RUNNINGTAB: DIRECT WORKSPACE INTERACTION WITH ENVIRONMENT-SIDE TABS

Anonymous authors
Paper under double-blind review

ABSTRACT

Much knowledge work produces new deliverables from files a workspace already holds, and LLM agents are beginning to take such work over. Through direct corpus interaction, an agent can search and read any of those files from a terminal with no indexing, and producing a deliverable from many of them in this way is what we call *direct workspace interaction* (DWI). Reaching the files, however, is only half the task: nothing keeps track of what the task asks for, what has been read, and what was listed but never opened, all of which slip through the context window without leaving a trace, so an agent may extract a figure and still deliver a report without it. To address this, we present RunningTab, a framework that equips direct workspace interaction with an *environment-side tab*: a per-task record of what the task still owes, kept by the environment alongside the agent. Specifically, the agent adds its requirements, while the environment records every file read as an excerpt with its provenance and every listed but unopened file as a candidate; the agent can then see each requirement beside its best-matching excerpts and top unopened candidates, resolve it against matching content or set it aside with a reason, and, should it try to finish with requirements still open, receive them in a finish check. We validate RunningTab on three benchmarks with three LLMs, where it consistently outperforms plain DWI and baselines that keep the record in the model, while its tab usually holds the values a deliverable needs once seen.

1 INTRODUCTION

Much of the work in an organization consists of producing new deliverables from files it already holds (the reports, spreadsheets, slide decks, and notes accumulated over years in a shared workspace) (Xu et al., 2025a; Patwardhan et al., 2025; Tang et al., 2026): a hiring summary is compiled from resumes and interview notes; a budget review is built from a year of monthly spreadsheets; a project proposal is drafted from meeting notes and a previous contract. Large Language Model (LLM) agents that operate a computer autonomously (Yao et al., 2023; Schick et al., 2023; Yang et al., 2024; Merrill et al., 2026) make it possible to hand such work over end to end, so that the agent itself locates the relevant files in the workspace, reads them, and writes the deliverable. A workspace task, then, is not a single retrieval but an accumulation over a long horizon: the deliverable brings together content scattered across many files, and the agent must reach each of them before it can write.

To reach the existing files, *direct corpus interaction* (Li et al., 2026b) opens the road: instead of a retriever over a prebuilt index (Karpukhin et al., 2020; Lewis et al., 2020), the agent works in a terminal and applies standard command-line utilities (listing a directory, searching for a term across files, opening a document) directly to the raw files, so that any file in the workspace can be reached as it is, with no indexing or preprocessing. Subsequent work scales it to larger corpora (Lu et al., 2026), synthesizes tasks for it (Salemi et al., 2026), and studies how agents search under it (Sen et al., 2026; Subramanian et al., 2026). However, when the corpus is such a workspace and the goal is not a short answer to a question but a new deliverable assembled from many of its files, the agent has to work inside the workspace over many steps and hand in the deliverable at the end; we refer to this setting as *direct workspace interaction* (DWI). Yet reaching the existing files is only the first half of the task.

Specifically, the agent has full access to the workspace, but nothing keeps track of the task it is carrying out. What the task asks for, which files have been read, and which were listed but never opened pass through the context window once and are soon displaced by new observations; nothing

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.

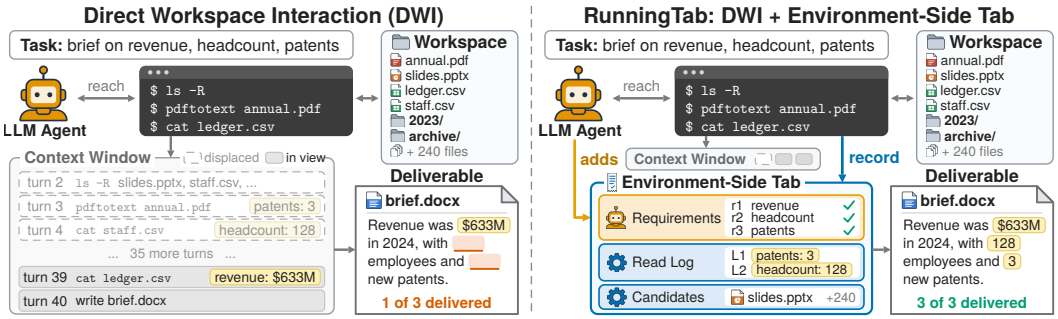


Figure 1: Direct workspace interaction (DWI) and RunningTab on an illustrative task: a brief that needs three facts scattered across a workspace. (Left) In DWI, the agent reaches every file through its terminal, but what it reads lives only in its context window: facts read early are displaced by later observations, and the brief is delivered without them. (Right) RunningTab keeps an environment-side tab beside the agent: the agent adds the requirements of the task, the environment records what is read and what is listed but not opened, and the facts remain available until the deliverable is written.

records which of them have been used, and nothing connects a requirement of the task to the file that satisfies it. As a consequence, an agent may open the right annual report, extract the exact figure it needs into its observations, and still deliver a report without that figure; or it may list a folder that contains the relevant file and never open it. Therefore, what is missing, we argue, is a record of what the task still owes, kept by the environment and available to the agent whenever it needs it.

In this work, we present **RunningTab**, a framework that equips direct workspace interaction with an *environment-side tab*: a per-task running record that the environment maintains for the agent (Figure 1). Specifically, the agent adds one *requirement* per fact, figure, or element the task asks for, while the environment records every file read as an excerpt with its provenance and every file listed but not opened as a *candidate*, further ranked by its relevance to the task. Because the tab keeps what would otherwise have passed out of view, the agent can consult it freely, returning to a stored excerpt or a listed file whenever it needs them instead of searching the workspace again. Furthermore, before resolving any requirement, the agent lists them and sees each open one beside its best-matching excerpts and its top unopened candidates, and a requirement is resolved only against an excerpt whose content matches it, or with a reason it cannot be met; should the agent attempt to finish with items open, the environment returns them once, in a *finish check*. The tab thus runs alongside the task, from the first requirement the agent writes to the finish check, holding what the model would otherwise have to retain and pointing back to the files it came from.

We validate RunningTab on three benchmarks (Tang et al., 2026; Xu et al., 2025a; Opsahl-Ong et al., 2026) with three LLMs, where it improves over plain DWI and over baselines in which the model itself keeps track of the task, in every case. Behind this improvement, ablations attribute most of the performance gain to what the environment captures on its own, the tab usually already holds the values a deliverable needs once the agent has seen them, and the gain grows with how much the agent reads. These results affirm that direct workspace interaction needs more than access to the files: the terminal reaches them, and the environment-side tab keeps what the task owes until it is delivered.

2 Related Work

Workspace Agents LLM agents that operate a computer have shown strong performance as software developers and terminal users (Yang et al., 2024; Wang et al., 2025) and are evaluated in terminal and desktop environments (Merrill et al., 2026; Xie et al., 2024). Building on these agents, recent benchmarks bring them to knowledge work: they pose tasks over realistic office and enterprise workspaces and judge the deliverable the agent produces (Xu et al., 2025a; Patwardhan et al., 2025; Tang et al., 2026; Wang et al., 2024; Opsahl-Ong et al., 2026; Chen et al., 2026). However, these studies find that the deliverables often lack content the workspace holds: Xu et al. (2025a) attribute failures in part to a lack of ability to understand documents, and Tang et al. (2026) name heterogeneous file understanding and lineage tracing as the bottlenecks. These diagnoses concern reaching and reading the files, whereas our own analysis (discussed in Section 3.1 with Figure 2) takes a different lens and finds a loss that these diagnoses do not cover: content from files the agent had already opened, which appeared in its observations yet is missing from the deliverable. In this

work, we address this part directly, by keeping beside the agent a record of what the task still owes and of what has been read, so that content once reached is not lost before delivery.

Direct Corpus Interaction Retrieval over a prebuilt index has long been the way a model reaches a corpus beyond its context (Karpukhin et al., 2020; Lewis et al., 2020), first as a single retrieval step and then, in agentic search, as repeated queries to the same retriever over many rounds (Jin et al., 2025; Li et al., 2025). Direct corpus interaction departs from this design: rather than a retriever that decides what the agent sees, the agent is given the corpus itself and searches it with terminal tools (Li et al., 2026b). In particular, Dr-DCI (Lu et al., 2026) lets the agent pull retrieved documents into a local workspace (where a workspace denotes the materialized set of pulled documents, whereas here it denotes the files an organization holds), GrepSeek (Salemi et al., 2026) trains compact agents for it, and related work compares terminal or keyword search with vector retrieval (Sen et al., 2026; Subramanian et al., 2026) or builds on it (Zhuang et al., 2026; Li et al., 2026a). However, these works typically target a short answer to a question and improve how an agent reaches the corpus; to our knowledge, none of them records which file meets which requirement of the task. Unlike these works, which improve how an agent reaches the files, RunningTab turns to what becomes of the content once it is reached: it changes nothing about how the agent reaches the files and records, beside the agent, what the task asks for and which opened file each item was closed against.

Agent Memory Memory for LLM agents is typically written by the model itself and kept across sessions or trials (Zhang et al., 2025): content is moved out of the context window into external storage and brought back when needed (Packer et al., 2023), or the agent records its experiences and reflections and retrieves them later (Park et al., 2023; Shinn et al., 2023; Xu et al., 2025b). Beyond memory that outlives a task, the model likewise keeps its own state within a single task: a plan written into its context (Wang et al., 2023), notes it writes on what it reads (Lanchantin et al., 2023), a compact internal state it rewrites at each turn (Zhou et al., 2025), or, in multi-agent systems such as Magentic-One (Fourney et al., 2024), a plan and a progress record that an orchestrator LLM writes and judges. In recent work, such a record is kept for the agent and updated as the task proceeds, by the model itself or by further models that supervise it (Ko et al., 2026; Wu et al., 2026; Ma et al., 2026). However, wherever these systems keep a record of the task, it is kept by a model, which writes it and marks its items complete. In contrast, in RunningTab, the tab is kept by the environment: the environment writes it from what the agent read and listed (the agent adds only its requirements), its items are marked done only against a captured excerpt or set aside with a reason, and, kept outside the context window, it is not displaced and is returned to the agent once, when it tries to finish.

3 METHOD

In this section, we first formalize direct workspace interaction, a setting in which the agent is given access to every file of a workspace but no persistent record of the state of its task, and then present RunningTab, which supplies this record in the form of an environment-side tab kept beside the agent.

3.1 DIRECT WORKSPACE INTERACTION

Workspace Tasks Let $\mathcal{W}$ denote a workspace, namely the collection of files an organization already holds (such as reports, spreadsheets, slide decks, and notes), and let t denote a task that asks for a new deliverable $\mathcal{D}$ (for instance, a report or a spreadsheet) to be produced from them. What characterizes such a task is that what it asks for rarely sits in one place: t decomposes into many individual elements (a fact to look up, a figure to extract, a section to write) whose sources are scattered across the files of $\mathcal{W}$, and the deliverable is complete only when every one of them has been carried into $\mathcal{D}$. The task is carried out by an agent, an LLM typically equipped with terminal tools (such as `read`, which opens a file, and `bash`, which runs a shell command), which works in turns: in each turn, the model issues tool calls and receives their observations, writing $\mathcal{D}$ into an output folder of $\mathcal{W}$ along the way, and it finishes when it responds without a further tool call, which we refer to as a stop.

Direct Workspace Interaction To reach the files, the agent follows direct corpus interaction (Li et al., 2026b): instead of querying a retriever over a prebuilt index, it applies its terminal tools directly to the raw files, so that any file in $\mathcal{W}$ can be listed, searched, and opened as it is. We then define *direct workspace interaction* (DWI) as the setting in which a workspace task is carried out through this form of access: given $\mathcal{W}$ and t, the agent interacts directly with the files of $\mathcal{W}$ over many turns and hands in $\mathcal{D}$ when it finishes. DWI thus inherits from direct corpus interaction how files are reached,

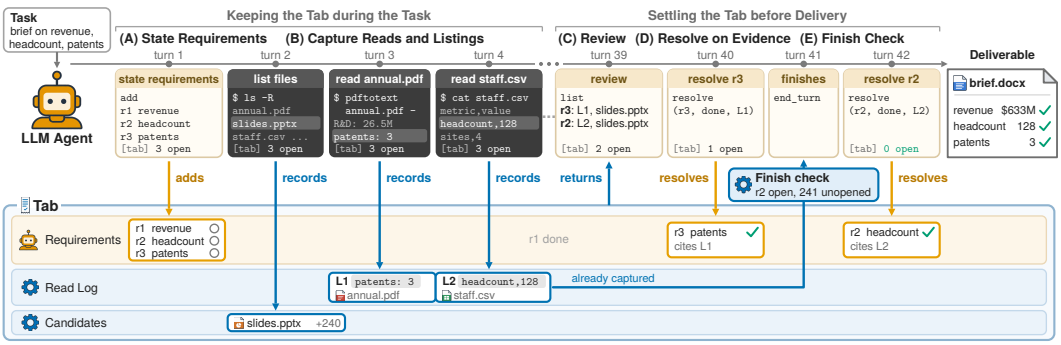


Figure 3: RunningTab over one task. The agent (top) works in turns, and the environment-side tab (bottom) fills under the turn that wrote each entry. (A) The agent states the requirements. (B) As the agent lists and reads files, the environment records each listed file as a candidate and each read as a read log entry with its anchor. (C) The review shows each open requirement beside its best read log match and the unopened files. (D) The agent resolves a requirement by citing the read log entry that holds its content. (E) At the stop, the environment returns one finish check with what is still open; here the tab had already captured the missing fact, so the agent closes it and delivers.

but differs from it in what is asked of the agent: there, the output is a short answer that a few relevant passages typically suffice to support; here, it is a deliverable that brings together content from many files, so that whatever the agent finds along the way should remain available until it is written into $\mathcal{D}$.

The Missing Record of the Task In its plain form, however, DWI leaves this entirely to the context window, where whatever the agent has found survives only as a passage somewhere in an ever-growing stream of observations. To measure how often content shown to the agent is missing from the deliverable, we analyze DCI (Li et al., 2026b), the plain form of DWI, with GPT-5.4 nano (OpenAI, 2026) on Workspace-Bench (Tang et al., 2026), which checks each deliverable against rubrics, over three runs of its 100 tasks. Specifically, we assign each rubric that DCI fails to a failure type with LLM annotators (whose labels agree with human labels at a Cohen's kappa of 0.83), and we check whether each numeric value a rubric expects appears in the observations of the agent and in its deliverable.

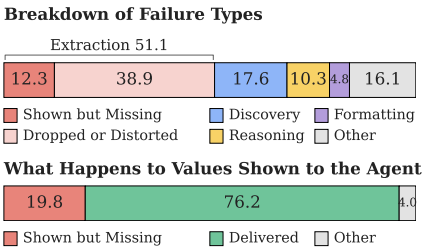


Figure 2: DCI with GPT-5.4 nano on Workspace-Bench, as shares (%). (Top) Lost points of the rubric pass rate by failure type. (Bottom) Rubric values shown to the agent, by outcome.

As shown in Figure 2 (top), extraction, in which the agent opened the right files but the deliverable lacks or distorts their content, accounts for 51.1% of the points that DCI loses, and it splits into two parts: Shown but Missing (12.3%), where the content is absent from the deliverable although it appears in the output of a file the agent had opened, and Dropped or Distorted (38.9%), where the content is either absent with no such evidence or present but inexact, misplaced, or misread. Beyond extraction, discovery, in which the right files are never opened (including files the agent listed but did not open), accounts for another 17.6%. Moreover, among the attempts that produced a deliverable, about one in five (19.8%) of the 891 distinctive numeric values that the rubrics expect and that appear in an observation of the agent are absent from the deliverable, and the rubrics that expect them are not satisfied (Figure 2, bottom). By contrast, a record of the state of the task, kept outside the context window, offers what the stream cannot: at any point, it shows which elements of t are still unmet, which observations have already been used in $\mathcal{D}$, and which listed files have yet to be opened.

Motivated by this, RunningTab is built around an object that keeps such a record, the *environment-side tab* (Figure 3), a per-task record of what the task still owes, maintained not by the model but by the environment (the loop that runs the model and its tool calls). Next, we define the tab (Section 3.2) and describe how it is kept during the task (Section 3.3) and settled before delivery (Section 3.4).

3.2 Environment-Side Tab

We now turn to the environment-side tab (Figure 3), the central component of RunningTab, and describe what it consists of, how the environment maintains it, and how the agent accesses it.

Composition of the Tab Recall that a deliverable is assembled from content scattered across many files, often long after that content was first encountered (Section 3.1). Consequently, by the time the agent writes, it should rely on the following (remaining accessible): what the task asks for (so that no element is left out), what it has already read (so that content found earlier can be reused without searching for it again), and which files it has seen in a listing but not yet opened (so that it knows where to look for what is still missing). To this end, the proposed environment-side tab maintains one kind of entry for each of them. Formally, for a task t, the tab is defined as $\mathcal{T}_t = (\mathcal{R}, \mathcal{L}, \mathcal{C})$, which consists of requirements ($\mathcal{R}$), a read log ($\mathcal{L}$), and candidates ($\mathcal{C}$), described as follows:

- **Requirements ($\mathcal{R}$).** The requirements are the individual items that the task asks for, each being one fact, figure, or element of the deliverable. For instance, for a task that asks for a short report with several figures from an annual report, one requirement would be the number of new invention patents in 2024. At the start of the task, the agent states these items, and the environment records each of them as a requirement $r_i \in \mathcal{R}$ and tracks its status from then on. Specifically, a requirement is `open` when it is recorded, and the agent could later resolve it in one of two ways: as `done`, when its content has been found in a file, in which case the agent cites the entry of the read log that contains it; or as `unavailable`, when no file supplies it (for instance, because it is to be computed or written by the agent), in which case the agent states the reason.
- **Read log ($\mathcal{L}$).** The read log keeps a copy of what the agent has read. Specifically, whenever the agent reads a file (for instance, by opening it or by extracting its text with a command), the environment records an entry $\ell_j \in \mathcal{L}$, which consists of an excerpt (the content exactly as shown to the model) and an anchor (its provenance: the path of the file and the command that produced it). In the example above, extracting the text of the annual report yields an entry whose excerpt contains the line with the patent count, so this line can be found again without reopening the report.
- **Candidates ($\mathcal{C}$).** The candidates are the files that the agent has come across but not yet looked into. Specifically, whenever a file appears in a listing (for instance, in the output of a command that lists a folder or searches for file names) without having been opened, the environment records it in $\mathcal{C} \subseteq \mathcal{W}$, and removes it once the agent opens it. In the same example, if the folder of the annual report also contains a presentation of the same year that the agent has not opened, this presentation remains among the candidates, as a place where a missing figure may still be found.

Note that it is the environment, rather than the model itself, that keeps and maintains every entry of the tab: it stores the requirements as the agent states them, and it captures the read log and the candidates on its own, since it observes every read and every listing, whereas a record that the model keeps for itself may miss some of these reads and listings. Moreover, the tab is stored outside the context window, so that its entries are not displaced by new observations, and outside the workspace, so that it is not mixed with the files the agent lists and reads or with the deliverable. In other words, being kept by the environment and stored outside the context window is what distinguishes the tab; a record that the model keeps on its own within its context has neither of these two properties.

Operations on the Tab Since the tab is stored outside the context window, the agent accesses it through four operations. Specifically, two of them update the requirements: `add` states one or more requirements, and `resolve` resolves a requirement as `done` or `unavailable`. The other two read the tab: `list` lists the requirements with their status, the candidates, or the entries of the read log, and `show` returns a single entry of the read log (its excerpt together with its anchor), so that content read earlier can be consulted again without reopening the file. We describe when and how these operations are used concretely in the following subsections.

3.3 Keeping the Tab during the Task

We now describe how the tab is kept while the agent works. Since the tab is useful at a given turn only if it already holds what the agent needs then, each kind of entry is recorded as it arises, by the party in a position to record it: the agent states the requirements at the start, while the environment captures reads and listings from the observations and reports the state of the tab alongside them.

Stating the Requirements Among the three kinds of entries, the requirements are the one the environment cannot capture on its own, since what the task asks for is written only in the task, and dividing it into items takes an understanding of the task that the model supplies. RunningTab therefore adds a short block to the system prompt of the agent, which asks it to begin by stating the requirements with `add` and then to work as usual.

Capturing Reads and Listings In contrast, the read log and the candidates need no action from the agent: the environment sees every observation, so it records both on its own. For each observation it decides which files the agent has read and which it has only seen listed. However, the terminal itself draws no such line: many commands read a file, and a single command can show the content of one file together with the names of many others. RunningTab therefore draws this line from the command and its observation: a file counts as read when its content appears, and as listed when only its path does. Notably, an entry of the read log stores the observation as it is, neither parsed nor summarized, so that capturing depends on no file format and the entry holds what the agent saw. Also, since one listing can name hundreds of files while a candidate carries nothing but its path, the candidates are ranked by the lexical relevance of their paths to the task with BM25 (Robertson & Zaragoza, 2009).

Reporting the State of the Tab Storing the tab outside the context window keeps its entries from being displaced, but the agent then sees none of them unless it calls an operation. The environment therefore appends to each observation a status line carrying the state of the tab: which requirements remain `open`, how much the read log and the candidates hold, and how long since its last operation.

3.4 Settling the Tab before Delivery

We now describe how the tab is settled before the deliverable is handed in. A requirement stays `open` until the agent resolves it, so a settled tab is one whose requirements have all been resolved. RunningTab works toward that state with three mechanisms: a review that shows the agent what may supply each open requirement; a resolution that names where the content of a requirement was found, or why no file supplies it; and a finish check sent at the stop, whatever the tab then shows.

Reviewing the Requirements Connecting a requirement to the file that supplies it is a link the context window leaves implicit, whereas the tab holds both ends of it explicitly: the requirements on one side, and the read log and the candidates on the other. RunningTab therefore gives the agent a review that joins them: whenever the agent lists its requirements with `list`, the tab returns each open requirement beside the read-log entries that best match it and the unopened files still to try, both ranked by BM25 against the requirement. The prompt block asks the agent to do so after exploring and before resolving any requirement, so that it can still act on what the review shows.

Resolving on Evidence To resolve a requirement as `done`, the agent cites the read-log entry its content came from, and RunningTab has the tab check the citation so that a `done` rests on what the agent read. Specifically, `resolve` refuses a citation whose path and excerpt share no term with the requirement and says what would resolve it, while `unavailable` requires a reason.

Checking at the Stop To give the agent a final pass over the tab, RunningTab places a check at the stop, the last moment the deliverable can still change. At the first stop, the environment sends one message (the finish check), which reports the status of the requirements together with the files listed but not opened, and asks the agent to add what is missing before finishing again.

4 Experimental Setup

In this section, we describe the benchmarks, the baselines, and the implementation details.

Benchmarks Direct workspace interaction targets tasks whose deliverable draws on many files of a workspace. We therefore use two benchmarks that pose such tasks and check the deliverable element by element (by rubrics or by checkpoints), and one benchmark of QA over enterprise documents:

- **Workspace-Bench** (Tang et al., 2026) simulates the workspaces of five professional roles (such as an operations manager and a researcher), each holding thousands of heterogeneous files, and asks for deliverables (such as reports, spreadsheets, and slide decks) that draw on files scattered across a workspace; it has 100 tasks and 1,850 evaluation instances (one per rubric), on which we report the Rubric Pass Rate (the share of satisfied rubrics) and the Task Completion Rate (TCR@70, the share of tasks with at least 70% of their rubrics satisfied), following its evaluation protocol.
- **TheAgentCompany** (Xu et al., 2025a) simulates a software company whose tasks are checked by programmatic evaluators at a series of checkpoints; we use a subset of its tasks that need no external service other than the file server of the company (such as other websites or external models), with 88 evaluation instances (one per checkpoint) in total, and report Success (the share of tasks whose

Table 1: Main results on three benchmarks with three LLMs, averaged over three runs (standard deviation in parentheses). Best results within each LLM are **bolded**; second best are underlined.

Method	Workspace-Bench		TheAgentCompany		OfficeQA Pro	
	Pass Rate	TCR@70	Score	Success	Acc@0%	Acc@1%
GPT-5.4 nano						
DCI	41.63 (±0.90)	20.67 (±1.70)	32.09 (±2.22)	17.86 (±5.05)	53.53 (±3.27)	65.38 (±2.83)
TODO	41.67 (±0.95)	23.33 (±2.62)	36.74 (±2.74)	22.62 (±3.37)	54.81 (±3.14)	65.71 (±3.27)
Self-Refine	41.32 (±2.15)	22.33 (±3.09)	35.76 (±2.97)	22.62 (±4.45)	54.17 (±2.40)	64.74 (±2.40)
Self-Tracking	42.93 (±0.44)	23.00 (±1.41)	34.30 (±1.52)	20.24 (±1.68)	55.13 (±2.97)	66.99 (±3.17)
RunningTab (Ours)	**44.73** (±0.56)	**25.67** (±2.49)	**39.12** (±1.22)	**23.81** (±1.68)	**60.58** (±0.79)	**71.15** (±0.79)
DeepSeek V4 Flash						
DCI	53.28 (±1.29)	38.67 (±1.25)	37.47 (±1.66)	21.43 (±2.92)	65.71 (±1.63)	74.68 (±1.20)
TODO	54.94 (±2.44)	38.00 (±2.16)	40.34 (±3.21)	25.00 (±5.05)	66.03 (±1.63)	73.08 (±2.83)
Self-Refine	53.44 (±0.53)	37.67 (±0.47)	40.24 (±1.76)	26.19 (±1.68)	64.74 (±1.63)	73.08 (±0.79)
Self-Tracking	53.56 (±1.57)	37.33 (±3.30)	39.24 (±5.21)	26.19 (±6.07)	64.74 (±1.63)	73.08 (±0.79)
RunningTab (Ours)	**59.99** (±0.60)	**41.00** (±1.63)	**43.11** (±0.61)	**29.76** (±1.68)	**68.59** (±1.20)	**77.56** (±1.20)
Gemini 3.8 Flash						
DCI	31.42 (±2.89)	23.33 (±1.70)	26.35 (±2.50)	19.05 (±1.68)	43.08 (±0.79)	45.64 (±0.91)
TODO	27.52 (±5.62)	22.33 (±3.30)	20.97 (±5.05)	14.29 (±5.05)	44.04 (±0.79)	47.24 (±0.45)
Self-Refine	32.73 (±2.92)	28.00 (±2.94)	26.68 (±6.54)	16.67 (±3.37)	41.79 (±1.20)	45.64 (±0.45)
Self-Tracking	26.87 (±1.23)	21.00 (±2.16)	21.68 (±6.23)	14.29 (±5.83)	45.00 (±2.72)	48.21 (±2.52)
RunningTab (Ours)	**46.91** (±1.18)	**36.33** (±3.30)	**32.93** (±0.41)	**22.62** (±1.68)	**55.77** (±0.79)	**62.82** (±1.98)

checkpoints all pass) and Score (the partial completion score, which gives each task half credit for the fraction of checkpoint points it earns and half for its success), as proposed by its authors.

- **OfficeQA Pro** (Opsahl-Ong et al., 2026) lets us check whether RunningTab also helps in the more conventional setting of question answering, where the deliverable is a single answer: its workspace holds 697 files (U.S. Treasury Bulletins spanning nearly a century, about 89,000 pages in total), and the answer to each question is found, and often computed, from these files. We use a subset of its questions that need no external data source (such as exchange rates or price indices), with 104 evaluation instances (one per question), and report the accuracy at an allowable absolute relative error of 0% and 1% (Acc@0% and Acc@1%), as in its evaluation protocol.

Baselines and Our Model We compare RunningTab against DCI and against three baselines in which the model, rather than an environment-side tab, keeps track of what the task asks for:

- **DCI** (Li et al., 2026b) is direct corpus interaction applied to the workspace: the agent works on the files with terminal tools, leaving the state of the task to its context window.
- **TODO** follows plan-first prompting (Wang et al., 2023), asking the agent to begin by writing a TODO list of every fact, figure, and deliverable element the task asks for.
- **Self-Refine** (Madaan et al., 2023) has the model review and refine its own output; at the first stop, one message asks the agent to re-check each of these elements against the source files it read and to make sure that each is in the deliverable before finishing again.
- **Self-Tracking** combines the two, as in model-kept records of task progress (Fourney et al., 2024; Ko et al., 2026): the agent lists these elements at the start and re-checks them at the end.
- **RunningTab** is our framework, in which the environment-side tab keeps track of the task.

Implementation Details Following Li et al. (2026b), all methods use DCI-Agent-Lite, a terminal agent built on the Pi harness[1], with its terminal tools (such as `read` and `bash`), a budget of 300 turns, and runtime context management at level 3 (which truncates each observation and clears older observations as they accumulate), to which we add a time limit of one hour per task. We instantiate every method with three LLMs: GPT-5.4 nano (OpenAI, 2026), DeepSeek V4 Flash (DeepSeek-AI, 2026), and Gemini 3.8 Flash (Google DeepMind, 2026). Where the evaluation calls for an LLM judge, we follow the protocol of each benchmark and use GPT-5.4 mini. For RunningTab, the calls to the operations on the tab are removed from the trajectory that the evaluators read, and its settings stay the same across benchmarks and models. We report the average over three runs of each configuration.

[1] https://github.com/earendil-works/pi

Table 2: What the tab holds per attempt on Workspace-Bench, reported as medians or, where marked, as shares of requirements (%).

	GPT	DeepSeek	Gemini
Requirements ($\mathcal{R}$)	4	5	4
resolved as `done` (%)	74.0	77.7	50.7
reviewed beside a matching excerpt (%)	87.1	88.0	69.9
Read-log entries ($\mathcal{L}$)	14	18	6
distinct files	5	11	5
excerpted text (K characters)	17.7	40.2	12.1
Candidates ($\mathcal{C}$), not opened	147	166	294

Table 3: How the agent operates the tab during a task, as the share of Workspace-Bench attempts (%) in which the agent does what each table row states.

Operation	**The agent**	GPT	DeepSeek	Gemini
`add`	states requirements before the check	56.9	100.0	100.0
`list`	pulls the review before the check	24.7	27.2	100.0
`resolve`	cites a reviewed entry for `done`	25.1	21.6	76.9
`show`	re-reads a stored entry	74.6	32.4	8.8
Finish check	operates the tab after it	72.6	8.7	5.1
	edits the deliverable after it	17.1	5.2	0.5
Candidates	opens a listed candidate	22.7	54.7	19.0

5 Experimental Results

Main Results We report the main results in Table 1, where RunningTab achieves the best performance on every benchmark, metric, and LLM. In contrast, the baselines in which the model itself keeps track of the task (by listing what the task asks for at the start, re-checking the deliverable at the end, or both) remain far less effective, which suggests that the gains of RunningTab come not from instructing the model to keep track of the task but from the record that the environment keeps on its behalf, outside the context window. Also, these gains hold across diverse workspace settings, from assembling deliverables from office files (Workspace-Bench) and carrying out the tasks of a company (TheAgentCompany) to answering questions from decades of office archives (OfficeQA Pro).

What the Tab Holds To understand what the environment-side tab provides to the agent, we first examine its contents on Workspace-Bench. As shown in Table 2, the tab holds a substantial record of each task: a median of four to five requirements, 6 to 18 read-log entries (each storing the text that the agent saw when it read a file, together with the path of that file) drawn from 5 to 11 distinct files and holding 12K to 40K characters of excerpted text (the file content exactly as the agent saw it), and 147 to 294 files that were listed but not opened. Moreover, the review brings this record to each requirement: when the agent pulls the review, 69.9% to 88.0% of its open requirements are shown beside a read-log entry whose excerpt contains a line with words of the requirement, so that, for most requirements, the review points the agent to a matching passage from a file it has already read.

Recovering Shown but Missing Content Recall that, with DCI, about one in five of the rubric values shown to the agent is absent from its deliverable (Figure 2, bottom). We check whether, with RunningTab, such values are held in the tab: for each numeric value that a failed rubric expects, that appears in an observation of the agent, and that is absent from the deliverable, we test whether the read log stores it. As shown in Figure 4 (left), with GPT-5.4 nano, the tab holds 94.8% of these values[2], so the missing content has in most cases already been captured by the environment. We further test whether it is usable: for the requirements whose values the deliverable left out entirely, the same LLM, given the task and the requirement with its values masked, recovers 8.4% of the values held in the tab without the tab, 45.8% from the review of the tab for that requirement and the entries it chooses to open, and 78.1% from the whole read log (Figure 4, right), so the review brings back more than half of what the full log provides while using only a small fraction of its text. Consistently, RunningTab succeeds on more of the rubrics in the Shown but Missing category (Figure 2, top) than DCI (27.0% against 20.7%).

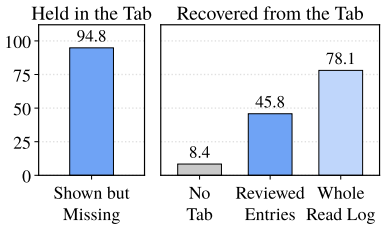


Figure 4: RunningTab with GPT-5.4 nano on Workspace-Bench (%). (Left) Values held in the tab, among those the agent was shown but its deliverable lacks. (Right) Held values that an LLM recovers when they are masked, by the part of the tab given.

How the Agent Operates the Tab We then examine how the agent operates the tab, reported in Table 3, which shows that each LLM draws on the tab in its own way. Specifically, Gemini 3.8 Flash works mainly through the review: it states its requirements and pulls the review before the finish check in every analyzed attempt, and in 76.9% of attempts it resolves a requirement as `done` by citing a read-log entry that the review listed for that requirement. GPT-5.4 nano instead relies on the

[2]The read log keeps only observations that show the content of a file (Section 3.3), so a value computed by the agent (such as a sum) or shown only in a listing (such as a file size) is not kept.

Table 4: Ablation of RunningTab with GPT-5.4 nano on Workspace-Bench (three runs), with drops from RunningTab in red.

Method	Pass Rate	TCR@70
RunningTab (Ours)	**44.73**	**25.67**
No Environmental Capture	41.36 −3.37	22.33 −3.34
No Review	41.87 −2.86	20.33 −5.34
No Evidence Check	42.89 −1.84	24.33 −1.34
No Finish Check	42.89 −1.84	23.00 −2.67
DCI	41.63	20.67

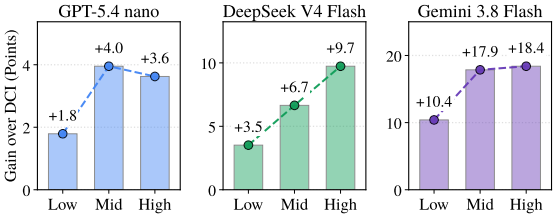


Figure 5: Gain of RunningTab over DCI in Pass Rate on tasks split into thirds by how much the agent reads.

finish check: it often turns to the tab only when the check prompts it, and after the check it operates the tab in 72.6% of attempts and edits its deliverable in 17.1%; it also re-reads stored entries with `show` in 74.6% of attempts. Lastly, DeepSeek V4 Flash works mainly through the candidates: like Gemini 3.8 Flash, it states its requirements before the finish check in every analyzed attempt, but it then explores more widely, reading the most distinct files (a median of 11 in Table 2) and opening a file that the tab listed as not yet opened in 54.7% of attempts.

Ablation Studies To examine how much each component of the environment-side tab contributes to the performance gains, we remove one component at a time on Workspace-Bench with GPT-5.4 nano and report the results in Table 4. From this, we observe that removing any of the four components lowers both the Pass Rate and TCR@70, indicating that each of them plays a role in these gains. Among them, removing the environmental capture (in which the environment no longer records the read log and the candidates, so that the review and the evidence check, which draw on them, are removed as well) leads to the largest drop in Pass Rate, down to the level of DCI (41.36 against 41.63), which indicates that the gains of RunningTab come largely from what the environment captures on its own, namely the files that the agent reads and lists. Following this, removing the review leads to the next largest drop, which suggests that the captured excerpts are most useful when the agent sees them beside the requirements they match. Lastly, the evidence check and the finish check complement the capture and the review at two moments of the task: when the agent resolves a requirement as `done`, the evidence check keeps that resolution grounded in what the agent read, and when the agent stops, the finish check gives it a final pass over the tab before delivery.

Gains on Reading-Heavy Tasks Since the tab keeps what the agent has read outside the context window, its benefit is expected to grow with how much the agent reads. To see whether this holds, we split the tasks of Workspace-Bench into thirds by reading volume (the characters that the agent receives from its reads and shell commands). As shown in Figure 5, the gain of RunningTab over DCI generally grows with reading volume across the three LLMs, which indicates that the tab is most valuable when the agent has the most to keep track of (on average, the gain on the third of tasks with the most reading is twice that on the third with the least). Also, it is worth noting that the reading-heavy tasks are where DCI performs worst (its Pass Rate drops by 16 to 27 points from the lightest to the heaviest third); this suggests that the content read earlier remains in the context yet goes unused as observations accumulate, which is precisely what the tab keeps within reach.

6 Conclusion

In this work, we presented RunningTab, a framework for direct workspace interaction, the setting in which an agent searches and reads the files of a workspace from a terminal to produce a deliverable. Motivated by the observation that what the task asks for and what the agent has read are typically left to the context window, we equipped the agent with an environment-side tab: a per-task record that the environment keeps outside the context window. Specifically, the agent states the requirements, and the environment records each read as an excerpt with its source and each file listed but not opened as a candidate. The agent can then review each open requirement beside its best-matching excerpts and candidates, resolve it by citing the excerpt that supplies it or by giving a reason, and receive a finish check at its stop. Empirical evaluations on three benchmarks with three LLMs show that RunningTab improves over direct workspace interaction alone and over baselines in which the model itself keeps track of the task. Ablations attribute most of this gain to what the environment captures on its own, and the tab usually holds the values a deliverable needs once seen. We believe these findings position the environment-side tab as a natural complement to direct access: terminal tools let the agent reach any file, and the tab keeps track of the task until the deliverable is handed in.

Ethics Statement

This work aims to improve how LLM agents produce deliverables from the files of a workspace, using existing models and publicly available benchmarks. However, agents equipped with RunningTab may retain the biases and errors of their underlying models, and since the tab stores excerpts of the files the agent reads, it may hold sensitive content when applied to real workspaces. We therefore encourage appropriate safeguards, such as keeping the tab under the same access controls as the workspace and reviewing the deliverables, when deploying such agents in real-world settings.

Reproducibility Statement

To facilitate reproducibility, we describe the environment-side tab in Section 3 and specify the benchmarks, baselines, LLMs, and evaluation protocols in Section 4, running each configuration three times. Upon acceptance, we will release our code to enable the full reproduction of our results.

AI Use Statement

We use large language models to assist with code implementation and debugging, the execution and analysis of experiments, and to aid in and polish writing. The core research ideas and the initial draft of the manuscript are entirely our own, and all AI-assisted code, results, and text are reviewed and verified by the authors, who take full responsibility for this work.

References

Shoufa Chen, Luyuan Wang, Xuan Yang, Zhiheng Liu, Yuren Cong, Yuanfeng Ji, Feiyan Zhou, Xiaohui Zhang, Fanny Yang, and Belinda Zeng. Tua-bench: A benchmark for general-purpose terminal-use agents. *arXiv preprint arXiv:2606.28480*, 2026. URL https://doi.org/10.48550/arXiv.2606.28480.

DeepSeek-AI. Deepseek-v4: Towards highly efficient million-token context intelligence. *arXiv preprint arXiv:2606.19348*, 2026. URL https://doi.org/10.48550/arXiv.2606.19348.

Adam Fourney, Gagan Bansal, Hussein Mozannar, Cheng Tan, Eduardo Salinas, Erkang Zhu, Friederike Niedtner, Grace Proebsting, Griffin Bassman, Jack Gerrits, Jacob Alber, Peter Chang, Ricky Loynd, Robert West, Victor Dibia, Ahmed Awadallah, Ece Kamar, Rafah Hosn, and Saleema Amershi. Magentic-one: A generalist multi-agent system for solving complex tasks. *arXiv preprint arXiv:2411.04468*, 2024. URL https://doi.org/10.48550/arXiv.2411.04468.

Google DeepMind. Gemini 3.8 Flash model card. Google DeepMind Model Card, September 2026. URL https://deepmind.google/models/model-cards/gemini-3-8-flash/.

Bowen Jin, Hansi Zeng, Zhenrui Yue, Dong Wang, Hamed Zamani, and Jiawei Han. Search-r1: Training llms to reason and leverage search engines with reinforcement learning. *arXiv preprint arXiv:2503.09516*, 2025. URL https://doi.org/10.48550/arXiv.2503.09516.

Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. Dense passage retrieval for open-domain question answering. In Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu (eds.), *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, November 16-20, 2020*, pp. 6769–6781. Association for Computational Linguistics, 2020. URL https://doi.org/10.18653/v1/2020.emnlp-main.550.

Dayoon Ko, Jihyuk Kim, Sohyeon Kim, Haeju Park, Dahyun Lee, Gunhee Kim, Moontae Lee, and Kyungjae Lee. When is enough not enough? illusory completion in search agents. *arXiv preprint arXiv:2602.07549*, 2026. URL https://doi.org/10.48550/arXiv.2602.07549.

Jack Lanchantin, Shubham Toshniwal, Jason Weston, Arthur Szlam, and Sainbayar Sukhbaatar. Learning to reason and memorize with self-notes. In Alice Oh, Tristan Naumann,

Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (eds.), *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/274d0146144643ee2459a602123c60ff-Abstract-Conference.html.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (eds.), *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/6b493230205f780e1bc26945df7481e5-Abstract.html.

Jiangnan Li, Yuqing Li, Mo Yu, Jinchao Zhang, and Jie Zhou. A new role for relevance: Guiding corpus interaction in agentic search. *arXiv preprint arXiv:2607.24223*, 2026a. URL https://doi.org/10.48550/arXiv.2607.24223.

Xiaoxi Li, Guanting Dong, Jiajie Jin, Yuyao Zhang, Yujia Zhou, Yutao Zhu, Peitian Zhang, and Zhicheng Dou. Search-o1: Agentic search-enhanced large reasoning models. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, EMNLP 2025, Suzhou, China, November 4-9, 2025*, pp. 5420–5438. Association for Computational Linguistics, 2025. URL https://doi.org/10.18653/v1/2025.emnlp-main.276.

Zhuofeng Li, Haoxiang Zhang, Cong Wei, Pan Lu, Ping Nie, Yi Lu, Yuyang Bai, Shangbin Feng, Hangxiao Zhu, Ming Zhong, Yuyu Zhang, Jianwen Xie, Yejin Choi, James Zou, Jiawei Han, Wenhu Chen, Jimmy Lin, Dongfu Jiang, and Yu Zhang. Beyond semantic similarity: Rethinking retrieval for agentic search via direct corpus interaction. *arXiv preprint arXiv:2605.05242*, 2026b. URL https://doi.org/10.48550/arXiv.2605.05242.

Yi Lu, Zhuofeng Li, Ping Nie, Haoxiang Zhang, Yuyu Zhang, Kai Zou, Wenhu Chen, Jimmy Lin, Dongfu Jiang, and Yu Zhang. Dr-dci: Scaling direct corpus interaction via dynamic workspace expansion. *arXiv preprint arXiv:2606.14885*, 2026. URL https://doi.org/10.48550/arXiv.2606.14885.

Ziyu Ma, Hailang Huang, Shun Zou, Yong Wang, Shidong Yang, Yiming Hu, Fei Wei, and Xiangxiang Chu. Longhorizon-harness: Advancing long-horizon agents for real-world tasks. *arXiv preprint arXiv:2608.01964*, 2026. URL https://doi.org/10.48550/arXiv.2608.01964.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (eds.), *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/91edff07232fb1b55a505a9e9f6c0ff3-Abstract-Conference.html.

Mike A. Merrill, Alexander Glenn Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, Lin Shi, Jeong Yeon Shin, Thomas Walshe, Estefany Kelly Buchanan, Junhong Shen, Guanghao Ye, Haowei Lin, Jason Poulos, Maoyu Wang, Marianna Nezhurina, Jenia Jitsev, Di Lu, Orfeas Menis-Mastromichalakis, Zhiwei Xu, Zizhao Chen, Yue Liu, Robert Zhang, Leon Liangyu Chen, Anurag Kashyap, Jan-Lucas Uslu, Jeffrey Li, Jianbo Wu, Minghao Yan, Song Bian, Vedang Sharma, Ke Sun, Steven Dillmann, Akshay Anand, Andrew Lanpouthakoun, Bardia Koopah, Changran Hu, Etash Kumar Guha, Gabriel H. S. Dreiman, Jiacheng Zhu, Karl Krauth, Li Zhong, Niklas Muennighoff, Robert Amanfu, Shangyin Tan, Shreyas Pimpalgaonkar, Tushar Aggarwal, Xiangning Lin, Xin Lan, Xuandong Zhao, Yiqing Liang, Yuanli Wang, Zilong Wang, Changzhi Zhou, David Heineman, Hange Liu, Harsh Trivedi, John Yang, Junhong Lin, Manish Shetty, Michael

Yang, Nabil Omi, Negin Raoof, Shanda Li, Terry Yue Zhuo, Wuwei Lin, Yiwei Dai, Yuxin Wang, Wenhao Chai, Shang Zhou, Dariush Wahdany, Ziyu She, Jiaming Hu, Zhikang Dong, Yuxuan Zhu, Sasha Cui, Ahson Saiyed, Arinbjörn Kolbeinsson, Jesse Hu, Christopher Michael Rytting, Ryan Marten, Yixin Wang, Alex Dimakis, Andy Konwinski, and Ludwig Schmidt. Terminal-bench: Benchmarking agents on hard, realistic tasks in command line interfaces. *arXiv preprint arXiv:2601.11868*, 2026. URL https://doi.org/10.48550/arXiv.2601.11868.

OpenAI. Introducing GPT-5.4 mini and nano. OpenAI Official Blog, March 2026. URL https://openai.com/index/introducing-gpt-5-4-mini-and-nano/.

Krista Opsahl-Ong, Arnav Singhvi, Jasmine Collins, Ivan Zhou, Cindy Wang, Ashutosh Baheti, Owen Oertell, Jacob Portes, Sam Havens, Erich Elsen, Michael Bendersky, Matei Zaharia, and Xing Chen. Officeqa pro: An enterprise benchmark for end-to-end grounded reasoning. *arXiv preprint arXiv:2603.08655*, 2026. URL https://doi.org/10.48550/arXiv.2603.08655.

Charles Packer, Vivian Fang, Shishir G. Patil, Kevin Lin, Sarah Wooders, and Joseph E. Gonzalez. Memgpt: Towards llms as operating systems. *arXiv preprint arXiv:2310.08560*, 2023. URL https://doi.org/10.48550/arXiv.2310.08560.

Joon Sung Park, Joseph C. O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Generative agents: Interactive simulacra of human behavior. In Sean Follmer, Jeff Han, Jürgen Steimle, and Nathalie Henry Riche (eds.), *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology, UIST 2023, San Francisco, CA, USA, 29 October 2023- 1 November 2023*, pp. 2:1–2:22. ACM, 2023. URL https://doi.org/10.1145/3586183.3606763.

Tejal Patwardhan, Rachel Dias, Elizabeth Proehl, Grace Kim, Michele Wang, Olivia Watkins, Simón Posada Fishman, Marwan Aljubeh, Phoebe Thacker, Laurance Fauconnet, Natalie S. Kim, Patrick Chao, Samuel Miserendino, Gildas Chabot, David Li, Michael Sharman, Alexandra Barr, Amelia Glaese, and Jerry Tworek. Gdpval: Evaluating AI model performance on real-world economically valuable tasks. *arXiv preprint arXiv:2510.04374*, 2025. URL https://doi.org/10.48550/arXiv.2510.04374.

Stephen E. Robertson and Hugo Zaragoza. The probabilistic relevance framework: BM25 and beyond. *Found. Trends Inf. Retr.*, 3(4):333–389, 2009. URL https://doi.org/10.1561/1500000019.

Alireza Salemi, Chang Zeng, Atharva Nijasure, Jui-Hui Chung, Razieh Rahimi, Fernando Diaz, and Hamed Zamani. Grepseek: Training search agents for direct corpus interaction. *arXiv preprint arXiv:2605.29307*, 2026. URL https://doi.org/10.48550/arXiv.2605.29307.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (eds.), *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/d842425e4bf79ba039352da0f658a906-Abstract-Conference.html.

Sahil Sen, Akhil Kasturi, Elias Lumer, Anmol Gulati, and Vamse Kumar Subbiah. Is grep all you need? how agent harnesses reshape agentic search. *arXiv preprint arXiv:2605.15184*, 2026. URL https://doi.org/10.48550/arXiv.2605.15184.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: language agents with verbal reinforcement learning. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (eds.), *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/1b44b878bb782e6954cd888628510e90-Abstract-Conference.html.

Shreyas Subramanian, Adewale Akinfaderin, Yanyan Zhang, Ishan Singh, Mani Khanuja, Sandeep Singh, and Maira Ladeira Tanke. Keyword search is all you need: Achieving rag-level performance without vector databases using agentic tool use. *arXiv preprint arXiv:2602.23368*, 2026. URL https://doi.org/10.48550/arXiv.2602.23368.

Zirui Tang, Xuanhe Zhou, Yumou Liu, Linchun Li, Yukai Wu, Weizheng Wang, Hongzhang Huang, Wei Zhou, Jun Zhou, Jiachen Song, Shaoli Yu, Jinqi Wang, Zihang Zhou, Hongyi Zhou, Yuting Lv, Jinyang Li, Jiashuo Liu, Ruoyu Chen, Chunwei Liu, Guoliang Li, Jihua Kang, and Fan Wu. Workspace-bench 1.0: Benchmarking AI agents on workspace tasks with large-scale file dependencies. *arXiv preprint arXiv:2605.03596*, 2026. URL https://doi.org/10.48550/arXiv.2605.03596.

Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim. Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models. In Anna Rogers, Jordan L. Boyd-Graber, and Naoaki Okazaki (eds.), *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2023, Toronto, Canada, July 9-14, 2023*, pp. 2609–2634. Association for Computational Linguistics, 2023. URL https://doi.org/10.18653/v1/2023.acl-long.147.

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, and et al. Openhands: An open platform for AI software developers as generalist agents. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=OJd3ayDDoF.

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://doi.org/10.48550/arXiv.2407.19056.

Yifan Wu, Lizhu Zhang, Yuhang Zhou, Mingyi Wang, Bo Peng, Serena Li, Xiangjun Fan, and Zhuokai Zhao. Remember when it matters: Proactive memory agent for long-horizon agents. *arXiv preprint arXiv:2607.08716*, 2026. URL https://doi.org/10.48550/arXiv.2607.08716.

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 Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems 37: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/5d413e48f84dc61244b6be550f1cd8f5-Abstract-Datasets_and_Benchmarks_Track.html.

Frank Fangzheng Xu, Yufan Song, Boxuan Li, Yuxuan Tang, Kritanjali Jain, Mengxue Bao, Zora Zhiruo Wang, Xuhui Zhou, Zhitong Guo, Murong Cao, Mingyang Yang, Hao Yang Lu, Amaad Martin, Zhe Su, Leander Maben, Raj Mehta, Wayne Chi, Lawrence Jang, Yiqing Xie, Shuyan Zhou, and Graham Neubig. Theagentcompany: Benchmarking LLM agents on consequential real world tasks. In Danielle Belgrave, Cheng Zhang, Laura N. Montoya, Hsuan-Tien Lin, Razvan Pascanu, Piotr Koniusz, Marzyeh Ghassemi, Nancy Chen, Iván Vladimir Meza Ruíz, and Arturo Loaiza-Bonilla (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2025, NeurIPS 2025, San Diego, CA, USA, December 2-7, 2025 / Mexico City, Mexico, November 30 - December 5, 2025*, 2025a. URL http://papers.nips.cc/paper_files/paper/2025/hash/0d744742f6fac4d1134c019b7cef3c8a-Abstract-Datasets_and_Benchmarks_Track.html.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-mem: Agentic memory for LLM agents. In Danielle Belgrave, Cheng Zhang, Laura N. Montoya, Hsuan-Tien

Lin, Razvan Pascanu, Piotr Koniusz, Marzyeh Ghassemi, Nancy Chen, Iván Vladimir Meza Ruíz, and Arturo Loaiza-Bonilla (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2025, NeurIPS 2025, San Diego, CA, USA, December 2-7, 2025 / Mexico City, Mexico, November 30 - December 5, 2025*, 2025b. URL http://papers.nips.cc/paper_files/paper/2025/hash/19909c36f51abc4856b4560aff3d36d6-Abstract-Conference.html.

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 Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems 37: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/5a7c947568c1b1328ccc5230172e1e7c-Abstract-Conference.html.

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, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

Zeyu Zhang, Quanyu Dai, Xiaohe Bo, Chen Ma, Rui Li, Xu Chen, Jieming Zhu, Zhenhua Dong, and Ji-Rong Wen. A survey on the memory mechanism of large language model-based agents. *ACM Trans. Inf. Syst.*, 43(6):155:1–155:47, 2025. URL https://doi.org/10.1145/3748302.

Zijian Zhou, Ao Qu, Zhaoxuan Wu, Sunghwan Kim, Alok Prakash, Daniela Rus, Jinhua Zhao, Bryan Kian Hsiang Low, and Paul Pu Liang. MEM1: learning to synergize memory and reasoning for efficient long-horizon agents. *arXiv preprint arXiv:2506.15841*, 2025. URL https://doi.org/10.48550/arXiv.2506.15841.

Shengyao Zhuang, Yuansheng Ni, Hengxin Fun, Jimmy Lin, and Xueguang Ma. Towards retrieving interaction spaces for agentic search. *arXiv preprint arXiv:2606.06880*, 2026. URL https://doi.org/10.48550/arXiv.2606.06880.