BUSYWORLD: BENCHMARKING COMPUTER-USE AGENT PLANNING IN A CONCURRENT WORLD

Anonymous authors
Paper under double-blind review

ABSTRACT

Computer-use agents are moving from controlled test environments to shared real-world environments. In shared environments, multiple actors may operate on the same resources concurrently, causing task conditions to change during execution. When multiple goals compete for a limited action budget, the agent must balance their importance and urgency while considering how current choices affect future opportunities. However, existing benchmarks provide limited evaluation of such planning trade-offs under concurrency. To address this gap, we introduce BUSYWORLD, a benchmark for evaluating multi-goal planning by computer-use agents in shared concurrent environments. The two benchmark dimensions are **(a) Concurrency**, implemented through NPC events that can change shared state during execution, and **(b) Planning trade-offs**, arising from differences in goal importance, urgency, and action cost under a shared action budget. The benchmark records state changes, public interface observations, agent actions, and goal outcomes on a common timeline, enabling controlled analysis of how concurrency changes planning choices. Extensive experiments on four advanced models across 71 evaluation instances show that competition consistently lowers task success, with larger losses as more goals share the budget, and that agents often make progress toward the right goals but fail to complete them once other actors change the shared state.

1 INTRODUCTION

As computer-use agents (CUAs) are applied to increasingly realistic and complex real-world tasks (Xie et al., 2024), benchmarks built on controlled, isolated environments no longer suffice for evaluating agent planning in shared environments. In a shared environment, other participants act concurrently with the agent. Their activity changes the environment independently of the agent, so the urgency of each part of a task can change during execution. When a task contains several goals, these goals also compete for a limited action budget, and a change in urgency alters which goals the remaining budget should serve. Planning under environmental change and a limited action budget is therefore a core capability for agents in shared environments.

Prior work has studied temporal planning and dynamic environments separately. TimeArena incorporates action duration, task dependencies, and resource occupancy into planning (Zhang et al., 2024). Trainee-Bench studies workflow scheduling with different priorities and temporal constraints (Fu et al., 2026). Gaia2 allows external events to occur independently of the agent, so the environment continues to evolve during execution (Froger et al., 2026). We study their interaction: concurrency changes task conditions, which in turn changes the relative value of execution choices. This requires the agent to reconsider how to use its remaining action budget during execution. The evaluation target therefore extends from single-action correctness to the cumulative outcome of the execution sequence. This leads to our central question: *can agents effectively balance current actions against future opportunities in shared concurrent environments to improve overall task outcomes?*

To study this question, we introduce BUSYWORLD, a benchmark for evaluating multi-goal planning by computer-use agents in shared concurrent environments. It covers two benchmark dimensions. (i) Concurrency is implemented through NPC events that can change shared state during execution and affect the feasibility of later actions. (ii) Planning trade-offs arise when goals differ in importance, urgency, and action cost under a shared action budget. Figure 1 contrasts a successful purchase in

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.

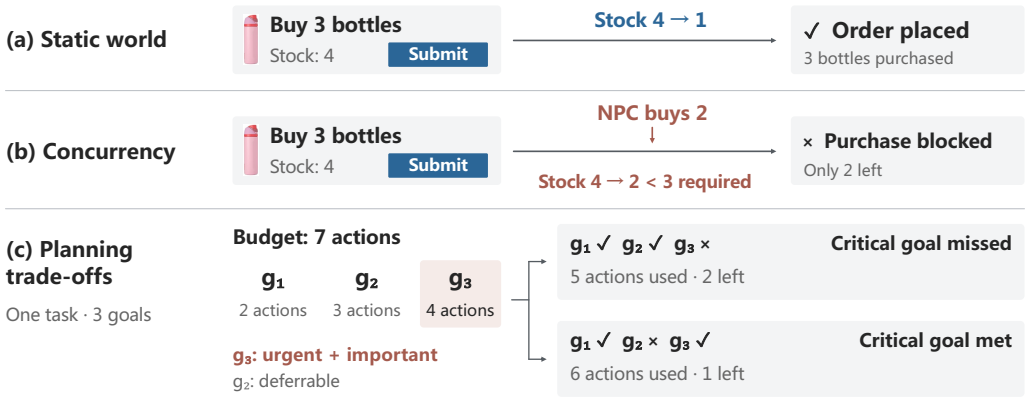

Figure 1: The two benchmark dimensions in BUSYWORLD. Panels (a) and (b) illustrate concurrency, and panel (c) illustrates planning trade-offs.

a static setting with the same purchase after an NPC consumes shared stock. It also shows how completing other goals first can leave too few actions for a critical goal.

Our contributions are as follows:

- We introduce BUSYWORLD, a computer-use benchmark in which NPCs change shared state during execution and goals compete for a shared action budget, with matched task variants that isolate concurrency and goal importance.
- We construct 71 controlled evaluation instances covering competition, multi-goal planning, attribute-based selection, and continual learning, with full execution traces for every run.
- Experiments on four advanced models show that competition consistently lowers task success, especially with several goals, that agents often reach the right goals but fail to complete them, and that earlier experience improves downstream success.

2 RELATED WORK

Dynamic Environments. WebArena and OSWorld evaluate computer-use agents on executable web and desktop tasks whose outcomes are verified from application state (Zhou et al., 2024; Xie et al., 2024). Later benchmarks let this state change during execution. DynamicGUIBench studies information missed because of interface dynamics, and D-GARA tests robustness to anomalies such as permission dialogs (Liu et al., 2026; Chen et al., 2025). Gaia2 introduces asynchronous events independent of agent actions, OSWorld 2.0 includes workflows in which new information arrives mid-task, and in τ^2-Bench a simulated user also acts on a shared world (Froger et al., 2026; Yuan et al., 2026; Barres et al., 2025). Work on time-of-check to time-of-use (TOCTOU) failures shows that state changes between observation and action can be exploited against browser and desktop agents (Jiang et al., 2026; Xu, 2026). BUSYWORLD studies the same gap from a planning perspective: another participant consumes resources the agent needs, so a valid goal can become infeasible before the agent reaches it.

Task Scheduling. Temporal planning benchmarks show that the order of otherwise valid actions matters. TimeArena models action durations, dependencies, and resource occupancy, and Robotouille studies the interleaving of activities with delayed effects (Zhang et al., 2024; Gonzalez-Pumariega et al., 2025). AsyncTool extends scheduling to tool calls with delayed feedback, and Trainee-Bench considers workplace tasks that arrive with different priorities (Shi et al., 2026; Fu et al., 2026). In these settings, the value of a schedule is determined by the task itself. In BUSYWORLD, it also depends on resources that another actor may consume, so the preferred order can change during execution.

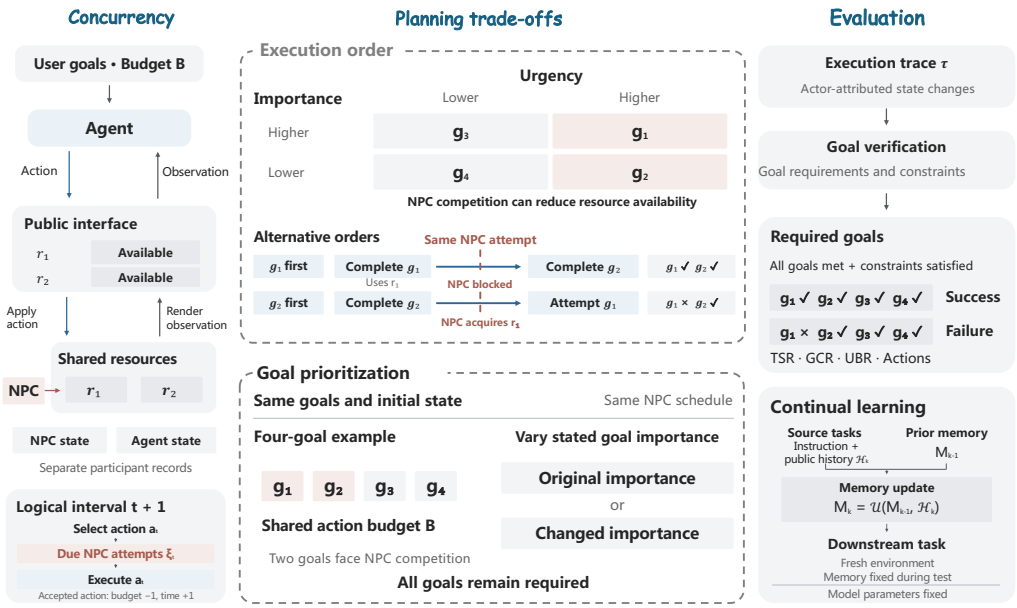

Figure 2: Overview of BUSYWORLD, from shared-resource interaction and multi-goal planning to outcome evaluation and continual learning.

Online Planning. InterruptBench studies how agents respond when user requests are added, revised, or withdrawn during web navigation, and Real-Time Reasoning Gym studies the trade-off between reasoning depth and response delay in evolving environments (Zou et al., 2026; Wen et al., 2025). In BUSYWORLD, the goals stay fixed while shared resources change the completion opportunities of the remaining goals. Matched variants vary NPC activity or stated goal importance without changing the initial state, goals, or action budget, which allows direct comparison of execution outcomes.

3 THE BUSYWORLD BENCHMARK

BUSYWORLD constructs tasks around goals that depend on shared resources. The *Concurrency* dimension introduces scheduled operations by non-player characters (NPCs) on these resources during execution. The *Planning trade-offs* dimension combines goals of different importance and urgency under a shared action budget. We also evaluate how experience retained across tasks affects subsequent planning. Figure 2 summarizes the task construction and evaluation.

3.1 CONCURRENCY

The agent and NPCs operate on the same resource state. NPC activity can reduce the resources available for a goal, while the agent's actions can change the outcome of later NPC operations. We specify this activity through scheduled operation attempts.

Shared State. A task instruction x specifies m required goals $\mathcal{G} = \{g_i\}_{i=1}^{m}$ and their constraints. The agent interacts with the application interface under an action budget B. The environment maintains shared resource state and participant-specific records. At step t, the agent receives a public observation o_t, together with feedback from its preceding action. It cannot access private NPC records or future event schedules.

Event Schedule. An NPC event e_j specifies a participant, a target resource, and operation parameters. The schedule $\Xi = ((d_j, e_j))_{j=1}^{K}$ assigns each attempt a logical interval d_j. Resource-specific streams are merged by interval, with simultaneous events following their declared stream order.

Each attempt acts on the resource state at execution and may be rejected if its preconditions no longer hold. The schedule therefore fixes external attempts while their outcomes depend on the agent's actions.

Execution Protocol. Each accepted agent action advances logical time by one interval, independently of model inference speed. The environment processes due NPC attempts after action selection and before action execution. Let s_t be the state after t accepted actions and h_t the agent's history of public observations, actions, and feedback. With policy π and the ordered NPC attempts ξ_t due at interval $t+1$, the transition is

$$\begin{aligned} &a_t \sim \pi(\cdot \mid x, h_t), \\ &\widetilde{s}_t = E(s_t, \xi_t), \qquad s_{t+1} = T(\widetilde{s}_t, a_t). \end{aligned} \tag{1}$$

Here, E advances the clock and applies due NPC attempts, and T executes the agent action in the resulting state $\widetilde{s}_t$. Thus, the resource state may change between the observation used to select an action and its execution. A run ends under the task's termination conditions or when its action budget or request cap is reached.

Observation. A change in shared state can affect a decision only after it enters the public observation. Let v_t denote the current page and viewport. The observation delivered to the agent is

$$o_t = \Omega(s_t, v_t), \tag{2}$$

where Ω selects the public content rendered on the page. A backend change may not yet be rendered, and rendered content may lie outside the viewport. Scrolling, clicking, and refreshing can reveal new information, but each such action consumes budget and advances logical time. The trace therefore records the environment state, the rendered page, and the delivered observation separately.

Single-goal tasks isolate resource competition from choices among goals. Each task is run with and without NPC activity, using the same instruction, initial state, action budget, and interface entry point.

3.2 Planning Trade-offs

With multiple goals, each action consumes the shared budget and advances the event schedule for all goals. Spending actions on one goal can therefore reduce the opportunity to complete another. We construct tasks with separate variations in importance and urgency to study how agents allocate actions under this constraint.

Goal Construction. A base task specifies the required goals $\mathcal{G}$, their resources and initial availability, and completion conditions. All goals are supplied at the start and share the action budget B. Selected goals have NPC event streams attached to their resources, while the remaining goals have no scheduled competitors. This creates differences in urgency, since delay can reduce resource availability for contested goals.

The instruction conveys importance through the consequences of leaving each goal unfinished. Assigning different importance levels within both resource groups yields four quadrants of high or low importance and high or low urgency. Each quadrant may contain multiple goals. An important goal can have little resource pressure, while a less important goal can face a short completion window. Section 4.1 specifies the evaluated instantiation.

Action cost, the number of actions needed to complete a goal, determines how much budget and time its pursuit consumes. The agent can interleave work across goals or use an action to advance several goals when the environment permits it. Completion is assessed from the resulting state, with no prescribed execution order.

Feasible Goal Sets. Let $\mathcal{R}_t \subseteq \mathcal{G}$ denote the goals still incomplete after t actions. The goal sets that remain jointly achievable are

$$\mathcal{F}_t = \big\{ S \subseteq \mathcal{R}_t : \text{ some continuation completes all goals in } S \text{ within the remaining budget } B - t \big\}. \tag{3}$$

$\mathcal{F}_t$ is defined relative to the current state and the event schedule. An NPC event can remove a contested goal from every set in $\mathcal{F}_t$, and spending actions on one goal can shrink the sets that include the others. A good plan therefore keeps high-importance goals in $\mathcal{F}_t$ while contested resources remain available. Each task instance is checked for reachability with a reference action sequence, and $\mathcal{F}_t$ is not provided to the agent.

Task Instantiation. The constructor binds goal declarations to environment entities and uses the same bindings to generate instructions, initial resource states, NPC targets, and completion checks. Substituting entity-specific facts in instruction templates produces task instances with different resources and the same competition pattern. Event timing, operation parameters, and ordering remain fixed for each resource role.

Each base task yields conditions without NPC activity, with competition, and with changed goal importance. The last condition changes the consequence statements while retaining the competition schedule. All conditions share the initial state, required goals, action budget, and completion checks. Section 4.1 gives the condition settings.

3.3 Evaluation

We evaluate planning through task outcomes under the shared action budget. The same completion criteria apply across task conditions and to downstream tests with retained memory.

Task Outcomes. At the end of each run, completion is checked against the resulting environment state and actor-attributed records. Task success requires all goals in $\mathcal{G}$ to be completed and all task-level constraints to be satisfied. Per-goal completion records partial progress when the full task is not completed. We report task success rate (TSR), goal completion rate (GCR), unintended behavior rate (UBR), and the mean number of accepted actions (Actions). Execution traces relate these measurements to execution order and changes in resource availability. Section 4.1 defines the metrics, and Appendix A.1 gives their calculation and aggregation rules.

Continual Learning. To assess how experience affects subsequent planning, we retain memory across source tasks while keeping model parameters fixed. After source task k, the memory update uses the previous memory M_{k-1} and the task instruction and public interaction history, denoted by $\mathcal{H}_k$. Starting from empty memory M_0, the update is

$$M_k = \mathcal{U}(M_{k-1}, \mathcal{H}_k), \tag{4}$$

where $\mathcal{U}$ operates on these public inputs under a fixed memory capacity. Appendix A specifies the update operations and capacity.

Each downstream task starts from a fresh environment, with M_k held fixed throughout execution. The policy in Equation 1 becomes $a_t \sim \pi(\cdot \mid x, h_t, M_k)$. We compare memory formed with and without resource competition, and after different numbers of source tasks. Testing these memory conditions on the same downstream task against the empty-memory baseline M_0 measures how the source and amount of experience affect task outcomes.

4 Experiments

We examine how competition, goal importance, and prior experience affect task completion. Single-goal tasks isolate the effect of NPC competition. Multi-goal tasks add a shared action budget and vary goal importance under the same competition schedule. Attribute-based selection distinguishes choosing the specified product from completing its order. Continual-learning evaluations test whether memory formed on earlier source tasks changes downstream performance. Together, these settings locate where agents lose success, from choosing a target to submitting its order.

4.1 Experimental Setup

Tasks. We use eight base tasks each for single-goal competition, four-goal planning, and attribute-based selection, three for two-goal planning, and one downstream task for the two memory comparisons. A base task fixes the goals and initial environment before a condition is applied. Four-goal

Table 1: Reference-model outcomes. N is the number of valid runs and rates are percentages. With one required goal, GCR equals TSR.

Condition	N	TSR ↑	GCR ↑	UBR ↓	Actions
(a) Single-goal competition					
Static	24	**79.2** (19/24)	**79.2**	12.5 (3/24)	14.0
Dynamic	24	50.0 (12/24)	50.0	12.5 (3/24)	23.7
(b) Four-goal planning					
M1 Static	22	**68.2** (15/22)	**78.4**	13.6 (3/22)	50.7
M1 Dynamic	19	0.0 (0/19)	0.0	0.0 (0/19)	55.6
M2 Dynamic	20	0.0 (0/20)	1.3	0.0 (0/20)	88.9
(c) Attribute-based selection					
Ratings count	24	8.3 (2/24)	8.3	4.2 (1/24)	74.8
Current price	24	**33.3** (8/24)	**33.3**	0.0 (0/24)	68.8
(d) Two-goal planning					
M1 Static	9	**77.8** (7/9)	**83.3**	0.0 (0/9)	27.7
M1 Dynamic	3	66.7 (2/3)	66.7	0.0 (0/3)	29.0
M2 Dynamic	5	20.0 (1/5)	20.0	0.0 (0/5)	16.8

tasks cover the four combinations of importance and urgency, with two goals exposed to NPC competition. Crossing base tasks with conditions gives 71 evaluation instances, whose composition Appendix A lists.

Agent Setup. DeepSeek-V4.1-Flash is the reference model. GPT-5.6 Luna, Gemini 3.8 Flash, and Qwen3.8-Flash are evaluated on single-goal competition, four-goal planning, and attribute-based selection. The reference model runs three repetitions per evaluation instance and each additional model runs one. Memory conditions use eight repetitions. All models share the agent framework, observation format, action protocol, within-run history, and input limit. Each run starts from a freshly reset environment with an action budget of 60 steps for single-goal tasks and 100 steps for multi-goal and selection tasks, together with a fixed request cap. Memory is provided only in continual-learning evaluations and stays fixed during testing.

Metrics. (i) *Task success rate (TSR)* is the fraction of runs that complete all required goals and satisfy task-level constraints. (ii) *Goal completion rate (GCR)* averages the fraction of required goals completed in each run. (iii) *Unintended behavior rate (UBR)* is the fraction of runs with at least one unauthorized commerce side effect. (iv) *Actions* is the mean number of accepted agent actions per run. Rates are computed over valid runs, and Appendix A lists the excluded runs. Appendix A.1 gives the formal definitions.

Experimental Groups. Static and Dynamic denote the absence and presence of NPC competition. M1 and M2 denote the original and changed importance assignments in multi-goal tasks. In four-goal tasks, M2 reverses importance within each urgency group. Memory conditions vary the source or number of source tasks. Table 4 in Appendix A summarizes all conditions.

4.2 Main Results

Table 1 reports reference-model outcomes, and Table 2 reports paired differences. M1 Dynamic versus M1 Static tests competition at fixed importance. M2 Dynamic versus M1 Dynamic tests importance under the same competition schedule.

Single-goal Competition. The two conditions share the instruction, initial state, and budget, and Dynamic adds NPC purchases on the required item. TSR falls from 79.2% to 50.0%, a paired decrease of 29.2 percentage points (pp) across 24 pairs. UBR stays at 12.5% in both conditions, so the additional failures are unfinished orders rather than unintended purchases. Actions rise from 14.0 to 23.7, and paired runs use 9.6 more actions under competition, so competition makes the agent spend more of its budget. Most failures still end with the agent stopping on its own, as 12 of

Table 2: Paired differences (first minus second). Rate differences are in percentage points.

Contrast	Pairs	ΔTSR	ΔGCR	ΔUBR	ΔActions
Single-goal competition					
Dynamic − Static	24	−29.2	−29.2	0.0	+9.6
Four-goal planning					
M1 Dynamic − M1 Static	19	−68.4	−76.3	−10.5	+6.5
M2 Dynamic − M1 Dynamic	18	0.0	+1.4	0.0	+34.7
Two-goal planning					
M1 Dynamic − M1 Static	3	−33.3	−33.3	0.0	+0.3
M2 Dynamic − M1 Dynamic	2	0.0	0.0	0.0	−21.0

the 17 failed runs across both conditions end normally with the order unfinished and five reach the request cap or action budget, as Appendix D.1 reports. In a matched case, the agent completes the Fire Kids order in the Static condition, whereas in the Dynamic condition NPCs consume the stock and the agent exhausts its budget without an order.

Four-goal Planning. All four goals are required in every condition. In M1 Static, the model completes 15 of 22 tasks and 69 of 88 goals, giving a TSR of 68.2% and a GCR of 78.4%. In M1 Dynamic, it completes none of 19 tasks and none of 76 goals, including the two goals per task that face no NPC competition. Across 19 pairs, TSR decreases by 68.4 pp and GCR by 76.3 pp, and Actions increase by 6.5. In 16 of the 19 M1 Dynamic runs, the agent adds target items to the cart, but none of these runs produces an order. Because the goals without NPC competition also remain unfinished, we attribute the failures to order completion under competition rather than to the loss of contested stock alone. UBR falls from 13.6% to zero because runs that submit no order cannot produce an unintended purchase. Unlike single-goal failures, four-goal failures rarely end with the agent stopping on its own. Of the 46 failures across the three four-goal conditions, 24 reach the request cap or action budget, 19 end with an invalid action or parse failure, and three end normally, so the agent mostly runs out of budget or breaks off while the goals remain open.

Importance Comparison. M2 Dynamic reverses importance within each urgency group while keeping the goals, initial state, budget, and NPC schedule. All 20 runs add target items to the cart, but only one produces an order, completing one of 80 goals for a GCR of 1.3% and a TSR of zero. Across 18 pairs, GCR differs from M1 Dynamic by $+1.4$ pp, and M2 Dynamic uses 34.7 more actions. Changing the business notes thus changes how long the agent interacts with the shop, from 55.6 to 88.9 actions per run, but not whether the goals reach a submitted order.

Two-goal Planning. Three base tasks reduce the required goals from four to two while keeping a contested goal and the three conditions. M1 Static completes seven of nine runs, and M1 Dynamic completes two of three. Competition lowers TSR by 33.3 pp across three pairs, compared with 68.4 pp across 19 pairs with four goals. Since the agent completes tasks under competition when two goals are required, the zero success rate with four goals arises from competition combined with the number of goals, and the contested goals remain attainable. Two-goal runs also use fewer actions, 16.8 to 29.0 per run compared with 50.7 to 88.9 with four goals. M2 Dynamic completes one of five runs. Across two pairs, changing importance leaves TSR unchanged, and M2 Dynamic uses 21.0 fewer actions.

4.3 Attribute-based Selection

The planning results show agents reaching the cart without submitting orders. Attribute-based selection tests whether this gap also appears without competition, by separating the choice of product from the order. Each selection task presents two products with fixed attributes. Ratings count requests the product with more ratings, and Current price requests the cheaper one, so the correct product changes with the instruction. No NPC events occur.

Table 3 separates product choice from order completion. A choice is unassessable when the trace does not identify the product. All identifiable choices are correct, whereas four of six correct choices

Table 3: Attribute-based selection over 24 runs per criterion.

Criterion	Correct choice	Wrong choice	Unassessable	Correct order	TSR ↑
Ratings count	6/24	0/24	18/24	2/24	8.3
Current price	9/24	0/24	15/24	8/24	**33.3**

under Ratings count and one of nine under Current price end without a correct order. Where the choice can be identified, the failures therefore arise after the choice rather than from wrong selections. Current price reaches a higher TSR than Ratings count, 33.3% versus 8.3%, mainly because more of its correct choices become correct orders, eight of nine versus two of six. Both criteria use about 70 actions per run, and of 38 failed selection runs, 27 reach the request cap or action budget and nine end with an invalid action or parse failure, as detailed in Appendix D.1. In one matched task, changing the criterion from Ratings count to Current price changes the product choice correctly and both orders succeed, while other runs select correctly but fail through duplicate orders. The only unintended purchase occurs under Ratings count.

4.4 Continual Learning

The continual-learning evaluation varies the memory formed from source tasks while keeping the downstream task fixed. A source task is an earlier task whose execution updates the memory. Each downstream run starts from the same initial state, with the same instruction, budget, and NPC schedule. Figure 3 reports success rates, and Table 11 in Appendix C.3 reports paired differences. Both comparisons share the No memory runs, and One task uses the Dynamic source runs.

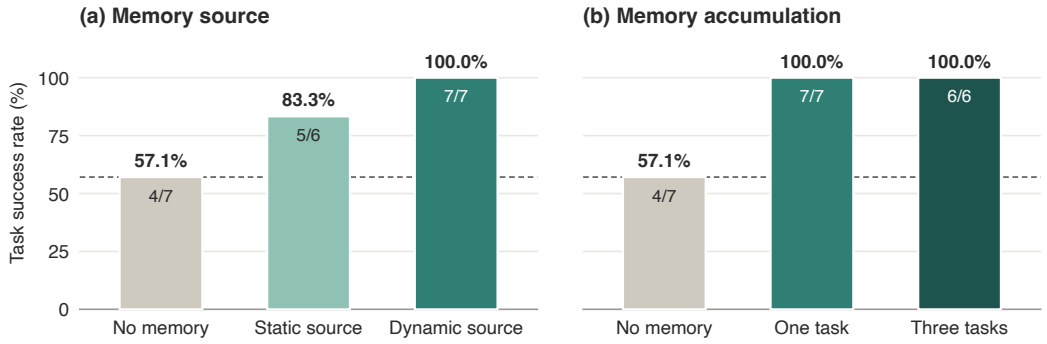

Figure 3: Downstream task success by memory condition, with successes/valid runs.

Memory Source. Without memory, the agent succeeds in four of seven runs. With Static source memory it succeeds in five of six, and with Dynamic source memory in seven of seven. On paired repetitions, both sources improve TSR by 33.3 pp over No memory, and Dynamic source exceeds Static source by 20.0 pp across five pairs, while using 13.8 fewer actions, 841,731 fewer input tokens, and 359.0 fewer seconds per run, as reported in Appendix C.3. Static source memory also lengthens the runs, to 23.3 actions compared with 10.0 without memory and 11.9 with Dynamic source memory. The failures show where memory helps. All four failures in this comparison end normally, three without memory and one with Static source memory. In a matched case, the agent without memory adds the requested quantity to the cart but stops before ordering, whereas with Static source memory it proceeds through checkout and completes the order. This case suggests that memory helps at the step where failures in the main tasks concentrate, turning cart progress into a submitted order.

Memory Accumulation. One task and Three tasks provide memory after one and three source tasks in a fixed order. They complete seven of seven and six of six downstream runs, respectively, and TSR is unchanged across six pairs. The improvement over No memory is therefore obtained

after the first source task. Later source tasks revise product and order details, and the memory grows from 268 to 319 characters, as recorded in Appendix C.4.

4.5 Model Comparison

We test whether the reference-model findings hold for other models. Figure 4 compares four models on the first repetition of each evaluation instance, giving eight runs per condition, except for one infrastructure failure each for GPT-5.6 Luna in M1 Static and Qwen3.8-Flash in Ratings count.

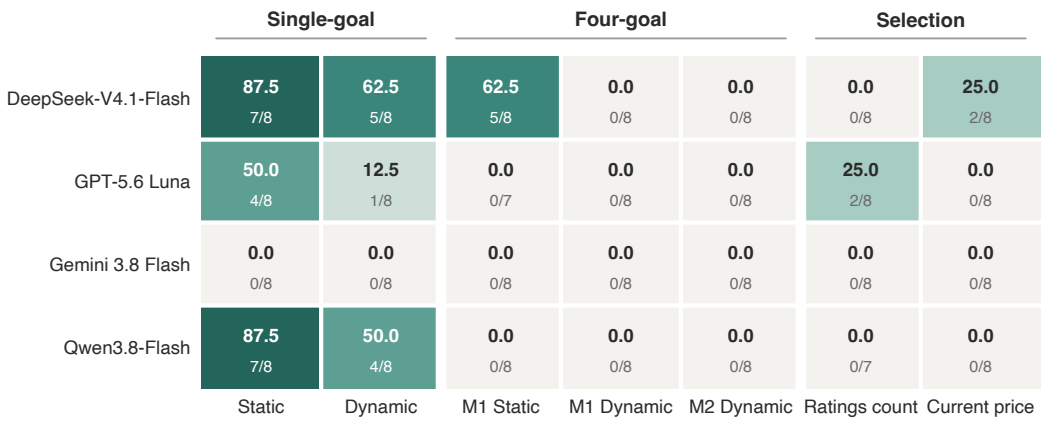

Figure 4: Task success rates (%) across models on the first repetition, with successes/valid runs.

Single-goal Competition. The effect of competition extends beyond the reference model. Under competition, successes fall from seven to five of eight runs for DeepSeek-V4.1-Flash, from seven to four for Qwen3.8-Flash, and from four to one for GPT-5.6 Luna, a drop of 25.0 pp for the first model and 37.5 pp for the other two. Every model that succeeds in the Static condition therefore declines under competition. Gemini 3.8 Flash completes no run and averages fewer than two actions per run, too few to complete a purchase.

Four-goal Planning. Four-goal tasks are the hardest setting for every model. No model completes a four-goal task under competition, and without competition only DeepSeek-V4.1-Flash succeeds, in five of eight M1 Static runs. GPT-5.6 Luna and Qwen3.8-Flash complete single-goal purchases but use only 17.9 and 19.1 actions per four-goal run, well below the 100-step budget, so their failures reflect the multi-goal setting.

Attribute-based Selection. Model rankings in selection depend on the criterion. GPT-5.6 Luna succeeds in two of eight Ratings count runs and DeepSeek-V4.1-Flash in two of eight Current price runs, each with none under the other criterion, while Qwen3.8-Flash and Gemini 3.8 Flash complete no selection run. No model is therefore best in every task family. Appendices B.2 and D give per-model counts, resource use, and failure analyses.

5 Conclusion

We introduce BusyWorld, a benchmark that makes concurrency and planning trade-offs controllable within the same computer-use task. Extensive experiments on four advanced models show that competition consistently lowers task success, especially when several goals share the budget, and that agents often make progress toward the right goals but fail to complete them once the shared state changes, while experience from an earlier task improves downstream success. Planning under concurrency therefore remains challenging for current agents, and future agents and models should be trained in realistic concurrent environments rather than only in isolated ones.

AI USE STATEMENT

Generative AI tools assisted with literature search, source-code inspection, refinement of the conceptual framework and experimental comparisons, interpretation of recorded results, manuscript drafting and revision, and figure preparation. The models evaluated in the benchmark are specified in Section 4.1. The authors are responsible for the accuracy of the reported results, citations, and scientific claims.

ETHICS STATEMENT

BUSYWORLD studies computer-use agents in controlled environments with shared resource state and scripted NPC activity. Resource competition is represented through changes to benchmark state. The benchmark studies planning under these conditions and does not authorize autonomous actions on live services. Applying such agents to real accounts requires appropriate permissions, privacy protection, and safeguards for consequential actions. Redistribution of environment assets and interaction traces should respect third-party rights and exclude credentials and personal information. Task completion and unintended behavior rates characterize the tested conditions and are not a certification of deployment safety.

REPRODUCIBILITY STATEMENT

Section 3 describes the shared-state dynamics, goal construction, and memory update procedure. Section 4.1 specifies the evaluated models, comparison groups, and repetition design. Appendix A provides event schedules, agent and memory settings, metric definitions, and run counts. Appendices B and C give resource costs, paired comparisons, and memory analyses. Appendix D classifies failed runs and presents matched cases. Appendix E gives the agent prompt and an example task instruction.

REFERENCES

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. τ^2-bench: Evaluating conversational agents in a dual-control environment, 2025. URL https://arxiv.org/abs/2506.07982.

Sen Chen, Tong Zhao, Yi Bin, Fei Ma, Wenqi Shao, and Zheng Wang. D-GARA: A dynamic benchmarking framework for GUI agent robustness in real-world anomalies, 2025. URL https://arxiv.org/abs/2511.16590.

Romain Froger, Pierre Andrews, Matteo Bettini, Amar Budhiraja, Ricardo Silveira Cabral, Virginie Do, Emilien Garreau, Jean-Baptiste Gaya, Hugo Laurençon, Maxime Lecanu, Kunal Malkan, Dheeraj Mekala, Pierre Ménard, Gerard Moreno-Torres Bertran, Ulyana Piterbarg, Mikhail Plekhanov, Mathieu Rita, Andrey Rusakov, Vladislav Vorotilov, Mengjue Wang, Ian Yu, Amine Benhalloum, Grégoire Mialon, and Thomas Scialom. Gaia2: Benchmarking LLM agents on dynamic and asynchronous environments, 2026. URL https://arxiv.org/abs/2602.11964.

Daocheng Fu, Jianbiao Mei, Rong Wu, Xuemeng Yang, Jia Xu, Ding Wang, Pinlong Cai, Yong Liu, Licheng Wen, and Botian Shi. The agent's first day: Benchmarking learning, exploration, and scheduling in the workplace scenarios, 2026. URL https://arxiv.org/abs/2601.08173.

Gonzalo Gonzalez-Pumariega, Leong Su Yean, Neha Sunkara, and Sanjiban Choudhury. Robotouille: An asynchronous planning benchmark for LLM agents, 2025. URL https://arxiv.org/abs/2502.05227.

Linxi Jiang, Zhijie Liu, Haotian Luo, and Zhiqiang Lin. Atomicity for agents: Exposing, exploiting, and mitigating TOCTOU vulnerabilities in browser-use agents, 2026. URL https://arxiv.org/abs/2603.00476.

Enqi Liu, Liyuan Pan, Zhi Gao, Yan Yang, Chenrui Shi, Yang Liu, Jingrong Wu, and Qing Li. Benchmarking and improving GUI agents in high-dynamic environments, 2026. URL https://arxiv.org/abs/2604.25380.

Kou Shi, Ziao Zhang, Shiting Huang, Avery Nie, Zhen Fang, Qiuchen Wang, Lin Chen, Huaian Chen, Zehui Chen, and Feng Zhao. AsyncTool: Evaluating the asynchronous function calling capability under multi-task scenarios, 2026. URL https://arxiv.org/abs/2605.27995.

Yule Wen, Yixin Ye, Yanzhe Zhang, Diyi Yang, and Hao Zhu. Real-time reasoning agents in evolving environments, 2025. URL https://arxiv.org/abs/2511.04898.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. OSWorld: Benchmarking multimodal agents for open-ended tasks in real computer environments. In *Advances in Neural Information Processing Systems (Datasets and Benchmarks Track)*, 2024. URL https://arxiv.org/abs/2404.07972.

Wenpeng Xu. Temporal UI state inconsistency in desktop GUI agents: Formalizing and defending against TOCTOU attacks on computer-use agents, 2026. URL https://arxiv.org/abs/2604.18860.

Mengqi Yuan, Zilong Zhou, Xinzhuang Xiong, et al. OSWorld 2.0: Benchmarking computer use agents on long-horizon real-world tasks, 2026. URL https://arxiv.org/abs/2606.29537.

Yikai Zhang, Siyu Yuan, Caiyu Hu, Kyle Richardson, Yanghua Xiao, and Jiangjie Chen. TimeArena: Shaping efficient multitasking language agents in a time-aware simulation. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3894–3916. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.acl-long.215. URL https://aclanthology.org/2024.acl-long.215/.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. WebArena: A realistic web environment for building autonomous agents. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2307.13854.

Henry Peng Zou, Chunyu Miao, Wei-Chieh Huang, Yankai Chen, Yue Zhou, Hanrong Zhang, Yaozu Wu, Liancheng Fang, Zhengyao Gu, Zhen Zhang, Kening Zheng, Fangxin Wang, Yi Nian, Shanghao Li, Wenzhe Fan, Langzhou He, Weizhi Zhang, Xue Liu, and Philip S. Yu. When users change their mind: Evaluating interruptible agents in long-horizon web navigation, 2026. URL https://arxiv.org/abs/2604.00892.

A Evaluation Protocol

Task Instantiation. The evaluated tasks use an executable, resettable shopping environment. The agent and NPCs draw from the same inventory, while carts and orders are recorded separately for each participant. Task construction binds instructions, initial resources, NPC targets, and completion checks to the same products and variants. Verification checks products, variants, sellers, quantities, and spending limits against the agent's own orders. Purchases from several orders may jointly satisfy a task, and a combined order may complete several goals.

Event Configuration. NPCs are scripted shoppers whose purchase attempts follow a fixed schedule in logical time. In single-goal competition, the schedule targets the required item. In four-goal planning, two streams of five attempts target the two contested goals, at logical intervals $(3, 6, 9, 12, 18)$ and $(15, 18, 21, 24, 30)$ with purchase quantities $(1, 2, 1, 2, 1)$ in each stream, and each attempt comes from a distinct NPC. At the shared interval 18, the first stream precedes the second. Each attempt follows normal transaction preconditions and is rejected when stock is insufficient. Conditions that vary importance share the NPC schedule, resource settings, and completion criteria.

Memory Configuration. The update in Equation 4 has a character capacity $C = 2{,}000$, with $|M_k|_{\text{chars}} \leq C$. Each source task permits one update that keeps, appends to, rewrites, or clears the memory. The update receives the previous memory, public task history, and a fixed instruction. The downstream environment is reset while the assigned memory is retained. Memory and model parameters remain fixed throughout each downstream test.

Evaluation Instances. A base task specifies the goals, initial resources, and constraints. Conditions vary NPC activity or stated importance for that task, and repetitions are separate executions of the same model, task, and condition. Single-goal competition, four-goal planning, and attribute-based selection contribute $8 \times 2 + 8 \times 3 + 8 \times 2 = 56$ evaluation instances. Three reference-model repetitions and one repetition for each of three additional models yield 336 planned runs. Two-goal planning adds nine evaluation instances and 27 reference-model runs. Each memory comparison contributes three conditions on one downstream task, bringing the total to 71 evaluation instances. With eight repetitions per condition, the memory comparisons contain 48 condition records drawn from 32 distinct downstream runs, since No memory and Dynamic source/One task are shared. Source tasks are counted separately from downstream tests.

Valid Runs. Rates use valid runs, in which the environment and protocol operate as specified. Runs interrupted by evaluation infrastructure are excluded. In single-goal competition, four-goal planning, and attribute-based selection, a run that ends at the request cap or action budget with unmet goals counts as a failure. The reference model has 24 valid runs per condition for single-goal competition and attribute-based selection, and 22, 19, and 20 of 24 planned runs for M1 Static, M1 Dynamic, and M2 Dynamic. Of the 168 additional-model runs, 166 are valid. Two-goal planning and continual-learning evaluations treat runs that reach the run limit before termination as censored and exclude them. In two-goal planning, seven of the 27 planned runs are censored and three are interrupted by infrastructure, leaving 9, 3, and 5 valid runs for M1 Static, M1 Dynamic, and M2 Dynamic. Of the 32 distinct downstream memory runs, six are censored (one No memory run, two Static source runs, one run shared by Dynamic source and One task, and two Three tasks runs), and the remaining 26 are valid. Runs that end with an invalid action or parse failure remain valid failures in all task families. Condition means use all valid runs, and paired differences use pairs in which both runs are valid.

Table 4: Experimental conditions.

Condition	**Definition**
Single-goal competition	
Static	One purchase goal without NPC competition.
Dynamic	Same goal, initial state, and budget, with scheduled NPC purchases.
Multi-goal planning	
M1 Static	Original goal importance, without NPC competition.
M1 Dynamic	Original goal importance, with NPC competition.
M2 Dynamic	Changed goal importance, with the M1 Dynamic schedule.
Attribute-based selection	
Ratings count / Current price	Select the product with more ratings or the lower price from the same pair.
Continual learning	
No memory	No retained experience before the downstream test.
Static source / Dynamic source	Memory formed from a Static or Dynamic source task.
One task / Three tasks	Memory after one or three Dynamic source tasks.

A.1 Metric Calculation

Let $c_i(\tau) \in \{0, 1\}$ indicate whether goal g_i is completed in trace τ, and let $q(\tau) \in \{0, 1\}$ indicate whether all task-level constraints are satisfied. For a task with m required goals, the success indicator is

$$Y(\tau) = q(\tau) \prod_{i=1}^{m} c_i(\tau). \tag{5}$$

Table 5: Common agent settings.

Setting	Configuration
Environment	Resettable shopping environment with shared inventory and participant-specific carts and orders.
Observation	Page URL, viewport, DOM, accessibility tree, task information, and a screenshot when visual input is enabled.
Within-run history	Up to three compact interaction turns.
Action	One structured browser action per turn.
Starting state	Home page of a freshly reset environment.
Hidden information	Future event schedule, NPC private state, verifier internals, and future states.
Memory	Absent in the main tasks and assigned by condition in downstream tests.
Input limit	224k tokens, with the High reasoning setting.
Action budget	60 steps for single-goal tasks and 100 steps for multi-goal and selection tasks, with a fixed request cap.
Event timing	Due NPC attempts are processed after action selection and before execution.

For an evaluation group of $N > 0$ valid runs, let τ_r denote the trace of run r, with m_r required goals and action budget B_r. Its accepted action count is $L_r \leq B_r$. Let $b(\tau_r) \in \{0, 1\}$ indicate at least one realized, unauthorized commerce side effect. The four metrics are

$$\begin{aligned} \mathrm{TSR} &= \frac{100}{N}\sum_{r=1}^{N} Y(\tau_r), \qquad \mathrm{GCR} = \frac{100}{N}\sum_{r=1}^{N}\frac{1}{m_r}\sum_{i=1}^{m_r} c_i(\tau_r), \\ \mathrm{UBR} &= \frac{100}{N}\sum_{r=1}^{N} b(\tau_r), \qquad \mathrm{Actions} = \frac{1}{N}\sum_{r=1}^{N} L_r. \end{aligned} \tag{6}$$

Each run receives equal weight in GCR and contributes at most once to UBR. Task failure alone does not count as an unintended behavior.

Paired Comparisons. A pair consists of the same base task and repetition in two conditions. Differences are computed within pairs and averaged, with rate differences in percentage points.

B Additional Results

B.1 Resource Use

Table 6 reports mean resource use by task condition, with model tokens following each provider's total-token convention, and Figure 5b summarizes the per-run means. Competition raises the cost of single-goal tasks from 14.0 to 23.7 actions, from 1.17M to 1.87M model tokens, and from 118.8 to 335.1 seconds per run, with 9.6 more actions across 24 pairs. In four-goal planning, M1 Dynamic and M2 Dynamic use 55.6 and 88.9 actions per run, compared with 50.7 for M1 Static, while completing almost no goals. M1 Dynamic uses 6.5 more actions than M1 Static across 19 pairs, and M2 Dynamic uses 34.7 more actions than M1 Dynamic across 18 pairs.

Continual Learning. For downstream tests with memory, Static source uses 1,614,879 model tokens and 455.9 seconds per run, compared with 874,691 tokens and 160.2 seconds for Dynamic source. Increasing memory from one to three source tasks lowers the mean from 11.9 to 11.5 actions, from 874,691 to 801,308 tokens, and from 160.2 to 153.8 seconds.

Table 7 reports totals per memory condition. Source gives the tokens of the source tasks, Update the input tokens of memory updates, and Test the input tokens of all downstream runs. USD is the settled downstream cost, and Time sums active downstream execution. Source tasks and memory updates add at most 5,698 and 5,520 tokens, below 0.05% of the downstream input tokens in every condition, so downstream execution determines the cost of each memory condition. Static source costs 11.57 USD, nearly twice the 6.09 USD of Dynamic source, in line with its higher per-run action count.

Table 6: Mean resource use per run for the reference model, by task condition (left) and memory condition (right).

Condition	Actions	Tokens	Time (s)
Single-goal competition			
Static	14.0	1,169,551	118.8
Dynamic	23.7	1,871,009	335.1
Four-goal planning			
M1 Static	50.7	4,718,785	642.8
M1 Dynamic	55.6	5,755,926	1,128.1
M2 Dynamic	88.9	8,856,909	1,674.5
Attribute-based selection			
Ratings count	74.8	4,550,896	606.7
Current price	68.8	4,814,922	573.6

Condition	Actions	Tokens	Time (s)
Memory source			
No memory	10.0	849,669	140.2
Static source	23.3	1,614,879	455.9
Dynamic source	11.9	874,691	160.2
Memory accumulation			
No memory	10.0	849,669	140.2
One task	11.9	874,691	160.2
Three tasks	11.5	801,308	153.8

Table 7: Resource totals for source tasks, memory updates, and downstream tests.

Condition	Source	Update	Test	USD	Time (s)
Memory source					
No memory	0	0	12,004,667	5.90	3,447.7
Static source	5,191	5,191	22,140,495	11.57	8,224.4
Dynamic source	5,193	5,193	12,483,778	6.09	3,518.6
Memory accumulation					
No memory	0	0	12,004,667	5.90	3,447.7
One task	5,193	5,193	12,483,778	6.09	3,518.6
Three tasks	5,698	5,520	17,155,048	8.83	5,994.2

Table 8: Successful runs across models and task conditions on the first repetition.

Condition	DeepSeek	GPT	Gemini	Qwen
Single-goal competition				
Static	7/8	4/8	0/8	7/8
Dynamic	5/8	1/8	0/8	4/8
Four-goal planning				
M1 Static	5/8	0/7	0/8	0/8
M1 Dynamic	0/8	0/8	0/8	0/8
M2 Dynamic	0/8	0/8	0/8	0/8
Attribute-based selection				
Ratings count	0/8	2/8	0/8	0/7
Current price	2/8	0/8	0/8	0/8

B.2 Model Results

Table 8 gives successes and valid runs on the first repetition. Table 9 reports outcomes and resource use by task family for the additional models. Success of the additional models concentrates in single-goal tasks. Qwen3.8-Flash completes 11 of 16 single-goal runs and GPT-5.6 Luna 5 of 16, while neither completes a four-goal task. Both use fewer than 20 actions per four-goal run, compared with 50.7 to 88.9 for the reference model, so their four-goal runs end much earlier. Qwen3.8-Flash uses 23.0 actions per selection run without a success. Gemini 3.8 Flash averages 1.4 to 1.6 actions in every task family, ending its runs after one or two actions, and completes no run.

Table 9: Outcomes and mean resource use across additional models. N denotes valid runs, and TSR is a percentage.

Task family	N	Successes	TSR ↑	Actions	Model tokens	Time (s)
GPT-5.6 Luna						
Single-goal competition	16	5	31.3	20.6	1,365,043	188.7
Four-goal planning	23	0	0.0	17.9	1,784,921	240.1
Attribute-based selection	16	2	12.5	14.7	1,194,059	165.3
Gemini 3.8 Flash						
Single-goal competition	16	0	0.0	1.5	367,467	47.9
Four-goal planning	24	0	0.0	1.4	339,839	42.4
Attribute-based selection	16	0	0.0	1.6	384,654	57.1
Qwen3.8-Flash						
Single-goal competition	16	11	68.8	14.4	1,122,372	222.6
Four-goal planning	24	0	0.0	19.1	2,102,956	399.5
Attribute-based selection	15	0	0.0	23.0	1,724,549	403.1

C Detailed Analyses

C.1 Attribute-based Selection

This task holds two candidate products and their public attributes fixed while changing the requested selection criterion. *Ratings count* asks for the product with more ratings, and *Current price* asks for the product with the lower current price. The criteria select different products, and no NPC events occur. Table 1(c) reports end-to-end outcomes, and Table 3 reports choice-level results on the same runs. A choice is unassessable when the trace does not identify the selected product. Unassessable choices account for 18 of 24 Ratings count runs and 15 of 24 Current price runs, and no assessable run selects the wrong product. Both criteria use about 70 actions per run, and 27 of 38 failures reach the request cap or action budget. Selection failures therefore arise mainly while executing the order after the choice. The only unintended behavior occurs under Ratings count.

C.2 Memory Outcomes

Table 10 reports downstream outcomes by memory source and accumulation stage, with No memory shared by both comparisons. Every memory condition exceeds the 57.1% TSR of No memory, and no condition records an unintended behavior. GCR equals TSR in every condition. Static source needs 23.3 actions per run, more than twice No memory and Dynamic source, so its gain in success comes with a higher execution cost.

Table 10: Downstream outcomes by memory source and accumulation stage. Completed gives successes/valid runs. Rates are percentages.

Condition	Completed	TSR ↑	GCR ↑	UBR ↓	Actions
(a) Memory source					
No memory	4/7	57.1	57.1	0.0	10.0
Static source	5/6	83.3	83.3	0.0	23.3
Dynamic source	7/7	100.0	100.0	0.0	11.9
(b) Memory accumulation					
No memory	4/7	57.1	57.1	0.0	10.0
One task	7/7	100.0	100.0	0.0	11.9
Three tasks	6/6	100.0	100.0	0.0	11.5

C.3 Paired Comparisons

Table 11 gives memory outcomes and costs on paired repetitions, where a pair shares the downstream task and repetition index. GCR equals TSR in every memory condition and no condition records an

unintended behavior, so the table omits both differences. The paired results agree with the condition means. Static source and Dynamic source both gain 33.3 pp over No memory, but Static source adds 13.7 actions and 730,528 tokens, whereas Dynamic source adds 1.2 actions and uses 3,080 fewer tokens. From One task to Three tasks, TSR is unchanged, and the pairs save 0.3 actions, 71,937 tokens, and 10.0 seconds on average.

Table 11: Paired memory outcomes and costs (first minus second). ΔTSR is in percentage points, tokens count input tokens, and time is in seconds.

Contrast	Pairs	ΔTSR	ΔActions	ΔTokens	ΔTime
Memory source					
Static source − No memory	6	+33.3	+13.7	+730,528	+325.7
Dynamic source − No memory	6	+33.3	+1.2	−3,080	+8.1
Dynamic source − Static source	5	+20.0	−13.8	−841,731	−359.0
Memory accumulation					
One task − No memory	6	+33.3	+1.2	−3,080	+8.1
Three tasks − No memory	5	+40.0	+1.0	−90,275	+0.9
Three tasks − One task	6	0.0	−0.3	−71,937	−10.0

C.4 Memory Contents

Table 12 records memory lengths and content changes. Static source and Dynamic source each append a record to empty memory, and the two- and three-task updates rewrite the preceding record. Dynamic source and One task share the same memory. The two-task state is an intermediate record and is not evaluated downstream. Every memory state stays at or below 319 characters, 16% of the 2,000-character capacity. The two rewrites add 47 and 4 characters while revising product and order information, and downstream success stays at 100% from One task to Three tasks.

Table 12: Memory updates and downstream completion. Length is in characters. Completed gives successes/valid runs.

Memory	Update	Length	Completed	Content change
Memory source				
Static source	Append	235	5/6	Creates a record from a Static source task.
Dynamic source	Append	268	7/7	Creates a record from a Dynamic source task.
Memory accumulation				
One task	Append	268	7/7	Shares the Dynamic source record.
Two tasks	Rewrite	315	Audit only	Revises product and order information.
Three tasks	Rewrite	319	6/6	Revises the preceding record.

D Failure Analysis

D.1 Terminal Modes

Figure 5a and Table 13 classify valid failed runs by how they end: normally without success, with an invalid action or parse failure, or at the request cap or action budget. In single-goal competition, 12 of 17 failures end normally with the order unfinished, and five reach the request cap or action budget. In four-goal planning, 24 of 46 failures reach the request cap or action budget, 19 end with an invalid action or parse failure, and only three end normally. In attribute-based selection, 27 of 38 failures reach the request cap or action budget and nine end with an invalid action or parse failure. Most selection failures thus use up the run without producing a correct order, in line with the gap between correct choices and correct orders in Table 3. Memory-source failures consist of three No memory runs and one Static source run, and memory-accumulation failures are the same three No memory runs. All memory failures end normally.

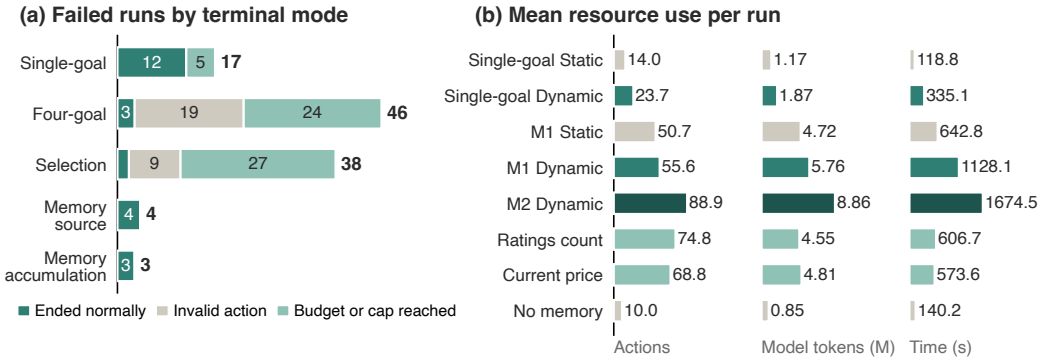

Figure 5: Terminal modes of valid failed runs and mean resource use per run (reference model).

Table 13: Valid failed runs by terminal mode for the reference model.

Terminal mode	**Single-goal**	**Four-goal**	**Selection**	**Memory source**	**Memory accumulation**
Ended normally	12	3	2	4	3
Invalid action	0	19	9	0	0
Budget or cap reached	5	24	27	0	0
Other agent failure	0	0	0	0	0
Total	17	46	38	4	3

D.2 Case Studies

Matched runs illustrate three patterns. (i) *Competition.* The Fire Kids order succeeds in the Static condition. In the Dynamic condition, NPCs consume the stock and the agent exhausts its action budget without completing the order. Nelko Label shows the same pattern. (ii) *Selection.* In one matched task, changing the criterion from Ratings count to Current price changes the product choice correctly, and both orders succeed. Other runs select correctly but fail through duplicate purchases. (iii) *Memory.* Without memory, the agent adds the requested quantity to the cart but stops before ordering. With Static source memory, it proceeds through checkout and completes the order.

E Agent Prompts

All models receive the same prompt, composed of a task-neutral preamble, the task instruction, and the action protocol. The prompt contains no strategy, priority, or timing hint beyond the task instruction itself. The preamble and action protocol are reproduced below; {`task instruction`} marks where the instruction is inserted.

Listing 1: System prompt shared by all models and conditions.

```
Complete the task using the web page. Use `key("F5")` if you choose to reload
the page before acting. `type(text)` enters text into the currently focused
element, so click an input before typing.

Task:
{task instruction}

Response format (required): return exactly one JSON object for one executable action, and
     nothing else. Do not use a Markdown code fence and do not include any explanation, prose,
      or text before or after the object. Every key listed by the schema is required: include
     all eight keys (`kind`, `x`, `y`, `text`, `combo`, `dx`, `dy`, `answer`) in every object,
     and set every key you are not using to `null`. Valid actions use `kind` equal to `click`,
     `double_click`, `type`, `key`, `scroll`, `wait`, or `done`; use non-null `x` and `y` for
     clicks, non-null `text` for type, non-null `combo` for key, non-null `answer` for done,
     and for `scroll` both `dx` and `dy` are required non-null integers (write `0` for the axis
      you are not scrolling). The `combo` must contain only a key name or '+'-separated key
     names. Do not put prose, JSON, error messages, whitespace, or punctuation in `combo`. Use
     `Control` as the preferred modifier spelling; `CTRL` remains a compatibility alias.
```

```
Task-neutral examples (copy the shape and the key set, not the example values):
Observation: a visible button is at viewport coordinates (120, 80).
Valid action: {"kind":"click","x":120,"y":80,"text":null,"combo":null,"dx":null,"dy":null,"
    answer":null}
Observation: no action is currently needed.
Valid action: {"kind":"wait","x":null,"y":null,"text":null,"combo":null,"dx":null,"dy":null,"
    answer":null}
For the current observation, output exactly one action JSON object.
Do not output a plan, action list, multiple actions, or a future done action.
Wait for the next observation before choosing any later action.
```

Listing 2 gives the M1 instruction of a four-goal task. The Fire HD and Owala goals face NPC competition, and the Anker and Patch goals do not. The business notes mark the Anker and Owala goals as important, so the four goals cover the four combinations of importance and urgency. The M2 instruction keeps the four requests and exchanges the business notes within each urgency group.

Listing 2: M1 task instruction of a four-goal task.

```
You are purchasing product samples and equipment for a small studio's client demonstration
    project and its internal demonstration area. Complete all four purchases below in this
    shopping session. The business notes explain the consequences if a request is not
    fulfilled; they do not cancel any of the four requests. Buy exactly the specified
    quantities and variants from the specified featured sellers. Do not substitute products
    or make extra or duplicate purchases. Separate orders or a combined order are both
    acceptable.

1. Purchase one 36-count pack of Mighty Patch from the featured Hero Cosmetics Store offer.
Business note: This pack is an optional sample for an internal product-photo rehearsal. The
    current client deliverables do not depend on that rehearsal.

2. Purchase one Anker Nano charger, Standard variant, from the featured Anker Store offer.
Business note: This charger is needed to power equipment for an agreed client demonstration.
    No suitable spare is available, so that part of the client delivery cannot be completed
    without this purchase.

3. Purchase two Fire HD 10 tablets, 32 GB · Tablet only variant, from the featured Amazon.com
    offer.
Business note: These tablets are optional upgrades for the studio's internal information
    display. Existing devices can continue to support the currently agreed work.

4. Purchase exactly two Owala FreeSip bottles, 32 oz · Blue variant, from the featured Owala
    Store offer. Spend no more than $70.00 total on these two bottles.
Business note: These exact bottles are required samples for an agreed hydration-product
    demonstration. There are no approved substitute samples, so that part of the client
    delivery cannot be completed without this purchase.
```

In continual-learning evaluations, the memory update after a source task receives three inputs: the previous memory, the public history of the source task, and a fixed update instruction. It returns a JSON object with the keys `operation` and `content`, where the operation is `KEEP`, `APPEND`, `REWRITE`, or `CLEAR`, and the resulting memory is limited to 2,000 characters. The downstream prompt adds the assigned memory to the prompt in Listing 1.

F LIMITATIONS

Environment Scope. The evaluated tasks use one shopping environment, where purchases give a direct and verifiable form of resource competition. The task constructor binds goals, NPC targets, and completion checks through declared resources, so other applications with shared resources can be instantiated in the same way.

Scripted Actors. NPCs follow fixed schedules and do not react to the agent. This keeps the competition identical across matched conditions and models, so paired differences reflect the agent's behavior. Logical time advances with accepted actions, so model latency does not change when NPC attempts occur. This removes inference speed as a factor in model comparisons. Adaptive actors and latency-dependent timing can be added through the same event interface.

Sample Size. The reference model runs three repetitions per evaluation instance and each additional model runs one. The memory comparisons use one downstream task with five or six pairs per contrast. We therefore report counts with every rate and compute differences on matched runs. The main conclusions rest on large differences, such as the 68.4 pp decrease across 19 pairs in four-goal planning and the decline under competition for every model with nonzero Static success. Smaller differences, such as the 20.0 pp contrast between memory sources across five pairs, are reported descriptively.

Goal Load. Under competition, no evaluated model completes a four-goal task, so the importance comparison at this load is measured through goal completion and actions. The two-goal setting restores success under competition and gives the corresponding comparison on task success.

Memory Form. Memory is a single text record of at most 2,000 characters, updated once per source task, with model parameters fixed. This form keeps every memory state inspectable, as the recorded lengths and content changes show. Retrieval over multiple records and parameter updates are other forms of retained experience, and the downstream protocol evaluates them without change.