ANYACT: UNIVERSAL ACTION FOR SELF-EVOLVING AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

As large language models (LLMs) advance, AI agents are increasingly deployed in open-world environments to tackle complex sequential tasks (*e.g.*, document processing, cross-application collaboration), relying heavily on actions ranging from GUI operations to semantic APIs. However, three core challenges persist: the "scale dilemma" of massive tool ecosystems exceeding LLM context windows, the "non-stationarity" of tool quality due to updates or outages, and the "heterogeneity" of feedback formats (pixels, text, structured data) creating information silos. To address these, we propose **AnyAct**, a universal action layer that unifies available capabilities into a self-evolving action space, enabling agents to operate efficiently and reliably in large-scale, dynamic tool ecosystems. AnyAct's core design focuses on two objectives: constructing this action space via hierarchical progressive retrieval (filtering task-relevant actions) and test-time reliability evolution (pruning unreliable actions), and enabling reliability-aware action orchestration through a heterogeneous observation grounding module that unifies multi-modal feedback. Additionally, it defines a hybrid action space (primitive + semantic actions) and optimizes for a balance between task success rate and execution cost. Evaluations on LiveMCPBench and OSMCP (a new benchmark we developed for multi-granularity action collaboration) demonstrate state-of-the-art performance. AnyAct delivers substantial performance gains over baseline methods across various LLM base models on LiveMCPBench and improvements are particularly notable for models with constrained native capabilities. On OSMCP, it achieves 77.27% overall success with only 50 steps, which is half the steps required by most competitors. We have made our AnyAct available anonymously at: https://anonymous.4open.science/r/AnyAct-ICLR-8A8C/.

1 INTRODUCTION

As large language model (LLM) capabilities continue to advance (Tran et al., 2025; Fang et al., 2025; Jiang & Huang, 2026), particularly in reasoning and tool utilization (Guo et al., 2025a; Li et al., 2025), AI agents have progressively moved from domain-specific applications (Sun et al., 2025; Yang et al., 2024) into open-world environments (Xie et al., 2024; Yang et al., 2025c), taking on complex sequential tasks such as document processing, cross-application collaboration, and data analysis. The successful completion of these tasks depends heavily on the agent's action space, ranging from pixel-level GUI interactions and command-line instructions to semantic APIs that encapsulate complex logic, and the richness of the tool ecosystem (Gan & Sun, 2025; Li et al., 2023; Lù et al., 2025) extends the agents' capability boundaries. In particular, with the widespread adoption of open protocols such as the Model Context Protocol (MCP) (Anthropic, 2024b), the number of available tools has grown explosively, creating a large ecosystem that spans multiple domains and granularities and providing agents with more efficient pathways for task execution.

However, agents acting in the open world still face three core challenges. First, the tool ecosystem's **"scale dilemma"**: the full set of API specifications often exceeds an LLM's context window capacity, and exposing all tools directly leads to inefficient retrieval and unclear task relevance (Shi et al., 2025a; Lin et al., 2025). Second, the **"non-stationarity"** of tool quality: in open environments tools may fail or degrade due to version updates, service outages, or misleading documentation, and static tool selection strategies cannot cope with such dynamic reliability changes. Third, the **"heterogeneity"** of feedback formats: GUI actions produce pixels, command-line operations return

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

text, and APIs yield structured data, and these modality differences make it difficult for agents to perform cross-tool collaborative reasoning, creating information silos. Existing methods either rely on fixed action sets and therefore lack generalization, or lack dynamic awareness of action reliability and therefore lack robustness; in neither case can the agent evolve its own action space from experience, making it difficult to achieve a balance between task success rate and execution efficiency.

To address the aforementioned challenges, we propose **AnyAct**, a universal action layer for self-evolving agents. It is designed to enable agents to efficiently and reliably accomplish open-world tasks within large-scale, dynamically evolving tool ecosystems. The core design of AnyAct focuses on two primary objectives. **Objective 1:** Constructing a self-evolving action space capable of precisely selecting task-relevant subsets from a massive tool repository while dynamically pruning unreliable tools; **Objective 2:** Implementing reliability-aware action orchestration that unifies heterogeneous tool feedback and coordinates tools of varying granularities for collaborative execution.

Specifically, AnyAct employs a hierarchical and progressive retrieval mechanism that utilizes the natural structure of the tool ecosystem to perform efficient coarse-grained filtering followed by fine-grained ranking. This enables the construction of a compact yet comprehensive action space during task initialization. Through a test-time reliability evolution mechanism, AnyAct dynamically updates reliability scores based on execution feedback, thereby guiding the agent to prioritize high-quality actions. A heterogeneous observation grounding module integrates feedback from multiple modalities into a unified knowledge space, eliminating information barriers across different actions. Furthermore, AnyAct defines a hybrid action space consisting of both primitive actions and task-relevant semantic actions. It formulates an optimization objective that balances task success rate with execution cost, achieving coordinated improvements in both robustness and efficiency.

AnyAct demonstrates strong performance on both the LiveMCPBench and OSMCP benchmarks.

OSMCP is constructed by extending OSWorld (Xie et al., 2024) and integrates a rich set of MCP servers that expose agent-native action interfaces, providing realistic support for evaluating multi-granularity action collaboration. We discuss related work in Appendix A. Our main contributions can be summarized as follows:

- **Universal Action Layer for Self-Evolving Agents.** We introduce the first universal action layer that unifies GUI, shell, and MCP actions and supports large-scale dynamic tool ecosystems, enabling self-evolving construction and maintenance of the action space. We also develop the OSMCP benchmark and its accompanying MCP toolset, filling an evaluation gap for multi-granularity action collaboration in open-world tasks.
- **Reliability-Aware Action Orchestration Strategy.** We design a reliability-aware action orchestration strategy that combines test-time evolution of action reliability with a primitive action fallback mechanism, efficiently adapting to challenges such as tool quality fluctuation and service outages in open environments.
- **Cross-Modal Observation Grounding Mechanism.** We develop a cross-modal observation grounding mechanism that maps GUI pixels, command-line text, and structured API feedback into a shared knowledge space, breaking heterogeneous information barriers and supporting collaborative reasoning across different types of actions.

2 PRELIMINARY

We consider an agent performing a task I through sequential interaction with an open-world environment. At each step t, the agent selects an action $a_t \in \mathcal{A}$ based on its observation o_t. The overall effectiveness is fundamentally driven by the co-design of the action hierarchy and the orchestration policy:

- **Action Space.** We define a unified action space $\mathcal{A} = \mathcal{A}_{\text{gui}} \cup \mathcal{A}_{\text{cmd}} \cup \mathcal{A}_{\text{api}}$. Specifically, $\mathcal{A}_{\text{gui}}$ provides atomic, pixel-level control for universal applicability. In contrast, $\mathcal{A}_{\text{cmd}}$ and $\mathcal{A}_{\text{api}}$ consist of semantic actions, which encapsulate high-level logic to perform complex tasks efficiently. The effective action space is further shaped by two practical considerations: (i) **Context-Aware Scaling:** Since the exhaustive library of API specifications is often too vast to be included in an LLM's context window, we retrieve a task-relevant subset $\mathcal{H}(I) \subseteq \mathcal{A}_{\text{api}}$ with $|\mathcal{H}(I)| \leq K$. This yields an instance-specific space $\mathcal{A}_I = \mathcal{A}_{\text{base}} \cup \mathcal{H}(I)$, where $\mathcal{A}_{\text{base}} = \mathcal{A}_{\text{gui}} \cup \mathcal{A}_{\text{cmd}}$ represents the invariant

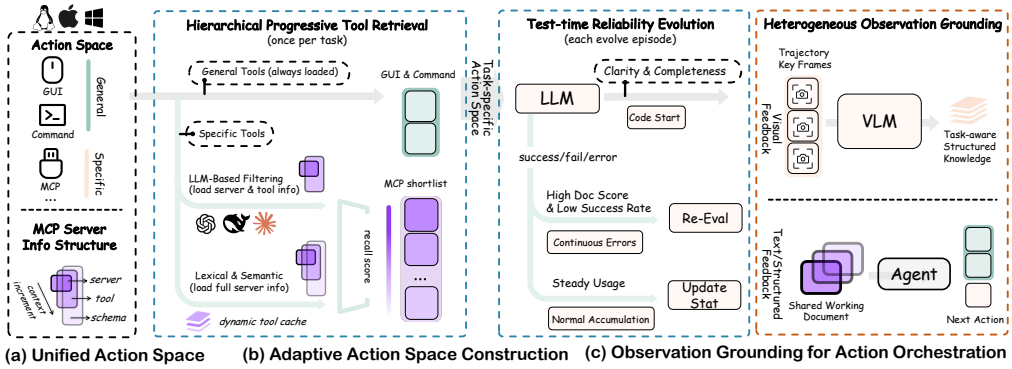


Figure 1: The overall framework of AnyAct for self-evolving agentic automation over a universal action space.

set of fundamental actions available across all tasks. (ii) **Execution Reliability:** Semantic actions may fail during execution due to environmental shifts or API instabilities. To account for this, we maintain a reliability score $r^{(n)}(a) \in [0, 1]$ that evolves across episodes to track the usability of each action.

- **Action Orchestration.** Different action types produce heterogeneous feedback, ranging from screenshots ($\mathcal{A}_{\text{gui}}$) and terminal logs ($\mathcal{A}_{\text{cmd}}$) to structured responses ($\mathcal{A}_{\text{api}}$). We aggregate these into a recursive context $c_{t+1} = \text{agg}(c_t, \phi(o_t))$ to guide subsequent decisions. The policy $a_t = \pi(I, c_t, \mathcal{A}_I, r^{(n)})$ implements a reliability-aware selection strategy: reliability influences the retrieved action priorities, while the LLM selects among available semantic and base actions using task context and execution feedback.

Objective. The agent aims to jointly optimize the action retrieval $\mathcal{H}$, the reliability estimator r, and the orchestration policy π. This joint optimization is formulated as:

$$\max_{\pi,\mathcal{H},r} \ \mathbb{P}(\text{success} \mid I) - \lambda|\tau|, \tag{1}$$

where $|\tau|$ denotes the trajectory length. This objective maximizes the success rate while penalizing execution cost via λ. Specifically, the penalty encourages the policy to leverage high-level actions in $\mathcal{H}$ to reduce steps, while r guides selection toward reliable actions. This formulation expresses the trade-off between task success and execution cost.

3 The AnyAct Framework

We present AnyAct, a universal action layer for scalable, reliable, and efficient self-evolving agent automation in open-world environments. The central goal of AnyAct is to enable an agent to operate in large evolving tool ecosystems while maintaining high task success rates and low execution cost. To this end, the framework addresses two tightly coupled challenges: (1) *adaptive action space construction*, which selects and maintains a task-relevant subset of actions from a massive and non-stationary action space and evolves it from execution feedback, and (2) *reliability-aware action orchestration*, which coordinates heterogeneous actions to complete tasks efficiently and robustly. In the following, we describe how these components are designed and integrated into a unified framework.

3.1 Adaptive Action Space Construction

The full action space $\mathcal{A}$, comprising low-level GUI primitives, shell commands, and high-level semantic APIs, supports a broad range of open-world tasks. In open tool ecosystems, however, directly exposing $\mathcal{A}$ to the agent is infeasible: $\mathcal{A}$ is both *too large* to fit within the context window and *non-stationary* at test time, as tools may be added, updated, or silently fail. Accordingly, the agent is exposed to an *evolving*, task-conditioned subset $\mathcal{A}_I^{(n)} \subset \mathcal{A}$, constructed per task I and revised across episodes n, while retaining base actions as a general-purpose fallback. Under these practical constraints, such an evolving action space is unavoidable, and it gives rise to two challenges. (i)

Retrieval efficiency versus coverage: refreshing $\mathcal{A}_I^{(n)}$ at every step can improve coverage but incurs repeated retrieval overhead, whereas one-shot retrieval is more efficient but may exclude task-critical actions. (ii) **Test-time reliability:** even retrieved actions may fail or drift during execution, requiring $\mathcal{A}_I^{(n)}$ to be re-ranked and revised based on execution feedback rather than static curation.

To address these challenges, we introduce two complementary mechanisms. **Hierarchical Progressive Retrieval** exploits the natural hierarchy of tool ecosystems to enable coarse-to-fine filtering, allowing single-shot retrieval at task initialization while preserving functional coverage. **Test-Time Reliability Evolution** tracks action usability through execution outcomes and demotes unreliable actions across episodes. Together, these mechanisms yield an action space that is both context-efficient and empirically dependable.

3.1.1 Hierarchical Progressive Retrieval

Our goal is to construct a compact set of task-relevant semantic actions $\mathcal{H}(I) \subseteq \mathcal{A}_{\text{api}}$ under a strict budget $|\mathcal{H}(I)| \leq K$, while avoiding repeated retrieval during execution. Accordingly, we retrieve only semantic actions in $\mathcal{A}_{\text{api}}$, and keep base actions $\mathcal{A}_{\text{base}}$ always available. This asymmetric design provides two benefits: (i) *fallback availability*, as $\mathcal{A}_{\text{base}}$ provides general-purpose alternatives when semantic tools are missing or fail; and (ii) *single-shot retrieval*, allowing the action space to be constructed once at task initialization. The resulting action space is $\mathcal{A}_I = \mathcal{A}_{\text{base}} \cup \mathcal{H}(I)$.

Large-scale tool ecosystems are naturally organized into hierarchical registries. At the top level, tools are grouped into providers or servers $s \in \mathcal{S}$ (*e.g.*, MCP servers), each hosting a subset of tools $\mathcal{A}_s \subseteq \mathcal{A}_{\text{api}}$. Each tool further admits a compact metadata view $m(a) = (\text{name}, \text{desc})$ that is distinct from its verbose specification $\text{spec}(a) = (\text{params}, \text{schema}, \text{examples})$. This structure enables a progressive disclosure strategy: metadata can be used for coarse filtering at scale, while full specifications are loaded only for a small set of selected tools.

Coarse-to-Fine Gating. We adopt a hierarchical retrieval strategy that combines LLM reasoning with embedding-based ranking. Embedding retrieval scales efficiently but suffers from an intent–capability mismatch: task instructions express user intent, whereas tool metadata describes declarative functions, and implicit requirements often emerge only after task decomposition. LLM reasoning can bridge this gap, but directly scoring thousands of tools is prohibitively expensive. To resolve this tension, we exploit the registry hierarchy by applying LLM reasoning only at the top level ($|\mathcal{S}| \ll |\mathcal{A}_{\text{api}}|$), followed by embedding-based ranking within the shortlisted servers.

For task I, let $\mathcal{S}_{\text{plan}}(I) \subseteq \mathcal{S}$ denote the coarse server shortlist and $\mathcal{C}_0 = \mathcal{A}_{\text{api}}$ the full metadata pool. When $|\mathcal{C}_0| > N_{\text{thresh}}$, the candidate pool is $\mathcal{C} = \bigcup_{s \in \mathcal{S}_{\text{plan}}(I)} \mathcal{A}_s$; otherwise, $\mathcal{C} = \mathcal{C}_0$. Reranking returns $\mathcal{H}(I) = \mathcal{H}_{\text{rank}}(I, \mathcal{C})$.

- **Server Shortlisting.** For very large action spaces, we first expose the LLM only to compact server-level metadata $\{(s, \text{desc}(s), \{m(a)\}_{a \in \mathcal{A}_s})\}_{s \in \mathcal{S}}$ (*i.e.*, server descriptions and tool names/descriptions). Given a task I, the LLM identifies (i) a set of high-confidence tools $\mathcal{A}_{\text{hc}} \subset \mathcal{A}_{\text{api}}$ matched by name, and (ii) a set of relevant servers $\mathcal{S}_{\text{sel}} \subseteq \mathcal{S}$ whose tools should be explored. The candidate pool is $\mathcal{C}_{\text{plan}} = \mathcal{A}_{\text{hc}} \cup \bigcup_{s \in \mathcal{S}_{\text{sel}}} \mathcal{A}_s$. If $|\mathcal{C}_{\text{plan}}| \leq K$, we return $\mathcal{H}(I) = \mathcal{C}_{\text{plan}}$ directly; otherwise, we rerank the candidates.
- **Tool Reranking.** When the candidate pool exceeds K, the LLM converts task I into capability-oriented keywords q. BM25 (Robertson et al., 1995) first selects a lexical candidate pool $\mathcal{C}_{\text{lex}} \subseteq \mathcal{C}$, which is then reranked by semantic similarity:

$$u(a; q) = \text{sim}_{\text{sem}}(q, a), \qquad a \in \mathcal{C}_{\text{lex}}, \tag{2}$$

  where sim_{sem} measures similarity between normalized dense embeddings. Lexical filtering narrows the pool, and semantic reranking orders the retained candidates.

After reranking, we form $\mathcal{H}(I)$ by retaining up to K actions from $\mathcal{C}_{\text{lex}}$ with the highest scores $u(a; q)$. Here, $\mathcal{C}$ is $\mathcal{A}_{\text{api}}$ when server shortlisting is skipped and $\mathcal{C}_{\text{plan}}$ otherwise.

3.1.2 Test-Time Reliability Evolution

While the above process determines *which* semantic actions to load, runtime reliability decides *whether* these actions remain usable during execution. Open tool ecosystems are non-stationary:

tools can fail due to outages, breaking updates, malformed schemas, or misleading documentation. To account for this, we maintain a per-action reliability score $r^{(n)}(a) \in [r_{\min}, 1]$, updated across episodes n, which acts as a multiplicative gate on the retrieval score:

$$u^{(n)}(a) \leftarrow u^{(n)}(a) \cdot r^{(n)}(a). \tag{3}$$

This ensures that persistently failing actions are gradually demoted, while still allowing recovery if subsequent executions succeed.

Execution Profiling. For each tool a, we maintain a recent execution history to capture runtime behavior, from which we compute two key statistics: the recent success rate $R_a^{(n)}$, summarizing stable performance, and the trailing consecutive failure count $C_a^{(n)}$, which quickly detects abrupt breakages. These statistics are tracked using a sliding window over recent executions, without exposing low-level details such as individual latency or error codes. Here, success denotes the runtime execution-status flag returned by the action backend; task completion is assessed separately by the benchmark evaluator.

Reliability Update. We update $r^{(n)}(a)$ from the recent success rate $R_a^{(n)}$ and consecutive failure count $C_a^{(n)}$ as follows:

- **Cold start.** Let $N_a^{(n)}$ denote the cumulative invocation count. For tools with limited history ($N_a^{(n)} < N_{\min}$), we set $r^{(n)}(a) = 1$ to avoid penalizing tools prematurely. This allows the system to collect sufficient evidence before demoting a tool.
- **Update rules.** After cold start, we subtract the consecutive-failure penalty from the base reliability and clip the result to $[r_{\min}, 1]$:

$$r^{(n)}(a) = \mathrm{clip}\Big(r_{\mathrm{base}}(R_a^{(n)}) - \delta_{\mathrm{streak}}(C_a^{(n)}),\ r_{\min}, 1\Big), \tag{4}$$

  where $\mathrm{clip}(x, l, u) = \min(u, \max(l, x))$. The base reliability downweights tools with low success rates:

$$r_{\mathrm{base}}(R) = \begin{cases} 1, & R \geq \theta_\rho, \\ 1 - (1 - r_{\min})^{\frac{\theta_\rho - R}{\theta_\rho}}, & R < \theta_\rho, \end{cases} \tag{5}$$

  and the streak penalty rapidly demotes tools with consecutive failures beyond a patience threshold F, with $[x]_+ = \max(x, 0)$:

$$\delta_{\mathrm{streak}}(C) = \min\left(\delta_{\max}, \delta_{\mathrm{step}} \cdot [C - F]_+\right). \tag{6}$$

- **Behavioral impact.** The update creates three regimes: (i) *Exploration*, for tools with insufficient history, where no penalty is applied; (ii) *Stable*, where tools with high success rates and few consecutive failures retain high rank, promoting efficient semantic execution; (iii) *Failure*, where low success or persistent failures gradually or rapidly demote tools, shifting selection toward reliable alternatives or fallback primitives. Clipping ensures $r^{(n)}(a) \geq r_{\min}$, so tools can recover in ranking once their execution improves.

Evolutionary Maintenance. To handle long-term ecosystem drift, we perform lightweight maintenance at session boundaries. Specifically, we compute a specification fingerprint $h(a)$ for each tool, capturing its core definition and usage requirements. When the specification changes, we refresh the cached documentation assessment while retaining the execution history. Tools that remain unreliable over extended periods can optionally be flagged for offline diagnosis or repair (*e.g.*, to improve documentation clarity and completeness).

3.2 Heterogeneous Observation Grounding

Agents acting over a universal action space receive feedback in multiple forms: visual observations from GUI interactions (screenshots), text output from shell commands, and structured responses from semantic APIs. While text and structured outputs can be directly consumed by reasoning modules, visual observations pose a challenge: it is often unclear whether an action succeeded and how to extract actionable information for subsequent steps.

Grounding Operator. To unify these heterogeneous signals, we introduce a grounding operator $\phi(\cdot)$ that converts all observations into a shared knowledge representation $\mathcal{K}$. For text and structured outputs, grounding applies lightweight normalization, such as truncating long outputs or linearizing structured data into compact textual records. The main challenge lies in visual observations, which require converting raw pixels into structured knowledge that downstream reasoning can use.

Visual Grounding. We use a vision-language model (VLM) to extract actionable elements from screenshots, such as text, numbers, tables, and identifiers like URLs, file paths, or element IDs, while ignoring decorative content. The main idea is to treat screenshots as documents to parse rather than scenes to describe, producing machine-readable knowledge. For sequences with multiple frames, we select representative keyframes to balance coverage and context efficiency.

Knowledge-Driven Planning. The unified knowledge representation $\phi(o_t) \in \mathcal{K}$ enables reasoning that is independent of the observation modality. Identifiers extracted from APIs can parameterize GUI actions, while states grounded from screenshots can verify the results of executed commands. Grounded observations are accumulated into an episodic context $c_{t+1} = \mathrm{agg}(c_t, \phi(o_t))$, which serves as input to the policy $\pi(I, c_t, \mathcal{A}_I, r^{(n)})$, providing a single, consistent substrate for sequential planning without modality-specific handling.

4 Evaluation

4.1 Experimental Setup

Benchmarks. We evaluate AnyAct on LiveMCPBench (Mo et al., 2025) and OSMCP. LiveMCP-Bench includes 70 MCP servers and 527 tools across six domains, with 95 practical, multi-step tasks solvable via MCP tools. OSMCP is a computer-use benchmark built on OSWorld (Xie et al., 2024): it selects 198 tasks (including 21 infeasible ones) from seven applications, *i.e.*, LibreOffice Calc, Impress, and Writer, GIMP, Thunderbird, VLC, and VS Code, preserving their instructions and evaluators. For these applications we develop 122 MCP tools (45 Atomic, 69 Compound, and 8 Workflow), so OSMCP currently covers only the OSWorld subset whose software has MCP support. See Appendix B for OSMCP statistics and Appendix C for the full tool list.

Models and Baselines. For the experiments on LiveMCPBench, we evaluate four base models: Claude-Sonnet-4 (Anthropic, 2025a), DeepSeek-V3 (Liu et al., 2024), DeepSeek-R1 (Guo et al., 2025a), and Qwen3-235B (Yang et al., 2025a). Following the setting in LiveMCPBench, we employ DeepSeek-V3 as our evaluation model. For the experiments on OSMCP, we use Claude-Sonnet-4.5 as the base LLM of AnyAct. Moreover, comparisons use two groups of baselines: (1) End-to-end GUI models: Claude-Sonnet-4.5 (Anthropic, 2025b), OpenCUA-72B (Wang et al., 2025b), UI-TARS-2 (Wang et al., 2025a), EvoCUA (Xue et al., 2026), DeepMiner-Mano-72B (Fu et al., 2025), and Seed-1.8 (ByteDance Seed, 2025). (2) Agentic frameworks: Jedi-7B (Xie et al., 2025), CoACT-1 (Song et al., 2025), GTA1 (Yang et al., 2025b), Agentic-Lybic-Maestro (Guo et al., 2025b), OS-Symphony (Yang et al., 2026), GBOX Agent (GBOX, n.d.), Agent S3 (Gonzalez-Pumariega et al., 2025), and UiPath Screen Agent (UiPath, n.d.).

4.2 Main Results on LiveMCPBench

We carry out extensive experiments on LiveMCPBench to evaluate the ability of AnyAct to solve problems via MCP services and to assess the effectiveness of key modules in the proposed method. The corresponding results are summarized in Table 1. Here, `Naive` refers to the LiveMCPBench baseline built on the ReAct (Yao et al., 2022) framework with the MCP-Zero (Fei et al., 2025) retrieval

Table 1: Main results on LiveMCPBench.

Model	Method	Office	Leisure	Travel	Lifestyle	Finance	Shopping	Overall (%)
Claude-Sonnet-4	Naive	90.32	64.29	75.00	80.00	78.57	66.67	78.95
	AnyAct w/o Retrieval	80.65	71.43	75.00	93.33	90.91	77.78	81.52
	AnyAct w/o Evolution	83.87	78.57	58.33	80.00	91.67	33.33	75.27
	AnyAct	**93.55**	**92.86**	**100.00**	**100.00**	**83.33**	**88.89**	**93.55**
DeepSeek-V3	Naive	41.94	42.86	50.00	40.00	28.57	55.56	42.11
	AnyAct w/o Retrieval	51.61	42.86	33.33	60.00	50.00	33.33	47.31
	AnyAct w/o Evolution	70.97	50.00	50.00	66.67	50.00	33.33	58.06
	AnyAct	**96.77**	**78.57**	**75.00**	**86.67**	**83.33**	**66.67**	**84.95**
DeepSeek-R1	Naive	41.94	50.00	58.33	46.67	50.00	55.56	48.42
	AnyAct w/o Retrieval	70.97	57.14	58.33	60.00	75.00	44.44	63.44
	AnyAct w/o Evolution	64.52	64.29	41.67	80.00	50.00	33.33	59.14
	AnyAct	**83.87**	**85.71**	**66.67**	**93.33**	**50.00**	**77.78**	**78.49**
Qwen3-235B	Naive	54.84	35.71	41.67	53.33	50.00	44.44	48.42
	AnyAct w/o Retrieval	58.06	57.14	50.00	53.33	58.33	66.67	56.99
	AnyAct w/o Evolution	74.19	57.14	66.67	60.00	58.33	66.67	65.59
	AnyAct	**80.65**	**78.57**	**66.67**	**86.67**	**75.00**	**66.67**	**77.42**

Table 2: Main results on OSMCP. Step budget denotes the maximum number of steps allowed for each system.

Method	Model	Steps	vlc	gimp	thunderbird	calc	writer	impress	vs_code	Overall (/)	Overall (%)
Foundation E2E GUI											
Claude-Sonnet-4.5	—	15	4.89/17	13.00/26	10.00/15	17.00/47	14.00/23	22.96/47	14.00/23	95.85/198	48.41
opencua-72b-preview	—	50	8.87/17	14.00/26	12.00/15	18.00/47	13.00/23	26.96/47	18.00/23	110.83/198	55.97
UI-TARS-2-2509	—	100	8.49/17	13.00/26	11.00/15	31.00/47	14.00/23	26.50/47	17.00/23	120.99/198	61.11
Claude-Sonnet-4.5	—	50	9.00/17	15.00/26	10.00/15	31.00/47	15.00/23	27.02/47	16.00/23	123.02/198	62.13
EvoCUA-20260105	—	50	8.39/17	20.00/26	12.00/15	26.00/47	16.0/23	27.96/47	20.00/23	130.35/198	65.83
DeepMiner-Mano-72B	—	100	6.00/17	23.00/26	10.00/15	27.00/47	18.00/23	28.96/47	18.00/23	130.96/198	66.14
Claude-Sonnet-4.5	—	100	9.89/17	14.00/26	9.00/15	34.00/47	18.97/23	31.96/47	17.00/23	134.82/198	68.09
Seed-1.8	—	100	10.67/17	18.00/26	12.00/15	33.00/47	18.88/23	28.62/47	17.00/23	138.17/198	69.78
Agentic Framework											
Jedi-7B	OpenAI-o3	100	9.25/17	18.00/26	12.00/15	20.00/47	13.00/23	20.96/47	19.00/23	112.21/198	56.67
CoACT-1	—	150	12.23/17	17.00/26	11.00/15	33.00/47	17.00/23	23.63/47	18.00/23	131.86/198	66.60
GTA1	GPT-5	100	9.84/17	20.00/26	12.00/15	30.00/47	13.97/23	30.80/47	19.00/23	135.61/198	68.49
Agentic-Lybic-Maestro	—	100	13.72/17	19.00/26	10.00/15	28.00/46	15.00/23	32.63/47	19.00/23	137.35/197	69.72
OS-Symphony	GPT-5	50	10.46/17	21.00/26	12.00/15	36.00/47	17.97/23	22.94/47	18.00/23	138.37/198	69.88
GBOX Agent	—	15	7.87/17	20.00/26	10.00/15	35.00/47	15.00/22	29.96/47	21.00/23	138.83/197	70.47
Agent S3 (N=1)	Claude-Opus-4.5	100	12.70/17	13.00/26	9.00/15	40.00/47	18.00/23	30.99/47	16.00/23	139.69/198	70.55
UiPath Screen Agent	Claude-Opus-4.5	100	10.34/17	18.00/26	11.00/15	37.00/47	16.00/23	32.99/47	18.00/23	143.33/198	72.39
AnyAct w/o GUI	Claude-Sonnet-4.5	15	11.00/17	15.00/26	11.00/15	30.00/47	13.00/23	20.00/47	17.00/23	117.00/198	59.09
AnyAct	Claude-Sonnet-4.5	50	13.00/17	22.00/26	10.00/15	42.00/47	18.00/23	29.00/47	19.00/23	153.00/198	77.27

strategy; `AnyAct w/o Retrieval` indicates a variant where the Hierarchical Progressive Retrieval module of AnyAct is replaced by the MCP-Zero retrieval strategy; and `AnyAct w/o Evolution` denotes a variant obtained by removing the Test-Time Reliability Evolution module from AnyAct.

(i) Strong performance across base models. From the results, AnyAct yields substantial gains over the `Naive` variant across different LLMs, with especially large improvements for DeepSeek-V3, DeepSeek-R1, and Qwen3-235B; performance roughly doubles for DeepSeek-V3. These results show that AnyAct benefits a range of base LLMs: when paired with LLMs of varying strengths, it can effectively enhance their ability to utilize MCP APIs. This improvement is attributable to our retrieval and evolution modules. Hierarchical Progressive Retrieval exploits the natural hierarchy in tool organization for multi-granularity filtering, greatly narrowing and optimizing the action space LLMs must handle; Test-Time Reliability Evolution maintains execution-based quality estimates for MCP APIs during testing, guiding subsequent retrieval toward reliable actions and improving task success rates.

(ii) Contributions of retrieval and reliability evolution. Both the `AnyAct w/o Retrieval` and `AnyAct w/o Evolution` variants perform substantially worse than the full AnyAct. The MCP-Zero comparison supports the effectiveness of Hierarchical Progressive Retrieval: by applying multi-granularity filtering to the tool set, it accurately identifies the tools required at the start of a task and markedly improves retrieval effectiveness. Meanwhile, Test-Time Reliability Evolution effectively addresses the challenge of heterogeneous tool quality within the tool pool; through updates to execution histories during testing, it steers LLMs toward using high-quality tools.

4.3 Main Results on OSMCP

To more comprehensively evaluate AnyAct's performance on real open-world tasks, we conducted experiments on OSMCP. This benchmark, developed on top of the computer-using benchmark OSWorld, integrates a rich set of MCP servers and can thoroughly assess AnyAct's ability to complete real open-world tasks using actions at different granularities (MCP APIs, shell commands, and GUI operations). We report the corresponding results in Table 2, where `AnyAct w/o GUI` denotes the variant that omits GUI actions and runs under a 15-step budget. We evaluate AnyAct under a 50-step budget, counting each GUI action within a subtask separately. Key findings are as follows:

(i) Strong performance under a 50-step budget. AnyAct achieves the highest overall score among the systems in Table 2, reaching 77.27% under a 50-step budget. Many competing systems use a 100-step budget. With Claude-Sonnet-4.5, AnyAct also outperforms the listed frameworks powered by Claude-Opus-4.5. These results reflect the combination of application-level MCP actions and AnyAct's retrieval and reliability mechanisms. MCP APIs let the agent perform complex application

Table 3: Case study: GUI-MCP backend switching for spreadsheet automation in LibreOffice Calc.

Task ID: `04d9aeaf-7bed-4024-bedb-e10e6f00eb7f`

Task Instruction: In a new sheet with 4 headers "Year", "CA changes", "FA changes", and "OA changes", calculate the percentage annual changes compared to last year in 2015 to 2019 for the Current Assets, Fixed Assets, and Other Assets columns.

Stepwise Execution (Reasoning → Tool Call):

`Step 0 (GUI):` "Check if LibreOffice Calc is open with the balance sheet file ready."
→ `gui_agent(task_description="Verify Calc open with SmallBalanceSheet.xlsx", max_steps=15)`

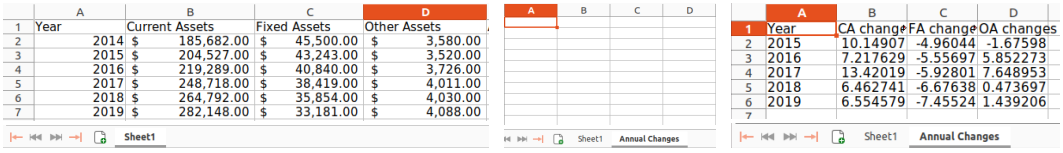
`Step 1 (MCP):` "Let me check the current document to understand the data structure."
→ `libreoffice-calc.get_all_document_contents(max_rows=100, max_cols=20)`

`Step 2 (MCP):` "Now I'll create a new sheet with the 4 headers."
→ `libreoffice-calc.create_new_sheet(sheet_name="Annual Changes")`

`Step 3 (MCP):` "Set up the headers and calculate the percentage changes."
→ `libreoffice-calc.set_cell_content(cells=[A1:D1 headers, A2:A6 years, B2:D6 formulas])`

`Complete:` "Created sheet 'Annual Changes' with percentage annual changes for 2015–2019."

Key Execution Snapshots (cropped worksheet and sheet-tab views):



	A	B	C	D
1	Year	Current Assets	Fixed Assets	Other Assets
2	2014	$ 185,682.00	$ 45,500.00	$ 3,580.00
3	2015	$ 204,527.00	$ 43,243.00	$ 3,520.00
4	2016	$ 219,289.00	$ 40,840.00	$ 3,726.00
5	2017	$ 248,718.00	$ 38,419.00	$ 4,011.00
6	2018	$ 264,792.00	$ 35,854.00	$ 4,030.00
7	2019	$ 282,148.00	$ 33,181.00	$ 4,088.00

Sheet1

(a) Initial state

Sheet1 Annual Changes

(b) Sheet created

	A	B	C	D
1	Year	CA change	FA change	OA changes
2	2015	10.14907	-4.96044	-1.67598
3	2016	7.217629	-5.55697	5.852273
4	2017	13.42019	-5.92801	7.648953
5	2018	6.462741	-6.67638	0.473697
6	2019	6.554579	-7.45524	1.439206
7				

Sheet1 Annual Changes

(c) Formulas populated

operations through a single invocation, complementing the flexibility of GUI and shell actions. The comparison demonstrates strong system-level performance under the reported budgets; different action granularities mean that step counts alone do not measure time or token cost.

(ii) Complementary roles of GUI and semantic actions. The full AnyAct achieves 77.27%, compared with 59.09% for `AnyAct w/o GUI`. This comparison uses different budgets (50 versus 15 steps), so it does not isolate the effect of GUI access. GUI actions nevertheless provide a way to handle visual state and interactions beyond the available semantic interfaces, while MCP APIs and shell commands support operations at higher granularities. Combining these action types lets the agent choose an interface suited to each subtask. The case study below illustrates this coordination: GUI interaction verifies the application state, and MCP actions perform the subsequent spreadsheet operations.

4.4 Case Study on OSMCP

To illustrate how AnyAct switches between action types during open-world tasks, we conduct a case study on OSMCP, shown in Table 3. The task involves adding a worksheet and computing annual percentage changes. We observe that the agent must first verify whether the balance sheet file is already open—a visual check performed through GUI interaction in this trajectory. At this point, GUI actions use visual feedback to confirm the application state. Subsequently, for worksheet creation and formula entry, APIs provided by MCP servers can accomplish the goals efficiently and accurately, avoiding the tedious, lengthy, and potentially error-prone nature of purely GUI-based actions. Therefore, for computer-use tasks, combining GUI actions with MCP APIs is a highly promising solution.

4.5 In-Depth Analysis

To comprehensively evaluate the effect of Test-Time Reliability Evolution, we compile statistics on how evolution influences different LLMs on LiveMCPBench; the corresponding results are shown in Figure 2. In addition, we select representative examples for case studies.

Effect of reliability evolution. Figure 2 shows that the evolution procedure yields substantial overall gains across different LLMs, validating the effectiveness of our method. For Claude-Sonnet-4, DeepSeek-V3, and Qwen3-235B, however, some tasks that were previously solvable fail after evolution. A likely reason is that the evolution module demotes tools that are unstable or have low success rates when no substitutes exist; tasks that had succeeded only by relying on such tools then fail because those tools are no longer retrieved. This underscores an inherent trade-off between pruning unreliable tools and preserving tools that are uniquely indispensable for certain tasks.

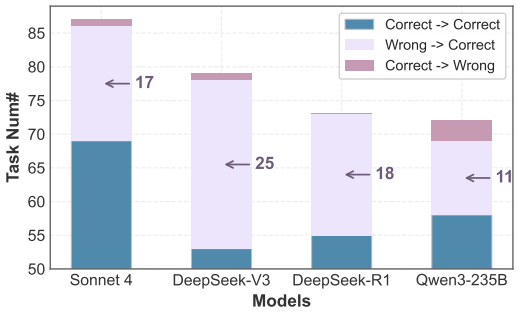


Figure 2: Evolution effect across different LLMs on LiveMCPBench. Blue: correct before and after evolution; light purple: incorrect-to-correct; mauve: correct-to-incorrect; deep-purple: net gain.

Case study on LiveMCPBench. We examine alternative tool use in two LiveMCPBench news-seeking tasks involving a recurring search-service failure. The tool `duckduckgo_web_search` consistently fails due to rate limiting (0/38 success; repeated "anomaly" warnings), and these failures are persistently recorded by the quality manager. These records can influence subsequent tool selection through reliability-weighted retrieval.

The two tasks use different instructions: **In Task 20** ("today's top LLM-related news"), `duckduckgo_web_search` is highly ranked by semantic similarity and is invoked once, immediately failing and adding $\sim$3 s of latency; the task completes only because alternative news tools (*e.g.*, `get-nytimes-news`, `get-theverge-news`, `get-infoq-news`) provide sufficient coverage. **In Task 77** ("big AI news today"), `duckduckgo_web_search` is not invoked; the agent instead uses reliable sources (*e.g.*, `get-infoq-news`, `get-theverge-news`, `get-bbc-news`, `get-36kr-trending`) to complete the task with zero failed calls. These trajectories illustrate how alternative tools can support task completion when a search service fails; the effect of reliability evolution is evaluated separately in Figure 2.

Robustness to description noise. Tool metadata is often incomplete or misleading in practice. We perturb the descriptions of the 109 cached OSMCP tools by deleting 5–30% of words or swapping descriptions across tools, keeping names and servers intact, and retrieve tools for 18 application queries (Figure 3; Appendix B.5). At $K = 10$, the dense/hybrid ranker retains 0.75 Top-10 overlap with clean retrieval under 30% deletion and 0.58 under 30% swapping, while its expected-server coverage stays at 18/18 throughout; the lexical control degrades faster (0.59 at 20% deletion). Expected-application coverage thus remains stable under this local ranking test, despite changes in the retrieved tools.

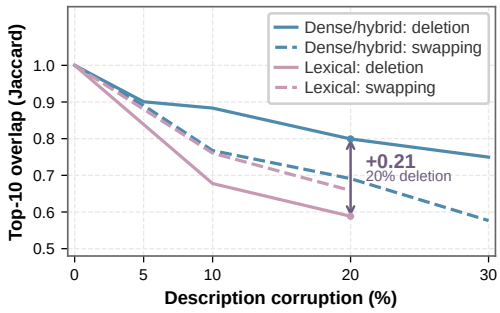


Figure 3: Top-10 retrieval stability under description noise. Blue: dense/hybrid; mauve: lexical. Solid: deletion; dashed: swapping.

5 Conclusion

We introduce AnyAct, a universal action layer for self-evolving agents operating in open-world tool ecosystems, designed to handle scale, non-stationarity, and heterogeneity. By combining hierarchical progressive retrieval, test-time reliability evolution, and cross-modal observation grounding, AnyAct adaptively builds and evolves its action space from execution feedback and robustly orchestrates actions, balancing both task success and efficiency. On LiveMCPBench and OSMCP, it achieves strong performance across base LLMs and combines GUI and semantic actions to complete tasks within the reported step budgets.

AI USE STATEMENT

We used generative AI tools exclusively to refine the language, grammar, and overall readability of this manuscript. We did not employ these tools for core research activities, such as generating synthetic datasets, developing theoretical models, formulating mathematical claims, or assisting with proofs. Furthermore, we did not use AI to propose hypotheses, design methodologies, implement methods, translate text, process data, or interpret results. We have rigorously reviewed all AI-assisted language revisions to ensure the scientific claims, experimental findings, and technical integrity of the paper remain entirely unaltered. We assume full responsibility for the final manuscript, including any text polished with the aid of AI.

REFERENCES

Anthropic. Developing a computer use model. In *https://www.anthropic.com/news/ developing-computer-use*, 2024a.

Anthropic. Introducing the model context protocol. In *https://www.anthropic.com/news/model-context-protocol*, 2024b.

Anthropic. Introducing Claude 4, 2025a. URL `https://www.anthropic.com/news/claude-4`.

Anthropic. Introducing Claude Sonnet 4.5, 2025b. URL `https://www.anthropic.com/news/claude-sonnet-4-5`.

ByteDance Seed. Seed1.8 model card: Towards generalized real-world agency, 2025. URL `https://seed.bytedance.com/seed1_8`.

Jinyuan Fang, Yanwen Peng, Xi Zhang, Yingxu Wang, Xinhao Yi, Guibin Zhang, Yi Xu, Bin Wu, Siwei Liu, Zihao Li, et al. A comprehensive survey of self-evolving ai agents: A new paradigm bridging foundation models and lifelong agentic systems. *arXiv preprint arXiv:2508.07407*, 2025.

Xiang Fei, Xiawu Zheng, and Hao Feng. Mcp-zero: Proactive toolchain construction for llm agents from scratch. *arXiv preprint arXiv:2506.01056*, 2025.

Tianyu Fu, Anyang Su, Chenxu Zhao, Hanning Wang, Minghui Wu, Zhe Yu, Fei Hu, Mingjia Shi, Wei Dong, Jiayao Wang, et al. Mano technical report. *arXiv preprint arXiv:2509.17336*, 2025.

Tiantian Gan and Qiyao Sun. Rag-mcp: Mitigating prompt bloat in llm tool selection via retrieval-augmented generation. *arXiv preprint arXiv:2505.03275*, 2025.

GBOX. GBOX agent, n.d. URL `https://gbox.ai`.

Gonzalo Gonzalez-Pumariega, Vincent Tu, Chih-Lun Lee, Jiachen Yang, Ang Li, and Xin Eric Wang. The unreasonable effectiveness of scaling agents for computer use. *arXiv preprint arXiv:2510.02250*, 2025.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025a.

Liangxuan Guo, Bin Zhu, Qingqian Tao, Kangning Liu, Xun Zhao, Xianzhe Qin, Jin Gao, and Guangfu Hao. Agentic lybic: Multi-agent execution system with tiered reasoning and orchestration. *arXiv preprint arXiv:2509.11067*, 2025b.

Yangqin Jiang and Chao Huang. Openphone: Mobile agentic foundation models. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 30362–30380, 2026.

Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. Api-bank: A comprehensive benchmark for tool-augmented llms. *arXiv preprint arXiv:2304.08244*, 2023.

Xiaoxi Li, Guanting Dong, Jiajie Jin, Yuyao Zhang, Yujia Zhou, Yutao Zhu, Peitian Zhang, and Zhicheng Dou. Search-o1: Agentic search-enhanced large reasoning models. *arXiv preprint arXiv:2501.05366*, 2025.

Jiaye Lin, Yifu Guo, Yuzhen Han, Sen Hu, Ziyi Ni, Licheng Wang, Mingguang Chen, Hongzhang Liu, Ronghao Chen, Yangfan He, Daxin Jiang, Binxing Jiao, Chen Hu, and Huacan Wang. Se-agent: Self-evolution trajectory optimization in multi-step reasoning with llm-based agents. *arXiv preprint arXiv:2508.02085*, 2025.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

Xing Han Lù, Gaurav Kamath, Marius Mosbach, and Siva Reddy. Build the web for agents, not agents for the web. *arXiv preprint arXiv:2506.10953*, 2025.

Yadong Lu, Jianwei Yang, Yelong Shen, and Ahmed Awadallah. Omniparser for pure vision based gui agent. *arXiv preprint arXiv:2408.00203*, 2024.

Guozhao Mo, Wenliang Zhong, Jiawei Chen, Xuanang Chen, Yaojie Lu, Hongyu Lin, Ben He, Xianpei Han, and Le Sun. Livemcpbench: Can agents navigate an ocean of mcp tools? *arXiv preprint arXiv:2508.01780*, 2025.

Ofir Press, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah A Smith, and Mike Lewis. Measuring and narrowing the compositionality gap in language models. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 5687–5711, 2023.

Yujia Qin, Yining Ye, Junjie Fang, Haoming Wang, Shihao Liang, Shizuo Tian, Junda Zhang, Jiahao Li, Yunxin Li, Shijue Huang, et al. Ui-tars: Pioneering automated gui interaction with native agents. *arXiv preprint arXiv:2501.12326*, 2025.

Stephen E Robertson, Steve Walker, Susan Jones, Micheline M Hancock-Beaulieu, Mike Gatford, et al. *Okapi at trec-3*. Nist Special Publication Sp, 1995.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems*, 36:68539–68551, 2023.

Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face. *Advances in Neural Information Processing Systems*, 36:38154–38180, 2023.

Yexuan Shi, Mingyu Wang, Yunxiang Cao, Hongjie Lai, Junjian Lan, Xin Han, Yu Wang, Jie Geng, Zhenan Li, Zihao Xia, et al. Aime: Towards fully-autonomous multi-agent framework. *arXiv preprint arXiv:2507.11988*, 2025a.

Zhengliang Shi, Yuhan Wang, Lingyong Yan, Pengjie Ren, Shuaiqiang Wang, Dawei Yin, and Zhaochun Ren. Retrieval models aren't tool-savvy: Benchmarking tool retrieval for large language models. *arXiv preprint arXiv:2503.01763*, 2025b.

Linxin Song, Yutong Dai, Viraj Prabhu, Jieyu Zhang, Taiwei Shi, Li Li, Junnan Li, Silvio Savarese, Zeyuan Chen, Jieyu Zhao, et al. Coact-1: Computer-using agents with coding as actions. *arXiv preprint arXiv:2508.03923*, 2025.

Qiushi Sun, Kanzhi Cheng, Zichen Ding, Chuanyang Jin, Yian Wang, Fangzhi Xu, Zhenyu Wu, Chengyou Jia, Liheng Chen, Zhoumianze Liu, et al. Os-genesis: Automating gui agent trajectory construction via reverse task synthesis. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 5555–5579, 2025.

Khanh-Tung Tran, Dung Dao, Minh-Duong Nguyen, Quoc-Viet Pham, Barry O'Sullivan, and Hoang D Nguyen. Multi-agent collaboration mechanisms: A survey of llms. *arXiv preprint arXiv:2501.06322*, 2025.

UiPath. UiPath screen agent. Project page, n.d. URL `https://www.uipath.com/ai/research/screen-agent`. Accessed September 26, 2026.

Haoming Wang, Haoyang Zou, Huatong Song, Jiazhan Feng, Junjie Fang, Junting Lu, Longxiang Liu, Qinyu Luo, Shihao Liang, Shijue Huang, et al. Ui-tars-2 technical report: Advancing gui agent with multi-turn reinforcement learning. *arXiv preprint arXiv:2509.02544*, 2025a.

Xinyuan Wang, Bowen Wang, Dunjie Lu, Junlin Yang, Tianbao Xie, Junli Wang, Jiaqi Deng, Xiaole Guo, Yiheng Xu, Chen Henry Wu, et al. Opencua: Open foundations for computer-use agents. *arXiv preprint arXiv:2508.09123*, 2025b.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh J Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, et al. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *Advances in Neural Information Processing Systems*, 37:52040–52094, 2024.

Tianbao Xie, Jiaqi Deng, Xiaochuan Li, Junlin Yang, Haoyuan Wu, Jixuan Chen, Wenjing Hu, Xinyuan Wang, Yuhui Xu, Zekun Wang, et al. Scaling computer-use grounding via user interface decomposition and synthesis. *arXiv preprint arXiv:2505.13227*, 2025.

Taofeng Xue, Chong Peng, Mianqiu Huang, Linsen Guo, Tiancheng Han, Haozhe Wang, Jianing Wang, Xiaocheng Zhang, Xin Yang, Dengchang Zhao, et al. Evocua: Evolving computer use agents via learning from scalable synthetic experience. *arXiv preprint arXiv:2601.15876*, 2026.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025a.

Bowen Yang, Kaiming Jin, Zhenyu Wu, Zhaoyang Liu, Qiushi Sun, Zehao Li, JingJing Xie, Zhoumianze Liu, Fangzhi Xu, Kanzhi Cheng, et al. Os-symphony: A holistic framework for robust and generalist computer-using agent. *arXiv preprint arXiv:2601.07779*, 2026.

John Yang, Carlos E Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering. *Advances in Neural Information Processing Systems*, 37:50528–50652, 2024.

Yan Yang, Dongxu Li, Yutong Dai, Yuhao Yang, Ziyang Luo, Zirui Zhao, Zhiyuan Hu, Junzhe Huang, Amrita Saha, Zeyuan Chen, et al. Gta1: Gui test-time scaling agent. *arXiv preprint arXiv:2507.05791*, 2025b.

Yingxuan Yang, Mulei Ma, Yuxuan Huang, Huacan Chai, Chenyu Gong, Haoran Geng, Yuanjian Zhou, Ying Wen, Meng Fang, Muhao Chen, et al. Agentic web: Weaving the next web with ai agents. *arXiv preprint arXiv:2507.21206*, 2025c.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *The eleventh international conference on learning representations*, 2022.

Zhiyuan Yao, Zishan Xu, Yifu Guo, Zhiguang Han, Cheng Yang, Shuo Zhang, Weinan Zhang, Xingshan Zeng, and Weiwen Liu. Toolace-mcp: Generalizing history-aware routing from mcp tools to the agent web. *arXiv preprint arXiv:2601.08276*, 2026.

Jiaru Zou, Ling Yang, Yunzhe Qi, Sirui Chen, Mengting Ai, Ke Shen, Jingrui He, and Mengdi Wang. Autotool: Dynamic tool selection and integration for agentic reasoning. *arXiv preprint arXiv:2512.13278*, 2025.

APPENDIX

A RELATED WORK

Tool-Augmented LLM Agents. The growing capability of large language models (LLMs) to use tools has attracted significant research interest, as it greatly extends their practical utility (Shi et al., 2025b). Early approaches (Press et al., 2023; Yao et al., 2022) relied on prompting strategies that interleaved reasoning with external queries or API calls, while others (Schick et al., 2023) proposed training-based methods to equip LLMs with tool-using skills. Following the emergence of open protocols like the Model Context Protocol (MCP) (Anthropic, 2024b), the tool ecosystem has shifted from closed to open systems. This shift introduces new challenges, including an explosive growth in the number of available tools and significant variation in their quality—issues that traditional context-only methods handle poorly (Shen et al., 2023). Recent efforts aim to address these problems. For example, AutoTool (Zou et al., 2025) improves model adaptability to evolving tool sets through multi-stage post-training, while ToolACE-MCP (Yao et al., 2026) enables precise routing within large-scale tool ecosystems by constructing trajectory graphs.

Computer-Using Agents. Advances in vision-language models and expanding agent applications have driven the development of computer-using agents, which are designed to perform diverse and complex tasks on computers (Anthropic, 2024a; Xie et al., 2024). The diversity and complexity of these tasks, along with their need for robust GUI-grounded capabilities, expose limitations in traditional tool-based approaches. Current research typically depends either on a model's inherent GUI proficiency (Lu et al., 2024; Anthropic, 2025a; Wang et al., 2025a) or on agentic frameworks (Yang et al., 2026; Song et al., 2025; Xie et al., 2025). For instance, UI-TARS (Qin et al., 2025) achieves strong GUI task performance via multi-stage post-training on targeted, high-quality data, whereas Agent S3 (Gonzalez-Pumariega et al., 2025) employs a framework that uses code-based actions to improve task success rates. With the growth of the MCP ecosystem, a promising direction is to integrate MCP's efficient interfacing with GUI interaction and more fine-grained, flexible, and generalizable tools to collaboratively solve computer-using tasks.

B OSMCP BENCHMARK DETAILS

B.1 TASK SELECTION AND OUTCOME EVALUATION

OSMCP contains all 198 tasks in the seven application domains listed in Table 4, with no additional filtering within those domains. This selection follows application-server coverage and represents a subset of OSWorld, excluding its dedicated Chrome, OS, and multi-application task groups. Comparison against the corresponding base snapshot confirms that all 198 task instructions and evaluator specifications are unchanged. The 17 VLC tasks differ only in runtime configuration and explanatory notes. Their initial-state requirements use either Chrome or base-setup VM snapshots, illustrating that task domains and runtime images serve different roles in the benchmark.

Table 4: OSMCP task coverage and static tool inventory. Infeasible tasks are included in the task counts.

Application	Tasks	Infeasible	Tools	Atomic	Compound	Workflow
LibreOffice Calc	47	1	32	6	21	5
LibreOffice Impress	47	0	23	3	20	0
LibreOffice Writer	23	1	21	5	14	2
GIMP	26	10	9	4	4	1
Thunderbird	15	1	13	10	3	0
VLC	17	3	11	8	3	0
VS Code	23	5	13	9	4	0
Total	**198**	**21**	**122**	**45**	**69**	**8**

The inherited executable evaluators assess application state and produced artifacts, including spreadsheets, documents, presentations, images, and configuration files. Task scores thus measure the final outcome beyond the success status of individual tool calls. The task set also includes 21 cases with the `infeasible` evaluator. These receive a score of one when the final action is `FAIL`, including its

structured-action form, and zero otherwise. They remain part of the 198-task denominator, allowing the evaluation to capture recognition of infeasibility. Table 2 includes the available task count for each system: Agentic-Lybic-Maestro and GBOX Agent have 197 outcomes, while the other rows cover 198 tasks.

B.2 Tool Interfaces and Granularity

The static tool catalog is annotated along two dimensions. Operation type describes functional intent through seven categories: Read, Write, Create, Transform, Control, Export, and Configuration. Granularity describes the scope of the exposed interface. Atomic tools provide individual application-level operations, such as `get_zoom` and `create_new_sheet`. Compound tools compose operations or process data in bulk; `Fill_blank_above`, for instance, fills blank cells using values from preceding cells, while `calculate_employee_ages` transforms a birthday column into ages. Workflow tools implement broader procedures, exemplified by `calculate_annual_asset_changes` and `summarize_revenue_by_promotion`. Table 5 summarizes both dimensions by application.

Table 5: Operation types and interface granularity in the OSMCP tool catalog.

Type	Calc	Impress	Writer	Thunderbird	VS Code	VLC	GIMP	Total
Operation type								
Read	3	3	1	4	5	3	1	20
Write	8	13	15	3	1	1	1	42
Create	6	2	1	3	0	0	0	12
Transform	9	0	3	0	0	0	5	17
Control	3	1	0	0	7	6	1	18
Export	2	3	1	0	0	1	1	8
Configuration	1	1	0	3	0	0	0	5
Granularity								
Atomic	6	3	5	10	9	8	4	45
Compound	21	20	14	3	4	3	4	69
Workflow	5	0	2	0	0	0	1	8
Total	**32**	**23**	**21**	**13**	**13**	**11**	**9**	**122**

These annotations characterize the functional scope presented to the agent. Internal execution may involve several low-level operations even for an Atomic interface, and a general dispatch interface may support a wide range of behaviors. Consequently, execution cost and task difficulty remain separate properties of a tool invocation.

B.3 Benchmark Construction and Generalization

The catalog was developed in the context of publicly available OSWorld tasks. Its reusable application controls support diverse goals, while specialized procedures expose useful operations for particular subtasks. For example, `calculate_employee_ages` is a Compound interface tailored to an age-calculation operation. Task specificity thus cuts across the three granularity levels: it depends on an interface's semantics, beyond the number or breadth of operations it exposes. The eight Workflow annotations capture only one part of this specialization.

OSMCP measures how agents use this mixture of interfaces on the covered tasks. Performance reflects both the available tool support and the agent's selection and coordination of tools. Because the tasks were public during catalog development, tool design can incorporate knowledge of the evaluation setting; prior model exposure and prompt or parameter tuning are further possible sources of adaptation, including for a training-free controller. The present results characterize this setting. Transfer to held-out task families or applications with a frozen catalog, and the effect of removing task-aligned interfaces across granularity levels, remain open empirical questions.

B.4 Benchmark Artifacts and Runtime Dependencies

The accompanying task manifest (`benchmark/osmcp_task_ids.json`) identifies the selected tasks, and the annotated catalog (`benchmark/osmcp_tool_catalog.csv`) records the static interfaces. The definition metadata (`benchmark/osmcp_definition.json`) supplies the base OSWorld snapshot identifier and file hashes. These artifacts specify the static task and tool inventory. Its realization

in a running system depends on server availability, tool schemas, application versions, dependencies, VM snapshots, input assets, and initialization settings. The interfaces available to an individual run can therefore differ from the static catalog. The definition files record benchmark coverage rather than every historical execution environment.

Execution histories are maintained per backend, server, and action name and retained across tasks. The implementation defaults to a window of 100 calls, a three-call cold start, a success-rate threshold of 0.4, and a minimum reliability multiplier of 0.2. Independently of the success-rate term, the consecutive-failure deduction is 0.1 at the third failure and increases by 0.1 per additional failure, capped at 0.3. These histories track the runtime execution-status signal defined in Section 3.

Table 2 presents the available systems with their native models, interfaces, and step budgets. A step may denote an agent decision, an MCP invocation, or a GUI action, with additional operations executed inside tools or shell commands. The resulting scores describe performance at the system level. Isolating the controller's contribution calls for a shared model and catalog under matched execution policies, while time and token measurements provide separate measures of cost.

B.5 Description-Noise Evaluation Setup

Figure 3 uses 18 queries, three per application, over 109 cached tools from six OSMCP applications. With names and server identities fixed, descriptions undergo deterministic word deletion or cyclic swapping at nominal rates of 5/10/20/30%; the lexical control covers 10/20%. We average the Jaccard overlap between clean and perturbed Top-10 tool-name sets over the queries, using one perturbation realization per rate. Expected-server coverage is the fraction of queries whose Top-10 contains at least one tool from the expected application. Dense and hybrid rankings coincide because the hybrid prefilter's 120-candidate limit retains all 109 tools; the lexical control uses normalized query-term overlap. This evaluation tests local ranking, without server planning or task execution.

C Complete List of MCP Tools

We provide the complete list of 122 MCP tools across seven applications in Tables 6–12.

Table 6: MCP tools for **Calc** (32 tools).

Tool Name	Description	Operation Type	Granularity
`Fill_blank_above`	Fill blank cells with values from cells above	Write	Compound
`add_new_row`	Add new row with data and formulas	Write	Compound
`calculate_annual_asset_changes`	Calculate year-over-year changes in asset values	Transform	Workflow
`calculate_employee_ages`	Calculate ages from birthday column	Transform	Compound
`clean_text_formatting`	Clean and format text in columns (remove whitespace, apply title case)	Transform	Compound
`copy_column`	Copy a column to a new sheet	Write	Compound
`create_dropdown_validation`	Create dropdown validation for column	Create	Compound
`create_line_chart`	Create line chart from data	Create	Workflow
`create_new_sheet`	Create a new empty sheet	Create	Atomic
`create_pivot_table`	Create pivot table from column data	Create	Workflow
`create_sheet_with_headers`	Create new sheet with merged header cells	Create	Compound
`export_csv`	Export sheet to CSV format	Export	Compound
`export_pdf`	Export spreadsheet to PDF with custom sizing	Export	Compound
`extract_unique_values`	Extract unique values from a column to another column	Transform	Compound
`fill_sequence_numbers`	Fill column with formatted sequence numbers	Write	Compound
`format_column_decimals`	Format column with decimal places and currency symbols	Write	Compound
`freeze_panes`	Freeze rows and/or columns	Control	Atomic
`get_all_document_contents`	Read all data from document	Read	Compound
`get_zoom`	Get current spreadsheet zoom level	Read	Atomic
`hide_rows_with_value`	Hide or show rows containing specific values	Control	Compound
`highlight_weekends`	Highlight weekend dates with background color	Write	Compound
`manage_sheets`	Perform sheet operations (rename, copy)	Write	Compound
`mcp_libreoffice_status`	Check LibreOffice setup status and get installation instructions	Read	Atomic

Table 6 (continued)

Tool Name	Description	Operation Type	Granularity
pad_numbers	Format numbers with custom padding (e.g., 123 → 0000123)	Transform	Compound
reorder_columns	Reorder columns in specified sequence	Transform	Compound
set_cell_content	Set content in specific cells with formatting	Write	Atomic
set_locale_formatting	Set locale-specific number formatting	Configuration	Compound
set_zoom	Set spreadsheet zoom level	Control	Atomic
setup_resource_sheets	Set up resource management sheets with rename, backup, and offline version	Create	Workflow
sort_data	Sort spreadsheet data by column	Transform	Compound
summarize_revenue_by_promotion	Create revenue summary grouped by promotion type	Transform	Workflow
transpose_table	Transpose table data (rows ↔ columns)	Transform	Compound

Table 7: MCP tools for **Impress** (23 tools).

Tool Name	Description	Operation Type	Granularity
mcp_libreoffice_add_image_to_slides	Add images to slides with configurable size and position	Write	Compound
mcp_libreoffice_add_slide_notes	Add notes to slides, either custom text or copying the slide's title text	Write	Compound
mcp_libreoffice_analyze_slides	Analyze and extract all text contents from slides with formatting info	Read	Compound
mcp_libreoffice_configure_auto_save	Configure auto-save functionality to automatically save every N minutes	Configuration	Atomic
mcp_libreoffice_create_new_slides	Create new blank slides with configurable positioning	Create	Compound
mcp_libreoffice_duplicate_slides	Duplicate specific slides or the last N slides	Create	Compound
mcp_libreoffice_export_to_pdf	Export presentation to PDF with full quality control	Export	Compound
mcp_libreoffice_export_to_png	Export presentation to PNG with flexible resolution settings	Export	Compound
mcp_libreoffice_get_slide_contents	Extract and analyze all text contents from slides	Read	Compound
mcp_libreoffice_reposition_images	Reposition existing images with predefined or custom positions	Write	Compound
mcp_libreoffice_reposition_title	Reposition title text on slides (top, middle, bottom)	Write	Compound
mcp_libreoffice_restore_slide_panel	Restore the left slide panel when accidentally closed	Control	Atomic
mcp_libreoffice_save_as_powerpoint	Save presentation as PowerPoint (.pptx/.ppt)	Export	Compound
mcp_libreoffice_set_font_styles	Set font style (font family) for text in specific slides	Write	Compound
mcp_libreoffice_set_slide_number_color	Set color for slide numbers	Write	Compound
mcp_libreoffice_set_slide_orientation	Set slide orientation (portrait/landscape)	Write	Compound
mcp_libreoffice_set_text_bold	Set text boldness with selective targeting (all/titles/content)	Write	Compound
mcp_libreoffice_set_text_colors	Set text colors for all textboxes in slides	Write	Compound
mcp_libreoffice_set_title_colors	Set title text color for specific slides	Write	Compound
mcp_libreoffice_set_title_font_size	Set title font size for specific slides	Write	Compound
mcp_libreoffice_set_title_text	Set title text for specific slides	Write	Compound
mcp_libreoffice_set_title_underlines	Set title text underline for specific slides	Write	Compound
mcp_libreoffice_status	Check LibreOffice setup status and get installation instructions	Read	Atomic

Table 8: MCP tools for **Writer** (21 tools).

Tool Name	Description	Operation Type	Granularity
mcp_libreoffice_add_content_at_cursor	Add content at the current cursor position	Write	Atomic
mcp_libreoffice_add_page_numbers	Add page numbers to the document	Write	Compound
mcp_libreoffice_append_content	Append plain text content to the document	Write	Atomic
mcp_libreoffice_append_formatted_content	Append formatted content to the document	Write	Compound
mcp_libreoffice_change_font	Change font throughout the entire document text	Write	Compound
mcp_libreoffice_conditional_text_formatting	Apply formatting based on text conditions (e.g., color words by pattern)	Write	Workflow
mcp_libreoffice_data_processing	Process structured data (remove duplicates, filter, sort, etc.)	Transform	Workflow

Table 8 (continued)

Tool Name	Description	Operation Type	Granularity
mcp_libreoffice_export_pdf	Export document to PDF	Export	Compound
mcp_libreoffice_insert_image	Insert an image at the current cursor position	Write	Compound
mcp_libreoffice_insert_table	Insert a table at the current cursor position	Create	Compound
mcp_libreoffice_italic_text_formatting	Modify italic text formatting for better discernibility	Write	Compound
mcp_libreoffice_line_spacing	Modify line spacing for specified text portions	Write	Compound
mcp_libreoffice_page_break	Insert page breaks to create blank pages or separate content	Write	Atomic
mcp_libreoffice_paragraph_formatting	Apply formatting to entire paragraphs (strikethrough, bold, etc.)	Write	Compound
mcp_libreoffice_remove_highlighting	Remove all highlighting/background colors from text	Write	Compound
mcp_libreoffice_replace_formatted_text	Replace text with formatted content	Write	Compound
mcp_libreoffice_replace_text	Replace text in the document	Write	Atomic
mcp_libreoffice_sentence_separation	Separate sentences by adding empty lines between them	Transform	Compound
mcp_libreoffice_status	Check LibreOffice setup status and get installation instructions	Read	Atomic
mcp_libreoffice_text_alignment	Change text alignment (left, center, right, justify)	Write	Compound
mcp_libreoffice_text_case_transformation	Transform text case (uppercase, lowercase, title case, etc.)	Transform	Compound

Table 9: MCP tools for **Thunderbird** (13 tools).

Tool Name	Description	Operation Type	Granularity
add_email_attachment	Add an attachment to a compose window	Write	Atomic
bulk_flag_folder	Flag all messages in a specific folder	Write	Compound
get_accounts	Get email accounts via API	Read	Atomic
get_auto_forward_rules	Get all configured automatic email forwarding rules	Read	Atomic
get_folders	Get folders via API	Read	Atomic
get_subject_filter_rules	Get all configured subject-based email filtering rules	Read	Atomic
mcp_thunderbird_create_folder	Create a new folder via API	Create	Atomic
mcp_thunderbird_new_identity	Create a new mail identity via API	Create	Atomic
mcp_thunderbird_theme_change	Change the theme/appearance	Configuration	Atomic
open_compose_window	Open a new email compose window	Create	Atomic
setup_auto_forward	Set up automatic email forwarding rule	Configuration	Compound
setup_subject_filter	Set up automatic email filtering based on subject keywords	Configuration	Compound
update_message	Update a message via API	Write	Atomic

Table 10: MCP tools for **VS Code** (13 tools).

Tool Name	Description	Operation Type	Granularity
code_checker	Retrieve diagnostics from language services for the active workspace	Read	Atomic
execute_command	Execute a command in an integrated terminal	Control	Compound
execute_vscode_command	Execute any VSCode command by its command ID	Control	Atomic
focus_editor	Open file in editor and navigate to specific line and column	Control	Atomic
get_terminal_output	Retrieve the output from a specific terminal by ID	Read	Atomic
list_debug_sessions	List all active debug sessions in the workspace	Read	Atomic
list_directory	List directory contents in a tree format	Read	Atomic
list_vscode_commands	List available VSCode commands with optional filtering	Read	Atomic
preview_url	Open a URL in VSCode's built-in simple browser	Control	Atomic
restart_debug_session	Restart a debug session with the provided configuration	Control	Compound
start_debug_session	Start a new debug session with the provided configuration	Control	Compound
stop_debug_session	Stop all debug sessions matching the provided session name	Control	Atomic

Table 10 (continued)

Tool Name	Description	Operation Type	Granularity
`text_editor`	File manipulation using VSCode's native APIs	Write	Compound

Table 11: MCP tools for **VLC** (11 tools).

Tool Name	Description	Operation Type	Granularity
`add_url_to_playlist`	Add a URL to playlist without playing immediately	Write	Atomic
`adjust_volume`	Increase or decrease the volume by a specified percentage	Control	Atomic
`get_available_videos`	Get all available videos with their path	Read	Atomic
`get_status`	Get the current status of playback	Read	Atomic
`get_volume`	Get the current volume level (as a percentage)	Read	Atomic
`seek`	Seek to a specific position in the video	Control	Atomic
`set_volume`	Set the volume level (0–200, where 100 is normal)	Control	Atomic
`show_video`	Show the video using path and subtitle language code	Control	Compound
`stream_url`	Stream a video/audio URL using the HTTP API	Control	Compound
`take_video_snapshot`	Take a snapshot of the current video scene	Export	Compound
`vlc_control`	Control playback with actions: play, pause, stop, fullscreen	Control	Atomic

Table 12: MCP tools for **GIMP** (9 tools).

Tool Name	Description	Operation Type	Granularity
`adjust_brightness_contrast`	Adjust the brightness and contrast of an image or layer	Transform	Atomic
`call_api`	Call GIMP API through the socket connection	Control	Workflow
`convert_to_palette_mode`	Convert an image to palette-based (indexed color) mode	Transform	Compound
`enhance_color_vibrancy`	Enhance color vibrancy and saturation using various methods	Transform	Compound
`export_image`	Export an image to a specified location with a given filename	Export	Compound
`flip_image`	Flip an image or layer horizontally or vertically	Transform	Atomic
`get_images`	Get information about currently open images	Read	Atomic
`make_background_transparent`	Make the background transparent using various methods	Transform	Compound
`move_text_layer`	Move a text layer in the specified direction	Write	Atomic