that align with real-world application workflows.

B.2 Subtask Synthesis

Trajectory Synthesis.

Evaluation Synthesis.

Cross-Verification. We design a cross-verification algorithm to optimize synthesized subtasks. Specifically, the algorithm performs N iterations, where an MLLM and a Code LLM specifically synthesize trajectories and evaluation functions based on the feedback from the previous iteration. The evaluation functions provide detailed failure feedback (*e.g.*, failing to click the "save" button), while the trajectories incorporate environmental state feedback, enabling an iterative refinement process for both trajectories and evaluation functions. To ensure the evaluation functions maintain sufficient discriminatory power, any function that passes cross-verification is further tested by evaluating three additional trajectories from other tasks.

B.3 Task Composition with Consistent Intent

Task Composition with Consistent Intent. To construct meaningful task graphs, we first curate a subtask pool by filtering out low-quality samples through a cross-verification process. A naïve bottom-up approach that directly connects subtasks based on shared input-output resources may generate incoherent tasks that lack a clear, unified goal. For example, an arbitrary combination might result in a task that simultaneously plays media from different applications without a meaningful relationship. To mitigate this issue, we explicitly extract overarching task intents from the subtask pool, such as "deploy a model," as illustrated in Figure 1. Each intent acts as an anchor, grouping subtasks that contribute to the same high-level objective. The task graph is then constructed by linking these subtasks based on resource dependencies. Since the bottom-up composition follows predefined rules, we maintain strict control over the synthesis process, ensuring that the resulting task graphs exhibit logical coherence and adjustable complexity. This structured approach prevents the generation of ill-formed tasks while supporting diverse and meaningful compositions.

B.4 Task Validation

To generate concise yet accurate task instructions for graph-structured tasks, we utilize GPT-4o to synthesize a task description by integrating subtask instructions and the structural information of the task graph. However, direct summarization may introduce inconsistencies, such as misrepresenting the graph's dependency structure or oversimplifying

it into a linear sequence, thereby distorting the original task complexity. To address this issue, we designed a consistency validation mechanism to assess the fidelity of the summarized instruction. In this process, GPT-4o is prompted to infer subtask dependencies purely from the generated task instruction, without access to the original task graph. The inferred dependency structure is then compared against the ground-truth graph. If discrepancies arise—such as missing parallel execution paths or incorrect sequencing—the instruction is flagged for revision and must be re-summarized to better preserve the intended graph structure. This validation ensures that the final task instruction accurately reflects the hierarchical and branching nature of the original task graph, improving the clarity and correctness of the synthesized tasks.

C Details of OmniBench

C.1 Subtask Format

C.2 Task Format

C.3 Task Graph Format

D Details of OmniEval

D.1 Capability Design

We construct 10 specialized sets of test tasks based on the five-dimensional complexity, each set designed to evaluate one of the 10 capabilities required for agents to fulfill user requests. The number of solid stars represents the level of complexity, while zero stars indicate that the complexity in that dimension can be arbitrary.

	Dependency	Instruction	Hierarchical	Branching	Knowledge
Parallel Planning	★★★	☆☆☆	☆☆☆	★★★	☆☆☆
Long-Range Planning	★★★	☆☆☆	★★★	☆☆☆	☆☆☆
Long-Sequence Reasoning	☆☆☆	★★★	★★★	☆☆☆	☆☆☆
Long Instruction Following	☆☆☆	☆☆☆	★☆☆	★★★	☆☆☆
Sequential Decision-Making	☆☆☆	☆☆☆	★★★	★★★	☆☆☆
Cross-Scenario Decision-Making	☆☆☆	☆☆☆	☆☆☆	★★★	★★★
Subtask Identification	★☆☆	★★★	☆☆☆	☆☆☆	☆☆☆
Dependency Identification	★★★	★☆☆	☆☆☆	☆☆☆	☆☆☆
Cross-Domain Knowledge	☆☆☆	☆☆☆	☆☆☆	☆☆☆	★★★
Domain-Specific Knowledge	☆☆☆	☆☆☆	☆☆☆	☆☆☆	★☆☆

Table 6. Constraining five fundamental task complexities (top) to construct test tasks across ten capability dimensions (left).

D.2 Experiment Setup

D.2.1 Settings

For the evaluation phase, we follow the practices of works such as Aguvis and UGround, ensuring consistency in preprocessing and maintaining comparability across different models. Specifically, we standardize the image resolution by scaling all input images to 1024×1024 pixels. This resolution is chosen to balance computational efficiency and visual detail, ensuring that models can effectively process graphical user interfaces (GUIs) without excessive memory overhead or loss of important information. The decision to use a fixed resolution is motivated by several key factors. First, many modern multimodal models and virtual agents, including those designed for GUI interaction, are trained on datasets with varying resolutions. Standardizing the input resolution reduces inconsistencies that may arise due to different input scales, helping models generalize better across diverse GUI layouts. Additionally, this resolution aligns with commonly used image sizes in recent benchmarks, facilitating direct comparison with prior works. Moreover, scaling images to 1024×1024 ensures that finer details within the GUI, such as text labels, buttons, and icons, remain distinguishable without introducing excessive noise or aliasing effects. Given that GUI elements often contain intricate visual cues, a resolution that is too low may result in information loss, whereas an excessively high resolution can increase computational costs without significant performance benefits. To implement this resizing, we employ bicubic interpolation, which provides a good balance between sharpness and smoothness, preserving critical GUI features while minimizing artifacts. We also ensure that aspect ratios are maintained whenever possible by applying padding techniques if necessary, preventing unintended distortions that could affect model predictions. By adopting this resolution standard, we align our evaluation methodology with established works like Aguvis and UGround, fostering reproducibility and enabling fair comparisons in GUI-based task assessments.

D.2.2 Baselines

We conducted a comprehensive evaluation of the following four types of models:

1) Closed-source MLLMs: These include GPT-4o (Hurst et al., 2024), Qwen-VL-Max (Bai et al., 2023), Claude-3.5-Sonnet, and Gemini-2.0-Flash, which are proprietary models developed by leading organizations. These models are selected due to their state-of-the-art performance on a wide range of multimodal tasks, including vision-language understanding, reasoning, and instruction following. Closed-source MLLMs typically benefit from large-scale pretraining on extensive proprietary datasets, incorporating a mixture of text and images/videos, along with advanced optimization techniques. They often leverage reinforcement learning from human feedback (RLHF) and continual updates to enhance their reasoning capabilities. These models are accessible via APIs, which enforce constraints on inference settings such as token limits, response latency, and proprietary decoding strategies. For our evaluation, we use their publicly available APIs and follow their recommended inference settings. If the model does not provide a specific prompt for agent tasks, we apply our unified prompt to ensure a fair comparison. Additionally, since these models lack direct access to GUI-specific training data, we evaluate

their adaptability by providing structured information about the interface components.

2) Open-source MLLMs: We also evaluate open-source models such as Qwen2-VL-7B-Instruct (Wang et al., 2024b), InternVL2-8B (Chen et al., 2024), and InternVL2.5-8B (Chen et al., 2024), which are widely recognized in the research community for their flexibility and strong performance. These models have been fine-tuned on large-scale multimodal datasets and provide strong generalization across diverse vision-language tasks. Open-source MLLMs offer several advantages, including transparency in training methodologies, customizability for domain-specific fine-tuning, and community-driven improvements. Unlike closed-source models, researchers can inspect their architectures, modify their training pipelines, and deploy them on local hardware without API restrictions. We use their fine-tuned weights and apply our unified prompt for tasks where no default prompt is provided. Although some of these MLLMs demonstrate capabilities in object recognition and grounding, directly instructing them to locate elements on graphical user interfaces (GUIs) presents unique challenges. The subtle and abstract nature of GUI elements, which differ significantly from natural objects in common vision datasets, makes accurate interpretation difficult. To mitigate this issue, we provide A11Y (accessibility) metadata, such as element descriptions and structural information, to aid inference.

3) Virtual Agents: In addition to MLLMs, we also evaluate various virtual agents, such as Aguvis (Xu et al., 2024b), OS-Atlas (Wu et al., 2024), and ShowUI (Lin et al., 2024). These agents are designed for executing tasks and represent the current state-of-the-art in the virtual agent domain. Unlike MLLMs, which primarily operate as general-purpose multimodal models, virtual agents are often specialized for task execution. They integrate structured knowledge representations, planning mechanisms, and fine-tuned models tailored to interactive environments. Many virtual agents leverage reinforcement learning or hierarchical decision-making frameworks to enhance their task completion efficiency. Some also incorporate retrieval-based techniques to access domain-specific knowledge dynamically. Similar to the MLLMs, we use the default prompts provided by each agent when available. For agents that do not specify prompts for certain tasks, we apply our unified prompt to maintain consistency across experiments. Given that virtual agents typically rely on structured inputs rather than free-form multimodal understanding, we evaluate their ability to process GUI environments by simulating real-world task execution scenarios.

4) Supervised Fine-Tuning Agents: We selected two backbones, OS-Atlas-Base-4B (Wu et al., 2024) and UGround-V1-7B (Gou et al., 2024), with different architectures and fine-tuned them using the synthetic data from OmniBench. Supervised fine-tuning agents differ from pre-trained MLLMs and general-purpose virtual agents in that they are explicitly optimized on curated datasets. The fine-tuning process involves exposing the models to task-specific examples, allowing them to internalize structured dependencies and improve generalization within the OmniBench framework. This approach enables agents to learn robust action sequences, refine their perception of GUI elements, and enhance their decision-making accuracy. By leveraging synthetic data, we ensure that these agents develop a structured understanding of GUI tasks while minimizing biases inherent in real-world datasets. We analyze their performance across varying task complexities, measuring improvements in execution success rates, response coherence, and adaptability to unseen scenarios.

D.3 Evaluation Details

For the evaluation phase, we follow the practices of works such as Aguvis and UGround, ensuring consistency in preprocessing and maintaining comparability across different models. Specifically, we standardize the image resolution by scaling all input images to 1024×1024 pixels. This resolution is chosen to balance computational efficiency and visual detail, ensuring that models can effectively process graphical user interfaces (GUIs) without excessive memory overhead or loss of important information. The decision to use a fixed resolution is motivated by several key factors. First, many modern multimodal models and virtual agents, including those designed for GUI interaction, are trained on datasets with varying resolutions. Standardizing the input resolution reduces inconsistencies that may arise due to different input scales, helping models generalize better across diverse GUI layouts. Additionally, this resolution aligns with commonly used image sizes in recent benchmarks, facilitating direct comparison with prior works. Moreover, scaling images to 1024×1024 ensures that finer details within the GUI, such as text labels, buttons, and icons, remain distinguishable without introducing excessive noise or aliasing effects. Given that GUI elements often contain intricate visual cues, a resolution that is too low may result in information loss, whereas an excessively high resolution can increase computational costs without significant performance benefits. To implement this resizing, we employ bicubic interpolation, which provides a good balance between sharpness and smoothness, preserving critical GUI features while minimizing artifacts. We also ensure that aspect ratios are maintained whenever possible by applying padding techniques if necessary, preventing unintended distortions that could affect model predictions. By adopting this resolution standard, we align our evaluation methodology with established works like Aguvis and UGround, fostering reproducibility and enabling fair comparisons in

GUI-based task assessments.

D.4 Training Details

E Qualitative Analysis

E.1 Success Cases

E.2 Failure Cases

F Limitations

F.1 Realism–Reproducibility Trade-off