

OS-Marathon: Benchmarking Computer-Use Agents on Vast-Horizon Repetitive Tasks

Jing Wu¹ *, Wenjie Ai⁴, Daphne Barretto², Yiye Chen³ *, Qingyu Chen⁵
Yuhang He², Pranit Chawla² †, Nicholas Gydé², Yanan Jian² †, Vibhav Vineet²

¹University of Oxford, ²Microsoft, ³Georgia Institute of Technology

⁴University of Surrey, ⁵University of Washington

Correspondence: jing.wu@eng.ox.ac.uk

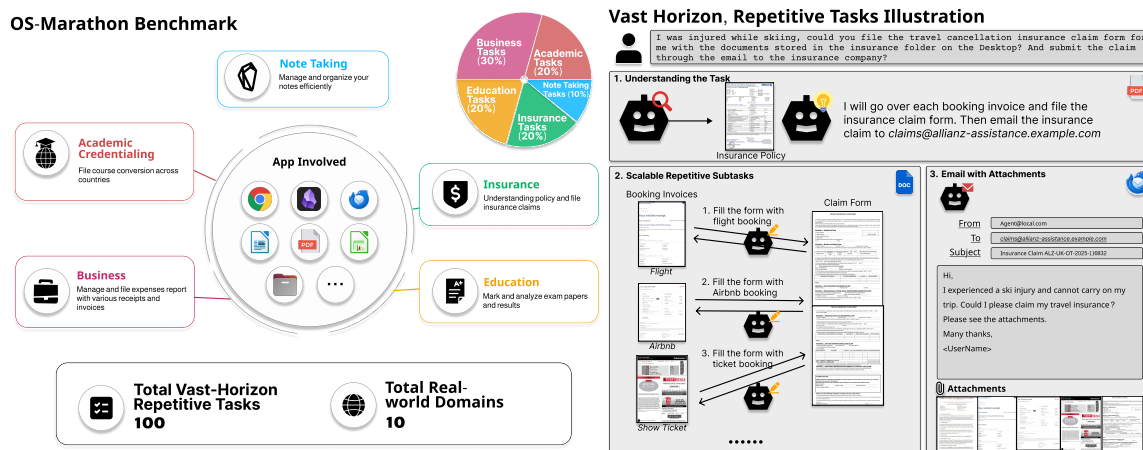


Figure 1: Overview of **OS-Marathon**. OS-Marathon contains 100 vast-horizon, repetitive tasks, spanning 5 scenarios, 10 domains, covering diverse daily use cases (Left). Successfully completing such a task requires three agent capabilities (Right): (i) interpreting a high-level user intention and understanding the task instruction, (ii) executing those subtasks iteratively and consistently, and (iii) performing the closing actions that wrap up the task once all subtasks are done.

Abstract

Vast-horizon, repetitive workflows are common in daily routines, e.g., processing expense reports from a stack of receipts, organising a collection of PDF annotations into structured notes, and are tedious for humans, with execution length scaling with the volume of data to process. They are, however, well-suited to autonomous agents, since their structured, recurring sub-workflows follow a logic that can be systematically learned. Yet no existing benchmark evaluates agents specifically on vast-horizon, repetitive capabilities, leaving the problem largely underexplored. To bridge this gap, we establish **OS-Marathon**, comprising 100 vast-horizon repetitive tasks across 5 scenarios and 10 domains, on which our evaluation reveals that leading state-of-the-art (SOTA) CUAs struggle substantially. We further find that introducing a task orchestrator to decompose the workflow into per-instance subtasks fails to mitigate the challenge: errors accumulate and propagate across solver

agents, indicating that naive decomposition is insufficient for these tasks. We then explore a cost-friendly personalisation strategy, i.e. GraphDemo, that adapts general agents to such tasks from a single human demonstration of the recurring sub-workflow logic. Extensive experiments show both the real-world challenge of vast-horizon, repetitive tasks and the improvement provided by human demonstration in this setting. Project website: <https://os-marathon.github.io/>.

1 Introduction

Vast-horizon, repetitive workflows are prevalent in both professional and personal contexts. It generally involves processing a collection of data instances, requiring executing a series of similar sub-workflows to handle each data instance iteratively. For example, in professional settings, an expense-reporting workflow requires entering a set of receipts into a single report. Analogously, in personal settings, individuals develop their own repetitive routines shaped by their specific role, tools, and

*Work completed during an internship at Microsoft.

†Work completed during employment at Microsoft.

habits, e.g., a researcher reorganising a set of PDF paper-reading annotations into a personal preferred note format, or a teacher marking and analysing student exam papers of a class in a fixed format. Such workflows are tedious for humans due to their extended duration and scalable repetitive nature.

Their repetitive structure, however, makes them a natural target for autonomous agents to alleviate human effort. Despite this potential, no existing benchmarks adequately stress-tests CUAs on real-world, vast-horizon, repetitive tasks. As illustrated in Fig. 2, existing benchmarks (Wang et al., 2024; Bonatti et al., 2025; Zhou et al., 2024; Xie et al., 2024; Wang et al.) concentrate on two task types: **self-contained atomic tasks**, independent of prior interactions or accumulated context, and **non-repetitive long-horizon workflows** whose instructions may explicitly enumerate the atomic tasks to perform (e.g., *save the files and place them in separate folders*). Neither captures the two challenges unique to real-world vast-horizon, repetitive tasks: **a scalable horizon** and **the need to decompose a high-level user intention**. The task horizon scales with the number of data instances, and the instruction itself also cannot be enumerated step-by-step in advance; for example, *file an expense report from the receipts on the desktop* requires the agent to infer the report’s internal structure and decompose the work into subtasks on its own. As a result, this task type lacks both a formal definition and a standardised evaluation framework for CUA capabilities.

To address this challenge, we formally define the problem and introduce **OS-Marathon**, a benchmark designed specifically to evaluate CUA performance on vast-horizon, repetitive workflows. OS-Marathon comprises 100 tasks, spanning 5 scenarios, 10 domains, covering both professional and personal use cases (Fig. 1 (Left)). Each OS-Marathon task tests three agent capabilities, illustrated in Fig. 1 (Right). First, the agent must interpret a high-level user intention and plan how to decompose the work into per-instance subtasks. Second, the agent needs to execute those subtasks repeatedly and consistently across all data instances; this is the stage that scales with the number of instances and drives the vast horizon of OS-Marathon tasks. Third, it must perform the closing actions that fall outside any individual subtask, e.g. completing summary sections or submitting the final output.

We evaluate leading CUAs on OS-Marathon

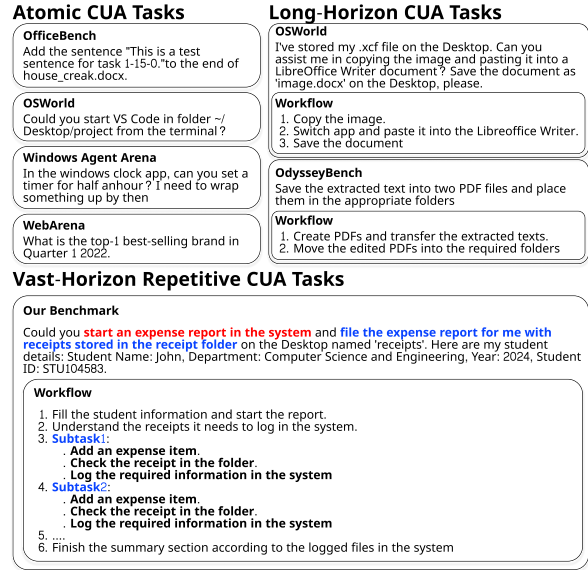


Figure 2: **Task comparison.** Atomic tasks are self-contained and require no long context. Long-horizon tasks chain atomic tasks into non-repetitive workflows whose instructions enumerate them explicitly. Vast-horizon, repetitive tasks scale with the number of data instances and are specified by high-level instructions that the agent must decompose itself.

and analyse their performance and bottlenecks at multiple granularity levels, e.g. atomic task execution, subtask-internal logic, and vast-horizon consistency etc, providing a fine-grained understanding of how and where each agent fails. The evaluation reveals that even leading CUAs fall behind human performance on OS-Marathon, leaving a substantial gap between general-purpose agents and the user-specific, vast-horizon, repetitive workflows they would need to execute in practice.

To bridge the performance gap, we explore two routes. We first investigate using a task orchestrator that decomposes the workflow into per-instance subtasks to help agents on this task, but find it counterproductive: *an error made by an earlier solver agent propagates forward and affects subsequent ones*, showing that this task is not solvable via simple decompositions. The second route exploits the repetitive structure of OS-Marathon and suggests a cost-friendly personalisation strategy GraphDemo: since every per-instance subtask shares a similar execution logic, a single human demonstration on a small number of subtasks could, in principle, teach the agent the execution logic of the remaining ones. We structure the demonstration as a causal graph to support the vast horizons of OS-Marathon tasks, providing the first reference point for personalising general CUAs on vast-horizon, repetitive tasks.

The contributions of this work are as follows:

- We formally define vast-horizon, repetitive CUA tasks, identifying two properties that distinguish them from prior benchmarks: a scalable repetitive sub-workflow structure that drives the task length, and high-level user intentions that the agent must decompose on its own.
- We introduce **OS-Marathon**, the first diverse benchmark tailored to this setting, comprising 100 tasks across 5 scenarios, 10 domains, and 12 execution environments.
- We evaluate leading CUAs on OS-Marathon across multiple granularity levels and benchmark them against human performance, demonstrating the real-world challenges that OS-Marathon poses.
- We explore agent personalisation from a single human demonstration as a first step toward bridging the gap between general-purpose agents and the user-specific, vast-horizon, repetitive workflows.

2 Related Works

Benchmarks. CUA benchmarks fall into two broad families. **Component-specific** benchmarks target fine-grained capabilities in isolation, such as UI grounding accuracy (Li et al., 2025; Liu et al., 2024a) and trajectory-level behaviour (Wang et al., 2025b). These benchmarks are useful for diagnosing low-level skills but do not measure how well an agent composes those skills into a complete workflow. **End-to-end** benchmarks measure task success across full executions and span desktop (Xie et al., 2024; Abhyankar et al., 2025), web (Deng et al., 2023), and mobile (Rawles et al., 2025; Lu et al., 2025; Liu et al., 2026) domains, where most tasks are short-horizon. OdysseyBench (Wang et al.; Xu et al., 2026) introduces longer office workflows, pushing the field toward more realistic horizons. However, it does not capture vast-horizon, repetitive capabilities, where the horizon scales with the number of data instances.

Computer-Use Agents. Existing computer-use agents follow two primary paradigms: **end-to-end specialised models** and **agentic frameworks**. End-to-end systems (Liu et al., 2024b; Fu et al., 2025; Wang et al., 2025b) train a monolithic model to manage the entire perception-reasoning-action loop, encompassing planning, grounding, execution, and reflection. Notable examples include AGUVIS (Xu et al., 2024) for unified vision-based

control, UI-TARS (Qin et al., 2025) for human-like keyboard and mouse input, and OpenAI’s Operator (OpenAI, 2025b), which combines GPT-4o with RL-based GUI control. In contrast, agentic frameworks (Xie et al., 2025; Yang et al., 2025; Yu et al., 2025; Guo et al., 2025; Cheng et al., 2024; Wu et al., 2025b) decompose the workflow into modular components (planning, grounding, execution) managed by an orchestrator. For instance, AgentS (Agashe et al., 2025a,b) sequences these modules explicitly. CoAct (Hou et al., 2024) coordinates distinct code and GUI agents for CUA tasks. Despite this progress, current agents struggle with vast-horizon, repetitive tasks: they fail to grasp the logic of individual subtasks and to execute that logic consistently across the vast horizon.

Agent Personalisation. Adapting agents to individual users’ workflows and preferences is an emerging challenge. Existing approaches fall into two categories: **memory-based** methods, which accumulate interaction experiences at inference time, and **demonstration-based** methods, which leverage explicit human demonstrations as guidance. Memory-based approaches include ICAL (Sarch et al., 2024), which distils interaction experiences into reusable memories for VLM agents, and PerPilot (Wang et al., 2025a), which personalises mobile agents through memory retrieval and reasoning-based exploration. Demonstration-based approaches include AdaptAgent (Verma et al., 2025) and LearnAct (Liu et al., 2026), which use few-shot demonstrations to aid web and mobile agent generalisation; ShowUI-Aloha (Zhang et al., 2026) and Synapse (Zheng et al., 2023), which inject full demonstration trajectories as in-context guidance for computer control; and IFR Agent (Wu et al., 2025a), which leverages demonstrations to interpret user intent in mobile settings. However, these methods target short-horizon tasks where the task demonstration fits within the model’s context window. For vast-horizon, repetitive workflows, this assumption breaks: injecting the entire task trajectory exceeds context limits, while retrieving complete demonstrations operates at the wrong granularity for tasks where the agent needs step-level guidance within a single extended execution.

3 OS-Marathon Benchmark

3.1 Task Definition

Preliminaries. We formulate a computer-use agent task as a Partially Observable Markov Deci-

sion Process (POMDP) $(\mathcal{S}, \mathcal{O}, \mathcal{A}, \mathcal{T}, \mathcal{R})$. Following OSWorld (Xie et al., 2024), the state space $\mathcal{S}$ captures the underlying desktop environment, and the observation space $\mathcal{O}$ collects the multimodal computer signals, including a screenshot, a natural-language instruction, an accessibility (a11y) tree, or any combination thereof, depending on available modalities. The action space $\mathcal{A}$ contains all actions executable by the agent. The transition function $\mathcal{T} : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$ governs how the desktop state evolves under each action. The reward function $\mathcal{R} : \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$ offers a sparse signal, assigning a reward of 1 only at the final step if and only if the task is completed.

Vast-horizon, repetitive tasks. We specialise this POMDP to tasks that consist of sequentially processing a set of N independent data instances $\{d_i\}_{i=1}^N$, where each d_i is a distinct item (e.g., one receipt to be logged to the expense system, or one student’s exam to be graded) and N denotes the total number of items to be processed. The complete workflow $\mathbf{M}$ decomposes into a pre-subtask phase $\mathbf{m}_{\text{pre}}$, N repetitive subtask phases $\{\mathbf{m}_i\}_{i=1}^N$, and a post-subtask phase $\mathbf{m}_{\text{post}}$:

$$\mathbf{M} = \{\mathbf{m}_{\text{pre}}, \mathbf{m}_1, \mathbf{m}_2, \dots, \mathbf{m}_N, \mathbf{m}_{\text{post}}\}.$$

Each $\mathbf{m}_i$ is a specific POMDP workflow with l_i POMDP steps used to process a subtask, corresponding to the data instance d_i (e.g., logging one receipt into the form).

$$\mathbf{m}_i = \{(S_{i,j}, O_{i,j}, A_{i,j}, T_{i,j}, R_{i,j}) | j \in [1, l_i]\}.$$

where the actions $A_{i,j}$ is conditioned on the specific information contained in d_i (e.g., `ExtractInfo( $d_i$ )` reads a vendor name from a receipt). In our formulation, the high-level execution logic of $\mathbf{m}_i$ is shared across i , while only the data d_i varies. The pre- and post-subtask phases $\mathbf{m}_{\text{pre}}$ and $\mathbf{m}_{\text{post}}$ are two one-time workflows: $\mathbf{m}_{\text{pre}}$ prepares the environment before any data is processed (e.g., filling in shared meta-information on the expense form, such as the employee’s name and ID) and $\mathbf{m}_{\text{post}}$ runs after all data instances are processed (e.g., emailing the completed insurance claim to the insurance company once every booking has been entered in the form). They are standard POMDP trajectories not directly conditioned on d_i , yet they share the execution environment with the per-instance subtasks $\mathbf{m}_i$ and are therefore indirectly shaped by the data through the environment state. The reward function assigns a reward of 1 only upon the completion of the post-subtask phase, i.e., $R_{\text{post}} = 1$.

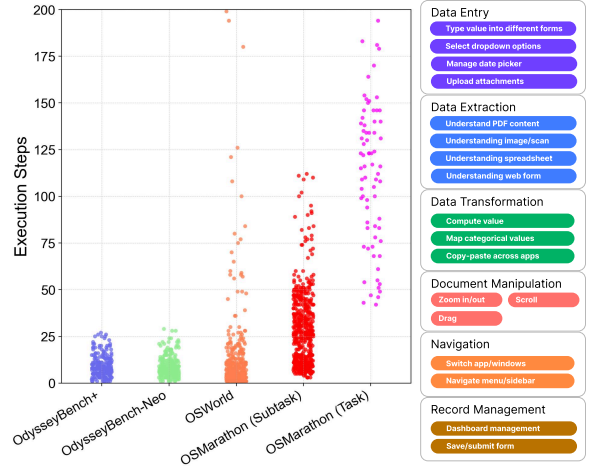


Figure 3: **Benchmark statistics.** Left: Distribution of human-execution steps required to complete tasks in OdysseyBench (Wang et al.), OSWorld (Xie et al., 2024), and OS-Marathon. Right: Atomic tasks covered by OS-Marathon.

3.2 Benchmark Statistics

OS-Marathon comprises 100 vast-horizon repetitive tasks, spanning 5 daily scenarios, Business, Academic Credentialing, Education, Note Taking, and Insurance, across 10 domains and 12 execution environments. Each task in OS-Marathon follows the three-phase structure defined in Sec. 3.1. All tasks in OS-Marathon are drawn from real-world workflows and built to match this repetition feature.

Diversity. OS-Marathon exhibits diversity at the atomic-operation, subtask, and task levels. At the *atomic-task level*, OS-Marathon includes 18 atomic actions common in daily digital workflows, such as navigating folders or entering values into a spreadsheet. These atomic tasks fall into 6 categories: data extraction, data entry, navigation, document manipulation, data transformation, and record management (Fig. 3 (Right)). At the *subtask level*, the data instances d_i that the agent must process are themselves heterogeneous: real images and PDFs captured under varying lighting and angles, in portrait or landscape orientation, etc. The PDFs alone span multiple layouts (one- or two-column), document functions (flight tickets, car-rental receipts, transcripts, insurance policies), text densities (dense or loose), and capture modalities (native, scanned, printed-from-email, long screenshot). At the *task level*, individual tasks span multiple applications combined following the real-world use-case, typical tasks involve the mix of Documents, Spreadsheet, Note-taking apps, Emails, PDF, OS-level utilities, etc.

Subtask dependency. Crucially, tasks in OS-

Marathon are not simple aggregations of independent subtasks. Because all subtasks share a single execution environment, **the outcome of an earlier subtask directly influences the success of later ones**. For example, for a business expense reporting case, the agent must log each receipt into the correct fields with the right amount, currency, etc. An earlier entry logged into the wrong field can displace a later entry, and an incorrectly completed subtask leaves the environment in a misleading state that cascades into subsequent ones. Furthermore, for the summary section in the form, any incorrect amount for the previous subtasks will lead to failure in the summary. This **error-compounding structure** is what makes OS-Marathon a challenging benchmark for real-world, vast-horizon, repetitive tasks.

Comparison to other CUA benchmarks. We compare OS-Marathon against two counterpart benchmarks: OSWorld (Xie et al., 2024; Abhyankar et al., 2025), a widely-used benchmark for general computer-use agent tasks, and OdysseyBench (Wang et al.), which targets long-horizon office workflows. Fig. 3 (Left) plots the required execution steps for tasks in these three benchmarks. Subtasks in OS-Marathon already match the horizon of full OSWorld and OdysseyBench tasks (typically < 30 steps), with a non-trivial fraction exceeding them. At the task level, OS-Marathon extends well beyond either prior benchmark, spanning roughly 50 to 200 steps end-to-end, and is further scalable through the number of data instances N each task processes. OS-Marathon thus poses a challenge at both granularities: each subtask is itself a multi-step workflow demanding coherent reasoning and execution, and the full task adds vast-horizon consistency across all N instances.

3.3 Infrastructures & Quality Control

Infrastructures. Each task is paired with a profile of supporting data, for example, a set of receipts for a business-trip expense report, a transcript for an academic credentialing task, a batch of student exam papers for grading, a collection of PDF notes for note organisation, or a bundle of travel bookings and insurance policies for a claim. To produce realistic and privacy-preserving file sets for each task, we use a hybrid data pipeline.

Editable data (digital-native PDFs such as travel bookings, insurance policies, and student papers) are generated using a *format-plus-content* procedure: we collect raw documents from public on-

line sources^{*†} to provide realistic templates, then manually edit all task-related data (names, policy numbers, account numbers, etc.) with synthetic values to keep edits minimal to preserve the documents’ real-world realism. *Uneditable data* (image receipts from existing OCR datasets (Kaggle, 2023; ExpressExpense, 2023) and scanned PDFs collected from the same online sources) are extended by annotating the additional fields each task requires (e.g., payment method). We anonymise the regions containing sensitive information to protect the privacy. This pipeline yields a dataset that preserves the visual and structural realism of the original documents without exposing real personal information. The full data-collection process spanned roughly three months, and all data sourced from existing datasets is used in compliance with the original license terms.

To support rigorous evaluation of the vast-horizon, repetitive workflows defined above, we built 12 interactive *execution environments*, visualisations of which are shown in Appendix A.3. Each environment is modelled directly on a real-world daily workflow, and the set as a whole spans four workflow categories: *web-based* workflows (e.g., corporate or university expense-submission portals), *spreadsheet-based* workflows (e.g., grading a batch of student exam papers and recording the marks in a sheet), *document-based* workflows (e.g., filling a multi-page insurance-claim form), and *markdown-based* workflows (e.g., reading and organising notes in Obsidian).

Quality control & human evaluation. Every annotation is cross-checked by a group of three annotators to guarantee correctness. We then conduct a human-evaluation study with five additional evaluators, as shown in Tab. 1 (Last Row), who were not involved in the annotation process and complete each task end-to-end under the same setup provided to the agents. This study serves two purposes. First, as a feasibility check, it confirms that tasks in OS-Marathon are understandable and solvable by a human, so the benchmark reflects genuine real-world difficulty rather than ambiguity or unsolvability and provides a reference for agent performance. Second, the human evaluation serves as a difficulty proxy: easier tasks require fewer steps and harder tasks require more. Based on this, we partition OS-Marathon into two tiers, i.e. OS-Marathon-Lite contains tasks humans complete

^{*}<https://www.scribd.com/>

[†]<https://www.slideshare.net/>

Agents	Business	Academic Cred.	Education	Note Taking	Insurance	Web	Spreadsheet	Document	Obsidian	2-app	3-app	4-app	Overall
OpenCUA7B	02.29 / 00.00	04.57 / 00.00	00.39 / 00.00	15.86 / 20.84	10.16 / 00.00	07.19 / 00.00	00.39 / 00.00	00.00 / 00.00	15.86 / 20.84	06.30 / 04.17	10.35 / 00.00	00.26 / 00.00	06.65 / 04.17
UITARS7B	00.70 / 00.00	00.00 / 00.00	00.00 / 00.00	14.47 / 19.45	02.34 / 00.00	01.50 / 00.00	00.00 / 00.00	00.00 / 00.00	14.47 / 19.45	03.68 / 04.32	02.34 / 00.00	00.00 / 00.00	03.50 / 03.89
AgentS2.5 w/ GPT5	47.93 / 30.61	40.76 / 37.08	02.73 / 00.00	40.49 / 34.71	35.59 / 14.82	50.55 / 38.88	05.30 / 00.41	30.56 / 00.00	40.49 / 34.71	50.53 / 39.89	26.90 / 15.44	11.10 / 00.00	33.50 / 23.44
UiPath w/ Opus 4.5	47.85 / 29.95	34.32 / 19.08	15.17 / 01.56	34.92 / 23.93	27.24 / 07.29	40.11 / 27.46	24.74 / 01.04	23.75 / 08.68	34.92 / 23.93	41.04 / 31.47	37.31 / 02.95	18.03 / 03.94	31.90 / 16.36
GPT5.4-Mini	19.27 / 21.41	02.79 / 01.00	02.28 / 00.00	15.07 / 14.37	10.11 / 00.00	12.53 / 12.80	05.58 / 00.74	08.75 / 00.00	15.07 / 14.37	13.25 / 15.67	11.82 / 01.11	04.44 / 00.00	09.90 / 07.35
GPT5.4-Nano	00.00 / 00.00	00.18 / 00.00	00.50 / 00.00	14.47 / 14.29	03.13 / 00.00	01.32 / 00.00	00.33 / 00.00	00.00 / 00.00	14.47 / 14.29	02.96 / 02.86	03.13 / 00.00	00.33 / 00.00	03.65 / 02.86
o3	00.00 / 00.00	00.38 / 00.50	00.50 / 00.00	19.44 / 16.50	03.91 / 00.00	01.71 / 00.20	00.33 / 00.00	00.00 / 00.00	19.44 / 16.50	04.04 / 03.50	03.91 / 00.00	00.00 / 00.00	04.85 / 03.40
GPT5.4-NCU	82.90 / 61.22	67.68 / 55.95	57.77 / 35.42	78.98 / 65.73	54.12 / 38.63	77.26 / 66.54	59.81 / 29.00	42.09 / 23.96	78.98 / 65.73	79.83 / 69.03	65.02 / 34.72	52.54 / 31.60	68.29 / 51.39
Human	93.31 / 82.84	87.63 / 75.80	84.40 / 76.77	94.10 / 88.26	86.51 / 74.16	94.42 / 83.47	86.45 / 76.39	87.16 / 77.45	94.10 / 88.26	93.61 / 82.59	87.41 / 79.88	83.73 / 76.46	88.38 / 80.52

Table 1: **Baseline performance on OS-Marathon.** Each cell reports the field-level / subtask-level success rate (%). Columns are grouped into four axes, distinguished by background colour: **five task scenarios** (light blue), **four workflow types** (light yellow), **#apps/task** (light green), and **overall** (light purple). See Sec. 3.2 for scenario and workflow-type definitions. Row colours mark agent categories (Sec. 4.1).

within 100 steps, and OS-Marathon-Pro contains tasks requiring 100 to 200 steps.

4 Benchmarking SOTA Agent Baselines

4.1 Baseline Experiment Setup

Agent baselines. To ensure a rigorous evaluation, we benchmark state-of-the-art VLM agents drawn from four categories: (i) **open-weight models**, OpenCUA-7B (Wang et al., 2025b) and UI-TARS-1.5-7B (Qin et al., 2025); (ii) **agentic frameworks**, AgentS2.5 with GPT-5 (Agashe et al., 2025b) and UiPath Screen Agent (UiPath, 2026) with Opus 4.5 (Anthropic, 2025); (iii) **proprietary reasoning models** from OpenAI, o3 (OpenAI, 2025a), GPT-5.4-mini (OpenAI Company, 2026), and GPT-5.4-nano (OpenAI Company, 2026); and (iv) an **agent with native computer use**, GPT-5.4 Native Computer Use (OpenAI, 2026), with high reasoning effort.

Evaluation protocol. We evaluate agents under budgets matched to human performance, following the evaluation protocol of OSWorld (Xie et al., 2024). To enable fine-grained analysis of the results, we report success rates at different granularity levels: (i) *field-level*, the correctness of each individual element (e.g., a single form field); (ii) *subtask-level*, the success rate over the group of individual elements that constitute one subtask (e.g., one receipt entered correctly in full). Due to the space limit, we provide further evaluation of the baseline performance on the end-to-end *task-level* success rate in Appendix A.1.1.

4.2 Results & Analysis

Tab. 1 reports field- and subtask-level success rates on OS-Marathon, broken down by scenario, workflow type, and number of apps per task.

Obs. 1: Humans still largely outperform SOTA agents.
Obs. 2: Best product-level native agent still struggles to solve vast-horizon, repetitive tasks.

Bottleneck. OpenCUA-7B (Wang et al., 2025b), UI-TARS-7B (Qin et al., 2025), o3 (OpenAI,

2025a), and GPT-5.4-nano (OpenAI Company, 2026) are bottlenecked at the atomic-operation level. The cause is not solely in raw grounding ability, but also in *high-level user-intention understanding*: the agent is unable to reliably decompose a high-level intention such as *file an expense report* into the concrete atomic steps it implies (e.g., opening a receipt, reading the payment amount, and entering it into the form). The remaining agents are able to execute atomic actions but bottleneck at the subtask and task levels, where we identify two failure modes. (i) **Visual-salience bias and downstream hallucination.** As shown in Fig. 4 (Step #2), the agent’s reasoning tends to be affected by the UI element that is most visually salient and most directly named in the instruction. This is likely related to how the model was pre-trained. *Expense report* and *upload receipt* commonly appear together in LLM training data, so during reasoning, the agent easily associates the two and jumps to *upload receipt* directly. The actions a global plan would require, e.g., first opening a receipt to read its content, are less directly linked to *expense report* in training, so the agent is more easily distracted by *upload receipt* and less likely to perform them. (ii) **Weak global planning and persistence.** Even when each subtask is executed correctly in isolation, agents fail to sustain the full N -instance loop. Without a global plan, they process data instances opportunistically rather than in a planned order, and the loop often terminates before all N instances are completed. We elaborate the analysis in Appendix A.1.3.

Analysis by scenario and workflow type. Across **scenarios**, education and insurance emerge as uniformly the hardest, which we attribute to their stronger *cross-referencing* dependencies: each subtask refers to information from multiple sources (e.g., an insurance claim must reference both the policy and the supporting documents), which is less intuitive for agents, making hallucination more likely. Tab. 1 (light green) also confirms this,

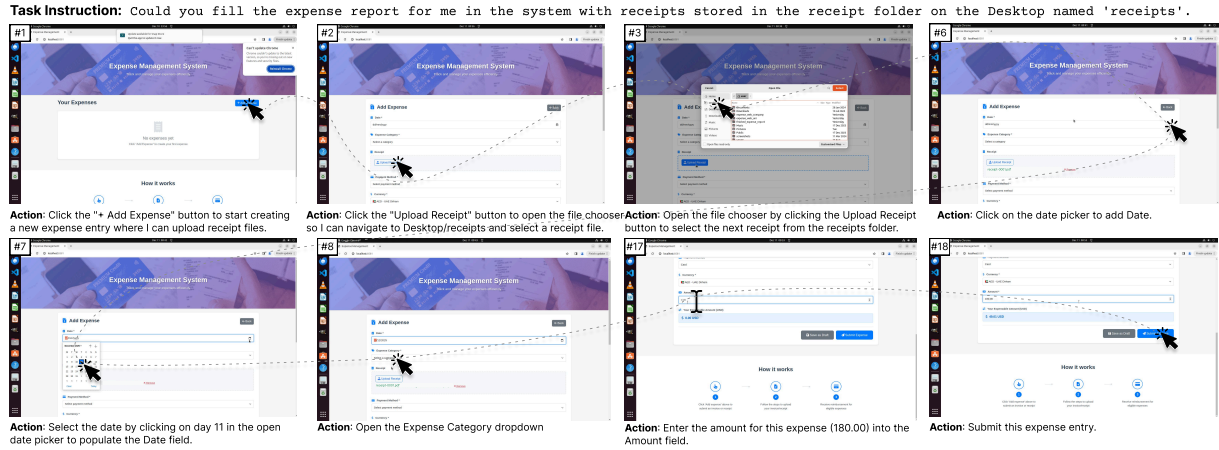


Figure 4: **Failure case analysis: AgentS2.5 + GPT-5.** The agent clicks the salient *Upload Receipt* button (#2) before reading any receipt, drawn by the LLM training association between *Upload Receipt* and *Fill an Expense Report*. Once on this trajectory, the agent hallucinates plausible values for the date (#7) and payment amount (#17) with no grounded source to draw from, compounding one planning error into a cascade of incorrect entries.

when tasks are grouped by #apps each must cross-reference, both field-level and subtask-level success drop sharply and near-monotonically as the #app/task grows. Across **workflow** types, performance varies with *error tolerance of the environment*: web workflows are more manageable because their actions are bounded and reversible, a mistake rarely renders a destructive result, while spreadsheet and document workflows expose a wider editing surface where a single destructive action (a stray drag, a misdirected paste) can corrupt the environment beyond recovery. More analysis is provided in Appendix A.1.4.

Cost and efficiency analysis. Fig. 7 (In Appendix) plots agent token usage, monetary cost, and step count against subtask-level accuracy, from which three observations stand out. *First*, the cheapest agent that achieves non-trivial subtask-level performance costs roughly \$5~\$10 per task on average (e.g. AgentS2.5 + GPT5 in Fig. 7 (Middle)), and this figure scales linearly with the number of data instances N , already prohibitive for everyday personal use. Two open questions follow from this. How can competitive performance be delivered from cheaper open-weight models? And how can a single human demonstration be used to personalise off-the-shelf models in a way that meaningfully reduces inference cost rather than merely improving accuracy at the same cost? *Second*, cost and accuracy are not tightly correlated across agents: UiPath + Opus 4.5 incurs a cost comparable to GPT-5.4 (high) but is substantially outperformed by it across all scenarios. *Third*, stronger agents are also more step-efficient: GPT-5.4 with native computer use achieves the highest success rate while

taking fewer steps than several substantially weaker baselines.

5 Agent Personalisation with GraphDemo

The baselines evaluated above are general-purpose computer-use agents, yet real-world, vast-horizon, repetitive workflows are highly personalised: each organisation has its own expense form, teachers in different countries follow different grading standards and conventions, and each insurer maintains its own claim portal and forms, etc. Adapting a general-purpose agent to a personalised workflow typically requires collecting supervised data and fine-tuning the underlying model, both prohibitively resource-intensive for individual users and small teams.

These costs, however, are largely avoidable for the workflows we target. Vast-horizon, repetitive tasks in OS-Marathon feature a repetitive structure, which opens a cost-friendly alternative: subtasks within a task share a similar execution logic, so a single human demonstration of a few subtasks can carry enough information for the agent to handle the rest. This raises the question we explore next: *how to use a single human demonstration to adapt an off-the-shelf agent to a personalised, vast-horizon, repetitive workflow?*

A straightforward approach is to inject the full demonstration trajectory, e.g., every screenshot and every action, into the agent’s context. However, this approach fails for two reasons in this case. (i) **Context overflow.** Demonstrations for vast-horizon, repetitive tasks far exceed agents’ context window: even a demonstration covering only a single sub-

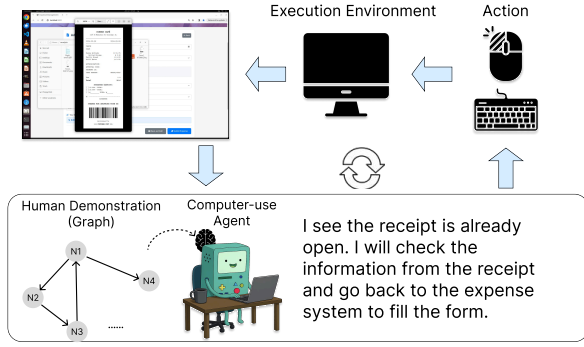


Figure 5: **GraphDemo pipeline.** A human demonstration is represented as a causal graph and incorporated into the agent’s context to guide agents’ reasoning.

task spans roughly 30 steps and 9M multimodal tokens. (ii) **Granularity mismatch.** Feeding the entire trajectory dilutes the useful signal: the agent only needs the slice relevant to its current step.

5.1 GraphDemo

Graph-based demonstration with on-demand retrieval. To resolve both issues, we represent the demonstration as a causal graph, termed as GraphDemo (Fig. 5). Each node in the graph represents one demonstrated step and stores two density levels, a textual description for compact retrieval and the screenshots for detailed visual context, while directed edges carry the executed action information. The textual abstraction of the graph is placed in the agent’s system prompt as a persistent workflow overview, giving the agent direct visibility over all candidate nodes. At each step, the agent retrieves the demonstration node best matching its planned action, along with its neighbours; the multimodal content of these nodes is injected into context during the reasoning of the next step. The full demonstration thus remains usable within the agent’s context window, while at any step the agent sees only the slice of the demonstration relevant to its action, addressing both the context-overflow and granularity-mismatch problems of the full-trajectory approach. We cover more details in Appendix A.2.2.

5.2 Experiments & Analysis

Agent personalisation experiment setup. We conduct the agent-personalisation experiments on AgentS2.5 with GPT-5 (Agashe et al., 2025b), which demonstrates solid atomic-level competence in our baseline results and therefore offers the headroom required to study higher-level personalisation effects. To investigate whether the difficulty of OS-Marathon can be reduced by explicit task de-

Agents	Business	Academic Cred.	Education	Note Taking	Insurance	Overall
S2.5 + GPTS	51.29 / 39.97	40.76 / 37.08	1.37 / 0.00	40.49 / 34.71	35.59 / 14.82	33.90 / 25.31
S2.5 + GPTS + Orch.	36.87 / 19.41	26.74 / 19.99	0.37 / 0.00	32.75 / 25.96	48.61 / 14.71	29.07 / 16.01
S2.5 + GPTS + GraphDemo	81.10 / 60.36	48.19 / 46.65	20.97 / 16.17	53.23 / 46.67	51.25 / 36.36	50.95 / 41.24
S2.5 + GPTS + Orch. + GraphDemo	64.12 / 42.80	43.26 / 41.33	13.45 / 0.00	38.55 / 38.48	44.53 / 14.38	40.78 / 27.40

Table 2: **Orchestrator-based task decomposition alone is counterproductive; GraphDemo improves performance in both settings.** Variants on AgentS2.5 + GPT-5: baseline, with orchestrator (+ Orch.), with GraphDemo alone, and with orchestrator + GraphDemo. Each cell reports field / subtask-level success rate (%).

composition, we adopt a complementary baseline: wrapping the agent as a solver in a task orchestrator that breaks the workflow into per-instance subtasks and assigns them to a solver agent one at a time. We then apply the graph-based demonstration to both setups, the vanilla agent and the orchestrator-wrapped agent, and evaluate all variants across OS-Marathon under the same protocol as the baseline experiments. We report the quantitative results in Tab. 2 and cover the additional results, analysis and implementation details for the orchestrator and demonstration integration are provided in Appendix A.2.

Obs. 3: Naively applying the orchestrator framework is largely counterproductive, providing only occasional gains, e.g., on the field-level metric for the insurance scenario.

Orchestrator decomposition analysis. The orchestrator mitigates task-level persistence, but the subtask-internal logic, which our analysis in Sec. 4.2 and Appendix A.1.3 identifies as a severe bottleneck, falls outside its scope. Furthermore, the framework introduces a new failure mode: a solver agent’s mistake corrupts the shared environment, and the next solver agent, with no context of the error, reasons from this corrupted state and compounds the failure.

We provide a failure case analysis of this in Fig. 6. The framework completes the first four subtasks successfully (Step #113) but drifts on the fifth: due to the incorrect subtask logic, the solver agent decides to remove the uploaded receipts (Step #156), believing them incorrect. When attempting to re-upload, it searches the wrong folder (**home**) (Step #164), fails to find the file (Step #165), and reports **FAIL**, leaving the environment in a corrupted state (Step #167). The next solver agent, initialised on this corrupted state with all numerical fields already filled, sees Upload Receipt as the only remaining action and misreads its instruction, “*Could you fill the expense report for me with receipt-0113.pdf from ~/Desktop/receipts?*”, as a request to simply upload the receipt. It re-uploads

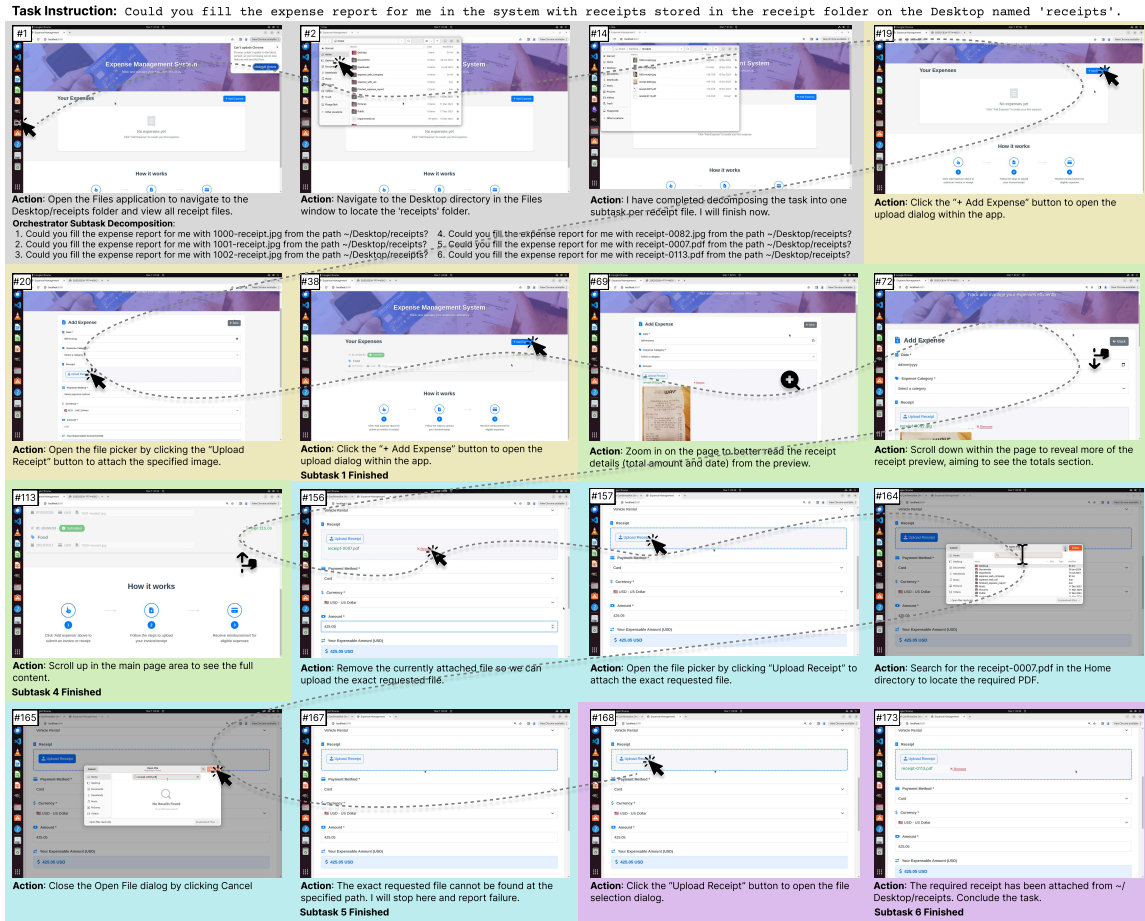


Figure 6: **Failure case analysis: AgentS2.5 + GPT-5 + Orchestrator.** The framework completes the first four subtasks successfully but drifts on the fifth: the solver mistakenly removes the uploaded receipts (#156) and fails to recover, leaving the environment in a corrupted state (#167). The next solver, initialised on this state, misreads its instruction as a request to simply upload the receipt (#168) and finishes the task incorrectly. Screenshot backgrounds denote the role of each agent: grey for the task orchestrator O, light yellow for solver S₁, light green for S₄, cyan for S₅, and light purple for S₆. S₂ and S₃ are omitted as they execute normally and are not part of the failure cascade.

(Step #168) and reports subtask 6 as completed, finishing the task incorrectly.

Obs. 4: Graph human demonstration improves general CUAs on vast-horizon, repetitive tasks, lifting weaker agents close to SOTA performance at a low cost.

GraphDemo analysis. GraphDemo improves both field-level and subtask-level success rates on the vanilla agent by guiding it through the correct execution logic. This lifts a weaker agent (AgentS2.5 + GPT-5) close to the performance of an SOTA native agent (GPT-5.4 with native computer use), giving general-purpose agents a substantial performance boost. When applied to the orchestrator framework, GraphDemo is still helpful: Orchestrator + Demo outperforms the Orchestrator alone, but its effect is weaker than in the vanilla setup, and in some cases falls below the vanilla agent itself. This is because GraphDemo acts as a soft guide, and the agent can still deviate. In this setting, when a solver agent’s mistake corrupts the shared environment,

even with the demonstration guide, the next solver agent still lacks context of the error and reasons from this corrupted state, compounding the failure.

6 Conclusion

This paper investigates vast-horizon, repetitive CUA tasks: workflows that process data instances iteratively over vast horizons. We introduce OS-Marathon, the first benchmark for this task type, spanning 100 tasks across 5 scenarios and 10 domains, and establish baselines on leading current agents. To bridge the observed performance gaps, we explore a cost-friendly personalisation strategy, GraphDemo, that encodes a single human demonstration as a causal graph and retrieves only step-relevant information into context. Extensive experiments demonstrate both the real-world challenges in this task and the effectiveness of GraphDemo, laying a foundation for future CUA research on vast-horizon, repetitive tasks.

Limitations

The human demonstration acts as a soft guide for personalisation: nothing constrains the agent to follow it consistently, and when the agent deviates, the demonstration cannot help it recover. Personalising general CUAs for vast-horizon, repetitive workflows, therefore, remains an open problem worth exploring.

Acknowledgments

This work was conducted during an internship at Microsoft; we thank Microsoft Research (MSR) and Windows Cloud Experience (WCX) for their support. We also thank the authors of the OS-World benchmark for their open-source infrastructure, which served as a foundation for this project.

References

- Reyna Abhyankar, Qi Qi, and Yiyang Zhang. 2025. OSWorld-human: Benchmarking the efficiency of computer-use agents. In *ICML 2025 Workshop on Computer-Use Agents*.
- Saaket Agashe, Jiuzhou Han, Shuyu Gan, Jiachen Yang, Ang Li, and Xin Eric Wang. 2025a. [Agent S: An Open Agentic Framework that Uses Computers Like a Human](#). In *International Conference on Learning Representations (ICLR)*.
- Saaket Agashe, Kyle Wong, Vincent Tu, Jiachen Yang, Ang Li, and Xin Eric Wang. 2025b. Agent S2: A compositional generalist-specialist framework for computer use agents. In *Second Conference on Language Modeling (COLM)*.
- Anthropic. 2025. Claude Opus 4.5 System Card. <https://assets.anthropic.com/m/64823ba7485345a7/Claude-Opus-4-5-System-Card.pdf>.
- Rogerio Bonatti, Dan Zhao, Francesco Bonacci, Dillon Dupont, Sara Abdali, Yinheng Li, Yadong Lu, Justin Wagle, Kazuhito Koishida, Arthur Bucker, Lawrence Keunho Jang, and Zheng Hui. 2025. [Windows agent arena: Evaluating multi-modal OS agents at scale](#). In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*.
- Kanzhi Cheng, Qiushi Sun, Yougang Chu, Fangzhi Xu, Li YanTao, Jianbing Zhang, and Zhiyong Wu. 2024. SeeClick: Harnessing GUI grounding for advanced visual GUI agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL)*.
- Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Samuel Stevens, Boshi Wang, Huan Sun, and Yu Su. 2023. [Mind2web: Towards a generalist agent for the web](#). In *Advances in Neural Information Processing Systems 36 Datasets and Benchmarks Track (NeurIPS)*.
- ExpressExpense. 2023. [Expressexpense srd](#).
- Tianyu Fu, Anyang Su, Chenxu Zhao, Hanning Wang, Minghui Wu, Zhe Yu, Fei Hu, Mingjia Shi, Wei Dong, Jiayao Wang, Yuyang Chen, Ruiyang Yu, Siran Peng, Menglin Li, Nan Huang, Haitian Wei, Jiawei Yu, Yi Xin, Xilin Zhao, and 4 others. 2025. Mano report. *arXiv preprint arXiv:2509.17336*.
- Liangxuan Guo, Bin Zhu, Qingqian Tao, Kangning Liu, Xun Zhao, Xianzhe Qin, Jin Gao, and Guangfu Hao. 2025. Agentic lybic: Multi-agent execution system with tiered reasoning and orchestration. *arXiv preprint arXiv:2509.11067*.
- Xinming Hou, Mingming Yang, Wenxiang Jiao, Xing Wang, Zhaopeng Tu, and Wayne Xin Zhao. 2024. Coact: A global-local hierarchy for autonomous agent collaboration. *arXiv preprint arXiv:2406.13381*.
- Kaggle. 2023. [Ocr receipts text detection - retail dataset](#).
- Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*.
- Kaixin Li, Ziyang Meng, Hongzhan Lin, Ziyang Luo, Yuchen Tian, Jing Ma, Zhiyong Huang, and Tat-Seng Chua. 2025. [ScreenSpot-Pro: GUI grounding for professional high-resolution computer use](#). In *Proceedings of the 33rd ACM International Conference on Multimedia (MM '25)*. ACM.
- Guangyi Liu, Pengxiang Zhao, Liang Liu, Zhiming Chen, Yuxiang Chai, Shuai Ren, Hao Wang, Shibo He, and Wenchao Meng. 2026. LearnAct: Few-shot mobile GUI agent with a unified demonstration benchmark. In *Findings of the Association for Computational Linguistics (ACL)*. Association for Computational Linguistics.
- Junpeng Liu, Yifan Song, Bill Yuchen Lin, Wai Lam, Graham Neubig, Yuanzhi Li, and Xiang Yue. 2024a. [Visualwebbench: How far have multimodal llms evolved in web page understanding and grounding?](#) In *First Conference on Language Modeling (COLM)*.
- Xiao Liu, Bo Qin, Dongzhu Liang, Guang Dong, Hanyu Lai, Hanchen Zhang, Hanlin Zhao, Iat Long Iong, Jiada Sun, Jiaqi Wang, Junjie Gao, Junjun Shan, Kangning Liu, Shudan Zhang, Shuntian Yao, Siyi Cheng, Wentao Yao, Wenyi Zhao, Xinghan Liu, and 11 others. 2024b. Autoglm: Autonomous foundation agents for guis. *arXiv preprint arXiv:2411.00820*.

- Quanfeng Lu, Wenqi Shao, Zitao Liu, Fanqing Meng, Boxuan Li, Botong Chen, Siyuan Huang, Kaipeng Zhang, Yu Qiao, and Ping Luo. 2025. Gui odyssey: A comprehensive dataset for cross-app gui navigation on mobile devices. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*.
- OpenAI. 2025a. Addendum to OpenAI o3 and o4-mini system card: OpenAI o3 Operator. https://cdn.openai.com/pdf/4375e605-f9a6-438d-bcc8-190599c183a6/o3_cua_system_card.pdf.
- OpenAI. 2025b. Computer-using agent: Powering operator with computer-using agent, a universal interface for ai to interact with the digital world.
- OpenAI. 2026. Introducing GPT-5.4.
- OpenAI Company. 2026. Introducing GPT-5.4 mini and nano. <https://openai.com/index/introducing-gpt-5-4-mini-and-nano/>.
- Yujia Qin, Yining Ye, Junjie Fang, Haoming Wang, Shihao Liang, Shizuo Tian, Junda Zhang, Jiahao Li, Yunxin Li, Shijue Huang, Wanjuan Zhong, Kuanye Li, Jiale Yang, Yu Miao, Woyu Lin, Longxiang Liu, Xu Jiang, Qianli Ma, Jingyu Li, and 16 others. 2025. UI-TARS: Pioneering automated GUI interaction with native agents. *arXiv preprint arXiv:2501.12326*.
- Christopher Rawles, Sarah Clinckemaulle, Yifan Chang, Jonathan Waltz, Gabrielle Lau, Marybeth Fair, Alice Li, William Bishop, Wei Li, Folawiyi Campbell-Ajala, Daniel Toyama, Robert Berry, Divya Tyamagundlu, Timothy Lillicrap, and Oriana Riva. 2025. Androidworld: A dynamic benchmarking environment for autonomous agents. In *The Thirteenth International Conference on Learning Representations (ICLR)*.
- Gabriel Herbert Sarch, Lawrence Jang, Michael J Tarr, William W Cohen, Kenneth Marino, and Katerina Fragkiadaki. 2024. Vlm agents generate their own memories: Distilling experience into embodied programs of thought. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems (NeurIPS)*.
- UiPath. 2026. UiPath Screen Agent (OSWorld implementation). https://github.com/xlang-ai/OSWorld/tree/main/mm_agents/uipath.
- Gaurav Verma, Rachneet Kaur, Nishan Srishankar, Zhen Zeng, Tucker Balch, and Manuela Veloso. 2025. AdaptAgent: Adapting multimodal web agents with few-shot learning from human demonstrations. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics.
- Weixuan Wang, Dongge Han, Daniel Madrigal Diaz, Jin Xu, Victor Rühle, and Saravan Rajmohan. Odyssey-bench: Evaluating llm agents on long-horizon complex office application workflows. *arXiv preprint arXiv:2508.09124*.
- Xin Wang, Zhiyao Cui, Hao Li, Ya Zeng, Chenxu Wang, Ruiqi Song, Yihang Chen, Kun Shao, Qiaosheng Zhang, Jinzhuo Liu, Siyue Ren, Shuyue Hu, and Zhen Wang. 2025a. Peripilot: Personalizing vlm-based mobile agents via memory and exploration. *arXiv preprint arXiv:2508.18040*.
- Xinyuan Wang, Bowen Wang, Dunjie Lu, Junlin Yang, Tianbao Xie, Junli Wang, Jiaqi Deng, Xiaole Guo, Yiheng Xu, Chen Henry Wu, Zhennan Shen, Zhuokai Li, Ryan Li, Xiaochuan Li, Junda Chen, Boyuan Zheng, Peihang Li, Fangyu Lei, Ruisheng Cao, and 23 others. 2025b. Opencua: Open foundations for computer-use agents. *arXiv preprint arXiv:2508.09123*.
- Zilong Wang, Yuedong Cui, Li Zhong, Zimin Zhang, Da Yin, Bill Yuchen Lin, and Jingbo Shang. 2024. Officebench: Benchmarking language agents across multiple applications for office automation. *arXiv preprint arXiv:2407.19056*.
- Zheng Wu, Heyuan Huang, Yanjia Yang, Yuanyi Song, Xingyu Lou, Weiwen Liu, Weinan Zhang, Jun Wang, and Zhuosheng Zhang. 2025a. Quick on the uptake: Eliciting implicit intents from human demonstrations for personalized mobile-use agents. *arXiv preprint arXiv:2508.08645*.
- Zhiyong Wu, Zhenyu Wu, Fangzhi Xu, Yian Wang, Qiushi Sun, Chengyou Jia, Kanzhi Cheng, Zichen Ding, Liheng Chen, Paul Pu Liang, and 1 others. 2025b. Os-atlas: Foundation action model for generalist gui agents. In *International Conference on Learning Representations (ICLR)*.
- Tianbao Xie, Jiaqi Deng, Xiaochuan Li, Junlin Yang, Haoyuan Wu, Jixuan Chen, Wenjing Hu, Xinyuan Wang, Yuhui Xu, Zekun Wang, Yiheng Xu, Junli Wang, Doyen Sahoo, Tao Yu, and Caiming Xiong. 2025. Scaling computer-use grounding via user interface decomposition and synthesis. In *Advances in Neural Information Processing Systems 38 Datasets and Benchmarks Track (NeurIPS)*.
- 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. 2024. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 37, pages 52040–52094.
- Fangzhi Xu, Hang Yan, Qiushi Sun, Jinyang Wu, Zixian Huang, Muye Huang, Jingyang Gong, Zichen Ding, Kanzhi Cheng, Yian Wang, Xinyu Che, Zeyi Sun, Jian Zhang, Zhangyue Yin, Haoran Luo, Xuanjing Huang, Ben Kao, Jun Liu, and Qika Lin. 2026.

Odysseyarena: Benchmarking large language models for long-horizon, active and inductive interactions. *arXiv preprint arXiv:2602.05843*.

Yiheng Xu, Zekun Wang, Junli Wang, Dunjie Lu, Tianbao Xie, Amrita Saha, Doyen Sahoo, Tao Yu, and Caiming Xiong. 2024. Aguis: Unified pure vision agents for autonomous gui interaction. <https://arxiv.org/abs/2412.04454>.

Yan Yang, Dongxu Li, Yutong Dai, Yuhao Yang, Ziyang Luo, Zirui Zhao, Zhiyuan Hu, Junzhe Huang, Amrita Saha, Zeyuan Chen, Ran Xu, Liyuan Pan, Silvio Savarese, Caiming Xiong, and Junnan Li. 2025. Gta1: Gui test-time scaling agent. *arXiv preprint arXiv:2507.05791*.

Chengyue Yu, Siyuan Lu, Chenyi Zhuang, Dong Wang, Qintong Wu, Zongyue Li, Runsheng Gan, Chunfeng Wang, Siqu Hou, Gaochi Huang, Wenlong Yan, Lifeng Hong, Aohui Xue, Yanfeng Wang, Jinjie Gu, David Tsai, and Tao Lin. 2025. Aworld: Orchestrating the training recipe for agentic ai. *arXiv preprint arXiv:2508.20404*.

Yichun Zhang, Xiangwu Guo, Yauhong Goh, Jessica Hu, Zhiheng Chen, Xin Wang, Difei Gao, and Mike Zheng Shou. 2026. Showui-aloha: Human-taught gui agent. *arXiv preprint arXiv:2601.07181*.

Longtao Zheng, Rundong Wang, Xinrun Wang, and Bo An. 2023. Synapse: Trajectory-as-exemplar prompting with memory for computer control. In *The Twelfth International Conference on Learning Representations (ICLR)*.

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. 2024. [WebArena: A realistic web environment for building autonomous agents](#). In *The Twelfth International Conference on Learning Representations (ICLR)*.

A Appendix

A.1 More Analysis

A.1.1 Task-level Success Rate Result

We run our experiments on a local VMWare virtual machine. For the open weights models, we host them using the vLLM (Kwon et al., 2023) framework.

Agent	Task-level SR
OpenCUA7B	0.00%
UITARS7B	0.00%
AgentS2.5 w/ GPT5	2.78%
UiPath w/ Opus 4.5	0.56%
GPT5.4-Mini	0.00%
GPT5.4-Nano	0.00%
o3	0.00%
GPT5.4 Native Computer Use	11.53%
Human	82.62%

Table 3: End-to-end task success rate for baseline agents. Agent colours follow the four categories in Sec. 4.1.

Tab. 3 reports the end-to-end task-level success rate of all baselines. Finishing a vast-horizon, repetitive task end-to-end is highly challenging for current agents: only the strongest native computer-use agent (GPT-5.4 Native Computer Use) achieves a non-trivial success rate (11.53%), while every other baseline falls below 3%. Human evaluators, in contrast, complete 82.62% of tasks, indicating that the gap reflects agent capability limits rather than task infeasibility. The low task-level success rate confirms that current agents struggle on all three capabilities required for a vast-horizon, repetitive task (Fig. 1, right): high-level planning, consistent iterative execution, and the implicit closing actions. Failure at any one collapses end-to-end success.

A.1.2 Analysis Across Difficulty Tiers

Tab. 4 reports baseline performance on the two tiers. A clear difficulty gap is visible across all three metrics: every baseline achieves higher field-level, subtask-level, and task-level SR on OS-Marathon-Lite than on OS-Marathon-Pro, confirming that the difficulty partition produces a meaningful performance distinction. The strongest agents can occasionally complete OS-Marathon-Lite tasks end-to-end, i.e. AgentS2.5, UiPath, and GPT5.4 NCU reach task-level SR of 5.56%, 1.11%, and 16.22%, but drop sharply on OS-Marathon-Pro (0%, 0%, and 6.83%). Weaker agents fail to complete tasks end-to-end on either tier, yet the difficulty gap re-

mains visible at the field and subtask levels (e.g., OpenCUA7B: 10.18% $\rightarrow$ 3.13% field-level).

A.1.3 Analysis on Subtask/Task Levels

We elaborate on **weak global planning and persistence**, identifying three common issues across baselines. First, **agents lack an ordered processing strategy**: rather than committing to a deterministic order (e.g., file-listing order or chronological by receipt date), they pick the next instance opportunistically, making it hard to verify exhaustive coverage. Second, **agents terminate prematurely**, declaring the task complete before all N instances are processed. As shown in Fig. 8, GPT5.4 Native Computer Use is required to fill the insurance form with the supporting files provided. There are five supporting files to fill out, but the agent only managed to add two out of five to the form, and it proceeded to email the form. This failure often compounds with visual-salience bias. For example, in an expense report case, after the agent submits a single invoice, the system displays a confirmation message such as *"Invoice submitted successfully"*, which the agent may misread as confirmation that the entire workflow is complete. Third, **implicit post-subtask actions that fall outside any single subtask, such as filling out the category-wise expense summary, are frequently skipped, especially when the environment imposes them implicitly**: the agent must discover them through exploration, recognise them as task requirements, and incorporate them into its plan. Together, these issues compound across the vast horizon, exposing why such tasks are challenging: completing them coherently requires global planning and persistence that current agents lack.

A.1.4 Analysis on Different Workflow Types

The **light yellow** columns of Tab. 1 group the results by workflow type, revealing a clear two-tier pattern: web and obsidian workflows are more manageable for most agents, whereas spreadsheet and document workflows pose a substantially larger challenge. Besides the analysis in Sec. 4.2, we further analyse the observation of agents' performance on obsidian tasks. The obsidian workflow asks the agent to organise paper-reading notes following a provided example: creating markdown notes in the appropriate folders, moving PDFs into matching directories, and structuring annotations in the example's format. Performance here varies with the granularity of the operations the agent must per-

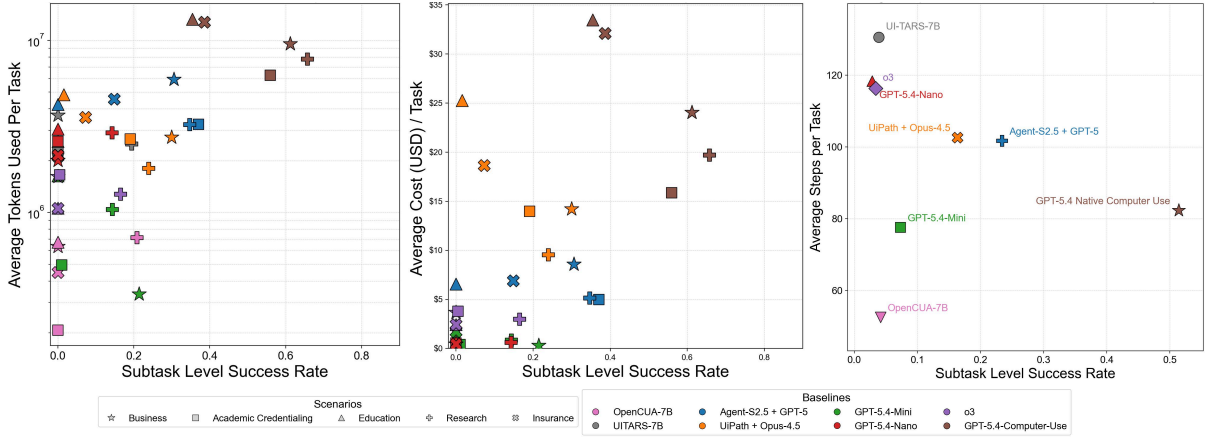


Figure 7: **Cost, efficiency and performance relationship across baseline agents on OS-Marathon.** Subtask-level success rate plotted against token usage (left), monetary cost in USD \$ (middle), and step count (right).

Agents	OS-Marathon-Lite			OS-Marathon-Pro		
	Field-level SR	Subtask-level SR	Task-level SR	Field-level SR	Subtask-level SR	Task-level SR
OpenCUA7B	10.18%	07.00%	00.00%	03.13%	01.33%	00.00%
UITARS7B	06.24%	06.44%	00.00%	00.77%	01.33%	00.00%
AgentS2.5 w/ GPT5	36.07%	26.33%	05.56%	30.38%	20.55%	00.00%
UiPath w/ Opus 4.5	34.06%	20.58%	01.11%	29.74%	12.15%	00.00%
GPT5.4-Mini	11.10%	08.33%	00.00%	08.70%	06.38%	00.00%
GPT5.4-Nano	04.16%	03.12%	00.00%	03.15%	02.59%	00.00%
o3	05.45%	03.68%	00.00%	04.24%	03.12%	00.00%
GPT5.4 NCU	70.12%	54.85%	16.22%	66.46%	47.93%	06.83%

Table 4: **Baseline performance comparison between the two OS-Marathon difficulty tiers.** We report the field-level, subtask-level, and task-level success rates (SR) on OS-Marathon-Lite and OS-Marathon-Pro. Agent colours follow the four categories in Sec. 4.1.

form. Agents reliably handle the explicit, OS-level structure, i.e. moving PDFs into the correct folders, creating notes under the matching directory layout. These operations are within reach even for weaker models such as OpenCUA-7B and o3, which is why those models score higher on this workflow than on others. However, they struggle with the finer content-level conventions the example demonstrates, such as the formatting of annotations within markdown notes and the backlinks between related notes.

A.2 Agent Personalisation Details

A.2.1 Orchestrator Framework

We adapt the orchestrator idea from (Wang et al.); the framework is illustrated in Fig. 9. The framework adopts multiple agents for distinct roles, one per phase of the three-phase task structure defined in Sec. 3.1: A **task orchestrator** **O** handles the pre-subtask phase: it explores the environment, parses the user intention, fills in the shared meta-information (e.g., the employee’s name and ID before starting an expense report), and decomposes

the workflow into per-instance subtask instructions. A set of **solver agents** $\{S_i\}_{i=1}^N$ executes the repetitive subtask phase. The framework iterates over the N data instances in a for-loop, initialising a solver agent with the corresponding subtask instruction at each iteration and moving on once the agent reports subtask completion. A **post-processor** **P** handles the post-subtask phase once all subtasks are completed. This framework theoretically decomposes a vast-horizon, repetitive task into shorter, more tractable subtasks. All three roles, i.e. orchestrator **O**, solver S_i , and post-processor **P**, are instantiated as separate AgentS2.5 + GPT-5 agents.

Adapt GraphDemo to the orchestrator framework. The graph-based demonstration is partitioned into three sub-graphs along the orchestrator’s three-phase decomposition, one for the task orchestrator **O**, one for the solver agents S_i , and one for the post-processor **P**, so each component receives only the demonstration content relevant to its role. Each subagent is initialised with the text-level abstraction of its own sub-graph in the system

Task instruction: Could you file the insurance claim form for me with the documents stored in the insurance folder on the Desktop? And submit the claim through the email to the insurance company? Refer to the `currency_conversion_rates_to_usd` file if you need the conversion rate.

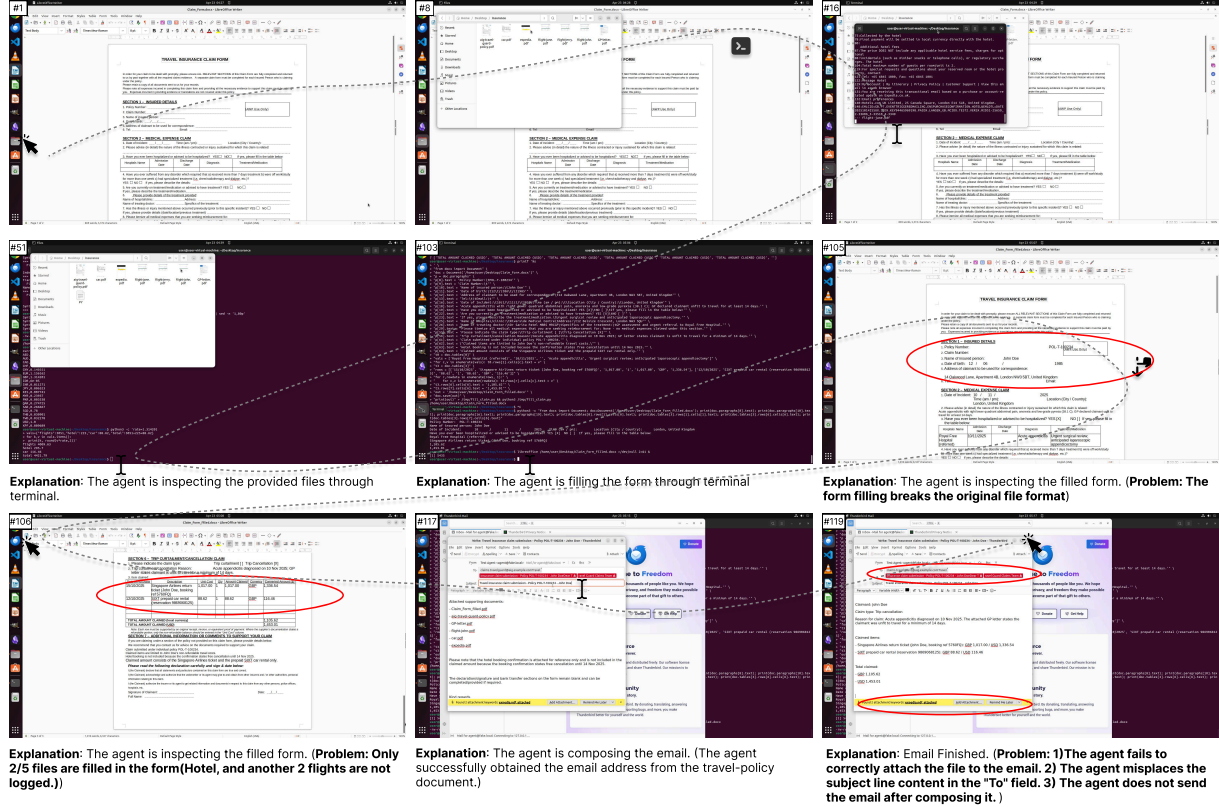


Figure 8: **Failure case analysis: GPT5.4 Native Computer Use (high reasoning).** The agent only processes two out of five files (subtasks) and terminates prematurely (#106). In the post-subtask phase, GPT5.4 attempts the closing actions but fails on three implicit requirements: it does not attach the relevant files, which is implied implicitly, misplaces the subject content in the *To* field, and does not send the email after composing (#119).

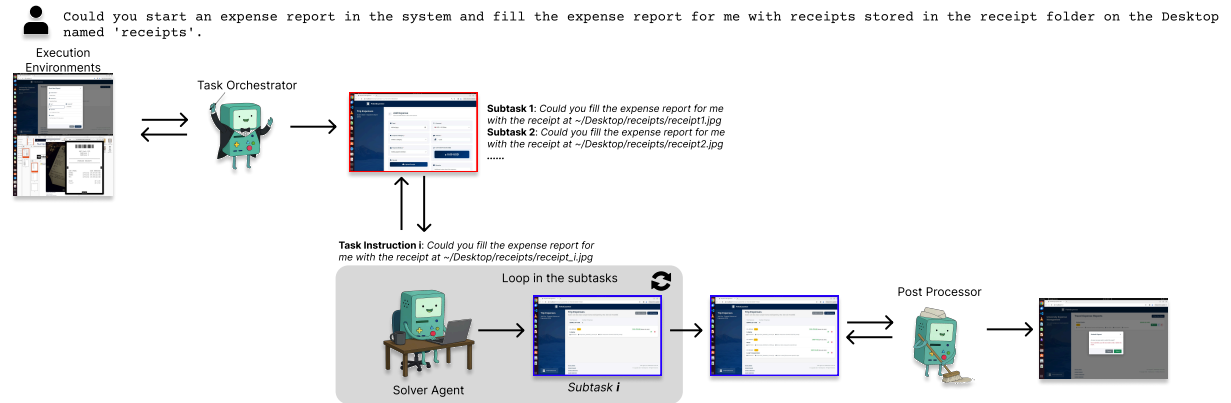


Figure 9: **Orchestrator framework.** The orchestrator handles the pre-subtask phase, a set of solver agents executes the repetitive phase, and the post-processor handles the post-subtask phase.

prompt. The rest of the demonstration segment retrieval, context embedding stays the same.

A.2.2 More details about GraphDemo

Implementation details. We further introduce the implementation details of GraphDemo. Nodes are organised hierarchically: root nodes capture higher-level semantic stages (e.g., `navigate_to_the_folder`), while

child nodes record the step-by-step trajectory within each stage (e.g. `navigate_to_Desktop`, `navigate_to_receipts_folder`, etc.). Directed edges carry the **action** executed between states. Each leaf node additionally stores **action_reason** and **action_goal** for context of the current action. This hierarchical organisation groups the demonstration into coherent semantic units,

Agents	Field-Level SR	Subtask-Level SR
Agent S2.5 + GPT5	33.90	25.31
Agent S2.5 + GPT5 + GraphDemo	50.95	41.24
Agent S2.5 + GPT5 + Orchestrator	29.07	16.01
Agent S2.5 + GPT5 + Orchestrator + GraphDemo	40.78	27.40
GPT5.4-Mini	09.90	07.35
GPT5.4-Mini + GraphDemo	27.34	18.66

Table 5: **GraphDemo provides consistent gains across different agent configurations.** Applied to three different setups (AgentS2.5 + GPT-5, AgentS2.5 + GPT-5 + Orchestrator, and GPT5.4-mini), GraphDemo improves both field-level and subtask-level success rates in every setting, disentangling the method’s contribution from the underlying agent’s capacity. Each cell reports the success rate in percentage (%).

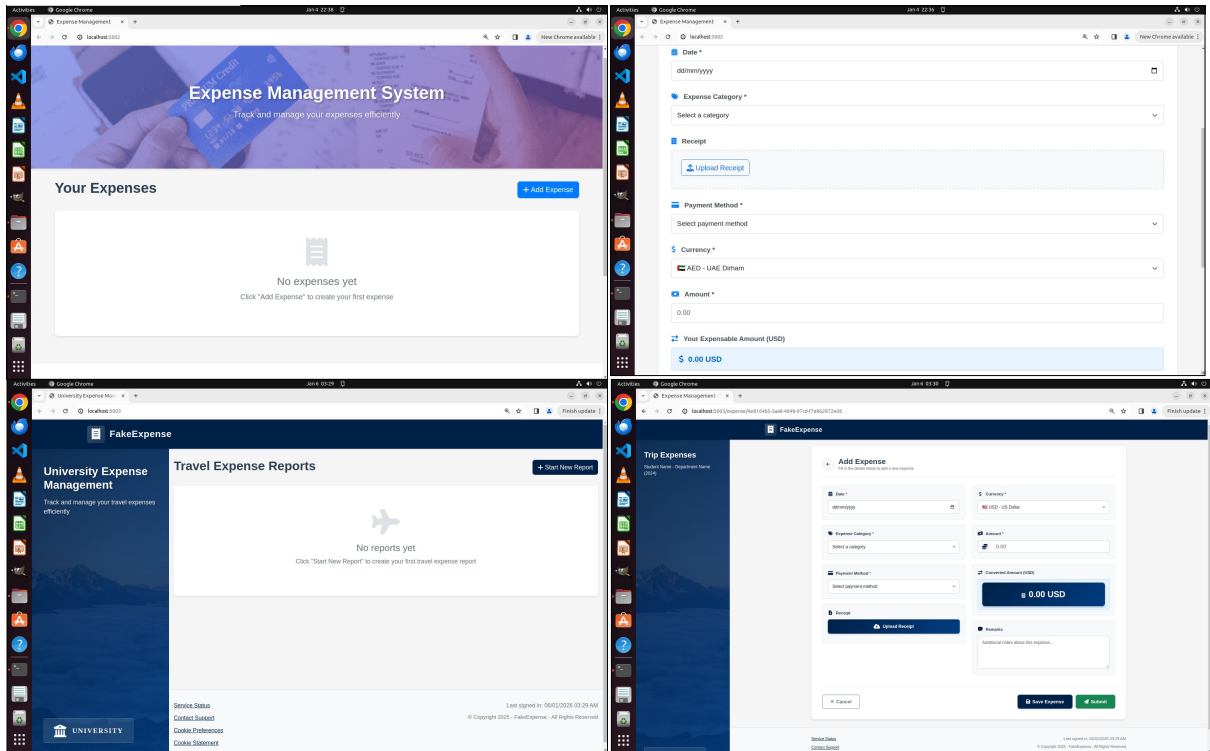
enabling more efficient retrieval.

More quantitative analysis of GraphDemo. To disentangle GraphDemo’s benefit from agent capacity, we extend Tab. 2 with results on the weaker GPT5.4-Mini. Tab. 5 reports field-level and subtask-level success rates for each configuration with and without GraphDemo. Across all three settings, GraphDemo consistently improves the performance, indicating that the gains do not come from any single agent’s underlying capacity.

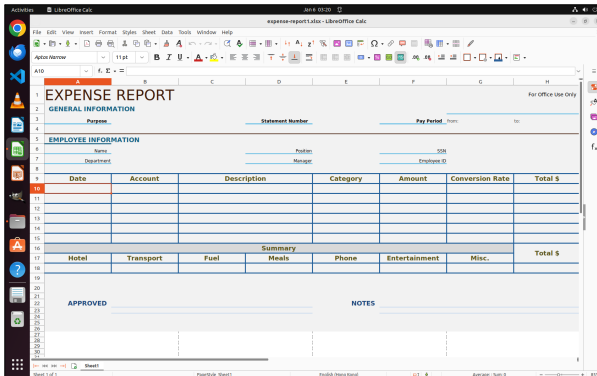
A.3 More Details About OS-Marathon

Execution Environment Examples. Figs. 10 to 13 visualise the all the execution environments included in OS-Marathon.

Business



Spreadsheet



Note Taking Following a Given Example

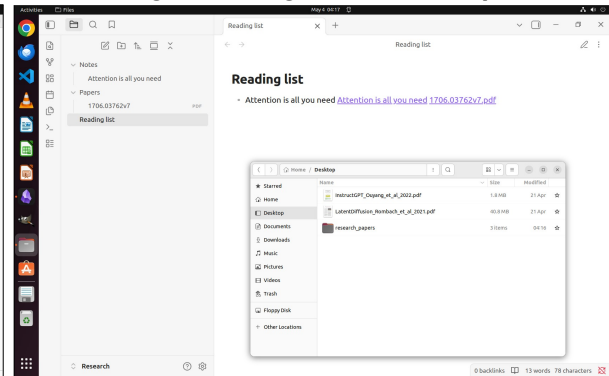


Figure 10: Examples of the execution environment in the business and note taking scenarios.

Academic Credentialing

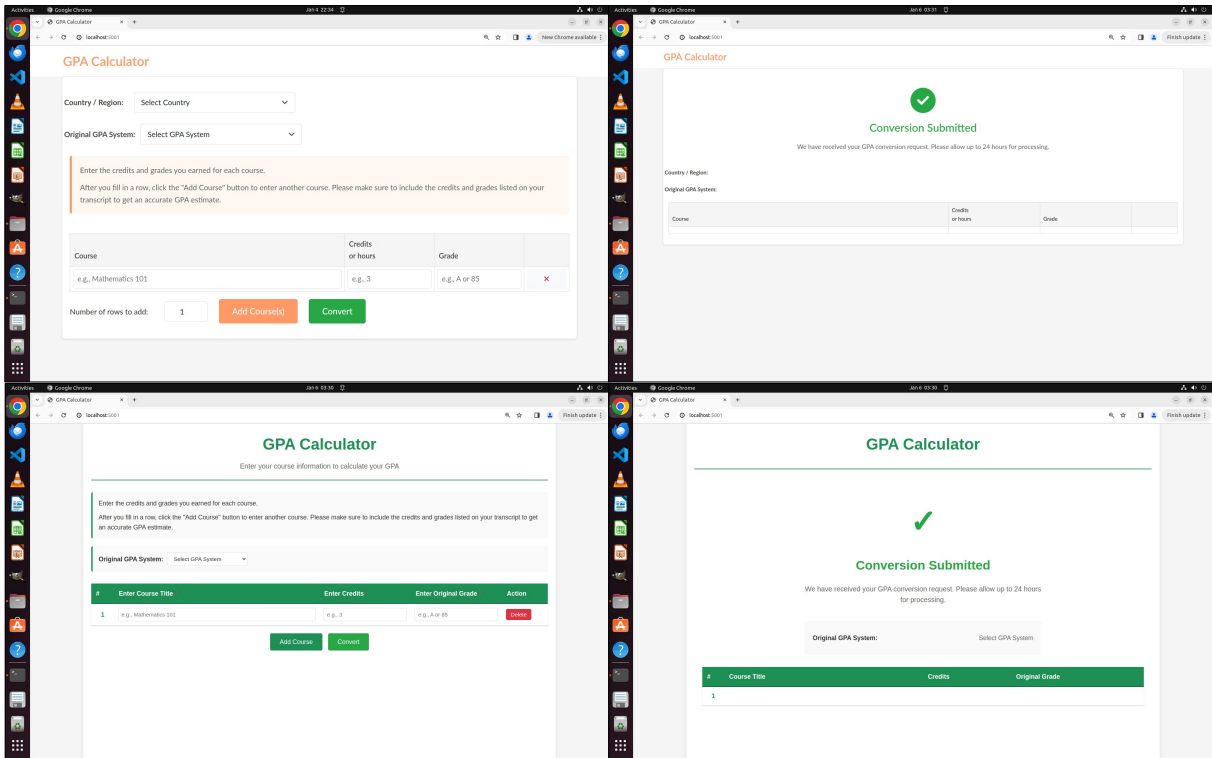


Figure 11: Examples of the execution environment in the academic credentialing scenario.

Education (Mark Exam / Analyse Exam Results)

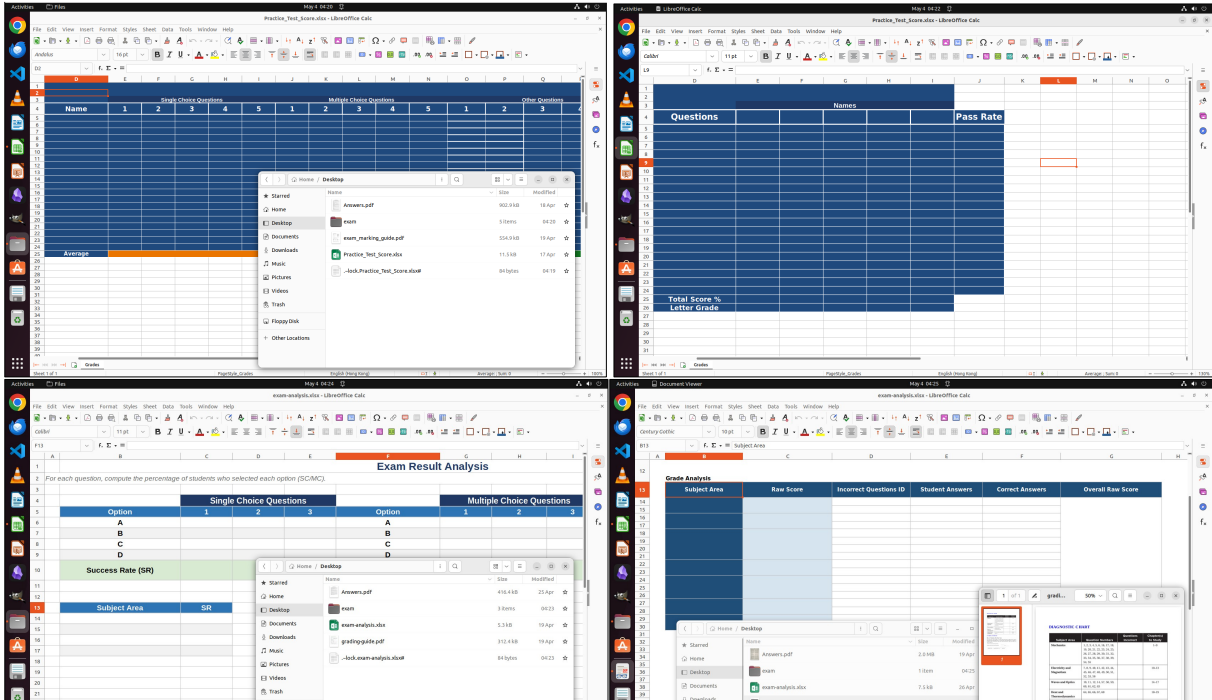


Figure 12: Examples of the execution environment in the education scenario.

56622