CHATDEV 3.0: ENABLING DISTRIBUTED MULTI-AGENT COLLABORATION UNDER EXECUTION ASYMMETRY

Anonymous authors
Paper under double-blind review

ABSTRACT

LLM-based agents are expanding into complex digital workflows that span diverse users and devices with heterogeneous information, authority, and execution states. We characterize these environments by *execution asymmetry*: while information can cross boundaries through communication, actions remain bounded by user authorization and environment-local interfaces. We present CHATDEV 3.0, a distributed multi-agent runtime that explicitly structures collaboration along user-authority and environment-locality boundaries. CHATDEV 3.0 employs *User Agents* to maintain user intent, context, and authority, and *Device Agents* to execute delegated tasks through local observations and action interfaces. A unified communication protocol and an asynchronous event-driven runtime allow agents to operate concurrently and synchronize only when task dependencies require it. Evaluations on cross-user coordination and cross-device execution benchmarks show that CHATDEV 3.0 achieves higher aggregate task success than the evaluated baselines, while also improving execution efficiency in cross-device tasks.

1 INTRODUCTION

As large language models (LLMs) continue to improve in reasoning Shao et al. (2024); Wang et al. (2023); Wei et al. (2022), planning Chen et al. (2024c); Qiao et al. (2024); Valmeekam et al. (2023), and tool use Qin et al. (2024); Schick et al. (2023), LLM-based agents are evolving from conversational assistants into computational actors that perceive their environments, formulate plans, and take actions Qian et al. (2024); XAgent Team (2023). Through code execution Hong et al. (2024); Yang et al. (2024), web browsing Gur et al. (2024); Deng et al. (2023), and graphical user interface interaction Qin et al. (2025); Zhang et al. (2025), agents can directly execute tasks in the digital world. These workflows form a closed loop between reasoning and execution: generated actions continuously alter persistent environment states, while subsequent decisions adapt to the evolving environment and resulting feedback Sun et al. (2025); Zhang et al. (2026c); Sun et al. (2026).

As the capabilities of a single agent remain constrained by its underlying model, available context, and accessible resources Chen et al. (2024a; 2025); Kim et al. (2026a), multi-agent systems (MAS) address these limitations by organizing complementary capabilities across agents through specialization Qian et al. (2024); Long et al. (2024), mutual verification and correction Qi et al. (2025); Qian et al. (2025), and parallel execution Team et al. (2026); Zhang et al. (2026b); Liu et al. (2026). A complementary line of MAS work models social systems by instantiating individuals as agents with distinct goals, memories, observations, and relationships. In these systems, information is inherently distributed because each agent has access only to its local context and internal state Park et al. (2023; 2026); Yang et al. (2025); Piao et al. (2026); Liu et al. (2024b).

Together, these lines of research reveal a common principle behind multi-agent collaboration: distribution creates the need for coordination, as shown in Figure 1. **Distributed capabilities** motivate the composition of complementary skills Hong et al. (2024); Li et al. (2025), while **asymmetric information** motivates the exchange of locally held knowledge Liu et al. (2024b); Wang et al. (2026). Digital-world tasks introduce a further dimension, which we term **execution asymmetry**. Unlike information or reasoning results, execution cannot generally be transferred through communication

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.

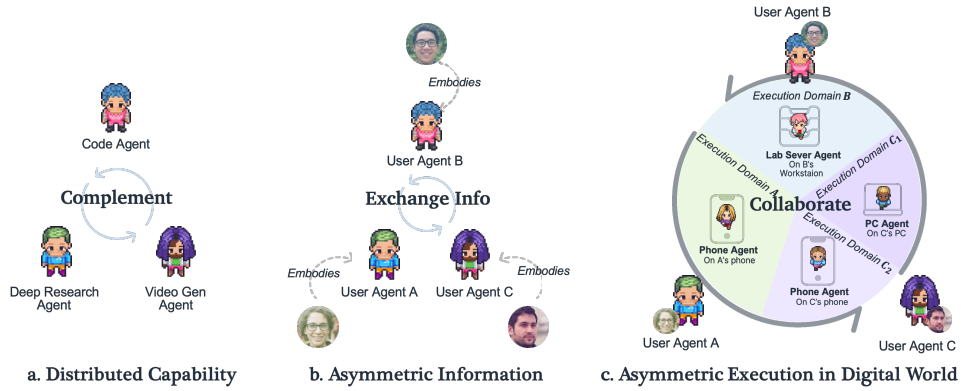


Figure 1: From distributed ability and information to distributed execution: digital-world collaboration requires coordinating across non-interchangeable user- and device-level execution boundaries.

alone: actions must be authorized by the relevant principal and grounded in the state and interfaces of the environment in which they are performed Rawles et al. (2025); Xie et al. (2024). For tasks spanning such non-interchangeable execution boundaries, collaboration is therefore a structural requirement for task completion rather than merely a means of improving reasoning.

Execution asymmetry arises along two complementary axes. At the user level, principal-specific authorization determines *who* is permitted to act South et al. (2025); Yang et al. (2026a); at the device level, environment-local state and action interfaces determine *where* an action can be realized Zhang et al. (2026a); Chen et al. (2026). For example, an agent acting for Bob cannot operate Alice's phone without her authorization; even an agent authorized by Alice cannot perform a phone-local action from her desktop without access to the phone's current state and interface. This structure creates a tension for centralized coordination. Consolidating multiple users into a shared agent can entangle user-specific decision contexts, privacy constraints, and authorization boundaries Rezazadeh et al. (2025); Yang et al. (2026a;b). Meanwhile, centralizing execution across devices requires a single reasoning context to track evolving states, histories, and feedback from multiple environments Zhang et al. (2026a). As tasks become longer and span more environments, this concentration of heterogeneous context can increase context-management overhead and make relevant information harder to utilize effectively Liu et al. (2024a); Zhang et al. (2024).

This motivates a factorized execution model that preserves user-level authority and environment-local state while enabling explicit coordination across them. We instantiate this model in ChatDev 3.0, a distributed multi-agent runtime comprising *User Agents*, which maintain user-specific intent, context, and authority, and *Device Agents*, which maintain environment-local state and execute actions through local interfaces. A unified protocol enables autonomous communication among User Agents and execution delegation to Device Agents, while an asynchronous event-driven runtime allows independent execution domains to progress concurrently. This design enables seamless cross-user and cross-device coordination while preserving the boundaries of information, authority, and execution. Across cross-user and cross-device tasks, ChatDev 3.0 attains the highest macro-average task success among the evaluated baselines, and the lowest execution time among the multi-agent systems, improving on both TeamBench Full Kim et al. (2026b) and UFO^3 Zhang et al. (2026a). The single-agent baselines remain faster by avoiding coordination altogether, but they do so by collapsing the user and device boundaries that ChatDev 3.0 is designed to preserve.

2 Method

To support effective and efficient execution of digital tasks spanning multiple users and devices, ChatDev 3.0 consists of three complementary components, as illustrated in Figure 2: (i) a *boundary-aligned agent architecture* that assigns user-specific authority and context to User Agents and environment-local state and execution to Device Agents, enabling reliable execution within each boundary; (ii) a *unified coordination protocol* that supports autonomous cross-boundary communication and coordination; and (iii) an *asynchronous event-driven runtime* that allows independent

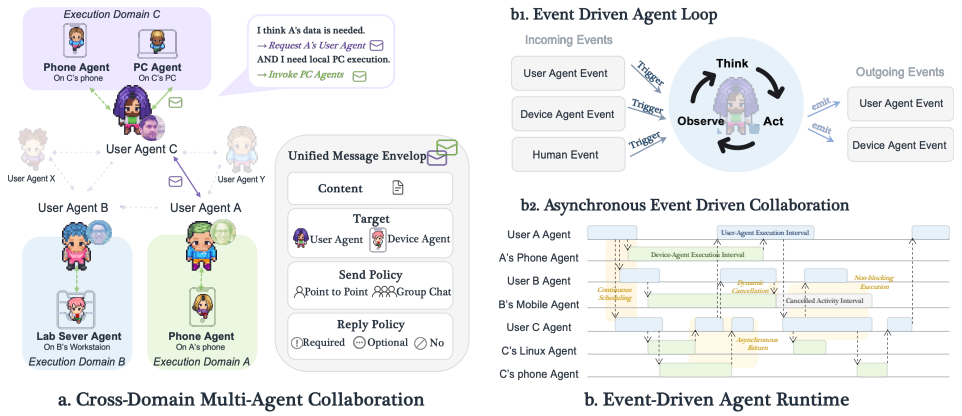


Figure 2: Overview of CHATDEV 3.0. The left shows the collaboration architecture across heterogeneous user and device agents, while the right depicts the asynchronous event-driven runtime of individual agents.

coordination and execution to proceed concurrently, improving efficiency under heterogeneous local latencies.

2.1 BOUNDARY-ALIGNED AGENT ARCHITECTURE

Digital actions operate within stateful environments with locally defined observations, actions, and dynamics. We define an *execution domain* $d \in \mathcal{D}$ as a distinct stateful environment, such as a physical device, virtual device, or isolated computational endpoint. Domain-local execution is modeled as a partially observable Markov decision process (POMDP) Xie et al. (2024); Kaelbling et al. (1998). Each domain has state space $\mathcal{S}_d$, observation space $\mathcal{O}_d$, action space $\mathcal{A}_d$, transition model $\mathcal{T}_d$, and observation model Ω_d. At local step h, an agent receives a partial observation $o_{d,h} \in \mathcal{O}_d$ of the underlying state $s_{d,h} \in \mathcal{S}_d$. It selects actions through the domain-specific interface, conditioned on its delegated objective and interaction history.

Although execution is domain-local, many multi-user and multi-device tasks require coordination across domains. For a feasible task G, let $\mathcal{K}_G \subseteq 2^{\mathcal{D}}$ denote the collection of inclusion-minimal domain sets sufficient for completion. Sufficiency is evaluated under fixed initial states, action interfaces, and user authorizations. We focus on *cross-domain tasks*, for which

$$\min_{K \in \mathcal{K}_G} |K| > 1, \tag{1}$$

meaning that no single execution domain suffices to complete the task. Such tasks require coordination across environments that maintain their own state.

This requirement motivates an architecture that preserves domain-local execution while coordinating across user and environment boundaries. As shown in Figure 2, CHATDEV 3.0 realizes this separation through User Agents and Device Agents. Each User Agent U_u maintains user-specific intent, context, and authorization and coordinates across accessible domains, while each Device Agent $D_{u,d}$ executes delegated subtasks within domain d under user u's authority using local observations and execution feedback. This factorization preserves user and environment execution boundaries while enabling coordinated task completion across them.

2.2 AUTONOMOUS COORDINATION THROUGH A UNIFIED PROTOCOL

Cross-domain tasks induce coordination patterns that depend on the evolving task state and locally available information. Rather than prescribing task-specific interaction sequences, fixed communication rounds, or static coordination topologies Liu et al. (2024c); Zhuge et al. (2024), CHATDEV 3.0 provides a unified communication protocol through which agents construct coordination decisions at runtime. Based on its local context, each agent determines whether communication is necessary, which agent should receive the message, what information should be conveyed, and whether subsequent execution depends on a response Chi et al. (2025); Li et al. (2026).

Both horizontal communication among User Agents and vertical communication between User and Device Agents use the same message primitive:

$$m = \langle \mu, \rho, c \rangle \in \mathcal{M}. \tag{2}$$

Here, $\mathcal{M}$ denotes the space of messages supported by the unified protocol. The routing field μ specifies the recipient set, supporting unicast and multicast. The response policy $\rho \in \{\text{required}, \text{optional}, \text{none}\}$ specifies whether the message introduces a dependency on subsequent feedback, and c denotes the message payload. By separating routing, content, and response dependency, the protocol allows communication patterns to adapt to the agent's current context without requiring a predefined workflow. The resulting interaction structure and synchronization dependencies are therefore determined during execution rather than fixed in advance.

Autonomous communication remains constrained by user authority. Cross-user information exchange is mediated by User Agents: a Device Agent $D_{u,d}$ reports execution feedback to its governing User Agent U_u, which determines whether subsequent coordination with other users is required. CHATDEV 3.0 thus enables adaptive cross-boundary coordination while preserving user-level control over information flow.

2.3 ASYNCHRONOUS EXECUTION RUNTIME

Collaborative tasks across multi-user and multi-device environments involve heterogeneous execution processes with significant execution latency variance. Device Agents may autonomously carry out long-running, multi-step local tasks, such as executing computational scripts Yang et al. (2024) or interacting with graphical user interfaces Xie et al. (2024), while human inputs Shao et al. (2026) and peer messages may arrive asynchronously. Globally synchronized decision rounds or fixed time steps would couple these independent processes, introducing unnecessary blocking delays for slower local execution and thereby reducing concurrency and responsiveness Hooper et al. (2026).

As illustrated in Figure 2(b1), CHATDEV 3.0 therefore organizes each User Agent around an event-driven loop. Rather than advancing according to globally synchronized rounds, a User Agent is activated by incoming events from three sources: human inputs, messages from other User Agents, and feedback from its Device Agents. Incoming events are queued and processed according to each User Agent's local event stream, allowing different agents to progress independently rather than through globally synchronized rounds. When an event is dequeued, it triggers a ReAct cycle Yao et al. (2022), during which the User Agent updates its local context, determines its next action, and may emit new events to other User Agents or Device Agents.

This event-driven design naturally enables asynchronous collaboration across User Agents and Device Agents, as illustrated in Figure 2(b2). Once a subtask is delegated, a Device Agent executes it independently of the User Agent loop and returns progress or completion through asynchronous events. The User Agent can therefore continuously schedule new work, process results from other devices, or coordinate with peer User Agents while previously dispatched tasks remain in execution. Ongoing activities may also be dynamically cancelled or revised in response to newly arriving events. Consequently, execution is synchronized only where an explicit dependency exists. An action that requires the result of a pending Device Agent waits for that result, while independent branches of coordination and execution continue without blocking. This prevents heterogeneous local execution latency from propagating into global synchronization, enabling concurrent and responsive collaboration across multi-user and multi-device environments.

3 EXPERIMENTS

3.1 RESEARCH QUESTIONS

We evaluate CHATDEV 3.0 along two complementary dimensions of distributed collaboration: *cross-user coordination* and *cross-device execution*. These settings examine the user- and device-level boundaries motivating the architecture in Section 2.1. Cross-user coordination evaluates task completion when information and responsibilities are partitioned across isolated user domains, using TeamBench Kim et al. (2026b). Cross-device execution evaluates task completion across heterogeneous device environments. We use MDCBench-CLI adapted from NebulaBench Zhang et al.

(2026a) and our newly curated MDCBench-GUI, including single-device reference tasks alongside cross-device workflows.

We ask: *RQ1: How reliably does* CHATDEV 3.0 *coordinate task completion when information and authority are partitioned across isolated user domains?*

RQ2: How effectively does CHATDEV 3.0 *scale execution across heterogeneous devices and interface modalities?*

3.2 RQ1: HOW RELIABLY DOES CHATDEV 3.0 COORDINATE TASK COMPLETION WHEN INFORMATION AND AUTHORITY ARE PARTITIONED ACROSS ISOLATED USER DOMAINS?

3.2.1 SETTINGS

Benchmark. To evaluate CHATDEV 3.0 under user-level information and role-specific authority boundaries, we use 88 tasks from TeamBench Kim et al. (2026b), spanning 21 task categories. TeamBench partitions the information, execution authority, and task responsibilities required for task completion across three roles: the Planner has access to the full task specification but not the workspace, the Executor receives a condensed brief and can operate on the workspace, and the Verifier inspects the resulting state and produces the final attestation. We map these roles to three User Agents in CHATDEV 3.0, each paired with a Device Agent operating within the corresponding access scope. This setup serves as a controlled proxy for cross-user collaboration, where task-relevant information and execution authority are distributed across distinct user domains. Task selection and role-domain mappings are detailed in Appendices B.1.1 and B.1.3.

Baselines. We compare CHATDEV 3.0 against two TeamBench baselines. **Solo** removes the role boundaries by consolidating specification access, workspace execution, and verification authority within a single agent. **Full** preserves TeamBench's native Planner–Executor–Verifier decomposition and access restrictions under its role-separated coordination protocol. Across all settings, we align the initial Docker environments, backbone models, and deterministic graders while retaining each system's native coordination and execution mechanisms. Environment-facing tool capabilities are broadly aligned; interface differences are documented in Appendix D.1.3.

Metrics. We report three metrics. **Solution Pass Rate** is the fraction of runs whose final workspace satisfies all functional and artifact checks. **Verification Pass Rate** additionally requires sign-off on 36 of the 88 tasks; on the remaining 52, it equals Solution Pass Rate. Solo grants final-attestation authority to its single agent. **Execution Time** is the mean wall-clock time from task submission to termination.

3.2.2 RESULTS

CHATDEV 3.0 improves collaboration performance under the same boundary constraints. As shown in Table 1, CHATDEV 3.0 improves the macro-average Solution Pass Rate over TeamBench Full from 53.41% to 57.10% and the Verification Pass Rate from 47.73% to 54.55%, while reducing mean execution time from 713.33 s to 512.45 s. Both pass rates improve on three of the four backbone models. Importantly, these gains are achieved under the same Planner–Executor–Verifier decomposition and role-level information and authority boundaries as TeamBench Full, indicating improved task completion and verification success without relaxing the underlying collaboration constraints.

CHATDEV 3.0 exceeds the centralized reference in task success despite coordination overhead. TeamBench Solo consolidates task information, workspace execution, and final-attestation authority within a single agent. It avoids cross-agent coordination and achieves the lowest mean execution time (125.55 s). By comparison, CHATDEV 3.0 preserves the information and authority boundaries, requiring agents to exchange information and coordinate execution. Despite this latency cost, CHATDEV 3.0 achieves higher macro-average Solution and Verification Pass Rates than Solo (57.10% vs. 47.73% and 54.55% vs. 46.31%, respectively). TeamBench Full also achieves a higher macro-average Verification Pass Rate than Solo (47.73% vs. 46.31%), suggesting that part of the gain may arise from separating execution and verification into distinct agent roles.

Table 1: Performance comparison on TeamBench. Best results are highlighted in blue and bold, and second-best distinct results are underlined.

Backbone	Method	Verification Pass Rate (%) ↑	Solution Pass Rate (%) ↑	Execution Time (s) ↓
Closed-source models				
GPT-5.6-Luna	TeamBench Solo	**55.68**	**56.82**	**107.82**
	TeamBench Full	32.95	53.41	1075.80
	Ours	53.41	**56.82**	462.65
Gemini-3.6-Flash	TeamBench Solo	39.77	40.91	**73.77**
	TeamBench Full	51.14	51.14	284.49
	Ours	**56.82**	**56.82**	447.29
Open-source models				
Qwen3.5-27B	TeamBench Solo	40.91	44.32	**147.07**
	TeamBench Full	50.00	51.14	866.98
	Ours	**57.95**	**57.95**	533.21
Kimi-K2.6	TeamBench Solo	48.86	48.86	**173.55**
	TeamBench Full	**56.82**	**57.95**	626.05
	Ours	50.00	56.82	606.66
Average	TeamBench Solo	46.31	47.73	**125.55**
	TeamBench Full	47.73	53.41	713.33
	Ours	**54.55**	**57.10**	512.45

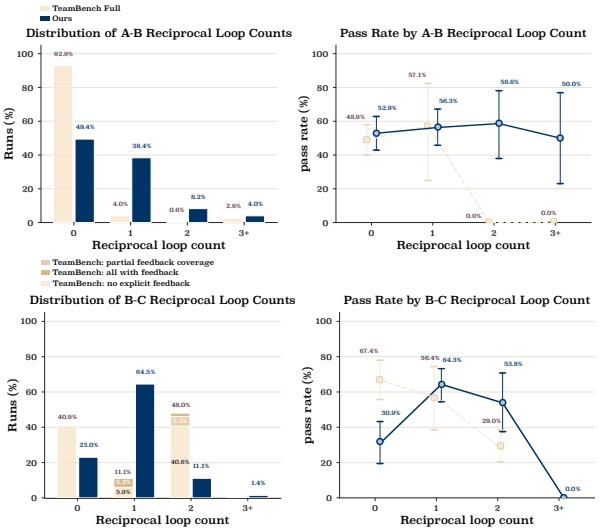


Figure 3: Reciprocal communication patterns and verification pass rates of CHATDEV 3.0 and TeamBench Full.

CHATDEV 3.0 enables autonomous and demand-driven reciprocal communication. Figure 3 illustrates distinct communication topologies between CHATDEV 3.0 and TeamBench Full. For Planner–Executor (A–B), TeamBench Full largely follows a prescribed one-way workflow, whereas CHATDEV 3.0 allows either User Agent to autonomously initiate communication based on its local needs. Such coordination can proceed independently of ongoing Device-Agent execution under the asynchronous runtime, rather than being serialized by delegation. For Executor–Verifier (B–C), TeamBench Full triggers revision through a predefined rule when the Verifier does not issue `PASS`, while CHATDEV 3.0 allows the Verifier to autonomously request further execution based on its local assessment. Accordingly, CHATDEV 3.0 exhibits a broader range of reciprocal loop depths, reflecting communication decisions made at runtime rather than fixed by the workflow. We further examine whether deeper reciprocal communication is associated with better outcomes. For A–B interactions, Verification Pass Rate shows no consistent trend with loop depth. For B–C interactions, it decreases from 64.3% to 53.8% in CHATDEV 3.0 and from 56.4% to 29.0% in TeamBench Full when moving from one to two reciprocal loops. These results suggest that additional exchanges are more likely to arise on trajectories that remain unresolved after an initial interaction, a pattern consistent with demand-driven communication rather than uniformly imposed interaction rounds.

3.3 RQ2: How effectively does ChatDev 3.0 scale execution across heterogeneous devices and interface modalities?

3.3.1 Settings

Benchmarks. To evaluate cross-device execution, we use two testbeds spanning command-line and graphical interfaces. We denote each task's annotated execution-domain count by n_{dom}. **MDCBench-CLI** adapts 45 of the 55 NebulaBench tasks Zhang et al. (2026a) through task filtering and execution-environment adaptation. It contains 44 cross-device tasks with $n_{\mathrm{dom}} \in \{2, 3, 4\}$ and one single-device reference task. **MDCBench-GUI** comprises 20 manually curated tasks, including 11 cross-device workflows spanning a Linux desktop GUI and an Android virtual device, and nine single-device reference tasks. Android actions use the Android Debug Bridge (ADB). Additional benchmark details are provided in Appendix B.1.2.

Baselines. We compare ChatDev 3.0 against two baselines: **TopoClaw**, a single root agent that directly invokes device-level tools Huang et al. (2026), and **UFO^3**, a centralized planner that decomposes and delegates tasks to device agents Zhang et al. (2026a). On MDCBench-CLI, all methods are evaluated with GPT-5.2, Gemini-3.5-Flash, Qwen3.5-27B, and Kimi-K2.5, using three trajectories per task. On MDCBench-GUI, we evaluate Kimi-K2.5 with one trajectory per task. Across methods, we align device environments, available action capabilities, and backbone models while preserving each system's native coordination logic and control loop.

Evaluation Metrics. We evaluate both task effectiveness and execution efficiency. For MDCBench-CLI, **Pass@1** is the mean success rate across the three trajectories per task, while **Pass@3** is the fraction of tasks solved by at least one of the three trajectories. We additionally report end-to-end execution time and the parallel efficiency metric from Zhang et al. (2026a),

$$E_r = 100 L_r / T_r^{\mathrm{E2E}}, \tag{3}$$

where for trajectory r, L_r is the reconstructed weighted critical-path duration and T_r^{E2E} is end-to-end execution time. Higher E_r indicates a larger critical-path fraction, conditional on the inferred graph and node durations, reflecting more effective exploitation of the concurrency available outside the task's critical dependency path (for details, see Appendix B.2.2). For MDCBench-GUI, we evaluate each task once and report **Pass@1**, together with the rubric-based **Process Score** that assesses intermediate execution quality.

3.3.2 Results

ChatDev 3.0 improves reliability and execution efficiency over UFO^3. As shown in Table 2, ChatDev 3.0 consistently achieves higher Pass@1 and critical-path efficiency with lower execution time than UFO^3 across all four backbone models, while matching or improving Pass@3. The advantage is even more pronounced on MDCBench-GUI in Table 3, where ChatDev 3.0 improves both task success and process quality by substantial margins. These results indicate that ChatDev 3.0 provides more reliable cross-device execution while better exploiting parallelism than centralized device-level orchestration.

Table 3: Performance on MDCBench-GUI.

Method	Pass@1 (%) ↑	Process Score (%) ↑
TopoClaw	25.00	44.33
UFO^3	35.00	59.65
Ours	**75.00**	**79.35**

ChatDev 3.0 improves task success on context-intensive cross-device tasks. Compared with the centralized TopoClaw baseline, ChatDev 3.0 achieves higher macro-average Pass@1 and Pass@3, with a substantially larger advantage on MDCBench-GUI as shown in Table 3. TopoClaw remains faster on task execution, reflecting the lower coordination overhead of centralized execution. However, long-horizon GUI tasks require maintaining richer and continually evolving state across multiple devices; a single root agent must track this information within one context, whereas ChatDev 3.0 distributes device-local state across Device Agents. This contrast suggests that centralized execution favors low coordination overhead, while distributed state management becomes increasingly beneficial as cross-device context grows.

Table 2: Overall performance on MDCBench-CLI. We compare CHATDEV 3.0 with UFO^3 and TopoClaw, a centralized single-agent baseline. Time is measured in seconds, and Efficiency denotes the mean critical-path fraction $100L_r/T_r^{\mathrm{E2E}}$ across runs, which is compared only between CHATDEV 3.0 and UFO^3. The final block reports macro averages across all backbones, and best and second-best results are computed across the reported methods.

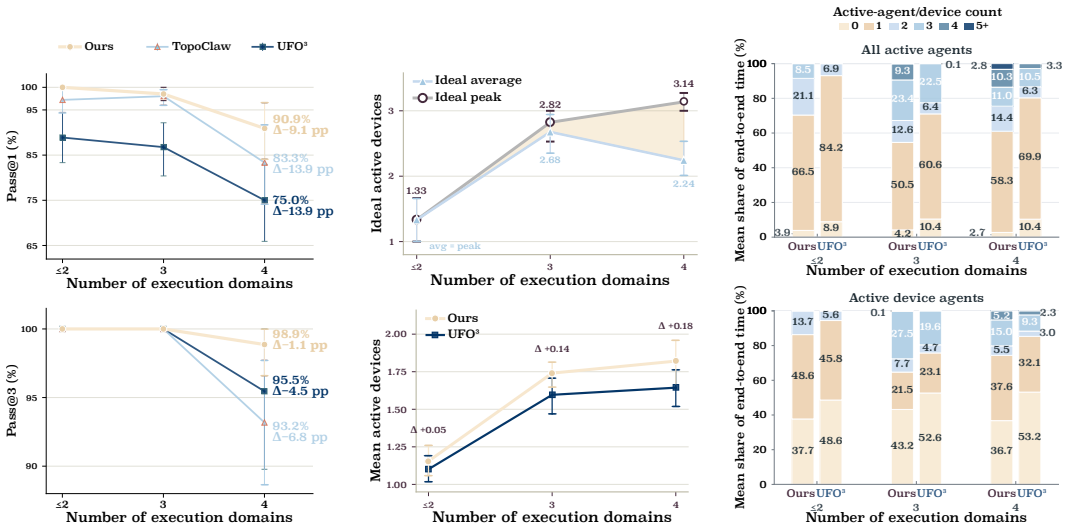
Backbone	Method	Pass@1 (%) ↑	Pass@3 (%) ↑	Time (s) ↓	Efficiency (%) ↑
Closed-source models					
GPT-5.2	TopoClaw	94.81	**100.00**	**29.82**	–
	UFO^3	85.19	**100.00**	238.50	42.27
	Ours	**97.78**	**100.00**	59.03	**56.43**
Gemini-3.5-Flash	TopoClaw	85.19	91.11	**57.56**	–
	UFO^3	91.11	**97.78**	155.82	46.59
	Ours	**96.30**	**97.78**	87.27	**55.84**
Open-source models					
Qwen3.5-27B	TopoClaw	**94.07**	95.56	**110.38**	–
	UFO^3	70.37	97.78	373.05	36.88
	Ours	**94.07**	**100.00**	118.17	**56.53**
Kimi-K2.5	TopoClaw	88.89	**100.00**	**96.46**	–
	UFO^3	78.52	95.56	520.04	42.17
	Ours	**91.85**	**100.00**	350.87	**45.14**
Average	TopoClaw	90.74	96.67	**73.55**	–
	UFO^3	81.30	97.78	321.85	41.98
	Ours	**95.00**	**99.44**	153.83	**53.49**

CHATDEV 3.0 maintains high task success and parallelism under scaling domain complexity. Tasks spanning more execution domains generally involve a broader execution scope and greater coordination complexity, as successful completion requires coordinating more independently stateful environments and cross-domain dependencies. As the minimum required execution scope increases from at most two domains to four domains, Pass@1 decreases for all methods (Fig. 4a). CHATDEV 3.0 degrades more gracefully than both TopoClaw and UFO^3 as execution spans more domains, with smaller declines in both Pass@1 and Pass@3, indicating that CHATDEV 3.0 maintains task success more consistently as execution spans more domains.

At the same time, larger execution scopes create greater opportunities for parallel execution. Ideal peak parallelism increases from 1.33 to 3.14 across the domain-count groups (Fig. 4b). CHATDEV 3.0 increasingly exploits this opportunity relative to UFO^3: its advantage in active devices per logical slice grows from 0.05 to 0.14 and 0.18 as the number of domains increases. Consistently, CHATDEV 3.0 spends a larger fraction of end-to-end execution time with multiple agents active across all domain-count groups (Fig. 4c). Together, these results show that CHATDEV 3.0 sustains higher cross-domain concurrency while exhibiting a smaller degradation in task success as execution scope expands.

4 RELATED WORK

Multi-Agent Systems LLM-based multi-agent systems distribute reasoning and decision-making across multiple agents to solve complex tasks Qian et al. (2024); Li et al. (2023); Chen et al. (2024b). Agents may specialize by role or expertise Dang et al. (2025); Long et al. (2024); Zhang et al. (2026d), or provide additional inference-time computation through collaboration Wang et al. (2025a); Du et al. (2024); Wunderlich et al. (2026). Complementary work models humans as agents with distinct goals, private information, and authorization boundaries Park et al. (2023); Zhou et al. (2024b); Rezazadeh et al. (2025); Yang et al. (2026b), supporting cross-user coordination Liu et al. (2024b); Jhamtani et al. (2025), memory sharing under access control Rezazadeh et al. (2025), and scoped delegation of authority Yang et al. (2026b).



(a) Success retention across execution domains

(b) Available and realized logical parallelism

(c) Sustained concurrency over end-to-end time

Figure 4: **Effectiveness and parallel execution across execution domains.** Tasks are grouped by their annotated execution-domain count n_{dom} ($\leq$ 2, 3, or 4). The first group includes one single-device reference task and five two-device tasks. **(a)** Cross-backbone macro-average Pass@1 and Pass@3. **(b)** Ideal parallelism treats reference-DAG nodes as unit-duration operations; observed parallelism counts active devices per event-defined slice. DAG reconstruction and observed-concurrency definitions appear in Appendix B.2.2. **(c)** Mean per-trajectory end-to-end time composition at each concurrency level. Active agents include the coordinating agent and device agents, whereas active devices count only distinct execution domains with an active device agent. The coordinating agent is the User Agent in ChatDev 3.0 and the Constellation Agent in UFO^3. Error bars denote 95% bootstrap confidence intervals.

Agents Acting in Digital Environments LLM agents interact with digital environments through command-line tools and code Yang et al. (2024); Wang et al. (2025b); Bin et al. (2026) and graphical interfaces Cheng et al. (2024); Zhang et al. (2025), performing tasks such as web browsing Zhou et al. (2024a); Deng et al. (2023) and multi-application workflows Xie et al. (2024); Wang et al. (2024). Beyond single-environment execution, cross-device workflows require agents to transfer intermediate results, maintain shared state, and reason over dependencies across environments Chen et al. (2026); Xu et al. (2025). They also introduce challenges in coordinating heterogeneous capabilities, asynchronous execution, and distributed task dependencies, motivating frameworks such as UFO^3 Zhang et al. (2026a). When devices belong to different users or administrative contexts, coordination must further respect identity and authorization boundaries Huang et al. (2026). ChatDev 3.0 separates user-level coordination from device-local execution to support collaboration across users and devices while preserving both boundaries.

5 Conclusion

We presented ChatDev 3.0, a distributed multi-agent runtime for collaboration under execution asymmetry. ChatDev 3.0 aligns agent roles with user-authority and environment-locality boundaries, enables autonomous cross-boundary coordination through a unified communication protocol, and decouples coordination from heterogeneous local execution through an asynchronous event-driven runtime. Experiments on TeamBench, MDCBench-CLI, and MDCBench-GUI show higher macro-average task success than the evaluated baselines, with higher macro-average verification performance on TeamBench. Mean execution time is lower than TeamBench Full on TeamBench and UFO^3, and critical-path efficiency on MDCBench-CLI is higher than UFO^3. These results support explicitly preserving and coordinating across user and device boundaries, while highlighting the trade-off between centralized latency and distributed state management.

REFERENCES

Yi Bin, Xiaoyang Yuan, Haoxi Zeng, Wencheng Ye, Wenqi Shao, Chen Qian, Wei Ye, Yujuan Ding, Zheng Wang, Pengpeng Zeng, Jingkuan Song, and Heng Tao Shen. Terminal agents: A survey of ai agents in command-line environments. arXiv preprint arXiv:2608.20485, 2026. URL https://arxiv.org/abs/2608.20485.

Lingjiao Chen, Jared Quincy Davis, Boris Hanin, Peter Bailis, Ion Stoica, Matei Zaharia, and James Zou. Are More LLM Calls All You Need? Towards the Scaling Properties of Compound AI Systems. In *Advances in Neural Information Processing Systems (NeurIPS 2024)*, 2024a. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/51173cf34c5faac9796a47dc2fdd3a71-Abstract-Conference.html.

Lingjiao Chen, Jared Quincy Davis, Boris Hanin, Peter Bailis, Matei Zaharia, James Zou, and Ion Stoica. Optimizing Model Selection for Compound AI Systems. arXiv preprint arXiv:2502.14815, 2025. URL https://arxiv.org/abs/2502.14815.

Weize Chen, Yusheng Su, Jingwei Zuo, Cheng Yang, Chenfei Yuan, Chi-Min Chan, Heyang Yu, Yaxi Lu, Yi-Hsin Hung, Chen Qian, Yujia Qin, Xin Cong, Ruobing Xie, Zhiyuan Liu, Maosong Sun, and Jie Zhou. AgentVerse: Facilitating Multi-Agent Collaboration and Exploring Emergent Behaviors. In *The Twelfth International Conference on Learning Representations (ICLR 2024)*, 2024b. URL https://openreview.net/forum?id=EHg5GDnyq1.

Ziru Chen, Michael White, Ray Mooney, Ali Payani, Yu Su, and Huan Sun. When is Tree Search Useful for LLM Planning? It Depends on the Discriminator. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL 2024)*, 2024c. URL https://aclanthology.org/2024.acl-long.738/.

Zixiang Chen, Yuheng Lu, Zihao Cheng, Zeming Liu, Jizeng Bai, Ziye Huang, Zhiyin Lin, Zihan Li, Yuhang Guo, Yunhong Wang, and Haifeng Wang. JarvisGUI: Towards cross-device GUI agents with dynamic task composition. arXiv preprint arXiv:2609.10451, 2026. URL https://arxiv.org/abs/2609.10451.

Kanzhi Cheng, Qiushi Sun, Yougang Chu, Fangzhi Xu, Yantao Li, Jianbing Zhang, and Zhiyong Wu. SeeClick: Harnessing GUI Grounding for Advanced Visual GUI Agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL 2024)*, 2024. URL https://aclanthology.org/2024.acl-long.505/.

Zewen Chi, Li Dong, Qingxiu Dong, Yaru Hao, Xun Wu, Shaohan Huang, and Furu Wei. The era of agentic organization: Learning to organize with language models, 2025. URL https://arxiv.org/abs/2510.26658.

Yufan Dang, Chen Qian, Xueheng Luo, Jingru Fan, Zihao Xie, Ruijie Shi, Weize Chen, Cheng Yang, Xiaoyin Che, Ye Tian, Xuantang Xiong, Lei Han, Zhiyuan Liu, and Maosong Sun. Multi-agent collaboration via evolving orchestration. In *Advances in Neural Information Processing Systems (NeurIPS 2025)*, 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/f1320d2e2842169c6fc89dcbd80e94d0-Abstract-Conference.html.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Samuel Stevens, Boshi Wang, Huan Sun, and Yu Su. Mind2Web: Towards a Generalist Agent for the Web. In *Advances in Neural Information Processing Systems, Datasets and Benchmarks Track (NeurIPS 2023)*, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/5950bf290a1570ea401bf98882128160-Abstract-Datasets_and_Benchmarks.html.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B. Tenenbaum, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate. In *Proceedings of the 41st International Conference on Machine Learning (ICML 2024)*, 2024. URL https://proceedings.mlr.press/v235/du24e.html.

Izzeddin Gur, Hiroki Furuta, Austin V Huang, Mustafa Safdari, Yutaka Matsuo, Douglas Eck, and Aleksandra Faust. A Real-World WebAgent with Planning, Long Context Understanding, and Program Synthesis. In *The Twelfth International Conference on Learning Representations (ICLR 2024)*, 2024. URL https://openreview.net/forum?id=9JQtrumvg8.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. MetaGPT: Meta Programming for A Multi-Agent Collaborative Framework. In *The Twelfth International Conference on Learning Representations (ICLR 2024)*, 2024. URL https://openreview.net/forum?id=VtmBAGCN7o.

Coleman Hooper, Minwoo Kang, Suhong Moon, Nicholas Lee, Eric Wen, John Wawrzynek, Michael W. Mahoney, Yakun Sophia Shao, Amir Gholami, and Kurt Keutzer. Speculative interaction agents: Building real-time agents with asynchronous i/o and speculative tool calling, 2026. URL https://arxiv.org/abs/2605.13360.

Heyuan Huang, Yeyi Guan, Jihong Wang, Mingzhi Wang, Jiamu Zhou, Xiangmou Qu, Jiaxin Yin, Xin Liao, Xingyu Lou, and Jun Wang. Topoclaw: A human-centric and topology-aware agent operating system. arXiv preprint arXiv:2605.15556, 2026. URL https://arxiv.org/abs/2605.15556.

Harsh Jhamtani, Jacob Andreas, and Benjamin Van Durme. LLM agents for coordinating multi-user information gathering. In *Findings of the Association for Computational Linguistics (Findings of ACL 2025)*, 2025. URL https://aclanthology.org/2025.findings-acl.916/.

Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 1998. doi: https://doi.org/10.1016/S0004-3702(98)00023-X. URL https://www.sciencedirect.com/science/article/pii/S000437029800023X.

Yubin Kim, Ken Gu, Chanwoo Park, Chunjong Park, Samuel Schmidgall, A. Ali Heydari, Yao Yan, Zhihan Zhang, Yuchen Zhuang, Yun Liu, Mark Malhotra, Paul Pu Liang, Hae Won Park, Yuzhe Yang, Xuhai Xu, Yilun Du, Shwetak Patel, Tim Althoff, Daniel McDuff, and Xin Liu. Capable language models can outgrow the benefits of collaboration. *Nature Machine Intelligence*, 2026a. URL https://www.nature.com/articles/s42256-026-01268-y.

Yubin Kim, Chanwoo Park, Taehan Kim, Eugene Park, Samuel Schmidgall, Salman Rahman, Chunjong Park, Cynthia Breazeal, Xin Liu, Hamid Palangi, Hae Won Park, and Daniel McDuff. Teambench: Evaluating agent coordination under enforced role separation. arXiv preprint arXiv:2605.07073, 2026b. URL https://arxiv.org/abs/2605.07073.

Guohao Li, Hasan Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. CAMEL: Communicative Agents for "Mind" Exploration of Large Language Model Society. In *Advances in Neural Information Processing Systems (NeurIPS 2023)*, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/a3621ee907def47c1b952ade25c67698-Abstract-Conference.html.

Haoran Li, Ziyi Su, Yun Xue, Zhiliang Tian, Yiping Song, and Minlie Huang. Advancing collaborative debates with role differentiation through multi-agent reinforcement learning. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (ACL 2025)*, July 2025.

Rui Li, Zeyu Zhang, Xiaohe Bo, Quanyu Dai, Chaozhuo Li, Feng Wen, and Xu Chen. Towards adaptive, scalable, and robust coordination of llm agents: A dynamic ad-hoc networking perspective, 2026. URL https://arxiv.org/abs/2602.08009.

Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics (TACL 2024)*, 2024a. URL https://aclanthology.org/2024.tacl-1.9/.

Shikun Liu, Mufei Li, Dongqi Fu, Haoyu Wang, Yinglong Xia, Hong Li, Hong Yan, and Pan Li. Towards direct latent-space synthesis for parallel branches in llm-agent workflows. arXiv preprint arXiv:2606.14672, 2026. URL https://arxiv.org/abs/2606.14672.

Wei Liu, Chenxi Wang, Yifei Wang, Zihao Xie, Rennai Qiu, Yufan Dang, Zhuoyun Du, Weize Chen, Cheng Yang, and Chen Qian. Autonomous agents for collaborative task under information asymmetry. In *Advances in Neural Information Processing Systems (NeurIPS 2024)*, 2024b. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/0534abc9e6db91683d82186ef0d68202-Abstract-Conference.html.

Zijun Liu, Yanzhe Zhang, Peng Li, Yang Liu, and Diyi Yang. A Dynamic LLM-Powered Agent Network for Task-Oriented Agent Collaboration. In *First Conference on Language Modeling (COLM)*, 2024c. URL https://openreview.net/forum?id=XII0Wp1XA9.

Do Xuan Long, Duong Ngoc Yen, Anh Tuan Luu, Kenji Kawaguchi, Min-Yen Kan, and Nancy F. Chen. Multi-expert Prompting Improves Reliability, Safety and Usefulness of Large Language Models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing (EMNLP 2024)*, 2024. URL https://aclanthology.org/2024.emnlp-main.1135/.

Joon Sung Park, Joseph C. O'Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology (UIST 2023)*, 2023. URL https://doi.org/10.1145/3586183.3606763.

Joon Sung Park, Carolyn Q. Zou, Jonne Kamphorst, Niles Egan, Aaron Shaw, Benjamin Mako Hill, Carrie Cai, Meredith Ringel Morris, Percy Liang, Robb Willer, and Michael S. Bernstein. Llm agents grounded in self-reports enable general-purpose simulation of individuals. arXiv preprint arXiv:2411.10109, 2026. URL https://arxiv.org/abs/2411.10109.

Jinghua Piao, Yuwei Yan, Jun Zhang, Nian Li, Junbo Yan, Xiaochong Lan, Zhihong Lu, Zhiheng Zheng, Jing Yi Wang, Di Zhou, Chen Gao, Fengli Xu, Fang Zhang, Ke Rong, Jun Su, and Yong Li. Agentsociety: Large-scale simulation of llm-driven generative agents advances understanding of human behaviors and society. *iFuture*, 2026. URL https://www.sciopen.com/article/10.26599/IF.2026.9710004.

Zhenting Qi, Mingyuan MA, Jiahang Xu, Li Lyna Zhang, Fan Yang, and Mao Yang. Mutual Reasoning Makes Smaller LLMs Stronger Problem-Solver. In *The Thirteenth International Conference on Learning Representations (ICLR 2025)*, 2025. URL https://openreview.net/forum?id=6aHUmotXaw.

Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, Juyuan Xu, Dahai Li, Zhiyuan Liu, and Maosong Sun. ChatDev: Communicative Agents for Software Development. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL 2024)*, 2024. URL https://aclanthology.org/2024.acl-long.810/.

Chen Qian, Zihao Xie, YiFei Wang, Wei Liu, Kunlun Zhu, Hanchen Xia, Yufan Dang, Zhuoyun Du, Weize Chen, Cheng Yang, Zhiyuan Liu, and Maosong Sun. Scaling Large Language Model-based Multi-Agent Collaboration. In *The Thirteenth International Conference on Learning Representations (ICLR 2025)*, 2025. URL https://openreview.net/forum?id=K3n5jPkrU6.

Shuofei Qiao, Runnan Fang, Ningyu Zhang, Yuqi Zhu, Xiang Chen, Shumin Deng, Yong Jiang, Pengjun Xie, Fei Huang, and Huajun Chen. Agent planning with world knowledge model. In *Advances in Neural Information Processing Systems (NeurIPS 2024)*, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/d032263772946dd5026e7f3cd22bce5b-Abstract-Conference.html.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. ToolLLM: Facilitating Large Language Models to Master 16000+ Real-world APIs. In *The Twelfth International Conference on Learning Representations (ICLR 2024)*, 2024. URL https://openreview.net/forum?id=dHng2O0Jjr.

Yujia Qin, Yining Ye, Junjie Fang, Haoming Wang, Shihao Liang, Shizuo Tian, Junda Zhang, Jiahao Li, Yunxin Li, Shijue Huang, Wanjun Zhong, Kuanye Li, Jiale Yang, Yu Miao, Woyu Lin, Longxiang Liu, Xu Jiang, Qianli Ma, Jingyu Li, Xiaojun Xiao, Kai Cai, Chuang Li, Yaowei Zheng, Chaolin Jin, Chen Li, Xiao Zhou, Minchao Wang, Haoli Chen, Zhaojian Li, Haihua Yang, Haifeng Liu, Feng Lin, Tao Peng, Xin Liu, and Guang Shi. Ui-tars: Pioneering automated gui interaction with native agents. arXiv preprint arXiv:2501.12326, 2025. URL https://arxiv.org/abs/2501.12326.

Christopher Rawles, Sarah Clinckemaillie, Yifan Chang, Jonathan Waltz, Gabrielle Lau, Marybeth Fair, Alice Li, William E. Bishop, Wei Li, Folawiyo Campbell-Ajala, Daniel Kenji Toyama, Robert James Berry, Divya Tyamagundlu, Timothy P. Lillicrap, and Oriana Riva. Androidworld: A dynamic benchmarking environment for autonomous agents. In *ICLR*. OpenReview.net, 2025.

Alireza Rezazadeh, Zichao Li, Ange Lou, Yuying Zhao, Wei Wei, and Yujia Bao. Collaborative memory: Multi-user memory sharing in llm agents with dynamic access control. arXiv preprint arXiv:2505.18279, 2025. URL https://arxiv.org/abs/2505.18279.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessi, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language Models Can Teach Themselves to Use Tools. In *Advances in Neural Information Processing Systems (NeurIPS 2023)*, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/d842425e4bf79ba039352da0f658a906-Abstract-Conference.html.

Yijia Shao, Vinay Samuel, Yucheng Jiang, John Yang, and Diyi Yang. Collaborative gym: A framework for enabling and evaluating human-agent collaboration, 2026. URL https://arxiv.org/abs/2412.15701.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. arXiv preprint arXiv:2402.03300, 2024. URL https://arxiv.org/abs/2402.03300.

Tobin South, Samuele Marro, Thomas Hardjono, Robert Mahari, Cedric Deslandes Whitney, Alan Chan, and Alex Pentland. Position: AI agents need authenticated delegation. In *Proceedings of the 42nd International Conference on Machine Learning (ICML 2025)*, 2025. URL https://proceedings.mlr.press/v267/south25a.html.

Weiwei Sun, Miao Lu, Zhan Ling, Kang Liu, Xuesong Yao, Yiming Yang, and Jiecao Chen. Scaling long-horizon llm agent via context-folding. arXiv preprint arXiv:2510.11967, 2025. URL https://arxiv.org/abs/2510.11967.

Yiyou Sun, Xinyang Han, Weichen Zhang, Yuanbo Pang, Tianyu Wang, Yuhan Cao, Yixiao Huang, Chris Duroiu, Haoyun Zhang, Jeffrey Lin, Weishu Zhang, Tyler Zeng, Ying Yan, Bo Liu, Hanson Wen, Mingyang Xu, Xiaoyuan Liu, Zimeng Chen, Weiyan Shi, Amanda Dsouza, Vincent Sunn Chen, Patrick Bryant, Carl Boettiger, Yamini Rangan, Bradley Rothenberg, Kyle Steinfeld, Arvind Rao, Tapio Schneider, Georgios Yannakakis, Laure Zanna, Kaan Ozbay, Ida Sim, Tarek Zohdi, George Em Karniadakis, Jack Gallant, Teresa Head-Gordon, Yushan Li, Wenxi Deng, Tao Sun, Huiqi Wang, Zhun Wang, Justin Xu, Chris Yuhao Liu, Yafei Cheng, Rongwang Hu, Aras Bacho, Shengcao Cao, Zengyi Qin, Yixiong Chen, Hengduan Fan, Hao Liu, Lin Zeng, Shashank Muralidhar Bharadwaj, Litian Gong, Yingxuan Yang, Maojia Song, Ruheng Wang, Zongzheng Zhang, Honglin Bao, Shuo Lu, Jianhong Tu, Zhonghua Wang, Zheng Zhang, Zijiao Chen, Yanqiong Jiang, Zhendong Li, Bohan Lyu, Chang Ma, Peiran Xu, Benran Zhang, Shangding Gu, Haoyue Hua, Haoyang Li, Wanzhe Liao, Chengzhi Liu, Junbo Peng, Haoran Sun, Zechen Xu, Bo Chen, Jiayi Cheng, Yi Jiang, Keying Kuang, Yuan Li, Youbang Pan, Ziyan Rao, Alexander Schubert, Yifan Shen, Vincent Siu, Xiatao Sun, Kangqi Zhang, Xiaopan Zhang, Yuchen Zhu, Ishaan Singh Chandok, Lei Ding, Jingxuan Fan, Andrew Glover, Jiaming Hu, Yiran Hu, Wenbo Huang, Zixin Jiang, Haoran Jin, Lukas Kim, Ming Liu, Yang Liu, Alireza Rafiei, Xuhuan Shen, Kunyang Sun, Sophia Sun, Ting Sun, Eric Wang, Yixin Wang, Hanwen Xing, Sihan Xu, Yuzheng Xu, Zhongxing Xu, Zhiling Yan, Boqin Yuan, Ruiqi Zhang, Yifan Zhang, Zibo Zhao, Liana, Santanu Bosu Antu, Haoyue Bai, Carlo Bosio, Joseph Cavanagh, Patricia Cavazos-Rehg, Tianxing Chen, Xuewen Chen, Yipu Chen, Chenyu Zhu, Chen Dai, Stefano De Castro,

Yunfu Deng, Kaustubh Dhole, Jiayuan Ding, Chenchen Du, Zhehang Du, Hao Fan, Run-Ze Fan, Hengyu Fu, Shi Gu, Yifan Gu, Charlie Guo, Baihe Huang, Baixiang Huang, Rimika Jaiswal, Zhihan Jiang, Ran Jin, Erin Kasson, Xin Lan, Joseph Lee, Deren Lei, Chenyu Li, Daofeng Li, Haitao Li, Hongwei Li, Jingyan Li, Xiao Li, Yi Li, Yinsheng Li, Yuangang Li, Zhixu Li, Wenyu Liang, Longtai Liao, Kevin Qinghong Lin, Andy Zeyi Liu, Che Liu, Jiaming Liu, Kaiyuan Liu, Xuan Liu, Pan Lu, Wenbo Lv, Yicheng Lyu, Qiuyang Mang, Kyle Montgomery, Yuzhou Nie, Ruoxi Ning, Jorin Overwiening, Xu Pan, Layna Paraboschi, Core Francisco Park, Justin Purnomo, Swati Rajwal, Scott Rankin, Bixuan Ren, Yiren Rong, HaoYang Shang, Ventus Shaw, Fiona Shen, Jiawei Shen, Minqi Shi, Shi Qiu, Huaxiu Yao, Tianneng Shi, Jonah So, Vladislav Susoy, Hannah Szlyk, Haocheng Wang, Jialu Wang, Wei Wang, Xinyu Wang, Zehao Wang, Dowling Wong, Angela Wu, Dehao Wu, Fangyu Wu, Mengyuan "Millie" Wu, Yu Wu, Yuchen Wu, Yuhao Wu, Qingpo Wuwu, Weihang Xiao, Yongyi Xiong, Fan Xu, Ruiling Xu, Mingxuan Yan, Benjamin Yang, Jirong Yang, Sen Yang, Xiaoli Yang, Yushi Yang, Haoran Ye, Xiaohu Yu, Zhengming Yu, Chenlong Zhang, Chi Zhang, Hanning Zhang, Hanwen Zhang, Junge Zhang, Kunpeng Zhang, Song Zhang, Wenjin Zhang, Wenshuo Zhang, Ying Zhang, Yizhi Zhang, Brian Zhao, Qijian Zhao, Yimin Zhao, Yuhaohua Zheng, Liwei Zhou, Tianyue Zhou, Sichen Zhu, Siqi Zhu, Yan Zhu, Yishu Zhu, Jierui Zuo, Chonghao Cai, Helena Casademunt, Wenjia Chen, Cheng Cheng, Nawen Deng, Rao Fu, Tianfu Fu, Yifan Han, He Ren, Zhenyu He, Qiao Jin, Langlang Li, Yuetai Li, Sylvia Liu, Lu Lu, Luqing Zhou, Subhabrata Mukherjee, Yunqi Ouyang, Yin Ren, Dawei Shi, Haoran Wu, Zhiyue Wu, Hannah Yao, Zhuoran Yi, Jenny Yu, Rhea Zhan, Hang Zhou, Blake Zhu, Junfan Zhu, Alan Yuille, Yang Liu, Russell Alan Poldrack, Jiachen Li, Zhenglu Li, Molei Tao, Jing Huang, Wenqi Shi, Costas Spanos, Lichao Sun, Chenguang Wang, Orson Xu, Zhen Dong, Hector Gomez, Aylin Caliskan, Ali Emami, Haimin Hu, Zhi Li, Lihui Liu, Murphy Niu, Yi Shao, Jianxin Sun, Mikko Tolonen, Ting Wang, Sanjiv Das, Yanjun Gao, Wenbo Guo, Erika J Schneider, Zhiyong Lu, Yian Ma, Mark Mueller, Radha Poovendran, Somayeh Sojoudi, Yinglun Zhu, and Dawn Song. Agents' last exam. arXiv preprint arXiv:2606.05405, 2026. URL https://arxiv.org/abs/2606.05405.

Kimi Team, Tongtong Bai, Yifan Bai, Yiping Bao, S. H. Cai, Yuan Cao, Y. Charles, H. S. Che, Cheng Chen, Guanduo Chen, Huarong Chen, Jia Chen, Jiahao Chen, Jianlong Chen, Jun Chen, Kefan Chen, Liang Chen, Ruijue Chen, Xinhao Chen, Yanru Chen, Yanxu Chen, Yicun Chen, Yimin Chen, Yingjiang Chen, Yuankun Chen, Yujie Chen, Yutian Chen, Zhirong Chen, Ziwei Chen, Dazhi Cheng, Minghan Chu, Jialei Cui, Jiaqi Deng, Muxi Diao, Hao Ding, Mengfan Dong, Mengnan Dong, Yuxin Dong, Yuhao Dong, Angang Du, Chenzhuang Du, Dikang Du, Lingxiao Du, Yulun Du, Yu Fan, Shengjun Fang, Qiulin Feng, Yichen Feng, Garimugai Fu, Kelin Fu, Hongcheng Gao, Tong Gao, Yuyao Ge, Shangyi Geng, Chengyang Gong, Xiaochen Gong, Zhuoma Gongque, Qizheng Gu, Xinran Gu, Yicheng Gu, Longyu Guan, Yuanying Guo, Xiaoru Hao, Weiran He, Wenyang He, Yunjia He, Chao Hong, Hao Hu, Jiaxi Hu, Yangyang Hu, Zhenxing Hu, Ke Huang, Ruiyuan Huang, Weixiao Huang, Zhiqi Huang, Tao Jiang, Zhejun Jiang, Xinyi Jin, Yu Jing, Guokun Lai, Aidi Li, C. Li, Cheng Li, Fang Li, Guanghe Li, Guanyu Li, Haitao Li, Haoyang Li, Jia Li, Jingwei Li, Junxiong Li, Lincan Li, Mo Li, Weihong Li, Wentao Li, Xinhang Li, Xinhao Li, Yang Li, Yanhao Li, Yiwei Li, Yuxiao Li, Zhaowei Li, Zheming Li, Weilong Liao, Jiawei Lin, Xiaohan Lin, Zhishan Lin, Zichao Lin, Cheng Liu, Chenyu Liu, Hongzhang Liu, Liang Liu, Shaowei Liu, Shudong Liu, Shuran Liu, Tianwei Liu, Tianyu Liu, Weizhou Liu, Xiangyan Liu, Yangyang Liu, Yanming Liu, Yibo Liu, Yuanxin Liu, Yue Liu, Zhengying Liu, Zhongnuo Liu, Enzhe Lu, Haoyu Lu, Zhiyuan Lu, Junyu Luo, Tongxu Luo, Yashuo Luo, Long Ma, Yingwei Ma, Shaoguang Mao, Yuan Mei, Xin Men, Fanqing Meng, Zhiyong Meng, Yibo Miao, Minqing Ni, Kun Ouyang, Siyuan Pan, Bo Pang, Yuchao Qian, Ruoyu Qin, Zeyu Qin, Jiezhong Qiu, Bowen Qu, Zeyu Shang, Youbo Shao, Tianxiao Shen, Zhennan Shen, Juanfeng Shi, Lidong Shi, Shengyuan Shi, Feifan Song, Pengwei Song, Tianhui Song, Xiaoxi Song, Hongjin Su, Jianlin Su, Zhaochen Su, Lin Sui, Jinsong Sun, Junyao Sun, Tongyu Sun, Flood Sung, Yunpeng Tai, Chuning Tang, Heyi Tang, Xiaojuan Tang, Zhengyang Tang, Jiawen Tao, Shiyuan Teng, Chaoran Tian, Pengfei Tian, Ao Wang, Bowen Wang, Chensi Wang, Chuang Wang, Congcong Wang, Dingkun Wang, Dinglu Wang, Dongliang Wang, Feng Wang, Hailong Wang, Haiming Wang, Hengzhi Wang, Huaqing Wang, Hui Wang, Jiahao Wang, Jinhong Wang, Jiuzheng Wang, Kaixin Wang, Linian Wang, Qibin Wang, Shengjie Wang, Shuyi Wang, Si Wang, Wei Wang, Xiaochen Wang, Xinyuan Wang, Yao Wang, Yejie Wang, Yipu Wang, Yiqin Wang, Yucheng Wang, Yuzhi Wang, Zhaoji Wang, Zhaowei Wang, Zhengtao Wang, Zhexu Wang, Zihan Wang, Zizhe Wang, Chu Wei, Ming Wei, Chuan Wen, Zichen Wen, Chengjie Wu, Haoning Wu, Junyan Wu, Rucong Wu, Wen-

hao Wu, Yuefeng Wu, Yuhao Wu, Yuxin Wu, Zijian Wu, Chenjun Xiao, Jin Xie, Xiaotong Xie, Yuchong Xie, Yifei Xin, Bowei Xing, Boyu Xu, Jianfan Xu, Jing Xu, Jinjing Xu, L. H. Xu, Lin Xu, Suting Xu, Weixin Xu, Xinbo Xu, Xinran Xu, Yangchuan Xu, Yichang Xu, Yuemeng Xu, Zelai Xu, Ziyao Xu, Junjie Yan, Yuzi Yan, Guangyao Yang, Hao Yang, Junwei Yang, Kai Yang, Ningyuan Yang, Ruihan Yang, Xiaofei Yang, Xinlong Yang, Ying Yang, Yi Yang, Yi Yang, Zhen Yang, Zhilin Yang, Zonghan Yang, Haotian Yao, Dan Ye, Wenjie Ye, Zhuorui Ye, Bohong Yin, Chengzhen Yu, Longhui Yu, Tao Yu, Tianxiang Yu, Enming Yuan, Mengjie Yuan, Xiaokun Yuan, Yang Yue, Weihao Zeng, Dunyuan Zha, Haobing Zhan, Dehao Zhang, Hao Zhang, Jin Zhang, Puqi Zhang, Qiao Zhang, Rui Zhang, Xiaobin Zhang, Y. Zhang, Yadong Zhang, Yangkun Zhang, Yichi Zhang, Yizhi Zhang, Yongting Zhang, Yu Zhang, Yushun Zhang, Yutao Zhang, Yutong Zhang, Zheng Zhang, Chenguang Zhao, Feifan Zhao, Jinxiang Zhao, Shuai Zhao, Xiangyu Zhao, Yikai Zhao, Zijia Zhao, Huabin Zheng, Ruihan Zheng, Shaojie Zheng, Tengyang Zheng, Junfeng Zhong, Longguang Zhong, Weiming Zhong, M. Zhou, Runjie Zhou, Xinyu Zhou, Zaida Zhou, Jinguo Zhu, Liya Zhu, Xinhao Zhu, Yuxuan Zhu, Zhen Zhu, Jingze Zhuang, Weiyu Zhuang, Ying Zou, and Xinxing Zu. Kimi k2.5: Visual agentic intelligence. arXiv preprint arXiv:2602.02276, 2026. URL https://arxiv.org/abs/2602.02276.

Karthik Valmeekam, Matthew Marquez, Sarath Sreedharan, and Subbarao Kambhampati. On the Planning Abilities of Large Language Models - A Critical Investigation. In *Advances in Neural Information Processing Systems (NeurIPS 2023)*, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/efb2072a358cefb75886a315a6fcf880-Abstract-Conference.html.

Chenxu Wang, Yongkun Yang, Boyuan Du, Shiwei Lin, and Huaping Liu. Llm agents for deliberative collaboration: A study on joint decision making under partial observability, 2026. URL https://arxiv.org/abs/2607.06157.

Junlin Wang, Jue Wang, Ben Athiwaratkun, Ce Zhang, and James Y. Zou. Mixture-of-agents enhances large language model capabilities. In *The Thirteenth International Conference on Learning Representations (ICLR 2025)*, 2025a. URL https://openreview.net/forum?id=h0ZfDIrj7T.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang H. Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng, Bill Qian, Yanjun Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, Junyang Lin, Robert Brennan, Hao Peng, Heng Ji, and Graham Neubig. OpenHands: An Open Platform for AI Software Developers as Generalist Agents. In *The Thirteenth International Conference on Learning Representations (ICLR 2025)*, 2025b. URL https://openreview.net/forum?id=OJd3ayDDoF.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-Consistency Improves Chain of Thought Reasoning in Language Models. In *The Eleventh International Conference on Learning Representations (ICLR 2023)*, 2023. URL https://openreview.net/forum?id=1PL1NIMMrw.

Zilong Wang, Yuedong Cui, Li Zhong, Zimin Zhang, Da Yin, Bill Yuchen Lin, and Jingbo Shang. Officebench: Benchmarking language agents across multiple applications for office automation. arXiv preprint arXiv:2407.19056, 2024. URL https://arxiv.org/abs/2407.19056.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems (NeurIPS 2022)*, 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/hash/9d5609613524ecf4f15af0f7b31abca4-Abstract-Conference.html.

Florian Valentin Wunderlich, Lars Benedikt Kaesberg, Jan Philip Wahle, Terry Ruas, and Bela Gipp. Multi-agent reasoning improves compute efficiency: Pareto-optimal test-time scaling. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics, Student Research Workshop (ACL 2026)*, 2026. URL https://aclanthology.org/2026.acl-srw.1/.

XAgent Team. XAgent: An Autonomous Agent for Complex Task Solving, 2023. URL https://github.com/OpenBMB/XAgent.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. OSWorld: Benchmarking Multimodal Agents for Open-Ended Tasks in Real Computer Environments. In *Advances in Neural Information Processing Systems, Datasets and Benchmarks Track (NeurIPS 2024)*, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/5d413e48f84dc61244b6be550f1cd8f5-Abstract-Datasets_and_Benchmarks_Track.html.

Tianqi Xu, Linyao Chen, Dai-Jie Wu, Yanjun Chen, Zecheng Zhang, Xiang Yao, Zhiqiang Xie, Yongchao Chen, Shilong Liu, Bochen Qian, Anjie Yang, Zhaoxuan Jin, Jianbo Deng, Philip Torr, Bernard Ghanem, and Guohao Li. CRAB: Cross-environment Agent Benchmark for Multimodal Language Model Agents. In *Findings of the Association for Computational Linguistics (Findings of ACL 2025)*, 2025. URL https://aclanthology.org/2025.findings-acl.1113/.

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. In *Advances in Neural Information Processing Systems (NeurIPS 2024)*, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/5a7c947568c1b1328ccc5230172e1e7c-Abstract-Conference.html.

Shu Yang, Shenzhe Zhu, Hao Zhu, José Ramón Enríquez, Di Wang, Alex Pentland, Michiel A. Bakker, and Jiaxin Pei. Multi-user large language model agents. arXiv preprint arXiv:2604.08567, 2026a. URL https://arxiv.org/abs/2604.08567.

Zhiqin Yang, Zhenyuan Zhang, Xianzhang Jia, Jun Song, Wei Xue, Yonggang Zhang, and Yike Guo. Clawnet: Human-symbiotic agent network for cross-user autonomous cooperation. arXiv preprint arXiv:2604.19211, 2026b. URL https://arxiv.org/abs/2604.19211.

Ziyi Yang, Zaibin Zhang, Zirui Zheng, Yuxian Jiang, Ziyue Gan, Zhiyu Wang, Zijian Ling, Jinsong Chen, Martz Ma, Bowen Dong, Prateek Gupta, Shuyue Hu, Zhenfei Yin, Guohao Li, Xu Jia, Lijun Wang, Bernard Ghanem, Huchuan Lu, Chaochao Lu, Wanli Ouyang, Yu Qiao, Philip Torr, and Jing Shao. Oasis: Open agent social interaction simulations with one million agents. arXiv preprint arXiv:2411.11581, 2025. URL https://arxiv.org/abs/2411.11581.

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.

Chaoyun Zhang, Liqun Li, He Huang, Chiming Ni, Bo Qiao, Si Qin, Yu Kang, Minghua Ma, Qingwei Lin, Saravan Rajmohan, and Dongmei Zhang. Ufo3: Weaving the digital agent galaxy. arXiv preprint arXiv:2511.11332, 2026a. URL https://arxiv.org/abs/2511.11332.

Chi Zhang, Zhao Yang, Jiaxuan Liu, Yanda Li, Yucheng Han, Xin Chen, Zebiao Huang, Bin Fu, and Gang Yu. Appagent: Multimodal agents as smartphone users. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems (CHI 2025)*, 2025. URL https://doi.org/10.1145/3706598.3713600.

Fan Zhang, Vireo Zhang, Shengju Qian, Haoxuan Li, Hao Wu, Jinyang Wu, Donghao Zhou, Zhihong Zhu, Zheng Lian, Xin Wang, and Pheng-Ann Heng. Orchestra-o1: Omnimodal agent orchestration. arXiv preprint arXiv:2606.13707, 2026b. URL https://arxiv.org/abs/2606.13707.

Jenny Zhang, Shengran Hu, Cong Lu, Robert Lange, and Jeff Clune. Darwin godel machine: Open-ended evolution of self-improving agents. arXiv preprint arXiv:2505.22954, 2026c. URL https://arxiv.org/abs/2505.22954.

Wentao Zhang, Liang Zeng, Yuzhen Xiao, Yongcong Li, Ce Cui, Yilei Zhao, Rui Hu, Yang Liu, Yahui Zhou, and Bo An. Agentorchestra: Orchestrating multi-agent intelligence with the tool-environment-agent(tea) protocol. arXiv preprint arXiv:2506.12508, 2026d. URL https://arxiv.org/abs/2506.12508.

Yusen Zhang, Ruoxi Sun, Yanfei Chen, Tomas Pfister, Rui Zhang, and Sercan Ö. Arik. Chain of agents: Large language models collaborating on long-context tasks. In *Advances in Neural Information Processing Systems (NeurIPS 2024)*, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/ee71a4b14ec26710b39ee6be113d7750-Abstract-Conference.html.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. WebArena: A Realistic Web Environment for Building Autonomous Agents. In *The Twelfth International Conference on Learning Representations (ICLR 2024)*, 2024a. URL https://openreview.net/forum?id=zlsj9akpaa.

Xuhui Zhou, Hao Zhu, Leena Mathur, Ruohong Zhang, Haofei Yu, Zhengyang Qi, Louis-Philippe Morency, Yonatan Bisk, Daniel Fried, Graham Neubig, and Maarten Sap. SOTOPIA: Interactive Evaluation for Social Intelligence in Language Agents. In *The Twelfth International Conference on Learning Representations (ICLR 2024)*, 2024b. URL https://openreview.net/forum?id=mM7VurbA4r.

Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and Jürgen Schmidhuber. GPTSwarm: Language Agents as Optimizable Graphs. In *Forty-first International Conference on Machine Learning (ICML)*, 2024. URL https://openreview.net/forum?id=uTC9AFXIhg.

A Appendix Overview

This appendix provides the supporting materials needed to interpret and reproduce our evaluation. The Evaluation Data section B.1 describes the construction of TeamBench and MDCBench, their task-level evaluation criteria, and their mapping to role and device domains. The Evaluation Details B.2 section defines the grading protocols, pass-rate metrics, timeout controls, concurrency measures, and human-evaluation procedures. The evaluation results section C provides model-specific analyzes of communication patterns, execution-domain scaling, task difficulty, and runtime concurrency. The Implementation Architecture D section describes the network topology, asynchronous orchestration, experimental execution backends, and additional system components. Finally, the Case Study E section illustrates how the proposed architecture coordinates heterogeneous agents and devices in representative workflows.

B Evaluation Implementation

B.1 Evaluation Data

B.1.1 TeamBench

We evaluate on TeamBench's Kim et al. (2026b) public 90-task leaderboard set, yielding a denominator of 88 after dropping two unusable tasks: one whose specification is missing and whose grader always returns zero, and another whose workspace contains no tests and whose grader only invokes a generic test runner. To ensure cross-batch score comparability and system stability, we identify and fix a series of upstream engineering defects in the original test harness and graders, and **we provide the corresponding repaired code and unified scripts in supplementary materials**. Specifically, these upstream issues stemmed from brittle evaluation logic, which led us to resolve tests incorrectly evaluated via log fragments instead of standard exit codes, inline checks that failed to parse due to indentation issues, sign-off checks accepting only a single lowercase token alongside write-read path mismatches for sign-off artifacts, scores overwritten through shared report directories across consecutive runs, program aborts upon emitting non-UTF-8 bytes, and version fragmentation caused by multiple historical grader versions in the repository—all of which were unified by re-scoring every TeamBench workspace with a single frozen copy using preinstalled offline dependencies, as shown in our supplementary materials.

B.1.2 MDCBench

MDCBench comprises a CLI subset adapted from NebulaBench Zhang et al. (2026a) and a newly authored GUI subset. Together, they evaluate execution across heterogeneous environments.

For the CLI part, we retain 45 of the original 55 NebulaBench tasks after excluding five single-device tasks and five additional tasks with unclear descriptions. The retained set includes one single-device task and 44 cross-device tasks; the device-count strata contain 6 tasks requiring at most two devices, 17 requiring three, and 22 requiring four. We adapt the Windows-side I/O in the original Windows-plus-Linux task specifications to a designated pc_workspace reporting directory. References to "Mac" in adapted task descriptions identify this reporting role rather than establishing the physical host operating system. Live services like SMTP, registry edits, and browser search are recast into reproducible local equivalents—specifically email drafts, mirrored configuration files, and curl. Complex actions such as cross-host image pushes and branch merges are rewritten as standard equivalents like crontab, tar distribution, and git merge-tree, with prompts explicitly enriched with absolute paths, sheet or column names, and JSON keys to ensure robust automated verification.

For the GUI part, we introduce a compact benchmark comprising 20 tasks—9 single-device tasks and 11 cross-device workflows—hand-crafted by 4 experienced computer science students to reflect realistic user goals. This GUI subset exercises a broad spectrum of real-world applications and online services, including Chrome, Calendar, Maps, Files, Weibo, YouTube, Inoreader, Keep Notes, Boostcamp, GitHub, and Bing. These tasks evaluate an agent's practical competence in navigating heterogeneous devices, retrieving dynamic online information, transferring content, manipulating files, and executing application-specific actions across real-world environments.

Table 4: Execution-domain mappings in the evaluation testbeds. TeamBench lists its prescribed role partition; MDCBench lists the annotated device domains required by task specifications.

Testbed	Local domains	Domain mapping	Annotated count
TeamBench	Planner, Executor, and Verifier	The three-role protocol assigns the full specification to the Planner, workspace operations to the Executor, and verification to the Verifier. The grader checks sign-off on 36 of 88 tasks.	3 prescribed roles,$n_{\mathrm{dom}} \in \{3\}$
MDCBench-CLI	Linux CLI endpoints and a PC reporting workspace	We collect the endpoints containing required nodes in the annotated task dependency graph. Each retained endpoint must provide a local action or endpoint-specific result needed by the final outcome.	$n_{\mathrm{dom}} \in \{1, 2, 3, 4\}$
MDCBench-GUI	Linux desktop GUI and Android GUI endpoints	Tasks are partitioned into single-device reference tasks (9 tasks, outcomes confined to one device) and cross-device tasks (11 tasks, outcomes span both devices with cross-boundary information flows).	$n_{\mathrm{dom}} \in \{1, 2\}$

The detailed data entries and specific statistics of these testbeds are summarized in the table 5 and table 6.

B.1.3 Mapping Testbeds to Execution Domains.

The execution supports in Section 2.1 are sets of domains under fixed authorization and access constraints, rather than sets of agent processes. For device tasks, we annotate the domains required by the task specification and reference graph. We denote their count by n_{dom}, separately from the number of agents or their instantaneous activity. These annotations identify the evaluated device scope; they do not constitute an exhaustive search over all sufficient domain subsets. TeamBench instead prescribes three role domains.

TeamBench tests whether ChatDev 3.0 can determine and communicate required contributions across fixed user-level boundaries. MDCBench then evaluates whether these contributions can be realized across device-level boundaries characterized by heterogeneous local execution context.The role and device-domain mappings across all evaluation testbeds are summarized in Table 4.

Table 5: Adapted NebulaBench tasks 1–45 and their task-level evaluation checks.

ID	Task	Task-level Evaluation Checks
1	Extract WARNING/ERROR lines from `/var/log/syslog` on Linux Server-1/2/3 and merge them into the `report` sheet of `benchmark/pc_workspace/log_detailed.xlsx` with columns Timestamp, Host, Level, Message (keep the template header). Write a simulated email body (To: `ops@example.com`) to `benchmark/pc_workspace/email_draft.txt`.	Did the agent keep the four headers and write WARNING/ERROR rows into the `report` sheet? Does `email_draft.txt` mention `ops@example.com`?
2	Search `Failed password` entries in `/var/log/auth.log` on Linux Server-1/2/3, count failures by source IP, and write `/root/top3_failed_ips.md` on Linux Server-1 listing the top three IPs with counts (count descending; ties by IP lexicographic order).	Does `/root/top3_failed_ips.md` exist on Linux Server-1, contain IPv4 addresses, and have at least three lines?

Continued on next page

ID	Task	Task-level Evaluation Checks
3	On Linux Server-1/2/3, if `/root/mock_app.log` is larger than 1 MB, gzip it to `/root/mock_app.log.1.gz` and recreate an empty log. After rotation the live log must be smaller than 1 MB.	Is `/root/mock_app.log` smaller than 1 MB on Linux Server-1, Linux Server-2, and Linux Server-3?
4	Count kernel OOM lines containing `Out of memory` or `oom_reaper`, reading `dmesg` first and falling back to `/var/log/benchmark/dmesg_oom_mock.log`. Reply as three `linux-N: <count>` lines plus `total: <sum>`.	Does the response list a numeric count for Linux Server-1, Linux Server-2, and Linux Server-3, plus a `total`?
5	Count whole-word WARN and ERROR (not prefixes such as WARNING) in Mac `benchmark/pc_workspace/mock_app.log` and each host's `/root/mock_app.log`. Reply as JSON with keys `mac`, `linux-1`, `linux-2`, `linux-3`, each holding integer WARN/ERROR.	Does the JSON include numeric WARN and ERROR fields for `mac` and Linux Server-1/2/3?
6	Parse the journalctl shim for the last 24 h (`Started`/`Stopped` lines) and write Host, Service, Started, Stopped into a `systemd_24h` sheet of `benchmark/pc_workspace/log_detailed.xlsx`, or an appended block with that marker.	Was `log_detailed.xlsx` modified with `systemd_24h` content covering Linux Server-1/2/3 and start/stop (or cron) text?
7	Read CPU model, cores, and max (else current) frequency with `lscpu`. Rank by cores descending, then frequency descending, then hostname ascending. Write one line `<hostname> <container IP>` to `benchmark/pc_workspace/benchmark_best_host.txt`.	Does `benchmark_best_host.txt` contain a single `linux-[123]` line followed by an IPv4 address?
8	Append `DEMO_ENV=staging` to `/etc/environment` on Linux Server-1 and Linux Server-2 (skip if present) so it matches from the start of the line. On Mac write the same line only to `benchmark/pc_workspace/demo_env.txt`; do not change shell rc files, LaunchAgents, or `defaults`. Reply JSON {`"linux-1":"ok","linux-2":"ok","mac":"ok"`}.	Does `demo_env.txt` exist? Does `/etc/environment` on Linux Server-1 and Linux Server-2 contain a leading `DEMO_ENV=staging` line?
9	Ensure user `sandbox` exists and is in group `sudo` on all Linux hosts (initially present but not sudo). Do not create a Mac system account; record Mac username, groups, and passwordless-sudo check in `benchmark/pc_workspace/sandbox_check.txt`.	Is `sandbox` in group `sudo` on Linux Server-1/2/3? Does `sandbox_check.txt` mention the Mac user, groups, and sudo?
10	Set `/root/backups` to mode 750 and owner `ops:ops` on Linux Server-1/2/3 (initially 755 `root:root`).	Does `stat` report `750 ops:ops` for `/root/backups` on all three hosts?

Continued on next page

ID	Task	Task-level Evaluation Checks
11	Write the single line `BENCH_CFG=enabled-2026` to Mac `benchmark/pc_workspace/mac_config.flag` and the same content to Linux Server-1 `/root/dev_path/linux_config.flag`. Confirm matching md5. Do not change Mac persistent config.	Do both flag files contain exactly `BENCH_CFG=enabled-2026`?
12	Restart cron on Linux Server-1/2/3 (initially stopped). There is no systemd; use `service` or the image's init tools. Report the final state per host.	Is the `cron` process running on Linux Server-1, Linux Server-2, and Linux Server-3?
13	Stop cron on Linux Server-1/2/3 using the image's service tools.	Is the `cron` process stopped on Linux Server-1, Linux Server-2, and Linux Server-3?
14	Run `/root/long_job.sh` concurrently on Linux Server-1/2/3, measure wall-clock seconds, and write three hostname-sorted lines `<host>: <seconds>` to `benchmark/pc_workspace/long_job_times.txt`.	Does `long_job_times.txt` list numeric times for Linux Server-1, Linux Server-2, and Linux Server-3?
15	Read `benchmark/pc_workspace/schedule.xlsx` (Sheet1 columns Task, Target, Command, Result Summary; Target in Linux Server-1/2/3/`all-linux`). Dispatch each Command via `docker compose exec`, not SSH (Mac DNS may resolve linux-N to fake-ip `198.18.0.x`). Write exit code and the first 120 characters of stdout/stderr back into Result Summary without adding or dropping columns.	Does every Result Summary cell record an exit/return code?
16	Add root crontab `0 * * * * /root/hello.sh` on Linux Server-1/2/3 (no systemd timers) and keep cron running. `crontab -l` must show the entry.	Does `crontab -l` mention `hello` on all three hosts, and is the cron process alive?
17	In each host's `/root/data.csv` (`date,value,duration`), filter rows whose date equals *that host's* today (YYYY-MM-DD); do not pass a literal `$(date ...)` to awk. Compute count/sum/avg of `value`. Reply JSON for Linux Server-1/2/3 plus `overall`.	Does the JSON provide a numeric `avg` for Linux Server-1, Linux Server-2, Linux Server-3, and `overall`?
18	As root, recursively find `*.sh` under `/root/` on Linux Server-1/2/3; record size and first line. Write a markdown table Host, Path, Size(bytes), FirstLine, grouped by host, to `benchmark/pc_workspace/sh_scan_summary.md`.	Does `sh_scan_summary.md` mention Linux Server-1/2/3 and `.sh` paths?

Continued on next page

ID	Task	Task-level Evaluation Checks
19	Treat column 3 of `data.csv` as duration in seconds. Average Mac `benchmark/pc_workspace/data.csv` and each Linux copy (two decimals), then an overall mean. Append one UTC ISO line `overall=... mac=... linux-1=...` to `benchmark/pc_workspace/avg_duration.txt`.	Does `avg_duration.txt` exist, and does the response state a decimal overall/mean?
20	List regular files under `/etc` modified in the last 48 h (no crossing mount points) on Linux Server-1/2/3. Write Host, Path, Mtime to an `etc_recent_48h` sheet or marked block in `benchmark/pc_workspace/log_detailed.xlsx`.	Does the workbook mention Linux Server-1/2/3 and `/etc` paths?
21	On Linux Server-1/2/3 (shared Docker socket): pull `nginx:alpine` if needed; run `bench_nginx_<hostname>` publishing port 80 with `--rm`; after about 3 s, HTTP 200 on `127.0.0.1:80` inside the container; then stop it. Report 200 and successful stop per host.	Does the response mention HTTP 200 and that the containers were stopped or removed?
22	On Linux Server-1 as root, create `/home/ops/demo` with a two-line Dockerfile (`FROM alpine:latest` and `CMD ["echo","bench-demo"]`), build `demo:test`, save `/home/ops/demo.tar`, and `chown` to ops. As ops, scp the tar to Linux Server-3 with the shared ed25519 key (no password). On Linux Server-3 as root, `docker load` and `docker run --rm demo:test`, expecting `bench-demo`. Keep the tar under `/home/ops` because `/root` is mode 700.	Does `/home/ops/demo.tar` exist on Linux Server-3, and does `docker run demo:test` print `bench-demo`?
23	The three hosts share one Docker engine, so image lists match. Instead, collect local TCP listen ports and process names, and write `benchmark/pc_workspace/listen_ports_diff.md` with per-host tables plus `## Host-unique ports` (`bench_http.py` makes 8001/8002/8003 unique).	Does `listen_ports_diff.md` name Linux Server-1/2/3 and discuss listen/TCP ports?
24	Clone `https://github.com/psf/requests.git` to `/root/requests_cmp` on Linux Server-1 and to `benchmark/pc_workspace/requests_cmp` on Mac. On Mac create `test-branch` from `origin/main` five commits back. Use `git merge-tree` to test a clean merge with `origin/main`. Write `clean` or conflicting paths to `benchmark/pc_workspace/merge_report.txt`.	Is `merge_report.txt` a single `clean` line, or does it list conflict-like paths?

Continued on next page

ID	Task	Task-level Evaluation Checks
25	Services listen on 5001/5002/5003. Measure 1-min load and available memory; deploy `service_1.py` to lowest load, `service_2.py` to median load, `service_3.py` to highest memory. Run them in the background, HTTP-probe every `<host>:<port>/health`, and write the assignment plus Src,Dst,Port,HTTPStatus matrix to `benchmark/pc_workspace/deploy_report.txt`.	Does `deploy_report.txt` name `service_1`/`service_2`/`service_3` and mention the ports or HTTP checks?
26	Make `/root/dev_path/requests` the latest `psf/requests` main on Linux Server-1/2/3 (clone or `fetch` plus hard reset). Quote matching short HEAD hashes in the response.	Does `/root/dev_path/requests/.git` exist on Linux Server-1, Linux Server-2, and Linux Server-3?
27	Clone `requests` under `/root/dev_path/` if needed. Create `.venv-linux-N` named from that host's hostname (do not pass a literal `$(hostname)`), `pip install .`, and import `requests.__version__` with that interpreter.	Are `.venv-linux-1/2/3/bin/python` present and executable?
28	Start JupyterLab on Linux Server-1 at `0.0.0.0:8080`, allow root, no browser, token `benchtoken`, cwd `/root`. Host maps 8080 to 18080. Reply with `http://127.0.0.1:18080/lab?token=benchtoken` (HTTP 200 or 302 from Mac).	Is a jupyter process matching port 8080 running on Linux Server-1, and does the response contain that URL?
29	Kill the entire JupyterLab process tree on Linux Server-1 so port 8080 is free; wait about 1 s. Killing only the parent PID is insufficient.	Is port 8080 on Linux Server-1 no longer listening?
30	Stop cron and nginx on Linux Server-1/2/3 (mark already-stopped if needed). Initially nginx runs and cron does not; sshd and `/root/init/bench_http.py` must stay up. List stopped vs. already-stopped per host.	Are both the cron and nginx processes absent on Linux Server-1, Linux Server-2, and Linux Server-3?
31	Ports 8001/8002/8003 already serve `/health=200`. From each linux host HTTP-GET all three `<host>:<port>/health` endpoints (3×3 rows) and write Src, Dst, Port, HTTPStatus to a `connectivity` sheet or block in `benchmark/pc_workspace/log_detailed.xlsx`.	Does the workbook mention Linux Server-1/2/3, ports 8001–8003, and HTTP 200?
32	Via `docker compose exec`, test whether `intranet.local` resolves on Linux Server-1 and Linux Server-2. If not, append `<linux-3-ip> intranet.local` to their `/etc/hosts`, re-ping from Mac through compose, and record the steps in `benchmark/pc_workspace/dns_check_notes.txt`.	Does `getent hosts intranet.local` succeed on Linux Server-1 and Linux Server-2?

Continued on next page

ID	Task	Task-level Evaluation Checks
33	If Linux Server-1 has no listener on 8080, start `python3 -m http.server` on `0.0.0.0:8080`. From Linux Server-2, `http://linux-1:8080/` must return 200/301/302/403/404. A Jupyter listener from task 28 also counts; do not bind two services.	Is Linux Server-1 listening on 8080, and can Linux Server-2 reach that URL?
34	Check `http://localhost:9090/health` on Linux Server-1 (host port 19090). If it fails, run a light HTTP container on 9090, verify 200 locally and from Linux Server-2, then remove the container.	Does `curl` to Linux Server-1 `localhost:9090/health` succeed?
35	As ops, scp `/home/ops/long_job.sh` (not `/root/long_job.sh`) to `/tmp/long_job.sh` for all 6 directed pairs among Linux Server-1/2/3 using the pre-shared ed25519 key. Write a 3×3 matrix to `benchmark/pc_workspace/scp_matrix.txt` (– on the diagonal, `ok` or the exit code).	Does `scp_matrix.txt` mention Linux Server-1/2/3 and `ok` with no fail/error text? Does `/tmp/long_job.sh` exist on each host?
36	On any linux host, open `https://www.python.org/downloads/` to obtain the latest stable 3.x version $X.Y.Z$; download `Python-<X.Y.Z>.tgz` to Mac `benchmark/pc_workspace/downloads/`. From Mac, scp it to `/home/ops/` on each linux host via `127.0.0.1:2201/2202/2203` with key `benchmark/.run/ops_ed25519` (do not SSH to `ops@linux-N`; Mac DNS may resolve linux-N to fake-ip `198.18.0.x`). List the tarball with `tar`; do not unpack or compile.	Does `/home/ops/Python-*.tgz` exist on Linux Server-1, Linux Server-2, and Linux Server-3?
37	Record root Used% on Linux Server-1/2/3. Write `benchmark/pc_workspace/disk_tips.txt` with section `Disk usage snapshot` (`<host>: <used%>` per line) and `Recommendations` with at least three disk-saving tips (agent knowledge allowed; no live crawl required).	Do both section titles appear in `disk_tips.txt`, with Linux Server-1/2/3 in the snapshot?
38	Run `/root/cpu_bench.py` on Linux Server-1/2/3. Write `benchmark/pc_workspace/cpu_bench_report.txt` with `Benchmark scores` (`<host>: elapsed_sec=<x>`) and 3–5 `Optimization tips` (no live crawl required).	Does the report contain three `elapsed_sec` lines and an optimization section?

Continued on next page

ID	Task	Task-level Evaluation Checks
39	Download Seattle weather CSV to `benchmark/pc_workspace/weather.csv`, scp to `/home/ops/weather.csv` via ports 2201–2203 with `ops_ed25519`, average column 3 (skip header, 4 decimals) into `/home/ops/weather_avg.txt` on each host, and confirm the three averages match.	Does the Mac `weather.csv` exist, and is `/home/ops/weather_avg.txt` non-empty on each linux host?
40	Query openssl CVEs from the CIRCL API (or three known CVE-IDs if the API fails). Record local openssl versions on Linux Server-1/2/3 in `benchmark/pc_workspace/cve_notes.txt` as CVE-ID / summary / version. Do not require a real upgrade.	Do the notes contain a CVE-ID and the string openssl?
41	Dispatch `schedule.xlsx` as in task 15 via `docker compose exec`, but do not modify the workbook. Write `benchmark/pc_workspace/schedule_summary.md` with Task, Target, exit code, truncated output, and a final `summary: K/N ok`.	Does `schedule_summary.md` include Task/Target, an exit code, and a `summary: K/N ok` line?
42	Read 1-min load on Linux Server-1/2/3, run `/root/long_job.sh` on the lowest-load host, and reply `chose linux-X because load=Y.YY` plus stdout containing `job completed`.	Does the response match `chose linux-[123]` with a numeric load?
43	Read integer root Used%. If any host is $\geq 90\%$ (free $\leq 10\%$), print `ALERT linux-N=<P>%` per such host; otherwise print only `OK`.	Is the response `OK`, or one or more `ALERT linux-N=<P>%` lines?
44	Run `/root/cpu_bench.py` on Linux Server-1/2/3 via `docker compose exec` (not SSH). Write exactly three hostname-sorted lines `<host>: <one-line output>` to `benchmark/pc_workspace/cpu_bench_all.txt`, with no connection/ERROR failure text.	Does `cpu_bench_all.txt` contain `elapsed_sec` for Linux Server-1/2/3 and no failure strings?
45	Collect kernel (`uname`) and OS/release info on Linux Server-1/2/3. Write `benchmark/pc_workspace/hosts_summary.txt` as three blocks `=== <host> ===` in hostname order.	Does `hosts_summary.txt` contain all three `=== linux-N ===` banners together with Linux/Ubuntu/version text?

Table 6: Tasks in MDCBench-GUI and their process-level evaluation rubrics.

ID	Task	Process-level Evaluation Rubric
1	Turn on the Bluetooth function on the mobile phone.	Does the agent successfully turn on Bluetooth on the mobile phone?
2	Create a new schedule titled “test” for today in the Calendar app on Android.	Does the agent successfully create a schedule titled “test” in the Calendar app?

Continued on next page

Table 6 continued

ID	Task	Process-level Evaluation Rubrics
3	Add an expense record of 10 US dollars in the Money Manager app on Android.	Does the agent successfully add a $10 expense record in the Money Manager app?
4	Search for images of "Mango Pudding" and "Matcha Pudding" separately using Chrome. Download one image from each set of search results, open the corresponding image-source pages, and then use the Files app to check the downloaded images.	Does the agent successfully search for images of "Mango Pudding"? Does the agent successfully search for images of "Matcha Pudding"? Does the agent successfully download one image from each set of search results? Does the agent successfully open the source page associated with each downloaded image? Does the agent successfully check the downloaded pictures in the Files app?
5	Look up the average human walking speed using Chrome. Then use the Calculator app to compute the total distance, in kilometers, traveled by walking for 30 minutes every day for one full year.	Does the agent successfully search for the average human walking speed? Does the agent use the Calculator app to complete the required walking-distance calculation?
6	Clear all search history in Weibo.	Does the agent successfully clear all search history in Weibo?
7	Open the Maps app, navigate from the current location to the nearest airport, and select the fastest available route.	Does the agent successfully open the Maps app? Does the agent successfully find a route from the current location to the nearest airport? Does the agent successfully select the fastest available route?
8	Clone the GitHub repository `https://github.com/psf/requests` on the PC.	Does the agent successfully clone the repository `https://github.com/psf/requests` on the PC?
9	Check the disk usage, CPU and GPU utilization, and latest system logs on the PC. Compile the collected information into a report and save it in the memo app on the mobile phone.	Does the agent successfully inspect the disk usage, CPU and GPU utilization, and latest system logs on the PC? Does the agent successfully compile the collected data into a report? Does the agent successfully save the report in the memo app on Android?
10	Search for "restaurants in New York City" using the mobile browser, and then record the content of the first search result in a TXT file on the PC.	Does the agent successfully search for "restaurants in New York City" using the mobile browser? Does the agent successfully save the content of the first search result in a TXT file on the PC?
11	Search for the latest technology news using the browser, summarize the first news article, and save the summary as `news.txt` on the PC desktop.	Does the agent successfully search for the latest technology news using the browser? Does the agent successfully summarize the first news article and save the summary as `news.txt` on the PC desktop?

Continued on next page

Table 6 continued

ID	Task	Process-level Evaluation Rubrics
12	Open the top news article in the Inoreader app installed on `phone`, refine and condense its content, and save the resulting text as `~/news2.txt` on `desktop`.	Does the agent successfully locate and open the top news article in the Inoreader app on `phone`? Does the agent successfully refine the article and save the condensed text as `~/news2.txt` on `desktop`?
13	Search for "Hotels" in the Android Maps app and record the first retrieved hotel. Create a Calendar event for a stay at this hotel tomorrow. Then open Chrome on the desktop and search for guest reviews of the selected hotel.	Does the agent successfully search for "Hotels" in the Android Maps app and record the first retrieved hotel? Does the agent successfully create a Calendar schedule for staying at the selected hotel tomorrow? Does the agent successfully search for guest reviews of the selected hotel using Chrome on the desktop?
14	Search for "travel plans to Qinghai China" using Chrome on `desktop`, and send the first search result to `phone`. Based on that result, create a feasible travel itinerary in the Keep Notes app on `phone`.	Does the agent successfully search for "travel plans to Qinghai China" on `desktop` and send the first search result to `phone`? Does the agent successfully create a feasible travel itinerary in the Keep Notes app on `phone` based on the search result?
15	Launch YouTube on Android and search for the latest movie trailers. Compile all titles shown in the search results and save the list as `title.txt` on the PC desktop.	Does the agent successfully search for the latest movie trailers using YouTube on Android? Does the agent successfully compile all titles from the search results and save them as `title.txt` on the PC desktop?
16	Search for a high-quality scenic landscape image on the PC, and then set the selected image as the user's Weibo profile avatar.	Does the agent successfully search for a high-quality scenic landscape image on the PC? Does the agent successfully set the selected image as the Weibo profile avatar?
17	Search for common sports on the PC and select the top three sports from the results. Send the three selected sports to the mobile phone. Use the Boostcamp app on the phone to find workout plans for each sport, and select the plan with the shortest duration. Finally, return to the PC and search for a complete training schedule for the sport associated with the selected plan.	Does the agent successfully search for common sports on the PC and select the top three sports from the results? Does the agent successfully use the Boostcamp app to find workout plans for each of the three selected sports? Does the agent successfully identify the workout plan with the shortest duration among the retrieved options? Does the agent successfully search on the PC for a complete training schedule corresponding to the selected sport?

Continued on next page

Table 6 continued

ID	Task	Process-level Evaluation Rubrics
18	Search for "events held in Beijing this month" using the PC browser. Select two or three events and record their venues and times. Send this information to the mobile phone and use the Maps app to check the driving time from Tiananmen Square to each venue. Select the event with the shortest driving time, compose a final event plan, and email it to `example@example.com` from the mobile phone. Finally, return to the PC, summarize the complete decision-making process, and save it as `decision.txt`.	Does the agent successfully search for events held in Beijing this month using the PC browser? Does the agent successfully select two or three events and record their venues and scheduled times? Does the agent successfully use the Maps app to check the driving time from Tiananmen Square to each event venue and select the event with the shortest driving time? Does the agent successfully compose the final event plan and send it to `example@example.com` from the mobile phone? Does the agent successfully summarize the complete decision-making process and save it as `decision.txt` on the PC?
19	Search for restaurants in New York on the PC and send the names of the top three restaurants to the mobile phone. Use Maps on the phone to retrieve their customer ratings and price ranges, and send the collected information back to the PC. Run `cost_perform.py` on the PC to calculate their cost-performance ranking and send the result to the phone. Finally, open the page of the top-ranked restaurant on the phone and bookmark it.	Does the agent successfully search for restaurants in New York on the PC and send the names of the top three restaurants to the mobile phone? Does the agent successfully retrieve the customer ratings and price ranges of the three restaurants using Maps on the phone and send the collected information back to the PC? Does the agent successfully write and run `cost_perform.py` on the PC to calculate the cost-performance ranking and send the result to the mobile phone? Does the agent successfully open the page of the top-ranked restaurant on the phone and bookmark it?
20	Find one paper in each of the following research areas: memory in multi-agent systems, reinforcement learning in multi-agent systems, and information asymmetry in multi-agent systems. Save the three paper titles in `~/Desktop/papers.txt` on `desktop`. Then open Bing in Chrome on `desktop`, search for each paper, send the first search result to `phone`, and save it in Keep Notes.	Does the agent successfully find one paper in each of the three specified multi-agent-system research subfields? Does the agent successfully save the titles of the three selected papers in `~/Desktop/papers.txt` on `desktop`? Does the agent successfully search for each selected paper using Bing in Chrome on `desktop`? Does the agent successfully send the first search result for each paper to `phone` and save it in Keep Notes?

B.2 Evaluation Details

B.2.1 Evaluation Details on TeamBench

Pass Rate Metrics and Sign-off Mechanism. We report two complementary pass rates on the same 88-task subset under a frozen, deterministic grader:

- **Solution Pass Rate:** Verifies functional and artifact correctness strictly through objective checks, without requiring a formal validation signature.
- **Verification Pass Rate:** Reproduces the policy enforced by TeamBench's native upstream grader. On the 36-task subset where TeamBench explicitly checks for a validation record, the verifier agent must evaluate its own output and write a formal *sign-off* to the designated verification file. On the remaining 52 tasks, it relaxes to the standard artifact check.

The two metrics distinguish artifact correctness from satisfaction of the additional sign-off requirement on the 36 applicable tasks. A correct artifact can fail Verification Pass Rate if the required sign-off is absent or invalid.

Termination and Timeout Controls. To prevent indefinite execution from infinite loops or unrecoverable error states, both systems enforce the following upper bounds:

- **Ours:** Enforces a strict hard ceiling of 1,200 seconds in wall-clock execution time.
- **TeamBench:** Caps the maximum allowed interaction loops between the Executor and the Verifier at two rounds.

These bounds limit execution under the respective configurations. Both limits are sufficiently generous and do not constitute effective resource constraints; their different forms simply reflect differences in the underlying implementation.

B.2.2 Evaluation Details on MDCBench-CLI

Pass@1 and Pass@3 Metrics. We evaluate system performance across 45 tasks stratified into three execution-domain groups ($\leq$ 2, 3, and 4 required devices). Each task is executed across 3 independent attempts over 4 model backbones.

- **Pass@1:** Measures the average success rate of a single trajectory attempt across all tasks within a domain group.
- **Pass@3:** Measures the proportion of tasks where at least one of the 3 independent attempts satisfies the grader.

Cross-backbone performance is reported as an unweighted macro-average across all 4 model backbones to ensure equal representation regardless of individual backbone capability.

Execution Time and Active Occupancy. For trajectory r of task t, end-to-end execution time T_r^{E2E} spans task initiation to completion or timeout. For our system, we use the logged task-start timestamp and the elapsed time reported by the task-completion event. Device activity is reconstructed by pairing execution-start and terminal events using their span identifiers. Intervals are clipped to the end-to-end window and unioned within each device, so overlapping episodes on the same device contribute only one active-device indicator. For occupancy calculations, an interval without a terminal event is closed at the end-to-end boundary.

Let $I_{r,d}(\tau)$ indicate whether device d is executing at time τ. The device and agent counts are

$$K_r^{\mathrm{dev}}(\tau) = \sum_d I_{r,d}(\tau), \qquad K_r^{\mathrm{agent}}(\tau) = K_r^{\mathrm{dev}}(\tau) + c_r(\tau), \tag{4}$$

where $c_r(\tau) \in \{0, 1\}$ indicates coordinator activity. In our system, coordinator activity is the union of logged user agent LLM-call intervals; in UFO^3, it is the union of native constellation-processing intervals. Device counts refer to execution domains, including containerized domains, and exclude the coordinator.

Logical-slice concurrency. The distinct start and end times of the merged device intervals partition the window into positive-duration slices. Let $\mathcal{S}_r^+$ contain only slices with at least one active device, and let $k_{r,s}$ be the device count in slice s. We compute

$$A_r^{\mathrm{obs}} = \frac{1}{|\mathcal{S}_r^+|} \sum_{s \in \mathcal{S}_r^+} k_{r,s}. \tag{5}$$

Each active slice has equal weight, irrespective of its duration; idle slices are excluded. The implementation returns zero if there are no active slices. This measure depends on the observed event boundaries and is distinct from both a time average and a graph-derived parallelism ratio.

Time-weighted occupancy. For $x \in \{\mathrm{dev}, \mathrm{agent}\}$ and concurrency level $k \geq 0$,

$$T^x_{r,k} = \int_0^{T_r^{\mathrm{E2E}}} \mathbf{1}\{K^x_r(\tau) = k\}\, d\tau, \qquad p^x_{r,k} = 100 \frac{T^x_{r,k}}{T_r^{\mathrm{E2E}}}. \tag{6}$$

Unlike the logical-slice mean, these percentages retain idle time in the denominator. Thus $\sum_{k\geq 0} p^x_{r,k} = 100$, and the time share with multiple active entities is $\sum_{k\geq 2} p^x_{r,k}$. Percentages are computed per trajectory and then averaged within each reported group.

Task Reference DAG Construction. We distinguish the task reference DAG $G_t = (V_t, \mathcal{E}_t)$ from trajectory-specific execution DAGs Zhang et al. (2026a). One reference DAG is constructed per task from the canonical task intent and executed task descriptions, with executed descriptions taking precedence for concrete operations, artifacts, and explicit exclusions; runtime ordering is not used to infer dependencies.

Each reference node denotes a maximal single-device episode of required work, merging local inspection, action, and confirmation unless continuation depends on a result from another device. Edges encode data or completion dependencies required for correctness; textual order alone does not create an edge. Retries, support work, benchmark-only verification, transport, waiting, and coordinator-only reasoning are excluded, with coordination represented through dependencies between device actions.

Using `gpt-5.6-luna`, we generate an initial annotation and conduct a separate review pass, followed by deterministic checks for node identity, device assignment, acyclicity, episode normalization, and transitive-edge removal. Human auditing covers all 45 tasks, with 22 receiving deeper review and 14 corrected. The resulting graphs are fixed evaluation annotations and are never provided to the evaluated system.

Reconstructing the Observed Execution DAG. For each trajectory, we extract device execution episodes from runtime logs and match them to the annotated reference DAG using their requests, results, tool traces, and textual evidence. Each matched episode retains its observed start time, end time, device, and duration, allowing us to measure the actual execution time associated with each reference node. Multiple episodes may correspond to the same reference node due to retries or support actions, and one episode may cover multiple nodes. Logical dependencies between matched episodes are inherited from their required result or completion dependencies, yielding the observed execution DAG $G_r = (\mathcal{V}_r, \mathcal{E}_r)$.

Work, Critical Path and Parallel Efficiency. Following UFO^3 Zhang et al. (2026a), we compute

$$W_r = \sum_{e \in \mathcal{V}_r} d_{r,e},\ \ell_r(e) \quad = d_{r,e} + \max_{u \in \mathrm{pred} * r(e)} \ell_r(u),\ L_r = \max * e \in \mathcal{V}_r \ell_r(e), \qquad P_r = \frac{W_r}{L_r}, \tag{7}$$

where an empty predecessor set contributes zero. Here, W_r is the total executed work, L_r is the critical-path duration of the reconstructed execution DAG, and P_r measures its available parallelism.

We report critical-path efficiency as

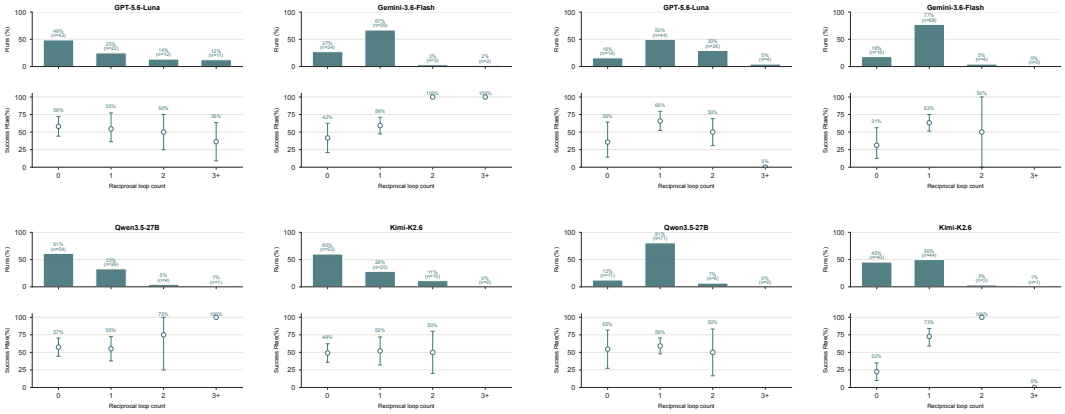
$$E_r = 100 \frac{L_r}{T_r^{\mathrm{E2E}}}, \tag{8}$$

and average E_r across trajectories. These metrics are defined over the reconstructed execution graph and therefore depend on its episode and dependency representation.

B.2.3 Evaluation Details on MDCBench-GUI

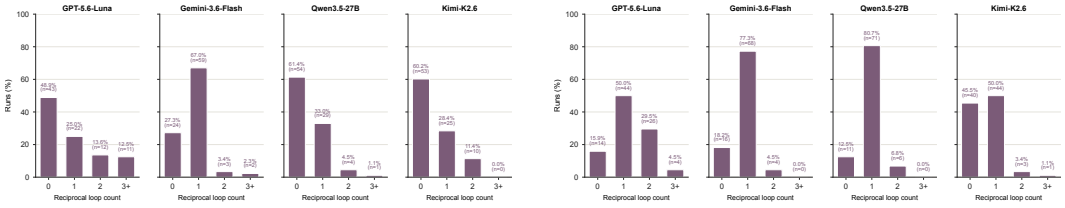
MDCBench-GUI was evaluated through human inspection of final environment states and execution traces across live services and graphical interfaces. The evaluation team comprised four computer science students who also annotated the tasks. Evaluators used the task-specific rubrics in Table 6 to assess overall completion and intermediate progress. These outcomes were reported as Pass@1 and Process Score, respectively.

C EVALUATION RESULTS



(a) Verification pass rate across $A - -B$ loop depths

(b) Verification pass rate across $B - -C$ loop depths



(c) Trajectory distribution over $A - -B$ loops

(d) Trajectory distribution over $B - -C$ loops

Figure 5: **Model-specific interaction loop depths, success rates, and trajectory distributions on TeamBench.** The top row shows CHATDEV 3.0's verification pass rate across varying loop depths for (a) Planner–Executor ($A - -B$) and (b) Executor–Verifier ($B - -C$) interaction pairs. The bottom row presents the corresponding trajectory frequency distributions for CHATDEV 3.0 across different model backbones for (c) $A - -B$ and (d) $B - -C$ loops.

Model-Specific Communication Topologies on TeamBench. To analyze how interaction structure influences performance, as in Figure 5, we measure the model-specific distributions of reciprocal loops in TeamBench, specifically focusing on the Planner–Executor (denoted as $A - -B$) and Executor–Verifier (denoted as $B - -C$) iteration loops, alongside their corresponding verification pass rates. Across different backbone models, the frequency distributions of these communication loops exhibit a remarkably consistent overall pattern. However, the exact correlation between loop counts and execution success rate remains strongly model-specific. This variation in success trajectories across models is partly attributable to variance stemming from smaller sample sizes in high-iteration buckets.

Model-Specific Execution-Domain Scaling and Task Difficulty Analysis on MDCBench-CLI. To evaluate system robustness and scalability across diverse foundation models, we conduct a fine-grained, model-specific breakdown of Pass@1, Pass@3, and execution latency on MDCBench-CLI across two distinct dimensions: task difficulty strata (Table 7) and execution-domain scale, measured by the number of target devices (Table 8).

As detailed in Table 7 and Table 8, as well as illustrated in Figure 6, the aggregate trends have model-specific exceptions:

- **Device-Count Stratification:** Macro-average Pass@1 is lower in the four-device group than in the $\leq$ 2-device group for all three methods. CHATDEV 3.0 has the highest macro-average Pass@1 in each group, but does not strictly outperform every baseline in every model–group combination: several results are tied, and TopoClaw exceeds CHATDEV 3.0 on Kimi-K2.5 in the three-device group (98.04% vs. 96.08%).
- **Task Difficulty Stratification:** CHATDEV 3.0 has the highest macro-average Pass@1 in each difficulty stratum, but the trend is not monotonically decreasing: its macro-average

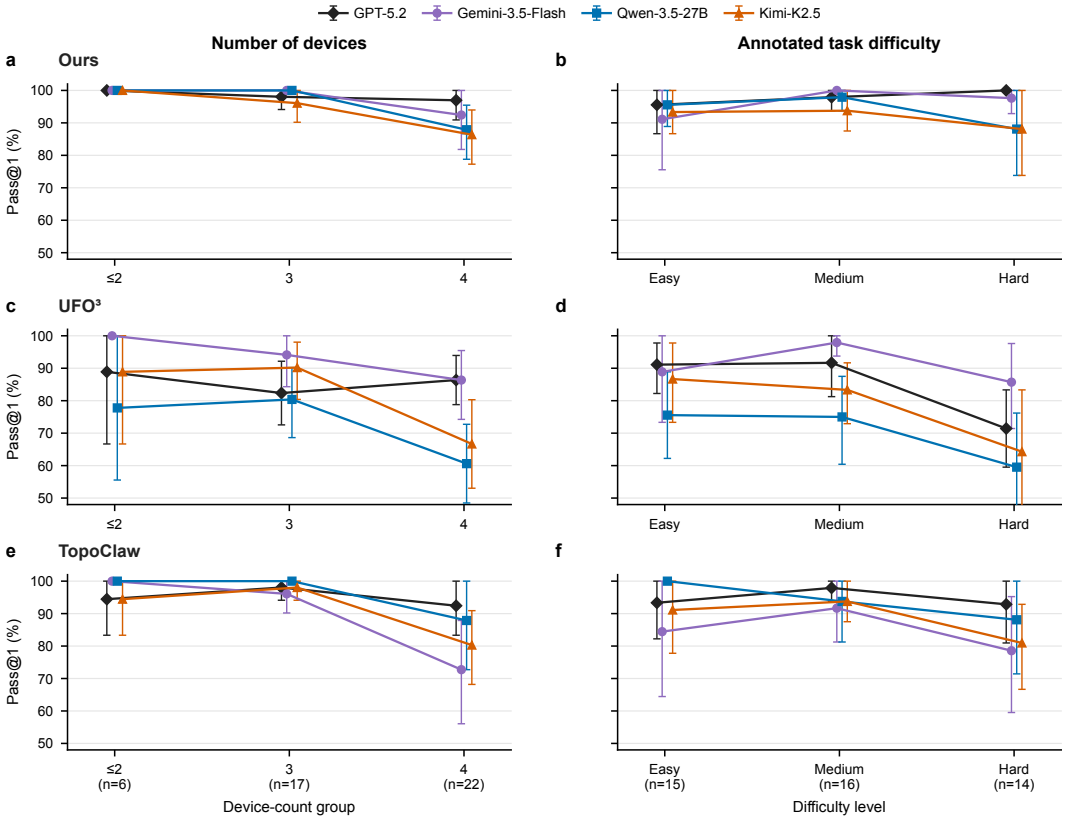


Figure 6: **Model-specific performance scaling and difficulty analysis on MDCBench-CLI.** Cross-model Pass@1 results stratified by execution-domain count (required device counts: $\leq$ 2, 3, and 4) and task difficulty tiers.

Pass@1 is 93.89%, 97.40%, and 93.45% on easy, medium, and hard tasks, respectively. There are also model-specific ties and exceptions; for example, TopoClaw exceeds ChatDev 3.0 on easy tasks with Qwen3.5-27B (100.00% vs. 95.56%).

Model-Specific Execution Concurrency and Active Agent Definitions. To provide further empirical evidence, we provide model-specific time-composition breakdowns across execution domains in Figure 7. Across all domain-count groups and all four foundation model backbones, ChatDev 3.0 consistently spends a larger fraction of end-to-end execution time with multiple active agents and active device agents compared to UFO^3. This demonstrates that our framework's high concurrency is structurally sustained throughout task execution rather than being purely instantaneous.

To ensure a fair and precise comparison of runtime parallelism across different architectures, we strictly define the counting scope for active entities as follows:

- **Active Device Agents (K_i^{dev}):** Counts distinct execution domains with an active device agent, including virtual and containerized environments. This metric strictly *excludes* high-level coordinating or routing entities—specifically, the *User Agent* in ChatDev 3.0 and the *Constellation Agent* in UFO^3.
- **Active Agents (K_i^{agent}):** Reflects total system-wide operational concurrency. This count *includes* both active device agents and the coordinating entity (the User Agent in ChatDev 3.0 or the Constellation Agent in UFO^3) whenever they are actively processing LLM reasoning or routing logic:

$$K_i^{\text{agent}}(t) = K_i^{\text{dev}}(t) + c_i(t), \tag{9}$$

where $c_i(t) \in \{0, 1\}$ indicates the active status of the central coordinator at time step t.

Table 7: Performance on MDCBench-CLI stratified by annotated task difficulty. Pass@1 is the mean success rate over three trajectories, while Pass@3 is the percentage of tasks completed in at least one trajectory. Time reports mean end-to-end execution time in seconds within each difficulty stratum. The dagger (†) marks the single-agent method. The final block reports macro averages across backbones. Best results are highlighted in blue and bold, while second-best distinct results are underlined.

Backbone	Method	Easy ($n = 15$)			Medium ($n = 16$)			Hard ($n = 14$)		
		Pass@1 (%) ↑	Pass@3 (%) ↑	Time (s) ↓	Pass@1 (%) ↑	Pass@3 (%) ↑	Time (s) ↓	Pass@1 (%) ↑	Pass@3 (%) ↑	Time (s) ↓
Closed-source models										
GPT-5.2	TopoClaw	93.33	**100.00**	**23.17**	**97.92**	**100.00**	**28.56**	92.86	**100.00**	**38.37**
	UFO^3	91.11	**100.00**	174.38	91.67	**100.00**	209.62	71.43	**100.00**	340.20
	Ours	**95.56**	**100.00**	49.81	**97.92**	**100.00**	54.61	**100.00**	**100.00**	73.94
Gemini-3.5-Flash	TopoClaw	84.44	86.67	44.88	91.67	**100.00**	**52.77**	78.57	85.71	**76.63**
	UFO^3	88.89	**93.33**	92.83	97.92	**100.00**	194.75	85.71	**100.00**	178.83
	Ours	**91.11**	**93.33**	**38.75**	**100.00**	**100.00**	81.47	**97.62**	**100.00**	145.88
Open-source models										
Qwen3.5-27B	TopoClaw	**100.00**	**100.00**	**60.89**	93.75	93.75	155.11	**88.10**	92.86	**112.27**
	UFO^3	75.56	**100.00**	273.40	75.00	**100.00**	356.03	59.52	92.86	499.27
	Ours	95.56	**100.00**	85.12	**97.92**	**100.00**	**97.91**	**88.10**	**100.00**	176.74
Kimi-K2.5	TopoClaw	91.11	**100.00**	**68.96**	**93.75**	**100.00**	**80.07**	80.95	**100.00**	**144.66**
	UFO^3	86.67	**100.00**	381.55	83.33	**100.00**	506.38	64.29	85.71	684.05
	Ours	**93.33**	**100.00**	259.13	**93.75**	**100.00**	282.03	**88.10**	**100.00**	527.83
Average	TopoClaw	92.22	96.67	**49.47**	94.27	98.44	**79.13**	85.12	94.64	**92.98**
	UFO^3	85.56	**98.33**	230.54	86.98	**100.00**	316.70	70.24	94.64	425.59
	Ours	**93.89**	**98.33**	108.20	**97.40**	**100.00**	129.01	**93.45**	**100.00**	231.10

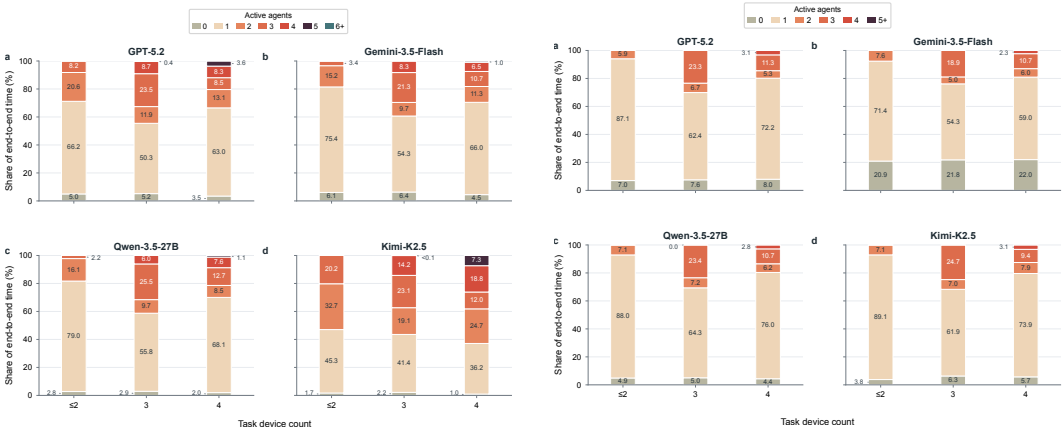
Note: Difficulty labels are inherited from the benchmark task annotations. Pass@1 averages the three binary trajectory outcomes for each task, whereas Pass@3 indicates whether at least one of the three trajectories succeeds. Rankings are computed separately within each backbone, difficulty stratum, and metric. Tied best and second-best distinct results are highlighted equally.

Table 8: Performance on MDCBench-CLI stratified by the number of devices involved in each task. Pass@1 is the mean success rate over three trajectories, while Pass@3 is the percentage of tasks completed in at least one trajectory. The Time column reports mean end-to-end execution time in seconds within each device-count stratum. The dagger (†) marks the single-agent method. The final block reports macro averages across backbones. Best results are highlighted in blue and bold, while second-best distinct results are underlined.

Backbone	Method	≤ 2 devices ($n = 6$)			3 devices ($n = 17$)			4 devices ($n = 22$)		
		Pass@1 (%) ↑	Pass@3 (%) ↑	Time (s) ↓	Pass@1 (%) ↑	Pass@3 (%) ↑	Time (s) ↓	Pass@1 (%) ↑	Pass@3 (%) ↑	Time (s) ↓
Closed-source models										
GPT-5.2	TopoClaw	94.44	**100.00**	**21.74**	**98.04**	**100.00**	**22.83**	92.42	**100.00**	**37.42**
	UFO^3	88.89	**100.00**	275.93	82.35	**100.00**	180.14	86.36	**100.00**	273.39
	Ours	**100.00**	**100.00**	52.37	**98.04**	**100.00**	48.03	**96.97**	**100.00**	69.34
Gemini-3.5-Flash	TopoClaw	**100.00**	**100.00**	**52.70**	96.08	**100.00**	**57.22**	72.73	81.82	**59.15**
	UFO^3	**100.00**	**100.00**	165.57	94.12	**100.00**	148.44	86.36	**95.45**	158.87
	Ours	**100.00**	**100.00**	64.78	**100.00**	**100.00**	98.65	**92.42**	**95.45**	84.61
Open-source models										
Qwen3.5-27B	TopoClaw	**100.00**	**100.00**	**75.80**	**100.00**	**100.00**	**75.37**	**87.88**	90.91	146.86
	UFO^3	77.78	**100.00**	345.43	80.39	**100.00**	337.81	60.61	95.45	407.82
	Ours	**100.00**	**100.00**	84.53	**100.00**	**100.00**	92.99	**87.88**	**100.00**	**146.81**
Kimi-K2.5	TopoClaw	94.44	**100.00**	**62.78**	**98.04**	**100.00**	**69.57**	80.30	**100.00**	**126.43**
	UFO^3	88.89	**100.00**	518.37	90.20	**100.00**	436.42	66.67	90.91	585.12
	Ours	**100.00**	**100.00**	304.26	96.08	**100.00**	203.43	**86.36**	**100.00**	477.51
Average	TopoClaw	97.22	**100.00**	**53.25**	98.04	**100.00**	**56.25**	83.33	93.18	**92.46**
	UFO^3	88.89	**100.00**	326.32	86.76	**100.00**	275.70	75.00	95.45	356.30
	Ours	**100.00**	**100.00**	126.48	**98.53**	**100.00**	110.77	**90.91**	**98.86**	194.57

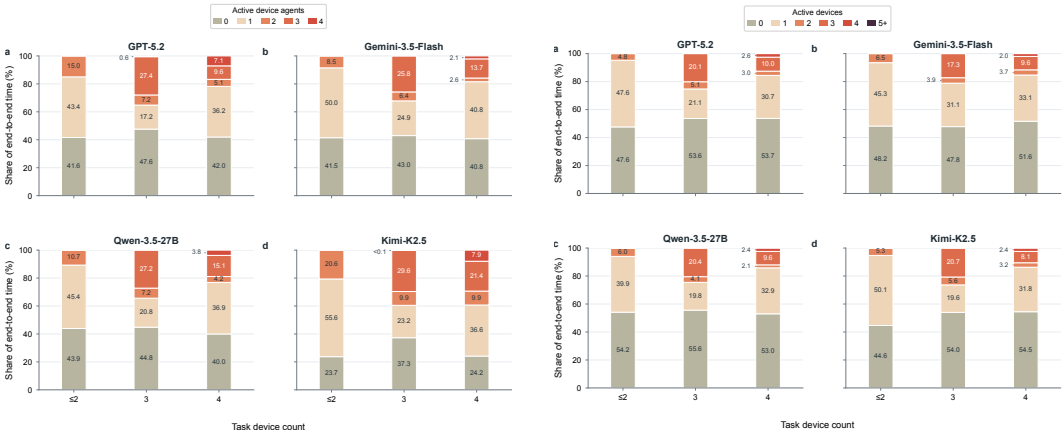
Note: The $\leq$ 2-device stratum combines one single-device task with five two-device tasks. Pass@1 averages the three binary trajectory outcomes for each task, whereas Pass@3 indicates whether at least one of the three trajectories succeeds. Rankings are computed separately within each backbone, device-count stratum, and metric. Tied best and second-best distinct results are highlighted equally.

By separating device-level execution from coordinator-level overhead, these results describe sustained overlap of logged execution activity across the evaluated model backbones. They do not measure physical hardware utilization.



(a) CHATDEV 3.0: Active-agent time composition

(b) UFO^3: Active-agent time composition



(c) CHATDEV 3.0: Device-parallel time composition

(d) UFO^3: Device-parallel time composition

Figure 7: **Model-specific execution time composition comparison between CHATDEV 3.0 and UFO^3.** Top row (a, b) compares overall system-level active agent concurrency (K^{agent}, including central coordinators) between CHATDEV 3.0 and UFO^3 across four model backbones. Bottom row (c, d) compares execution-domain parallelism (K^{dev}, strictly excluding central coordinators). Across all backbone models, CHATDEV 3.0 spends a substantially larger fraction of end-to-end time maintaining multi-agent execution.

D IMPLEMENTATION ARCHITECTURE

D.1 MAIN ARCHITECTURE

This section describes the concrete realization of the boundary-aligned architecture and asynchronous runtime introduced in Section 2. We focus on the network topology, orchestrator scheduling, and device-routing mechanisms. The execution backends used in the experiments are described separately below.

D.1.1 NETWORKED SERVICE TOPOLOGY.

CHATDEV 3.0 is implemented as a three-tier network deployment comprising a shared Hub Server, one User Server for each active user, and independently deployed Device Agents. The Hub Server implements the shared control plane. It maintains user identities, authorization relationships, device ownership, online User Server instances, and cross-user event routes. It does not execute User Agent policies or directly operate device environments. Each User Server is bound to the credentials of one user and embeds the corresponding stateful User Agent orchestrator. It additionally hosts the device-connection registry, device-dispatch service, orchestrator dispatcher, task-state store, and bridges for device results and cross-user messages. Device Agents run as separate processes within, or adjacent to, the environments that they control.

Two layers of persistent WebSocket connections link these services. First, each User Server establishes an authenticated connection to the Hub Server. Through this connection, the Hub Server pushes owner requests, cross-user messages, remote device results, and control-plane updates to the appropriate User Server. The receiver acknowledges processed records and temporarily caches their identifiers to suppress duplicate processing after reconnection or repeated delivery.

Second, each Device Agent establishes a persistent WebSocket connection to its governing User Server. During connection setup, it submits a registration record containing its `user_id`, `device_id`, `client_id`, platform, and execution-capability profile. After registration, task requests, progress updates, cancellation signals, and terminal results are transmitted over the same connection. Periodic heartbeat events allow the User Server to maintain device reachability and last-activity information.

HTTP endpoints are used for authentication, initial user and device registration, health checks, administrative operations, and optional execution-backend registration. WebSocket connections carry the continuous task and event traffic. The resulting cross-user communication path is

$$\text{User Server}_u \longleftrightarrow \text{Hub Server} \longleftrightarrow \text{User Server}_v, \tag{10}$$

whereas device delegation follows

$$\text{User Server}_u \longleftrightarrow \text{Device Agent}_{u,d} \tag{11}$$

A Device Agent does not establish a direct connection to another user's User Server. Information that must cross a user boundary is first returned to the governing User Server and subsequently routed through the Hub Server under the corresponding authorization policy.

D.1.2 ORCHESTRATOR SCHEDULING.

All inputs that require a User Agent decision are submitted to **a unified orchestrator dispatcher** within the corresponding User Server. The dispatcher is implemented using an `asyncio` priority queue and a configurable worker pool. Scheduling priorities distinguish among direct owner requests, execution results returned by Device Agents, messages from peer User Agents, and triggers generated by the proactive runtime.

The dispatcher maintains an asynchronous lock for each `session_id`. Events belonging to the same session enter orchestrator decision turns sequentially, preventing concurrent modification of the same interaction history, task state, and collaboration context. Events associated with different

sessions may be processed by different workers. This design preserves state consistency within a session while allowing independent collaboration processes to progress concurrently.

Device-dispatch and cross-user communication actions typically initiate downstream operations with substantial or unpredictable execution latency. When a decision turn produces such an action, the corresponding service validates the target identity, authorization, and current reachability before constructing and transmitting the outgoing request. Once transmission succeeds, the orchestrator receives an `accepted` or `dispatched` result. This result confirms only that the remote action has been submitted, rather than that the downstream task has completed. The current decision turn then terminates, releasing the worker to process other queued events while the downstream operation proceeds independently.

Progress and terminal results returned later by a Device Agent, peer User Agent, or messages from human, are submitted to the dispatcher as new events and initiate subsequent decision turns. Thus, the asynchronous execution described in Section 2.3 is realized through short-lived queue jobs, concurrent workers, per-session locks, and event-triggered continuations, rather than through a globally blocking call that waits for all remote operations to finish.

D.1.3 Experimental Execution Backends.

The experiments used Linux CLI agents, Linux desktop GUI agents, and Android GUI agents. Table 9 separates the benchmark configurations from the recorded case studies. The desktop GUI backend is also illustrated in Appendix E.2. Although their tool interfaces are modality-specific, these backends provide environment-observation and environment-manipulation functions that are broadly equivalent to those exposed by the corresponding baselines. For the TeamBench comparison below, we counted environment-related function calls (FCs) from the recorded trajectories. These counts describe interaction volume rather than elapsed-time overhead.

Table 9: Reported execution backends and evidence scope. Endpoint roles do not establish a particular physical host layout or operating system.

Evaluation	Execution backends	Evidence scope
TeamBench	Linux CLI agents with isolated role permissions	88 tasks with Planner, Executor and Verifier roles.
MDCBench-CLI	Linux CLI endpoints and a PC reporting workspace	45 adapted tasks; three attempts per task and backbone.
MDCBench-GUI	Linux desktop GUI and Android GUI agents	20 tasks; one attempt per task with Kimi-K2.5.

The case descriptions document individual archived executions. Their narrative summaries do not establish which attempts contributed to the aggregate benchmark results.

Linux Agent For TeamBench, the number of environment-related operations was defined as

$$FC_{\text{env}}^{\text{TB}} = N(\texttt{run}) + N(\texttt{read}) + N(\texttt{write}), \tag{12}$$

where `run` executes commands in the task workspace, `read` retrieves file contents, and `write` creates or overwrites files. Communication calls, such as `send_message`, were included in the overall FC count but excluded from $FC_{\text{env}}^{\text{TB}}$.

For ChatDev 3.0, the Linux device agent exposes five environment-facing tools: `shell`, `read`, `search`, `patch`, and `write`. We excluded orchestrator-level calls for task dispatch, progress monitoring, device and directory discovery, and inter-agent communication. The environment-operation count for the Linux device agent was therefore defined as

$$\begin{aligned} FC_{\text{env,linux}}^{\text{ChatDev3.0}} = {} & N_{\text{device}}(\texttt{shell}) + N_{\text{device}}(\texttt{read}) + N_{\text{device}}(\texttt{search}) \\ & + N_{\text{device}}(\texttt{patch}) + N_{\text{device}}(\texttt{write}). \end{aligned} \tag{13}$$

The resulting counts across 352 task–model runs per method (88 tasks $\times$ four backbones) in the TeamBench comparison are summarized in Table 10.

Table 10: Breakdown of environment-related function calls in the TeamBench comparison.

Harness	Tool	Count
TeamBench Solo	`run`	3,148
	`read`	1,926
	`write`	845
	Total	**5,919**
TeamBench Full	`run`	22,965
	`read`	13,615
	`write`	1,164
	Total	**37,744**
CHATDEV 3.0 Linux	`shell`	7,760
	`read`	5,747
	`search`	1,136
	`patch`	859
	`write`	663
	Total	**16,165**

At the functional level, the Linux interfaces are closely aligned. TeamBench's `run` tool broadly corresponds to CHATDEV 3.0's `shell` tool, while `read` and `write` have direct counterparts in both systems. Nevertheless, the interfaces are not identical. CHATDEV 3.0 exposes `search` and `patch` as separate tools, whereas TeamBench agents can perform equivalent operations within a `run` call, for example, by invoking `grep`, `find`, or other shell utilities. Patch-like operations may also be implemented by rewriting a file through `write`.

Android Agent The Android device agent uses a corresponding set of observation and action FCs for screen-based interaction. Its environment-facing calls include `android_screenshot`, `android_tap`, `android_long_tap`, `android_swipe`, `android_write_text`, `android_key_press`, `android_exec_adb`, and `android_open_app_drawer`. The `android_screenshot` call captures the current device state, while the remaining calls manipulate the interface through ADB. Model-level control outputs, including `wait`, `interact`, `answer`, and `terminate`, were excluded because they do not directly invoke the device environment.

These Android FCs are functionally aligned with the interaction primitives used by screen-based baselines. Screenshot acquisition corresponds to visual observation, while tap, long-tap, swipe, text-entry, and key-press calls correspond to pointing, scrolling, keyboard input, and system-button operations. Although the interfaces differ in their names and underlying implementations, they provide comparable environment-level capabilities.

D.2 ADDITIONAL IMPLEMENTED COMPONENTS

Beyond the core collaboration path and the execution backends used in the experiments, the complete CHATDEV 3.0 implementation includes a structured long-term memory system, additional Device Agent runtimes, a provider-based extension interface, and support for independently deployed execution backends. **These components are implemented in the complete system but were excluded from all reported benchmark configurations. Specifically, optional mechanisms such as long-term memory were disabled, and additional Device Agent backends were not instantiated.**

Structured Long-Term Memory. The User Agent orchestrator stores long-term memory in a structured `MEMORY.md` file. Memory entries are divided into six types: `user` entries record dynamic user preferences and working patterns; `feedback` entries record corrections and confirmations of Agent behavior; `entity` entries describe observations about other users and devices; `project` entries track ongoing objectives; `reference` entries point to external resources; and `playbook` entries store reusable task experience and operating procedures.

Memory modification follows a four-operation protocol: `ADD`, `UPDATE`, `DELETE`, and `NOOP`. `ADD` creates a new entry, `UPDATE` corrects or supplements an existing entry, `DELETE` removes information that is outdated or has been invalidated, and `NOOP` explicitly indicates that an interaction contains no information worth persisting. Stable identity fields are stored in the file frontmatter, whereas the body contains only information expected to evolve through interaction.

The memory system excludes state that can be queried directly at execution time, including current device availability, live file contents, raw tool outputs, and intermediate results from the active session. Such records would quickly become stale and are therefore unsuitable as long-term decision inputs. The implementation treats stale records as memory drift: memory may provide preferences, recurring patterns, and prior experience, but the orchestrator must re-query the relevant tools before taking an action that depends on the current world state.

To limit prompt size, the orchestrator does not inject the complete memory store into every decision turn. Instead, the system prompt contains a compact memory index and short summaries of the available entries. Detailed entity records, individual reflections, and full playbooks are loaded on demand through memory-specific tools.

Asynchronous Memory Maintenance. Memory maintenance is divided between the main orchestrator and a background memory sub-agent. During ordinary decision turns, the main orchestrator retrieves memory and may update entries concerning the current user, behavioral feedback, and relevant entities. Reflection over completed tasks, consolidation of recurring experience, and periodic memory maintenance are delegated to the background sub-agent, preventing these operations from occupying the main task-processing context.

The memory sub-agent runs asynchronously in a separate context window. It may be triggered when a task completes, when execution switches to another session, after a configurable number of turns, or when the main context approaches its capacity limit. The sub-agent examines recent interaction history, device-dispatch traces, and cross-user communication records to identify persistent preferences, corrections, entity observations, and reusable task experience. If no information should be retained, it returns `NOOP`.

The sub-agent is restricted to memory-specific operations. It cannot dispatch tasks to Device Agents, send messages to other users, or invoke environment tools. It may therefore summarize and reorganize completed interaction traces but cannot initiate new external actions while running in the background.

Reusable task experience is first recorded as a category-specific reflection. After a sufficient number of reflections have accumulated for the same category, the memory system synthesizes them into a structured playbook. Each playbook contains a *Trigger*, *Steps*, *Expected Result*, and *Pitfalls*. When a later request matches a playbook summary, the orchestrator can retrieve the complete procedure and use it as an execution reference.

Extensible Device-Agent Backends. CHATDEV 3.0 separates the generic Device Agent lifecycle from environment-specific execution logic. The generic Device Agent handles authentication, WebSocket connections, registration, heartbeat transmission, task reception, progress reporting, cancellation, state tracking, and result delivery. Each environment-specific runtime implements a common execution interface and returns a normalized `DeviceAgentRuntimeResult` containing its status, execution mode, textual outcome, updated session context, and diagnostic metadata.

Runtimes are loaded through a provider registry. Each provider defines a `RuntimeSpec` containing its runtime kind, platform, configuration mappings, and factory. A new backend can therefore be added by implementing the runtime interface and registering its provider, without modifying the Hub Server, User Server, shared event protocol, or generic Device Agent lifecycle.

Beyond the evaluated Linux CLI, Linux desktop GUI, and Android GUI backends, the implementation includes simulated IoT control, local coding-agent CLI harnesses, and adapters for external agent platforms. Backends requiring process, container, remote-host, or permission isolation can expose their tools through the Model Context Protocol (MCP). The User Server binds an MCP endpoint to a `device_id` and forwards the binding to the corresponding Device Agent. The runtime becomes available only after receiving a valid binding.

These additional backends reuse the same registration, dispatch, progress, cancellation, and result interfaces. However, they were not instantiated in the reported benchmark configurations. We therefore make no empirical claim about their task performance or execution efficiency.

E CASE STUDY

E.1 CLI CASE: ASYNCHRONOUS MULTI-HOST DEPLOYMENT

CLI Case — Three-Host Deployment and Health Verification

This case follows a recorded run of MDCBench-CLI Task 25, a DevOps deployment task, executed by our Kimi-K2.5 configuration during the timeout-retry run. Five agents take part. A User Agent plans the work and dispatches requests. Three Linux CLI agents, `Linux Server-1`, `Linux Server-2`, and `Linux Server-3` (hosts `linux-1`, `linux-2`, and `linux-3`), report load and available memory, launch the services assigned to them, and probe one another's health endpoints. A Mac-side reporting agent, `Personal Computer` (`mac-pc`), writes the final deployment report. The archive records `status=ok`, `verified=True`, and a benchmark runtime of 460.19 s. Here "Mac-side" denotes the role specified by the task; the archived runtime for `mac-pc` is labelled `linux_mcp`, so the device name does not establish execution on physical macOS hardware.

Task prompt. The scripts `service_1.py`, `service_2.py`, and `service_3.py` under `/root/dev_path/` listen on ports 5001, 5002, and 5003, respectively. (1) Collect the one-minute load average and available memory on `linux-1`, `linux-2`, and `linux-3`. (2) Assign `service_1` to the host with the lowest load, `service_2` to the host with the middle-ranked load, and `service_3` to the host with the most available memory. (3) Run each assigned `/root/dev_path/service_N.py` in the background on its target host, redirecting stdout and stderr to `/var/log/bench_service_N.log`. (4) From each Linux host, send HTTP requests to every deployed `<target_host>:<port>/health` endpoint and collect the status codes into a cross-host access matrix. (5) On the local Mac, write the service-to-host deployment plan and the matrix, with columns `Src,Dst,Port,HTTPStatus`, to `benchmark/pc_workspace/deploy_report.txt`.

Presentation convention. Each bubble is an abridged summary of the corresponding agent activity or message, not a verbatim transcript. A device task may involve multiple ReAct steps and function calls before producing a returned event; intermediate reasoning and tool interactions are omitted for readability. Dispatches and returned events remain distinct.

Time convention. T+0 is the first request dispatch. Dispatch times refer to request creation; device-action times refer to runtime execution starts; returned-event times refer to result creation, not the later User Agent reply. WAIT and FINAL times use the User Agent reply log. Where shown, "UA follow-up logged" identifies that later log timestamp, not the first receipt of the result. All times are relative seconds rounded to two decimal places. The 460.19 s benchmark runtime includes work outside the first-dispatch-to-final-reply interval.

Trace scope. The archive contains 43 requests and 43 corresponding terminal results: 30 metric requests, six deployment requests, six health-check requests, and one report-writing request. The first metric set is shown individually; the remaining nine sets are compacted separately for each device. Repeated WAIT replies are also compacted. Each timestamp marks the start of the activity or the event time. For a compacted group, the left-hand timestamp marks its first activity or event; individual metric-return times remain listed in the body. Records can overlap and do not denote serial stages. Bubbles paraphrase requests, local actions, and results rather than representing a verbatim chat transcript. DISPATCH identifies a request, DEVICE ACTION a local action, EVENT RETURN a device result, and WAIT an orchestrator waiting reply.

T+0.00 s — DISPATCH

User Agent:

To Linux Server-1, Linux Server-2, and Linux Server-3: read `/proc/loadavg` and `/proc/meminfo`; return the one-minute load average and available memory. This is the first set of three metric requests.

T+0.00 s — DEVICE ACTION

Linux Server-1 Agent:

Reads `/proc/loadavg` and `/proc/meminfo`.

T+0.02 s — DEVICE ACTION

Linux Server-2 Agent:

Reads `/proc/loadavg` and `/proc/meminfo`.

T+0.03 s — DEVICE ACTION

Linux Server-3 Agent:

Reads `/proc/loadavg` and `/proc/meminfo`.

T+5.06 s — DISPATCH

User Agent:

Repeats metric collection in nine further sets of three requests: nine additional requests per Linux agent. Together with the first set, 30 metric requests are dispatched.

T+10.10 s — EVENT RETURN

Linux Server-1 Agent:

Load: 1.80; available memory: 959399732 KB.

UA follow-up logged at T+90.30 s.

T+10.11 s — DEVICE ACTION

Linux Server-1 Agent:

Begins the first of nine further metric-collection tasks. Each task returns its own result.

T+10.18 s — EVENT RETURN

Linux Server-3 Agent:

Load: 1.80; available memory: 959390924 KB.

UA follow-up logged at T+96.30 s.

T+10.18 s — DEVICE ACTION

Linux Server-3 Agent:

Begins the first of nine further metric-collection tasks. Each task returns its own result.

T+11.45 s — EVENT RETURN

Linux Server-2 Agent:

Load: 1.82; available memory: 959387728 KB.

UA follow-up logged at T+135.30 s.

T+11.45 s — DEVICE ACTION

Linux Server-2 Agent:

Begins the first of nine further metric-collection tasks. Each task returns its own result.

T+20.82 s

Linux Server-1 Agent:

Nine further results (time: load; available memory):
T+20.82 s: 1.69; 959379340 KB;
T+33.59 s: 1.58; 959257948 KB;
T+42.17 s: 1.57; 959392740 KB;
T+52.99 s: 1.49; 959243828 KB;
T+60.94 s: 1.41; 959431500 KB;
T+73.43 s: 1.42; 959450896 KB;
T+88.60 s: 1.48; 959461872 KB;
T+97.68 s: 1.41; 959470152 KB;
T+105.93 s: 1.42; 959480536 KB.

Nine separate returned events, compacted for this agent.

T+105.93 s

9 separate events, show in one bubble

EVENT RETURN

T+21.21 s

Linux Server-3 Agent:

Nine further results (time: load; available memory):
T+21.21 s: 1.69; 959390712 KB;
T+30.88 s: 1.58; 959390348 KB;
T+39.67 s: 1.57; 959270980 KB;
T+51.25 s: 1.49; 959410656 KB;
T+60.48 s: 1.41; 959422028 KB;
T+70.76 s: 1.46; 959442924 KB;
T+78.75 s: 1.39; 959461108 KB;
T+89.84 s: 1.48; 959460372 KB;
T+98.46 s: 1.41; 959401440 KB.

Nine separate returned events, compacted for this agent.

T+98.46 s

9 separate events, show in one bubble

EVENT RETURN

T+23.27 s

Linux Server-2 Agent:

Nine further results (time: load; available memory):
T+23.27 s: 1.69; 959392408 KB;
T+32.74 s: 1.58; 959393616 KB;
T+44.51 s: 1.53; 959269756 KB;
T+55.65 s: 1.45; 959406316 KB;
T+70.06 s: 1.46; 959443668 KB;
T+84.77 s: 1.44; 959456052 KB;
T+94.30 s: 1.44; 959467152 KB;
T+104.32 s: 1.42; 959478288 KB;
T+115.04 s: 1.43; 959486676 KB.

T+115.04 s

9 separate events, show in one bubble

EVENT RETURN

T+85.30 s

User Agent:

WAIT

Let me wait for the device results and check what we have.

T+90.30 s

User Agent:

WAIT

Acknowledges Linux Server-1: load 1.80; available memory 959399732 KB. Still waiting for Linux Server-2 and Linux Server-3.

T+96.30 s

User Agent:

WAIT

Acknowledges Linux Server-3: load 1.80; available memory 959390924 KB. Still waiting for Linux Server-2.

T+127.97 s

User Agent:

DISPATCH

Deploy `service_1.py` on Linux Server-1, `service_2.py` on Linux Server-3, and `service_3.py` on Linux Server-2. Run from `/root/dev_path/` with `nohup python3`, redirect stdout/stderr to `/var/log/bench_service_N.log`, and return the service, port, status, and PID.

T+127.97 s — DEVICE ACTION

Linux Server-1 Agent:

Launches `service_1.py`. Checks the process and service status.

T+127.98 s — DEVICE ACTION

Linux Server-3 Agent:

Launches `service_2.py`. Checks the process and service status.

T+128.00 s — DEVICE ACTION

Linux Server-2 Agent:

Launches `service_3.py`. Checks the process and service status.

T+135.30 s — WAIT

User Agent:

Services dispatched. Waiting for deployment confirmations.

T+139.30 s — WAIT

User Agent:

Repeatedly reports pending deployment confirmations while earlier metric-result follow-ups are still being processed. These waiting replies overlap device executions and repeated deployment requests.

T+149.34 s — EVENT RETURN

Linux Server-2 Agent:

`service_3`: port 5003; reported status `running`; reported PID 976.

UA follow-up logged at T+350.30 s.

T+150.97 s — EVENT RETURN

Linux Server-3 Agent:

`service_2`: port 5002; reported status `running`; reported PID 960.

UA follow-up logged at T+354.30 s.

T+152.03 s — EVENT RETURN

Linux Server-1 Agent:

`service_1`: port 5001; reported status `running`; reported PID 994.

UA follow-up logged at T+391.30 s.

T+310.07 s — DISPATCH

User Agent:

Repeats the deployment requests: `service_1.py` on Linux Server-1, `service_2.py` on Linux Server-3, and `service_3.py` on Linux Server-2. Run from `/root/dev_path/` with `nohup python3`, redirect stdout/stderr to `/var/log/bench_service_N.log`, and return the service, port, status, and PID.

T+310.08 s — DEVICE ACTION

Linux Server-1 Agent:

Repeats the background-launch command for `service_1.py`. The log reports "Address already in use"; `ss` still shows the existing listener on port 5001 (PID 994).

T+310.09 s — DEVICE ACTION

Linux Server-3 Agent:

Repeats the background-launch command for `service_2.py`. Checks the process and service status.

T+310.10 s — DEVICE ACTION

Linux Server-2 Agent:

Repeats the background-launch command for `service_3.py`. Checks the process and service status.

T+329.11 s — EVENT RETURN

Linux Server-3 Agent:

`service_2`: port 5002; reported status `running`; reported PID 959.

UA follow-up logged at T+395.30 s.

T+336.79 s — EVENT RETURN

Linux Server-1 Agent:

service_1: port 5001; reported status running; reported PID 993. The returned message also acknowledges the address-in-use error.

UA follow-up logged at T+401.30 s.

T+341.00 s — EVENT RETURN

Linux Server-2 Agent:

service_3: port 5003; reported status running; reported PID 976.

UA follow-up logged at T+404.30 s.

T+347.30 s — WAIT

User Agent:

All metric collection tasks completed. Now waiting for service deployment confirmations.

T+350.30 s — WAIT

User Agent:

Acknowledges Linux Server-2: service_3, port 5003, PID 976. Still waiting for Linux Server-1 and Linux Server-3.

T+354.30 s — WAIT

User Agent:

Acknowledges Linux Server-3: service_2, port 5002, PID 960. Still waiting for Linux Server-1.

T+365.24 s — DISPATCH

User Agent:

To Linux Server-1, Linux Server-2, and Linux Server-3: Use curl to check linux-1:5001/health, linux-3:5002/health, and linux-2:5003/health; return the HTTP status codes.

T+365.24 s — DEVICE ACTION

Linux Server-1 Agent:

Probes linux-1:5001/health, linux-3:5002/health, and linux-2:5003/health with curl.

T+365.25 s — DEVICE ACTION

Linux Server-2 Agent:

Probes linux-1:5001/health, linux-3:5002/health, and linux-2:5003/health with curl.

T+365.26 s — DEVICE ACTION

Linux Server-3 Agent:

Probes linux-1:5001/health, linux-3:5002/health, and linux-2:5003/health with curl.

T+373.46 s — EVENT RETURN

Linux Server-3 Agent:

linux-1:5001: HTTP 200; linux-3:5002: HTTP 200; linux-2:5003: HTTP 200.

UA follow-up logged at T+408.30 s.

T+374.24 s — EVENT RETURN

Linux Server-1 Agent:

linux-1:5001: HTTP 200; linux-3:5002: HTTP 200; linux-2:5003: HTTP 200.

UA follow-up logged at T+413.30 s.

T+374.26 s — EVENT RETURN

Linux Server-2 Agent:

linux-1:5001: HTTP 200; linux-3:5002: HTTP 200; linux-2:5003: HTTP 200.

UA follow-up logged at T+425.30 s.

T+387.67 s
DISPATCH

User Agent:

To Linux Server-1, Linux Server-2, and Linux Server-3: repeat the health checks after `sleep 2`. Use `curl` to check `linux-1:5001/health`, `linux-3:5002/health`, and `linux-2:5003/health`; return the HTTP status codes.

T+387.67 s
DEVICE ACTION

Linux Server-1 Agent:

Runs `sleep 2`, then probes `linux-1:5001/health`, `linux-3:5002/health`, and `linux-2:5003/health` with `curl`.

T+387.68 s
DEVICE ACTION

Linux Server-2 Agent:

Runs `sleep 2`, then probes `linux-1:5001/health`, `linux-3:5002/health`, and `linux-2:5003/health` with `curl`.

T+387.69 s
DEVICE ACTION

Linux Server-3 Agent:

Runs `sleep 2`, then probes `linux-1:5001/health`, `linux-3:5002/health`, and `linux-2:5003/health` with `curl`.

T+391.30 s
WAIT

User Agent:

Reports that health checks have been dispatched and waits for results. At T+404.30 s, reports that all services are confirmed running.

T+399.94 s
EVENT RETURN

Linux Server-2 Agent:

`linux-1:5001`: HTTP 200; `linux-3:5002`: HTTP 200; `linux-2:5003`: HTTP 200.

UA follow-up logged at T+430.30 s.

T+399.95 s
EVENT RETURN

Linux Server-1 Agent:

`linux-1:5001`: HTTP 200; `linux-3:5002`: HTTP 200; `linux-2:5003`: HTTP 200.

UA follow-up logged at T+434.30 s.

T+400.22 s
EVENT RETURN

Linux Server-3 Agent:

`linux-1:5001`: HTTP 200; `linux-3:5002`: HTTP 200; `linux-2:5003`: HTTP 200.

UA follow-up logged at T+438.30 s.

T+408.30 s
WAIT

User Agent:

Acknowledges Linux Server-3: all three HTTP statuses are 200. Still waiting for Linux Server-1 and Linux Server-2.

T+413.30 s
WAIT

User Agent:

Acknowledges Linux Server-1: all three HTTP statuses are 200. Still waiting for Linux Server-2.

T+420.72 s
DISPATCH

User Agent:

To Personal Computer: create `benchmark/pc_workspace/deploy_report.txt` with the supplied service assignment and all nine source–destination–port health-check rows.

T+420.72 s
DEVICE ACTION

Personal Computer Agent:

Writes the supplied deployment plan and nine-row health-check matrix to `benchmark/pc_workspace/deploy_report.txt`.

T+425.30 s
WAIT

User Agent:

Reports that the report-writing request has been dispatched to Personal Computer and waits for file-creation confirmation.

T+434.19 s
EVENT RETURN

Personal Computer Agent:

Returns `status=success` and the report path `benchmark/pc_workspace/deploy_report.txt`.

UA follow-up logged at T+447.30 s.

T+447.30 s

User Agent:

Reports successful completion: all nine unique health-check combinations return HTTP 200, and the deployment report has been created. The final response lists `Linux Server-1service_15001PID 993`, `Linux Server-3service_25002PID 959`, and `Linux Server-2service_35003PID 976`.

Verification boundary. The two health-check rounds produced 18 HTTP 200 observations over nine unique source–destination combinations (including same-host checks). The report contains those nine combinations. The final PID values are reported metadata, not fully validated process identifiers: the tool traces show listening Python PIDs 994 on Linux Server-1 and 960 on Linux Server-3, whereas the repeated deployment results and final reply report 993 and 959. On Linux Server-1, PID 993 is the shell wrapper, not the listening Python process. The archived report-file verification does not resolve this discrepancy.

E.2 GUI Case: Cross-Device Restaurant Decision

GUI Case — Restaurant Search, Ranking, and Bookmarking

This case follows a recorded cross-device GUI run from the MDC orchestrator-state archive, in which restaurant search, rating and price lookup, script-based ranking, and bookmarking must be coordinated across two devices. Three agents take part. A User Agent plans the work and dispatches requests. A Linux desktop GUI agent, `Personal Computer` (device `slot08-linux`, agent `slot08-linux-gui`), runs the restaurant searches and the ranking attempts. An Android GUI agent, `Android Phone` (device `slot08-android`, agent `slot08-android-gui`), runs the Maps lookup and the final bookmarking. From first dispatch to final reply the run spans 3879.90 s (approximately 64 min 40 s), or 3900.69 s measured from the initial user request. Both are session-derived durations, not independently verified benchmark runtimes.

Task prompt (original wording).

1. Search for restaurants in New York on the PC, send the names of the top three restaurants to the mobile phone
2. Look up the customer ratings and price ranges of these 3 restaurants via Maps on the phone, then send the retrieved data back to the PC
3. Execute the script `cost_perform.py` on the PC to calculate the cost-performance ranking of the restaurants, then send the ranking result to the mobile phone
4. Open the page of the top-ranked restaurant on your phone and bookmark it

Presentation convention. Each bubble is an abridged summary of the corresponding agent activity or message, not a verbatim transcript. A device task may involve multiple ReAct steps and function calls before producing a returned event; intermediate reasoning and tool interactions are omitted for readability. Dispatches and returned events remain distinct. The prompt states the requested workflow; the trace and verification boundary below distinguish those requirements from the outcomes supported by the archive.

Time convention. T+0 is the first database-recorded request dispatch. DISPATCH times use request creation timestamps; returned-event times use the corresponding result's arrival in the User Agent conversation. WAIT and FINAL use that conversation's assistant-message timestamps. All times are relative seconds rounded to two decimal places. Desktop result-creation timestamps in the database are approximately 29 s later than their conversation-arrival timestamps; these fields are not mixed to infer device execution duration or network latency. Local-action bubbles summarize the recorded actions or device reports. "During Gxx" associates an action with its request; it does not assign an unverified execution start or end time.

Trace scope. All 12 dispatches and their 12 corresponding terminal results are shown separately as G01–G12: three PC searches, one Android lookup, seven PC ranking/recovery requests, and one Android bookmarking request. Selected WAIT replies show the intervening orchestrator state. The same bubble layout as the CLI case distinguishes DISPATCH, DEVICE ACTION, EVENT RETURN (or EVENT RETURN), and WAIT. The text is a source-grounded paraphrase, not a verbatim chat transcript.

T+0.00 s
DISPATCH

User Agent:

G01. To Personal Computer: search for restaurants in New York and return the first three restaurant names found.

During G01
DEVICE ACTION

Personal Computer Agent:

Searches for restaurants in New York and prepares a three-name result.

T+11.64 s
DISPATCH

User Agent:

G02. To Personal Computer: repeat the request to search for restaurants in New York and return the first three names.

During G02
DEVICE ACTION

Personal Computer Agent:

Performs the repeated restaurant search and prepares another candidate list.

T+19.71 s
DISPATCH

User Agent:

G03. To Personal Computer: dispatch a third request for the same restaurant search.

During G03
DEVICE ACTION

Personal Computer Agent:

Performs the third restaurant search and prepares another candidate list.

T+24.11 s
WAIT

User Agent:

Waiting for the PC restaurant-search results before dispatching the phone lookup.

T+86.37 s
EVENT RETURN

Personal Computer Agent:

G01. Returns Tokyo Hibachi Catering (sponsored; 5.0 stars; approximately 3.9K reviews), Gramercy Tavern (4.6 stars; approximately 5K reviews), and Monkey Bar (4.5 stars; approximately 1.9K reviews).

T+90.83 s
DISPATCH

User Agent:

G04. To Android Phone: look up Tokyo Hibachi Catering, Gramercy Tavern, and Monkey Bar in Google Maps; return each rating and price range.

During G04
DEVICE ACTION

Android Phone Agent:

Looks up ratings and price ranges for the three supplied names in Google Maps.

T+93.72 s
WAIT

User Agent:

Waiting for the phone to return ratings and price ranges.

T+95.17s
EVENT RETURN

Personal Computer Agent:

G02. Returns Gramercy Tavern, Monkey Bar, and The Smith. This differs from the first candidate list.

T+99.80s
WAIT

User Agent:

Acknowledges a second, different PC candidate list. The Android lookup for Tokyo Hibachi Catering, Gramercy Tavern, and Monkey Bar remains pending.

T+110.48s
EVENT RETURN

Personal Computer Agent:

G03. Returns Gramercy Tavern, Monkey Bar, and The Smith.

T+125.64s
WAIT

User Agent:

Reports that all PC search results have arrived, but names the third candidate ambiguously as "Tokyo Hibachi Catering/The Smith". Still waits for the phone lookup.

T+973.93s
EVENT RETURN

Android Phone Agent:

G04. Returns Tokyo Hibachi Catering: 5.0 stars, 3,866 reviews, price not listed (mobile caterer); Gramercy Tavern: 4.6 stars, 4,974 reviews, `$$$$` (USD 100+); Monkey Bar: 4.5 stars, 1,903 reviews, `$$$$` (USD 100+).

T+992.93s
DISPATCH

User Agent:

G05. To Personal Computer: execute `cost_perform.py` using the Android results and return the cost-performance scores and ranking.

During G05
DEVICE ACTION

Personal Computer Agent:

Attempts to enter a shell command through PyAutoGUI. The generated Python input code has an unterminated string literal.

T+996.39s
WAIT

User Agent:

Waiting for the PC cost-performance calculation.

T+1187.90s
EVENT RETURN

Personal Computer Agent:

G05. Returns `failed: SyntaxError: unterminated string literal` in the PyAutoGUI input code. No successful execution of `cost_perform.py` is established.

T+1201.05s
DISPATCH

User Agent:

G06. To Personal Computer: retry `cost_perform.py` from the benchmark directory using the same three restaurants and their rating/price data.

During G06
DEVICE ACTION

Personal Computer Agent:

Attempts to enter `find . -name "cost_perform.py"` through PyAutoGUI; the generated Python input code again has an unterminated string literal.

T+1202.81s
WAIT

User Agent:

Waiting for the retried script-execution task.

T+1298.64s
EVENT RETURN

Personal Computer Agent:

G06. Returns `failed`: another `SyntaxError: unterminated string literal` while attempting to enter the script-search command.

T+1310.99s
DISPATCH

User Agent:

G07. To Personal Computer: rank the three restaurants manually. The request introduces the assumption that a mobile caterer is typically more affordable, although Tokyo Hibachi has no observed price.

During G07
DEVICE ACTION

Personal Computer Agent:

Reports collecting data for 12 restaurants and creating `restaurant_data.csv`. Attempts to create an analysis script encounter syntax errors; the step limit is reached without a ranking.

T+1316.89s
WAIT

User Agent:

Waiting for the manual ranking task on PC.

T+2782.88s
EVENT RETURN

Personal Computer Agent:

G07. The event status is `completed`, but the result explicitly states `INCOMPLETE`. Reports a 12-restaurant CSV; no ranked list was generated. The proposed calculation would exclude Tokyo Hibachi because its price is missing.

T+2806.46s
DISPATCH

User Agent:

G08. To Personal Computer: use `awk` on `restaurant_data.csv` to sort rating-to-estimated-price ratios and return a ranking for the original three restaurants.

During G08
DEVICE ACTION

Personal Computer Agent:

Attempts desktop text input for the calculation command; `desktop_paste_text` fails because `xclip` is unavailable.

T+2811.65s
WAIT

User Agent:

Waiting for the PC calculation command.

T+2825.55s
EVENT RETURN

Personal Computer Agent:

G08. Returns `failed`: `desktop_paste_text` cannot find `xclip`.

T+2877.55s
DISPATCH

User Agent:

G09. To Personal Computer: retry a simple ranking with brief reasoning. The request again describes Tokyo Hibachi as typically affordable despite its missing price.

During G09
DEVICE ACTION

Personal Computer Agent:

Attempts desktop text input for the simplified ranking task; `desktop_paste_text` fails because `xclip` is unavailable.

T+2907.66s
WAIT

User Agent:

Waiting for the simplified ranking task.

T+3621.16s
EVENT RETURN

Personal Computer Agent:

G09. Returns `failed`: `desktop_paste_text` cannot find `xclip`.

T+3642.47s
DISPATCH

User Agent:

G10. To Personal Computer: type the supplied ranking—Tokyo Hibachi first, Gramercy Tavern second, Monkey Bar third—and confirm completion.

During G10
DEVICE ACTION

Personal Computer Agent:

Attempts the supplied-text task; `desktop_read_clipboard` fails because `xclip` is unavailable.

`T+3646.57 s`
WAIT

User Agent:

Waiting for the PC ranking-analysis task.

`T+3650.51 s`
EVENT RETURN

Personal Computer Agent:

G10. Returns `failed`: `desktop_read_clipboard` reports that the clipboard command `xclip` is missing.

`T+3666.62 s`
DISPATCH

User Agent:

G11. To Personal Computer: "Output this exact text as your response." The supplied text ranks Tokyo Hibachi first, Gramercy Tavern second, and Monkey Bar third, and asserts that Tokyo Hibachi has lower catering prices.

During G11
DEVICE ACTION

Personal Computer Agent:

Prepares a textual response reproducing the User Agent's supplied ranking; this is not an independently calculated ranking.

`T+3672.49 s`
WAIT

User Agent:

Waiting for the PC to output the ranking text.

`T+3718.46 s`
EVENT RETURN

Personal Computer Agent:

G11. Returns the supplied text: Tokyo Hibachi Catering first, Gramercy Tavern second, and Monkey Bar third. This is a verbatim ranking-text echo, not evidence of successful script execution or a measured price comparison.

`T+3723.43 s`
DISPATCH

User Agent:

G12. To Android Phone: open Tokyo Hibachi Catering in Google Maps and bookmark its page.

During G12
DEVICE ACTION

Android Phone Agent:

Reports opening the Tokyo Hibachi Catering listing in Google Maps and saving it to Favorites.

`T+3727.40 s`
WAIT

User Agent:

Waiting for the Android bookmarking result.

`T+3873.98 s`
EVENT RETURN

Android Phone Agent:

G12. Reports that Tokyo Hibachi Catering was saved to Favorites and that Google Maps displayed "Saved in Favorites". The returned listing is at 932 McBride Ave, Woodland Park, NJ, with 5.0 stars and 3,866 reviews.

`T+3879.90 s`

User Agent:

Declares that all four steps completed successfully and identifies Tokyo Hibachi Catering as the best-value restaurant, now bookmarked. This is the User Agent's final claim; the recorded execution does not establish successful completion of the requested script-based ranking.

Verification boundary. The ranking stage contains five failed events, one event marked `completed` whose body explicitly reports an incomplete analysis, and one completed echo of a User Agent-supplied ranking. Neither `cost_perform.py` nor the attempted replace-

ment analysis is confirmed to have produced a ranking. Tokyo Hibachi's price was not listed; the claim that it was cheaper was introduced in the User Agent's requests, not established by the Maps data. Its returned address is in Woodland Park, New Jersey, rather than New York City.
The Android result reports a successful Favorites save, but the reviewed evidence consists of event and conversation text, not independent screenshot verification. No independent `verified=True` benchmark record was established from these GUI sources. This case therefore documents an asynchronous workflow that continued through retries and a supplied-text fallback to a reported bookmark, not verified completion of every requirement in the original task.

F DISCLOSURE OF AI ASSISTANCE

Generative AI tools, e.g., ChatGPT, were used during the preparation of this manuscript to assist with language polishing and improving clarity of presentation. All research ideas, methodology, experiments, analysis, and conclusions were developed and verified by the authors.