Complete Cyclic Subtask Graphs for Tool-Using LLM Agents: Flexibility, Cost, and Bottlenecks in Long-Horizon Workflows

Anonymous authors
Paper under double-blind review

Abstract

Long-horizon tool-using tasks sometimes benefit from revisiting earlier subtasks, but explicit revisitation also adds routing, coordination, and token cost. We study complete cyclic subtask graphs for large language model (LLM) agents: a workflow controller in which executable subtasks are fully connected and a unified state-analysis-and-routing agent selects transitions from natural-language criteria. We evaluate task-specific (Spec-Cyc) and benchmark-generic (Gen-Cyc) cyclic graphs on TextCraft, ALFWorld, and Finance-Agent against ReAct and dependency-directed acyclic workflows. The results expose three task regimes. TextCraft behaves like a prerequisite-chain setting, where cyclic routing often adds overhead. ALFWorld behaves like a partially observable recovery setting, where explicit revisitation improves exploration and success. Finance-Agent behaves like an open-ended evidence-synthesis setting, where retrieval, grounding, and synthesis bottlenecks limit all controllers. We add a task-regime selection matrix, fault-injection robustness analysis, token-cost accounting, and scaling discussion. Overall, complete cyclic subtask graphs are best understood as a diagnostic workflow-control tool: they quantify when flexible backtracking is worth its cost and when simpler or sparsified controllers are preferable.

1 Introduction

Large language models (LLMs) are increasingly deployed as tool-using agents that interact with external environments such as the web, APIs, simulators, and embodied text worlds. Recent systems range from ReAct-style closed-loop reasoning/action to explicit planner-executor decompositions and state- or graph-based controllers Yao et al. (2022); Erdogan et al. (2025); Wu et al. (2024); Zhuge et al. (2024). In these long-horizon settings, errors compound across multi-step trajectories and partial progress must often be preserved, motivating controllers that can verify state, replan, and recover rather than only continue forward. Recent work similarly treats LLM agents as long-horizon learning systems whose reliability depends on roles, execution, external knowledge, reflection, and workflow planning rather than on prompting alone (Chen et al., 2026). The same design pressure appears in analyses of LLM reasoning failures and efficient reasoning, which emphasize both robustness failures and the token/latency cost of longer reasoning traces (Song et al., 2026; Sui et al., 2025).

Despite this shift, revisitation is rarely treated as a first-class capability: "going back" is often implemented through retries, edited step lists, or ad hoc recovery states rather than through an explicit workflow representation. This distinction matters because control-flow structure shapes both behavior (e.g., how readily an agent can backtrack or explore) and diagnosis (e.g., whether repeated behavior reflects productive recovery or unproductive looping).

We study complete cyclic task graphs as an explicit workflow representation for tool-using agents. The task is decomposed into subtask nodes, and a unified analyzer+router selects transitions by evaluating natural-language edge criteria against a rolling trajectory state (Sec. 3). We instantiate Spec-Cyc (task-specific graphs) and Gen-Cyc (benchmark-generic graphs reused across instances) to test whether cyclic revisitation

*Large-language-model assistance was used for language editing. All ideas, claims, experiments, results, and final manuscript decisions are the authors' responsibility.

acts as a transferable control policy rather than a per-task artifact. Our goal is not to claim complete connectivity is universally optimal; rather, we use it as a lens to understand when revisitation helps and when it degenerates into thrashing.

This work is best viewed as a controlled study of workflow flexibility versus coordination cost rather than as a pure leaderboard comparison. Our complete cyclic graph is a deliberately permissive multi-agent regime: because every subtask can transition to every other, the controller is not confined to a predefined trajectory, and revisitation, recovery, and exploration are always structurally available. The central scientific question is therefore when that added control-flow freedom buys meaningful recovery relative to a simpler ReAct agent, and when it mostly adds coordination and token overhead. This framing also motivates our model choices. We intentionally center the controlled comparisons on `gpt-4o-mini`, while also comparing against stronger `gpt-5-mini` settings, because once stronger single-agent or high-tier multi-agent configurations approach saturation, there is far less headroom to observe the recovery effects of orchestration itself.

We define the graph at the level of executable subtasks of the task itself. Each node corresponds to an action-producing subproblem, and the router selects among those subtask-specific action spaces using natural-language transition criteria. Because every subtask can route to every other subtask, nothing is pruned a priori. We use this deliberately maximally flexible design as an experimental lens: it lets us study when unrestricted revisitation improves recovery or exploration, when it induces thrashing or routing hallucinations, and how those effects interact with router quality, tool exposure, robustness perturbations, and token cost.

We evaluate on three structurally distinct long-horizon benchmarks TextCraft, ALFWorld, and Finance-Agent against ReAct and a dependency-direct acyclic graph (DAG or DepDAG) workflow, with ablations over orchestration tiering, tool access (generalist full-tool executors vs tool-restricted specialists), optional n-shot successful-trajectory conditioning (akin to workflow/routine induction and reuse for long-horizon agents Wang et al. (2025b)), and robustness under control-flow perturbations.

The three benchmarks stress different failure modes. TextCraft primarily rewards correct execution of a prerequisite chain once that chain is identified, so extra routing flexibility can become coordination overhead. ALFWorld is partially observable and interaction-heavy: agents must search, recover from mis-localization, and revisit earlier subtasks after failed or misleading actions, so explicit backtracking is often useful. Finance-Agent is open-world and evidence-heavy; here low success across all methods suggests that retrieval, grounding, and evidence synthesis remain major bottlenecks, so cyclic control alone yields only modest gains. This difference in structure helps explain why cyclic control is most compelling in ALFWorld, mixed in Finance-Agent, and often less efficient than a single ReAct agent in TextCraft, consistent with recent findings that multi-agent coordination can underperform single-agent execution on sequential, dependency-heavy planning tasks such as PlanCraft Kim et al. (2025).

Contributions.

- We formalize complete cyclic subtask graphs for tool-using large language model agents, a deliberately maximally flexible workflow controller in which executable subtasks are fully connected and transitions are selected by a unified analyzer+router using natural-language criteria.
- We introduce a task-regime taxonomy and workflow-selection matrix that maps prerequisite density, uncertainty, and repair requirements to suitable workflow controllers.
- We evaluate two regimes, Spec-Cyc (task-specific) and Gen-Cyc (benchmark-generic and reusable), against ReAct and a dependency-directed acyclic graph workflow across three benchmarks chosen to stress different failure modes.
- We report revisitation, unique-transition, exploration, token-cost, and fault-injection robustness metrics to make the flexibility–cost–reliability tradeoff explicit.

Taken together, our results suggest that making revisitation explicit through cyclic workflow control can be a useful and interpretable design point for long-horizon tool-using agents, while clarifying that performance hinges on orchestration quality and tool exposure, and that unrestricted revisitation can either enable recovery or amplify thrashing.

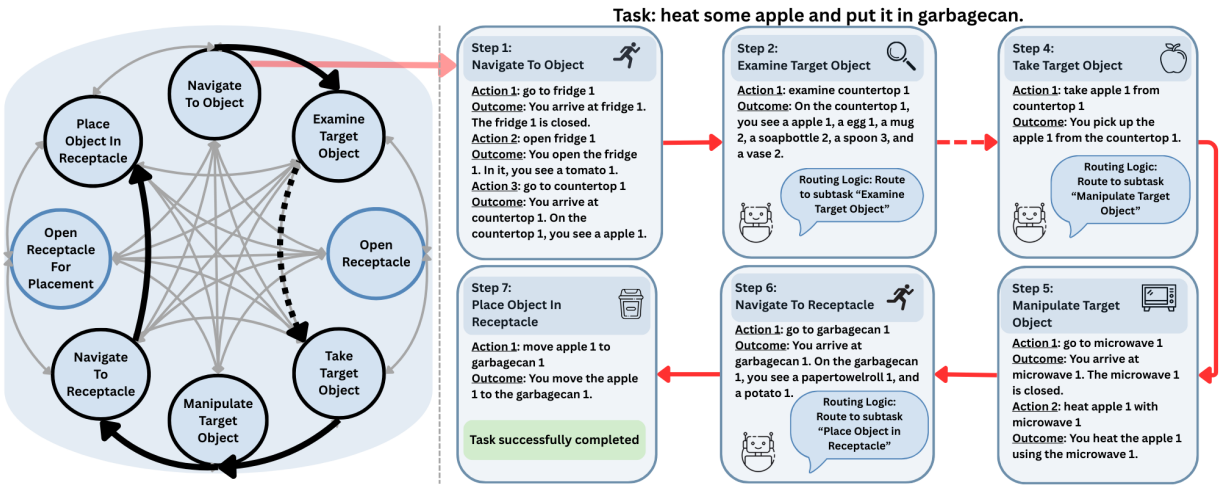

Figure 1: Diagram showing a complete cyclic graph from a real ALFWorld task experiment with executions and routing transitions (the routing criteria and some subtasks and routing logics are omitted for visual purposes). Dashed lines indicate that other subtasks were visited within the trajectory.

2 Related Work

Agentic LLM systems increasingly close the loop between reasoning, tool use, and external feedback. ReAct interleaves reasoning and actions during environment interaction Yao et al. (2022), Reflexion adds self-evaluation and memory across attempts Shinn et al. (2023), and related work studies alternative action interfaces such as executable code actions Wang et al. (2024). A broader line separates planning from execution for long-horizon tasks: Plan-and-Act studies planner–executor decomposition in web navigation Erdogan et al. (2025), ADaPT recursively decomposes subtasks and refines plans online when execution stalls Prasad et al. (2024), CoPE studies concurrent planning and acting under commitment constraints Coles et al. (2025), OPEx analyzes embodied agents by swapping Observer/Planner/Executor components Shi et al. (2024), and ReWOO decouples reasoning traces from tool observations for more efficient tool use Xu et al. (2023). Our work shares this closed-loop view, but focuses on a different design variable: the workflow representation itself. We ask what changes when the controller is an explicit complete cyclic subtask graph whose revisitation behavior can be directly measured.

Several recent frameworks make workflow control more explicit. AIME coordinates iterative planning and dispatch through centralized progress management Shi et al. (2025a), and AgentOrchestra maintains plan state through step creation, update, and deletion operations Zhang et al. (2025e). State-machine and graph-based approaches make the controller even more explicit: StateFlow represents task solving as a state machine with rule- or LLM-governed transitions Wu et al. (2024), MetaAgent uses a finite-state-machine backbone for constructing multi-agent systems Zhang et al. (2025f), and GPTSwarm frames language-agent systems as optimizable graphs Zhuge et al. (2024). Recent surveys of data-science agents, GUI agents, and test-time-compute search similarly emphasize the separation of planning, execution, reflection, external evidence, and routing policies (Chen et al., 2026; Zhang et al., 2025a; Li, 2025). Our contribution is narrower but complementary: rather than optimizing a general agent graph or hand-designing a finite state machine, we isolate cyclic subtask revisitation as the control-flow variable and measure when unrestricted revisitation improves success, exploration, and recovery relative to its cost.

A large body of work also studies how multiple LLM agents should be organized, communicate, and coordinate. Orchestration frameworks can dispatch specialized agents and integrate tool calls through centralized schedulers Song et al. (2025), while other systems treat multi-agent communication topology as a first-class design object, including G-Designer, MaAS, MAS-GPT, and large-scale DAG collaboration networks such as MacNet Zhang et al. (2024; 2025b); Ye et al. (2025); Qian et al. (2024). Multi-agent debate and structured interaction are likewise used to improve reasoning through aggregation or competition Smit

et al. (2024); Chen et al. (2024); Klein et al. (2024). These works show that coordination structure matters, but they typically focus on agent-to-agent communication or agent-graph organization. In contrast, our graph nodes are executable subtasks of the task itself, and the complete graph is used as a diagnostic workflow controller for studying recovery, exploration, and redundant revisitation.

Tool selection and specialization are another practical constraint in tool-using agents. ToolLLM/ToolBench, ToolRet, Meta-Tool/Meta-Bench, and Gorilla study API retrieval and open-world tool use, showing that narrowing candidate tools can reduce hallucination but introduces a retrieval dependency that can affect downstream success Qin et al. (2024); Shi et al. (2025b); Qin et al. (2025); Patil et al. (2024). Other systems argue for role or tool specialization to reduce interference between high-level reasoning and low-level execution Wang et al. (2025a); Song et al. (2024). Our experiments directly ablate this issue by comparing generalist full-tool executors against tool-restricted specialists within the same cyclic controller, allowing us to separate tool-selection effects from coordination and hand-off effects.

Finally, robustness and debugging are increasingly central in multi-agent systems, including fault-injected collaboration studies, failure attribution, and perturbation-based testing Huang et al. (2025); Zhang et al. (2025d;c); Ma et al. (2024). Our robustness study is aligned with this stress-testing perspective but targets control flow specifically: we inject random subtask redirections to test whether cyclic controllers can recover under disrupted routing, and we pair success rates with behavioral measures of revisitation and exploration.

3 Methodology

Given an objective O and tool/environment interface $\mathcal{U}$, we execute via a complete directed subtask graph in which each node is handled by a tool-using executor and a separate analyzer+router selects the next node by evaluating natural-language transition criteria on outgoing edges, making backtracking/revisitation always available.

3.1 Problem setting and time scales

We consider episodic interaction via tools $\mathcal{U}$. Let k index tool calls and let m index subtask segments: within segment m, executor α_{i_m} may issue multiple tool calls, and routing occurs only at segment boundaries.

3.2 Graph representation

A task graph is a labeled directed graph $G = (T, E, C)$ where $T = \{t_1, \ldots, t_n\}$ is a set of subtasks and $E = T \times T$ is the complete directed edge set (self-loops included). Each edge $(t_i, t_j) \in E$ carries a natural-language transition criterion $c_{i\to j} = C(t_i, t_j) \in \Sigma$ describing when control should move from t_i to t_j (e.g., "if verification fails, return to Execute"), where Σ denotes natural-language strings. We use t_i for the static node identity and i_m for the runtime-selected node index at segment m (full toy trace in a footnote).[1]

Although E is complete by construction, the realized transition sequence is induced by router decisions and may be cyclic or acyclic; the goal is to make revisitation available and inspectable rather than to force cycles.

3.2.1 Agents and memories

We use three roles: planner π (constructs (T, C)), executors $\{\alpha_i\}_{i=1}^n$ (each executes subtask t_i with tool set $\mathcal{U}_i \subseteq \mathcal{U}$), and a unified analyzer+router ρ (updates trajectory memory and selects the next node by criterion evaluation).

Let $\Delta k_m \in \mathbb{N}$ denote the number of tool calls issued by executor α_{i_m} within segment m. We maintain a global tool-call counter k_m (at segment boundaries) with update

$$k_{m+1} = k_m + \Delta k_m, \qquad k_0 = 0. \tag{1}$$

[1]Toy trace: let $T = \{t_1, t_2, t_3\}$ with $t_1 = \textsc{Search}$, $t_2 = \textsc{Plan}$, $t_3 = \textsc{Execute}$. A possible run selects indices $i_0 = 2$ (so $t_{i_0} = \textsc{Plan}$), $i_1 = 1$ (so $t_{i_1} = \textsc{Search}$), $i_2 = 1$ (self-loop on $\textsc{Search}$), and $i_3 = 3$ (so $t_{i_3} = \textsc{Execute}$).

Within each segment we enforce a local (per-segment) budget $\Delta k_m \leq C_l$. At the end of segment m we maintain

$$S_m := (k_m, i_m, H_m, M_m), \tag{2}$$

where k_m is the current global tool-call count, $i_m \in \{1, \ldots, n\}$ is the current node index, H_m is the segment-level tool input/output summary (and raw trace) passed from the executor, and M_m is a structured rolling memory maintained by ρ and is passed to the next executor as soft guidance (the controller-level state variables and execution budgets are summarized in Appendix Secs. B and B.2).

3.2.2 Graph construction: Spec-Cyc vs. Gen-Cyc

We instantiate two construction regimes. Spec-Cyc constructs a task-specific graph per instance:

$$\pi_{\text{spec}} : (O, \mathcal{B}, \mathcal{D}) \mapsto (T, C), \tag{3}$$

where $\mathcal{B}$ is a benchmark description and $\mathcal{D} = \{d^{(r)}\}_{r=1}^{n}$ is the set of optional n demonstration summaries of successful episode trajectories, and subtasks/criteria may reference instance-specific entities and failure modes. Gen-Cyc constructs a benchmark-generic graph,

$$\pi_{\text{gen}} : (\mathcal{B}, \mathcal{D}) \mapsto (T, C) \tag{4}$$

once per benchmark family and reuses it across instances to test transfer of cyclic revisitation rules.

3.2.3 Runtime execution and routing

Execution alternates between (i) running the current executor and (ii) routing to the next node. Given (S_m, t_{i_m}), the executor runs an inner tool-call loop and the router updates memory

$$M_{m+1} = \textsc{Update}_\rho(M_m, t_{i_m}, H_m), \tag{5}$$

then selects the next node by criterion evaluation

$$i_{m+1} = \textsc{Select}_\rho(M_{m+1}, \{c_{i_m \to j}\}_{j=1}^{n}), \tag{6}$$

where self-loops ($i_{m+1} = i_m$) correspond to repeating the current subtask when criteria indicate "continue current work." Termination occurs if the environment signals success or the global tool-call budget is exhausted ($k_{m+1} \geq C_g$).

3.3 Task-regime taxonomy and workflow selection

The complete graph is a diagnostic instrument rather than a universal deployment prescription. Its purpose is to expose the reliability benefit and cost of unrestricted revisitation before a practitioner commits to a sparser controller. We therefore characterize each task family by an *workflow signature*: prerequisite density, state uncertainty, observability, and repair semantics. Table 1 summarizes the resulting workflow-selection matrix.

This matrix also explains the empirical split observed later. TextCraft has the workflow signature of a prerequisite-chain automation problem: after the correct dependency order is known, extra routing freedom is often a tax. ALFWorld has the signature of a fault-tolerant interactive controller: repeated search, verification, and correction are part of normal operation. Finance-Agent has the signature of an evidence-synthesis pipeline, where the limiting factor may be the quality of retrieved evidence rather than the local control policy.

The complete graph should therefore be read as a diagnostic workflow controller rather than a replacement for every directed acyclic graph, finite-state machine, or ReAct-style loop. Static DAGs remain preferable when dependencies are stable, state machines are preferable when legal states can be enumerated, and ReAct remains attractive when the workflow can be inferred internally without explicit subtask routing. The complete cyclic graph is most useful when recovery routes are uncertain before deployment; after measurement, the observed behavior can be distilled into a sparse graph or state-machine variant that preserves useful repair routes while reducing routing cost.

Table 1: Workflow selection matrix for choosing a control architecture from task structure. The complete cyclic subtask graph is most valuable when recovery, verification, or exploration must be available at many points in the workflow; it is mainly diagnostic when the task is a stable prerequisite chain.

Task regime	Prerequisite density	Uncertainty / observability	Dominant failure mode	Recommended workflow control
Prerequisite-chain execution (e.g., TextCraft)	High and mostly known	Low to moderate; state is inspectable	Skipping or misordering required intermediates	ReAct or static directed acyclic graph; use complete cyclic graph as a diagnostic stress test, then prune.
Stateful recovery and exploration (e.g., ALFWorld)	Moderate; ordering depends on discovered state	High; partial observability and misleading actions	Mis-localization, failed manipulation, stale state, need to revisit search/verification	Complete cyclic graph or a state machine with explicit recovery loops; retain backtracking and verification edges.
Open-world evidence synthesis (e.g., Finance-Agent)	Moderate and data-dependent	High; retrieved evidence is noisy and incomplete	Retrieval miss, source mismatch, weak grounding, unsupported synthesis	Workflow control plus evidence-quality gates, retrieval auditing, and final-answer verification; cyclic routing alone is insufficient.
Mixed enterprise automation	Heterogeneous across subtasks	Variable; external tools and humans may change state	Local failures propagate across APIs, documents, and decisions	Start with a complete cyclic diagnostic graph, measure recovery/cost, then learn or hand-prune a sparse deployment graph.

3.4 Computational overhead and scaling

A complete graph over n subtasks stores $O(n^2)$ natural-language transition criteria, but a router positioned at a single node evaluates only the outgoing candidate set for that segment. If all outgoing criteria are included in one router call, the number of large-language-model calls per segment is constant, while the input length grows linearly in n and in the average criterion length. With bounded rolling memory, the routing input-token cost at segment m can be summarized as

$$C_{\text{route}}(m) \in O\big(n(\bar{\ell}_c + \bar{\ell}_t) + |M_m|\big)\,, \tag{7}$$

where $\bar{\ell}_c$ is the average transition-criterion length, $\bar{\ell}_t$ is the average subtask-name/description length, and $|M_m|$ is the bounded router memory supplied to the decision.

In our experiments, the extracted graphs remain small: across Spec-Cyc and Gen-Cyc settings, TextCraft uses roughly 4–7 nodes, ALFWorld roughly 5–9 nodes, and Finance-Agent roughly 3–7 nodes, with short natural-language criteria of about 7–33 words per edge (Appendix Tables 15 and 17). Thus, complete routing is tractable at the benchmark scale studied here; the real scaling concern is not the present graph size, but larger enterprise workflows where tens or hundreds of subtasks would increase router prompt length, criterion maintenance burden, and the chance of spurious transitions. The practical path suggested by our results is cost-aware sparsification: begin with complete connectivity when the recovery structure is unknown, then

remove never-useful transitions, downweight loop-inducing edges, and retrieve over candidate transitions before routing.

4 Experiments

4.1 Experimental Setup

This section describes the evaluation benchmarks, agent configurations, ablation studies, and reported metrics. Unless otherwise specified, every configuration in every ablation is run with three random seeds; we report mean and standard deviation over seeds. All agents are run using either `gpt-4o-mini` and/or `gpt-5-mini`.

4.1.1 Benchmarks

We evaluate on three long-horizon tool-using benchmarks: TextCraft (depth 2/3/4 crafting tasks with increasing horizon) Prasad et al. (2024), ALFWorld (interactive household tasks emphasizing exploration and recovery) Shridhar et al. (2021), and Finance-Agent (open-world financial question answering with web research and evidence aggregation) Bigeard et al. (2025). A run is successful if the benchmark goal is achieved within the episode tool-call budget. For Finance-Agent, the publicly available validation set contains 50 cases; in our protocol, a fixed subset is used only for optional n-shot successful-trajectory summary construction, and the remainder is used for held-out evaluation. Exact split details for all benchmarks, including exceptions such as TextCraft-4 and the benchmark-provided ALFWorld partitions, are given in Appendix B.1.

4.1.2 Methods Compared

We compare two baselines to our methods: ReAct (`4o-mini` and `5-mini`), a tool-using agent that interleaves reasoning and actions in a closed loop; DepDAG (`4o-mini` all), a dependency-structured workflow that executes subtasks under a directed acyclic (forward-biased) control graph. DepDAG is a forward-only dependency workflow: a planner LLM generates an ordered subtask list. An analyzer LLM, ρ is still invoked between steps, but its primary role is to use its structured memory and provide soft guidance to the next executor, rather than to select among loops or recovery routes. We present two complete cyclic workflow variants, Spec-Cyc and Gen-Cyc, which represent tasks as fully connected subtask graphs where each node is executed by a tool-using executor and a unified state-analysis-and-routing controller selects transitions by evaluating natural-language criteria. Spec-Cyc is generated per task instance, while Gen-Cyc is generated once per benchmark family and reused to test whether cyclic revisitation transfers across tasks.

4.1.3 Metrics

We report success rate (SR), successful-episode tool calls (TC), and efficiency STR = SR/TC. For cyclic runs, we additionally report subtask-visitation statistics on successful episodes: average total subtask visits (ATS), average unique subtasks visited (AUS), and average revisitation ASR = ATS − AUS. We also report unique transitions (UT) over full runs. For fault-injection experiments, we report the success-retention ratio and successful-episode tool-call overhead,

$$\mathrm{SRR} = \frac{\mathrm{SR}_{\mathrm{fault}}}{\mathrm{SR}_{\mathrm{nominal}}}, \qquad \Delta\mathrm{TC} = \mathrm{TC}_{\mathrm{fault}} - \mathrm{TC}_{\mathrm{nominal}}. \tag{8}$$

The current instrumentation records token counts and tool-call counts, not wall-clock latency; we therefore treat tokens and calls as compute proxies and identify latency instrumentation as a deployment requirement.

4.1.4 Ablation Studies and Experimental Factors

We study six controlled factors. A1 is the main no-n-shot comparison among ReAct, DepDAG, Spec-Cyc, and Gen-Cyc. A2 varies planner, executor, and router model tiers to test orchestration sensitivity. A3 adds successful-trajectory summaries to test whether experience-derived routines improve planning and routing. A4 compares generalist full-tool executors with tool-restricted specialists on TextCraft and Finance-Agent. A5 injects random subtask redirections to stress-test recovery under disrupted control flow. A6 measures ALFWorld environment-level exploration through unique visited states.

Table 2: Overall comparison of ReAct, DepDAG, Spec-Cyc, and Gen-Cyc across benchmarks without n-shot summaries.

Benchmark	ReAct			DepDAG			Spec-Cyc			Gen-Cyc		
	SR (%)	TC	STR	SR (%)	TC	STR	SR (%)	TC	STR	SR (%)	TC	STR
TextCraft-2	**94.3% ± 0.2%**	7.4 ± 4.3	12.7	58.6% ± 5.4%	13.2 ± 6.1	4.4	85.2% ± 0.8%	11.3 ± 6.5	7.5	93.9% ± 1.3%	9.4 ± 5.6	10.0
TextCraft-3	**82.5% ± 2.1%**	16.0 ± 8.8	5.2	17.9% ± 9.0	15.4 ± 6.6	1.2	50.4% ± 2.3%	21.6 ± 11.8	2.3	71.5% ± 1.5%	22.1 ± 11.9	3.2
TextCraft-4	**45.5% ± 14.8%**	23.4 ± 8.4	1.9	0.0% ± 0.0%	D/A	D/A	15.2% ± 8.6%	68.8 ± 20.1	0.2	36.4% ± 0.0%	69.1 ± 15.0	0.5
ALFWorld	33.8% ± 0.4%	12.9 ± 7.5	2.6	52.2% ± 1.6%	28.1 ± 14.4	1.9	**58.2% ± 4.6%**	30.2 ± 18.5	1.9	56.7% ± 3.2%	28.7 ± 19.9	2.0
Finance-Agent	12.4% ± 1.3%	3.0 ± 0.9	4.1	9.5% ± 1.3%	4.5 ± 2.2	2.1	14.3% ± 0.0%	5.3 ± 2.2	2.7	**15.2% ± 2.7%**	6.0 ± 2.5	2.5

4.1.5 Budgets, Hyperparameters, and Reproducibility

To comprehensively measure performance, for all experiments below, we set specific global tool-call limits: 30, 50, and 100 for TextCraft tasks at depth 2, 3, and 4 respectively, 75 for ALFWorld, and 20 for Finance-Agent. Additionally, the local subtask limit is fixed at 5 for TextCraft, 15 for ALFWorld, and 20 for Finance-Agent for both cyclic and DAG models. The ReAct agents use a unified limit equal to the global limits. Benchmark splits, evaluation budgets, and protocol details are summarized in Appendix Sec. B.2. Verbatim prompt templates, structured-output schemas, and tool specifications are intended for release with the anonymized code package; the aggregate comparisons reported here do not require the reader to run new code.

4.1.6 Train/Test Split and n-Shot Construction

For TextCraft depths 2 and 3 and for Finance-Agent, we reserve a fixed subset of instances for constructing optional n-shot successful-trajectory summaries and evaluate on the remaining held-out instances. For TextCraft-4, only 11 instances are available, so all 11 are used for evaluation and no separate summary-construction split is formed. For ALFWorld, we follow the benchmark's provided partitions: benchmark training games are used only for optional summary construction.

Concretely, when n-shot summaries are enabled, we run Spec-Cyc on the designated summary-construction subset, summarize successful trajectories, and then freeze that summary set for the rest of the benchmark. All reported results are on held-out evaluation instances; the only difference in the n-shot setting is that the planner receives the fixed summary set as additional context. Exact benchmark counts are listed in Appendix A.

4.2 Experimental Results

We organize results around the controlled comparisons in Table **??**: the uniform head-to-head without n-shot summaries (Table 2), graph/role sensitivity with n-shot summaries (Fig. 2), and success-conditioned revisitation (Fig. 3). Remaining sweeps, robustness results, routing-complexity statistics, and qualitative cases are summarized in Appendix Secs. D–G.

A1: Overall comparison without n-shot summaries. Table 2 gives the cleanest apples-to-apples comparison of single-agent and multi-agent control: generalist executors, no successful-trajectory summaries, and `gpt-4o-mini` across all non-ReAct roles. The benchmarks separate clearly by structure. TextCraft is largely a prerequisite-chain domain, so ReAct is strongest overall and cyclic control often pays a coordination cost in TC/STR even when it partially recovers harder cases. ALFWorld is the clearest setting where maximally flexible multi-agent control helps: both Spec-Cyc and Gen-Cyc exceed ReAct in SR, consistent with the need to search, backtrack, and recover under partial observability. Finance-Agent is different again. Success remains low across all methods, and the small gains of cyclic workflows should therefore be read as contrastive rather than celebratory: this benchmark is bottlenecked less by control-flow flexibility than by open-world retrieval, grounding, and evidence synthesis. In other words, the multi-agent architecture matters, but it cannot by itself remove the dominant task bottlenecks.

These outcome differences should also be read jointly with inference cost. On the shared-win intersection against ReAct, cyclic control is frequently much more expensive in total tokens, especially beyond TextCraft-2. We therefore interpret complete cyclic graphs as a higher-flexibility, higher-cost regime: they can recover

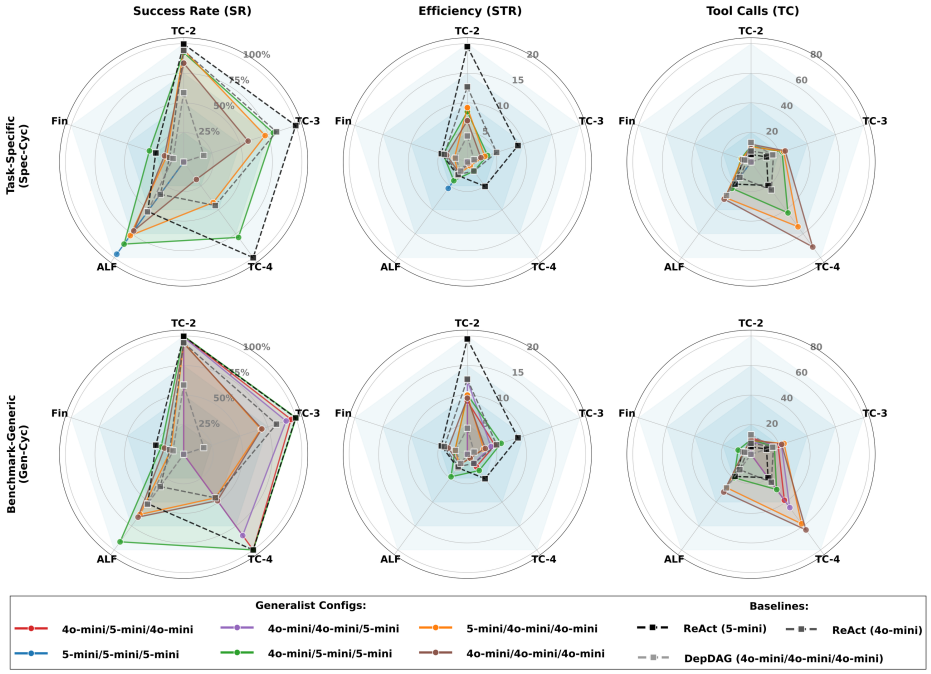

Figure 2: Comparison of graph construction regimes across benchmarks (n-shots included). The top row shows task-specific Spec-Cyc graphs and the bottom row shows benchmark-generic Gen-Cyc graphs. Columns report (left to right) success rate (SR), success-to-tool-call ratio (STR), and tool-call count (TC). Each subplot includes planner–router–executor LLM configurations as well as both ReAct tiers and DepDAG baselines.

failures that a simpler agent does not, but that recovery is often purchased with materially greater inference expenditure rather than coming for free (Appendix Table 8).

A2/A3: Ablations across graph regimes and role allocations (n-shot). Figure 2 summarizes n-shot performance with generalist executors across (i) Spec-Cyc vs. Gen-Cyc graph regimes and (ii) planner-router-executor tier allocations (with ReAct tiers and DepDAG baselines). Across benchmarks, (1) role allocation materially affects cyclic performance (stronger routing/execution shifts behavior from loops to recovery), and (2) benchmark-generic graphs are often competitive at long horizons, in several settings matching or exceeding task-specific graphs, while in others the task-specific regime remains stronger. Full per-configuration results appear in Appendix Table 9; the main visual summary is shown in Fig. 2.

A4: Tool exposure. We additionally vary executor tool exposure on TextCraft and Finance-Agent, comparing generalist (full-tool) vs. tool-restricted specialized executors. We omit ALFWorld from the specialization sweep because, unlike TextCraft (naturally separable `get`/`craft`-style interactions) and Finance-Agent (pre-packaged multiple tools), ALFWorld exposes a single, monolithic action interface; any "tool" partition would be an arbitrary design choice with many plausible alternatives. As a result, specialist performance on ALFWorld would likely be sensitive to the chosen partitioning scheme, so we exclude it to avoid biasing the comparison. Tool restriction is consistently harmful here: specialized executors substantially reduce SR/STR on TextCraft and Finance-Agent, with the largest degradation under Spec-Cyc. Gen-Cyc is more robust under tool restriction and remains the preferred cyclic regime when tools are limited (Appendix Tables 10, 11, and 12). The corresponding numerical summaries appear in Appendix Tables 10, 11, and 12.

4.3 Robustness and recovery efficiency

The fault-injection study is promoted here from an auxiliary ablation to an robustness test. At segment boundaries, the nominal router is randomly overridden with a 50% probability and redirected to a different

Table 3: Episode-level recovery efficiency under control-flow fault injection for generalist executors (all components `gpt-4o-mini`). SRR is perturbed success divided by nominal success; ΔTC is the change in successful-episode tool calls.

Benchmark	Graph	SRR	ΔTC	Interpretation
TextCraft-2	Spec-Cyc	0.591	+7.5	Task-specific criteria are brittle when forced off-route.
TextCraft-2	Gen-Cyc	0.961	+1.2	Generic rules preserve most success with small recovery overhead.
TextCraft-3	Spec-Cyc	0.340	+12.3	Perturbations amplify loop and misrouting cost.
TextCraft-3	Gen-Cyc	0.941	+3.4	Strong retention despite deeper prerequisite chains.
TextCraft-4	Spec-Cyc	0.000	D/A	Low nominal success leaves little recoverable margin.
TextCraft-4	Gen-Cyc	0.874	−6.2	Perturbed successes are fewer but not more call-intensive.
ALFWorld	Spec-Cyc	0.865	−1.5	Recovery routes remain useful under partial observability.
ALFWorld	Gen-Cyc	0.769	−1.6	Success drops more than Spec-Cyc, but retained wins remain efficient.

subtask. A linear directed acyclic workflow has no natural semantics for such arbitrary backtracking: an invalid hand-off is typically treated as a failed step, a restart, or a manually enumerated exception. In contrast, the complete cyclic graph keeps every subtask reachable, so recovery can be measured as the ability to retain success under disrupted control flow.

Table 3 summarizes the generalist-executor setting. The strongest reliability signal is Gen-Cyc on TextCraft: despite perturbation, it retains 96.1% of nominal success on TextCraft-2 and 94.1% on TextCraft-3, with only 1.2 and 3.4 additional successful-episode tool calls, respectively. Spec-Cyc is less stable under the same perturbation, dropping to 59.1% retention on TextCraft-2 and 34.0% retention on TextCraft-3. On ALFWorld, both cyclic variants retain substantial success under perturbation, consistent with the benchmark's need for search and recovery. These results quantify the core workflow-control trade-off: flexible routing is not free, but in regimes where the generic transition rules are stable, a small call overhead can buy high fault tolerance.

This is still an episode-level robustness measure rather than an event-level repair-latency measure. The present traces do not separately log "fault occurrence → corrected state" intervals, so we cannot claim a precise number of cycles or seconds per individual recovery event. The available evidence supports a weaker but deployment-relevant conclusion: under harsh randomized routing disruption, some cyclic controllers, especially Gen-Cyc in TextCraft, retain most of their nominal success with modest successful-episode call overhead, whereas a forward-only controller would require externally specified exception handling or restart logic.

Success-conditioned revisitation behavior. Beyond aggregate SR/TC, Figure 3 decomposes subtask visitation within successful episodes only. Across benchmarks and configurations, successful cyclic trajectories typically leverage revisitation (error correction and re-planning) rather than only first-pass progress, but because this view is conditioned on success it mixes wasteful detours with the corrective revisits that ultimately enable completion; we interpret these effects in the Discussion. The appendix includes routing-complexity tables and qualitative traces that illustrate the same revisitation patterns.

4.4 Finance-Agent as an evidence-synthesis pipeline

Finance-Agent behaves differently from ALFWorld because the bottleneck is not primarily whether the controller can revisit earlier subtasks, but whether the system retrieves and grounds the right evidence. The main comparison shows low absolute success across all methods: ReAct reaches 12.4%, DepDAG 9.5%, Spec-Cyc 14.3%, and Gen-Cyc 15.2% without n-shot summaries (Table 2). This suggests that cyclic control can provide modest gains, but cannot by itself solve query formulation, source coverage, citation-level grounding, or evidence synthesis errors. The workflow-design lesson is therefore distinct from ALFWorld: revisitation is useful only when it changes the evidence set or improves grounding; otherwise, the system pays routing and token cost while cycling over the same unresolved information gap.

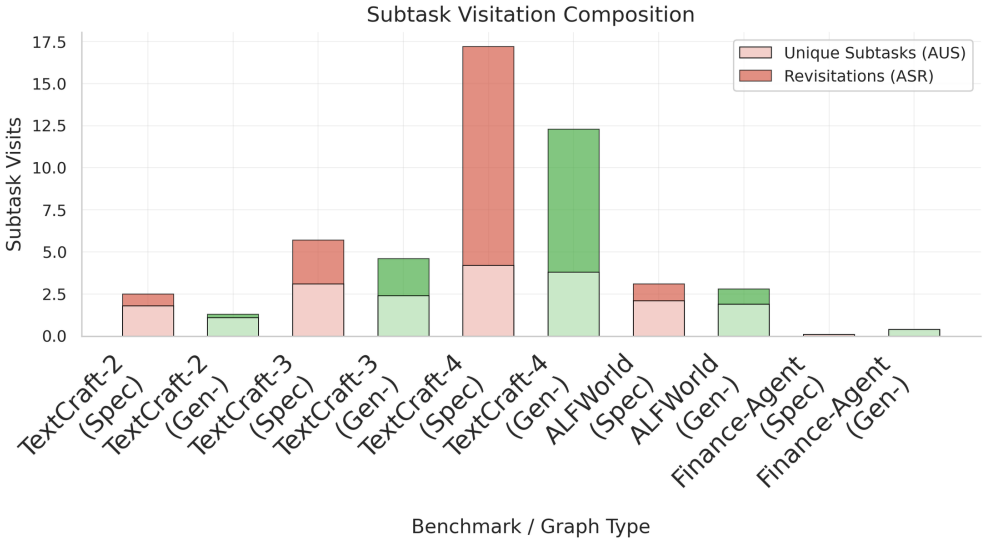

Figure 3: Success-only subtask visitation decomposition.

5 Discussion

Complete cyclic subtask graphs are best understood here as a deliberately maximally flexible multi-agent regime rather than as a claim that fully connected control is always optimal. Because every executable subtask can transition to every other subtask, the controller is never forced into a predefined trajectory: recovery, revisitation, and exploratory detours are always structurally available. That design choice is exactly what makes the method informative. It lets us observe when flexibility translates into useful recovery, when it turns into coordination overhead or router hallucination, and when the main bottleneck lies outside the workflow controller altogether.

Benchmark structure determines whether flexibility is an asset or a tax. The three benchmarks expose three different regimes. In ALFWorld, partial observability and interaction-heavy search make backtracking genuinely useful, so cyclic control shows its clearest advantage. In TextCraft, once the correct prerequisite chain is identified, the problem is closer to disciplined forward execution, so the same routing flexibility can become extra coordination cost. Finance-Agent exposes a third regime: open-world retrieval and evidence aggregation dominate, so even a flexible multi-agent controller yields only modest gains because the harder problem is grounding and synthesis, not just workflow navigation.

Cost matters, and weaker-tier models are the right lens for seeing it. The central controlled setting uses weaker-tier models on purpose. The point is not simply to maximize benchmark scores, but to measure whether multi-agent flexibility can recover mistakes that a weaker single agent makes. This is also the regime where the tradeoff is visible: cyclic control can help, but it often requires substantially more inference. Once stronger agents already saturate a benchmark, the scientific value of additional workflow flexibility becomes harder to distinguish from raw model capability.

Why Gen-Cyc can sometimes outperform Spec-Cyc. Benchmark-generic graphs can act as a regularizing bias because their subtasks and routing criteria are typically simpler, reusable, and less entangled with instance-specific wording. Spec-Cyc can be more expressive, but that same expressivity can also produce brittle high-precision transitions, especially when the router over-trusts subtask intent relative to the actually observed state. Our results therefore do not suggest that Gen-Cyc is uniformly superior; rather, they indicate that the simpler reusable regime is often competitive and, in several long-horizon or perturbed settings, more stable and recoverable than task-specific graphs.

Latency remains a deployment metric rather than a measured benchmark metric. The current experiments record tool calls and token use, which are reliable proxies for resource consumption across model configurations, but they do not record wall-clock time-to-completion. In production, latency would include the number of model calls, model-specific response time, tool latency, and any parallelism across subtasks. Because the complete cyclic graph adds router calls and longer routing prompts, it should be expected to increase latency unless routing is batched, cached, or pruned. This is why the scalability analysis treats the complete graph as a diagnostic starting point rather than a final enterprise controller.

The main optimization opportunity is to keep the lens, then sparsify it. The complete graph is useful experimentally because it reveals where freedom helps and where it wastes budget. The natural next step is therefore not to abandon the subtask-level controller, but to learn which transitions should remain. The unique-transition analysis suggests a concrete path: use never-won transitions, recurrent loop patterns, and contradiction signals from structured state analysis to prune or downweight edges. In that sense, the current architecture is a maximally informative starting point for learning better, sparser multi-agent workflows.

6 Limitations and Reproducibility Notes

The experiments characterize workflow-control behavior under the prompts, tools, model tiers, and budgets reported in the appendix. They should not be read as a claim that complete cyclic graphs are universally optimal. The current logs support token and tool-call cost accounting, but they do not contain wall-clock latency traces; therefore, latency is discussed as a deployment instrumentation requirement rather than reported as a measured metric. Similarly, the fault-injection analysis uses episode-level recovery proxies rather than per-failure cycle-to-recovery traces.

An anonymized code repository, including prompt templates, execution scripts, and result-processing utilities, is included in the supplementary material for review. If the paper is accepted, we will make the repository publicly available with the final artifact release.

7 Broader Impact Statement

This work studies workflow-control structures for tool-using LLM agents. The main positive impact is improved diagnosis of when flexible revisitation and recovery are worth their cost in long-horizon automation. The same mechanisms could be misapplied to high-stakes domains if users treat successful task completion as a substitute for evidence quality, domain validation, or human oversight. The Finance-Agent results in particular argue against such over-reliance: retrieval, grounding, and synthesis bottlenecks remain limiting failure modes even when the controller is made more flexible.

8 Conclusions

We studied complete cyclic subtask graphs as a maximally flexible workflow-control design for automated large-language-model workflows. The value of this design is not that full connectivity is universally best, but that it makes the flexibility–cost–reliability tradeoff directly measurable at the level of executable subtasks. The workflow-selection matrix clarifies when cyclic control is appropriate: it helps most when recovery and exploration are central, adds overhead in stable prerequisite-chain tasks, and cannot by itself solve evidence-quality bottlenecks in open-world financial research. The robustness results further show that generic cyclic transition rules can retain high success under harsh routing perturbations with modest successful-episode call overhead, while task-specific high-precision routes can be brittle. Overall, complete cyclic subtask graphs provide a systematic diagnostic lens for deciding when multi-agent flexibility is worth its price and when practitioners should instead deploy a simpler directed acyclic graph, state machine, or sparsified cyclic controller.

References

Antoine Bigeard, Rayan Krishnan, Shirley Wu, and Langston Nashold. Finance agent benchmark: Benchmarking llms on real-world financial research tasks. *arXiv preprint arXiv:2508.00828*, 2025. URL https://arxiv.org/abs/2508.00828.

Justin Chih-Yao Chen, Swarnadeep Saha, Elias Stengel-Eskin, and Mohit Bansal. MAGDi: Structured distillation of multi-agent interaction graphs improves reasoning in smaller language models. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 7220–7235, 2024.

Ke Chen, Peiran Wang, Yaoning Yu, Xianyang Zhan, and Haohan Wang. Large language model-based data science agent: A survey. *Transactions on Machine Learning Research*, February 2026. URL https://openreview.net/forum?id=ZT5SJQN0CS.

Andrew Coles, Erez Karpas, Eyal Shimony, Shahaf Shperberg, and Wheeler Ruml. Concurrent planning and execution using dispatch-dependent values. In James Kwok (ed.), *Proceedings of the Thirty-Fourth International Joint Conference on Artificial Intelligence, IJCAI-25*, pp. 8483–8490, 2025.

Lutfi Eren Erdogan, Nicholas Lee, Sehoon Kim, Suhong Moon, Hiroki Furuta, Gopala Anumanchipalli, Kurt Keutzer, and Amir Gholami. Plan-and-act: Improving planning of agents for long-horizon tasks. In *Proceedings of the 42nd International Conference on Machine Learning*, 2025. arXiv:2503.09572.

Jen-tse Huang, Jiaxu Zhou, Tailin Jin, Xuhui Zhou, Zixi Chen, Wenxuan Wang, Youliang Yuan, Michael R. Lyu, and Maarten Sap. On the resilience of llm-based multi-agent collaboration with faulty agents. In *Proceedings of the 42nd International Conference on Machine Learning*, 2025. arXiv:2408.00989.

Yubin Kim, Ken Gu, Chanwoo Park, Chunjong Park, Samuel Schmidgall, A Ali Heydari, Yao Yan, Zhihan Zhang, Yuchen Zhuang, Mark Malhotra, et al. Towards a science of scaling agent systems. *arXiv preprint arXiv:2512.08296*, 2025.

Lars Klein, Nearchos Potamitis, Roland Aydin, Robert West, Caglar Gulcehre, and Akhil Arora. Fleet of agents: Coordinated problem solving with large language models. *arXiv preprint arXiv:2405.06691*, 2024.

Xinzhe Li. A survey on llm test-time compute via search: Tasks, llm profiling, search algorithms, and relevant frameworks. *Transactions on Machine Learning Research*, May 2025. URL https://openreview.net/forum?id=x9VQFjtOPS.

Xuyan Ma, Yawen Wang, Junjie Wang, Xiaofei Xie, Boyu Wu, Shoubin Li, Fanjiang Xu, and Qing Wang. Enhancing multi-agent system testing with diversity-guided exploration and adaptive critical state exploitation. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*, pp. 1491–1503, 2024.

Shishir G Patil, Tianjun Zhang, Xin Wang, and Joseph E Gonzalez. Gorilla: Large language model connected with massive apis. *Advances in Neural Information Processing Systems*, 37:126544–126565, 2024.

Archiki Prasad, Alexander Koller, Mareike Hartmann, Peter Clark, Ashish Sabharwal, Mohit Bansal, and Tushar Khot. Adapt: As-needed decomposition and planning with language models. In *Findings of the Association for Computational Linguistics: NAACL 2024*, pp. 4226–4252, 2024.

Chen Qian, Zihao Xie, Yifei Wang, Wei Liu, Kunlun Zhu, Hanchen Xia, Yufan Dang, Zhuoyun Du, Weize Chen, Cheng Yang, et al. Scaling large language model-based multi-agent collaboration. *arXiv preprint arXiv:2406.07155*, 2024.

Shengqian Qin, Yakun Zhu, Linjie Mu, Shaoting Zhang, and Xiaofan Zhang. Meta-tool: Unleash open-world function calling capabilities of general-purpose large language models. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 30653–30677, 2025.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. Toolllm: Facilitating large language models to master 16000+ real-world apis. In *The Twelfth International Conference on Learning Representations (ICLR 2024)*, 2024.

Haochen Shi, Zhiyuan Sun, Xingdi Yuan, Marc-Alexandre Côté, and Bang Liu. OPEx: A large language model-powered framework for embodied instruction following. In N. Alechina, V. Dignum, M. Dastani, and J. S. Sichman (eds.), *Proceedings of the 23rd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2024)*, pp. 2465–2467, May 2024. Extended Abstract.

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

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

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36:8634–8652, 2023.

Mohit Shridhar, Jesse Thomason, Daniel Gordon, Yonatan Bisk, Winson Han, Roozbeh Mottaghi, Luke Zettlemoyer, and Dieter Fox. Alfred: A benchmark for interpreting grounded instructions for everyday tasks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 10740–10749, 2020.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. Alfworld: Aligning text and embodied environments for interactive learning. In *International Conference on Learning Representations (ICLR)*, 2021.

Andries Petrus Smit, Nathan Grinsztajn, Paul Duckworth, Thomas D. Barrett, and Arnu Pretorius. Should we be going MAD? A look at multi-agent debate strategies for LLMs. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 45883–45905. PMLR, 2024.

Linxin Song, Jiale Liu, Jieyu Zhang, Shaokun Zhang, Ao Luo, Shijian Wang, Qingyun Wu, and Chi Wang. Adaptive in-conversation team building for language model agents. *arXiv preprint arXiv:2405.19425*, 2024.

Peiyang Song, Pengrui Han, and Noah Goodman. Large language model reasoning failures. *Transactions on Machine Learning Research*, January 2026. URL `https://openreview.net/forum?id=vnX1WHMNmz`.

Xinyuan Song, Zeyu Wang, Siyi Wu, Tianyu Shi, and Lynn Ai. Gradientsys: A multi-agent llm scheduler with react orchestration. *arXiv preprint arXiv:2507.06520*, 2025.

Yang Sui, Yu-Neng Chuang, Guanchu Wang, Jiamu Zhang, Tianyi Zhang, Jiayi Yuan, Hongyi Liu, Andrew Wen, Shaochen Zhong, Na Zou, Hanjie Chen, and Xia Hu. Stop overthinking: A survey on efficient reasoning for large language models. *Transactions on Machine Learning Research*, August 2025. URL `https://openreview.net/forum?id=HvoG8SxggZ`.

Dayu Wang, Jiaye Yang, Weikang Li, Jiahui Liang, and Yang Li. Reducing cognitive overhead in tool use via multi-small-agent reinforcement learning. *arXiv preprint arXiv:2508.08882*, 2025a.

Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. Executable code actions elicit better llm agents. In *Forty-first International Conference on Machine Learning*, 2024.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. In *International Conference on Machine Learning (ICML)*, 2025b. Also appears as ICML 2025 poster / proceedings.

Yiran Wu, Tianwei Yue, Shaokun Zhang, Chi Wang, and Qingyun Wu. Stateflow: Enhancing LLM task-solving through state-driven workflows. In *Conference on Language Modeling (COLM 2024)*, 2024.

Binfeng Xu, Zhiyuan Peng, Bowen Lei, Subhabrata Mukherjee, Yuchen Liu, and Dongkuan Xu. Rewoo: Decoupling reasoning from observations for efficient augmented language models. *arXiv preprint arXiv:2305.18323*, 2023.

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

Rui Ye, Shuo Tang, Rui Ge, Yaxin Du, Zhenfei Yin, Siheng Chen, and Jing Shao. Mas-gpt: Training llms to build llm-based multi-agent systems. *arXiv preprint arXiv:2503.03686*, 2025.

Chaoyun Zhang, Shilin He, Jiaxu Qian, Bowen Li, Liqun Li, Si Qin, Yu Kang, Minghua Ma, Guyue Liu, Qingwei Lin, Saravan Rajmohan, Dongmei Zhang, and Qi Zhang. Large language model-brained gui agents: A survey. *Transactions on Machine Learning Research*, June 2025a. URL `https://openreview.net/forum?id=xChvYjvXTp`.

Guibin Zhang, Yanwei Yue, Xiangguo Sun, Guancheng Wan, Miao Yu, Junfeng Fang, Kun Wang, Tianlong Chen, and Dawei Cheng. G-designer: Architecting multi-agent communication topologies via graph neural networks. *arXiv preprint arXiv:2410.11782*, 2024.

Guibin Zhang, Luyang Niu, Junfeng Fang, Kun Wang, Lei Bai, and Xiang Wang. Multi-agent architecture search via agentic supernet. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 75834–75852, 2025b.

Guibin Zhang, Junhao Wang, Junjie Chen, Wangchunshu Zhou, Kun Wang, and Shuicheng Yan. Agentracer: Who is inducing failure in the llm agentic systems? *arXiv preprint arXiv:2509.03312*, 2025c.

Shaokun Zhang, Ming Yin, Jieyu Zhang, Jiale Liu, Zhiguang Han, Jingyang Zhang, Beibin Li, Chi Wang, Huazheng Wang, Yiran Chen, and Qingyun Wu. Which agent causes task failures and when? On automated failure attribution of LLM multi-agent systems. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 76583–76599, 2025d.

Wentao Zhang, Ce Cui, Yilei Zhao, Yang Liu, and Bo An. Agentorchestra: A hierarchical multi-agent framework for general-purpose task solving. *arXiv preprint arXiv:2506.12508*, 2025e.

Yaolun Zhang, Xiaogeng Liu, and Chaowei Xiao. MetaAgent: Automatically constructing multi-agent systems based on finite state machines. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 75667–75694, 2025f. ICML 2025 Poster.

Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and Jürgen Schmidhuber. Gptswarm: Language agents as optimizable graphs. In *Forty-first International Conference on Machine Learning*, 2024.

A Protocol, Key Tables, and Case Studies

This appendix retains the material most relevant for auditing the main claims in the paper, including protocol details, token accounting, ablation tables, routing-complexity statistics, and qualitative traces. Full prompt dumps and execution logs are intended for release with an anonymized code package upon acceptance.

B Benchmark and protocol details

B.1 Benchmark Details

We evaluate on three long-horizon tool-using benchmarks that differ in environment dynamics, observability, and failure modes.

B.1.1 TextCraft

TextCraft. TextCraft is a text-based crafting environment that requires multi-step composition of actions and intermediate prerequisites. We use three difficulty tiers (TextCraft-2/3/4), where higher indices correspond to deeper dependency chains and longer horizons. We treat each instance as successful if the target crafting goal is achieved within the allowed tool-call budget.

The tool set supports item acquisition, crafting operations, and inventory management.

`textcraft_inventory`
: Check the current contents of the agent's inventory.

- **Parameters:** None
- **Returns:** A listing of all items currently in the inventory with their quantities

`textcraft_get_item`
: Acquire items from the environment. Not all items can be obtained directly—some must be crafted.

- **Parameters:**
  - `reasoning` (List[str]): Step-by-step reasoning assessing which item to get
  - `item_name` (Literal): The name of the item to acquire, constrained to obtainable items
  - `num_total_items_needed` (int, ≥ 1): The quantity of items to acquire
- **Returns:** The result of the acquisition attempt

`textcraft_craft`
: Craft items using materials in the inventory according to available recipes.

- **Parameters:**
  - `reasoning` (List[str]): Step-by-step reasoning assessing which item to craft
  - `crafting_command` (Literal): The crafting command in the format "craft *N item* using *M ingredient1*, ...", constrained to valid recipes
- **Returns:** The result of the crafting action, including success or failure with explanation

`textcraft_select_command`
: A unified command interface combining all TextCraft operations.

- **Parameters:**
  - `reasoning` (List[str]): Step-by-step reasoning assessing which command to execute
  - `command` (str): One of:
    - `get <number> <item>` – Acquire items from the environment
    - `craft <crafting_command>` – Craft items using inventory materials
    - `inventory` – Check inventory contents
- **Returns:** The observation after executing the command

B.1.2 ALFWorld

ALFWorld. ALFWorld is an interactive household environment where agents must navigate, manipulate objects, and complete multi-step tasks. Compared to TextCraft, ALFWorld places greater emphasis on exploration, partial observability, and recovery after incorrect actions. We treat each episode as successful if the specified goal condition is satisfied within the tool-call budget.

The tool set is dynamically constrained based on the current game state, only valid commands for the current context are exposed to the agent.

`alfworld_select_command`
Select and execute a command from the list of currently admissible commands.

- **Parameters:**
  - `reasoning` (List[str]): Step-by-step reasoning assessing which command to execute
  - `command` (Literal): The command to execute, constrained to the set of admissible commands
- **Returns:** The next observation of the environment after executing the command

`alfworld_manage_inventory`
Manage the agent's inventory by checking contents, taking items from receptacles, or placing items.

- **Parameters:**
  - `reasoning` (List[str]): Step-by-step reasoning assessing which inventory command to execute
  - `command` (Literal): One of:
    - `inventory` – Check current inventory contents
    - `take <object> from <receptacle>` – Take an item from a receptacle
    - `put <object> in/on <receptacle>` – Place an item in or on a receptacle
    - `examine <inventory_item>` – Examine an item in inventory
- **Returns:** The observation after the inventory operation

`alfworld_explore`
Navigate and explore the environment to discover objects and locations.

- **Parameters:**
  - `reasoning` (List[str]): Step-by-step reasoning about which navigational command to execute
  - `command` (Literal): One of:
    - `look` – Look around the current location
    - `go to <receptacle>` – Move to a specific furniture or receptacle
    - `examine <receptacle>` – Get detailed information about a receptacle
    - `open <receptacle>` – Open a closed receptacle
    - `close <receptacle>` – Close an open receptacle
- **Returns:** The observation after the exploration action

`alfworld_manipulate_object`
Perform physical manipulations on objects using tools or appliances.

- **Parameters:**
  - `reasoning` (List[str]): Step-by-step reasoning assessing which manipulation to perform
  - `command` (Literal): One of:
    - `use <object>` – Use an item in inventory
    - `heat <object> with <appliance>` – Heat an item using an appliance (e.g., microwave)
    - `cool <object> with <appliance>` – Cool an item using an appliance (e.g., fridge)
    - `clean <object> with <receptacle>` – Clean an item using a receptacle (e.g., sink)
    - `slice <object> with <tool>` – Cut an item using a tool (e.g., knife)
- **Returns:** The observation after the manipulation

B.1.3 Finance-Agent

Finance-Agent. Finance-Agent consists of information-seeking tasks requiring web research and evidence aggregation to produce a final, grounded financial answer. The environment is open-world and tool interactions can return large, noisy observations. An episode is successful if the produced final answer matches the benchmark's correctness criteria under the allowed tool-call budget. We evaluate only on the publicly available validation split, which contains 50 cases, and report success under the same tool-call budget and evaluation criteria as in the benchmark.

The tool set enables systematic gathering and processing of financial data.

`finance_agent_edgar_search`
Search the SEC EDGAR database for regulatory filings.

- **Parameters:**
  - `query` (str): Keyword or phrase to search (e.g., "substantial doubt" OR "material weakness")
  - `form_types` (List[str], optional): Filter by form types (e.g., ["10-K", "10-Q", "8-K"])
  - `ciks` (List[str], optional): Filter by company CIK numbers
  - `start_date` (str, optional): Start date in yyyy-mm-dd format
  - `end_date` (str, optional): End date in yyyy-mm-dd format
  - `page` (int, default=1): Page number for pagination
  - `top_n_results` (int, default=10): Maximum number of results to return
- **Returns:** Filing metadata including company name, ticker, CIK, form type, filing date, description, and document URL

`finance_agent_google_web_search`
Search the web for financial information using Google Search.

- **Parameters:**
  - `search_query` (str): The query to search for on the web
- **Returns:** Summarized search results with relevant information extracted

`finance_agent_parse_html_page`
Fetch and parse an HTML page, storing the extracted text content for later retrieval.

- **Parameters:**
  - `url` (str): The URL of the HTML page to parse
  - `key` (str): The key under which to store the parsed content for later retrieval
- **Returns:** Confirmation of success with content length and current data storage keys

`finance_agent_retrieve_information`
Retrieve and summarize previously stored documents using placeholder substitution.

- **Parameters:**
  - `prompt` (str): A prompt containing `{{key_name}}` placeholders that will be replaced with stored content (e.g., "Summarize this filing: {{company_10k}}")
- **Returns:** The prompt with placeholders replaced by summarized document content

`finance_agent_final_answer`
Submit the final answer to the financial question.

- **Parameters:**
  - `answer` (str): The final answer to the financial question
  - `sources` (List[SourceInfo], optional): List of sources with URLs and descriptive names
- **Returns:** Confirmation that the answer has been submitted

B.2 Experimental Protocol Overview

This appendix documents the train/test splits used for each benchmark. All splits are deterministically shuffled using a fixed random seed of 42 to ensure reproducibility.

B.2.1 Summary

Table 4: Overview of benchmark dataset sizes and train/test splits.

Benchmark	Total Seeds	Train Set	Test Set
TextCraft (Depth 2)	291	88 (30%)	203 (70%)
TextCraft (Depth 3)	117	35 (30%)	82 (70%)
TextCraft (Depth 4)	11	0 (0%)	11 (100%)
ALFWorld	3,827	3,553 (train)	134 (unseen) / 140 (seen)
Finance-Agent	50	15 (30%)	35 (70%)

B.2.2 TextCraft

TextCraft tasks are stratified by recipe depth, representing the number of intermediate crafting steps required to produce the target item. Each depth level constitutes an independent dataset with its own train/test split.

Table 5: TextCraft dataset splits by recipe depth.

Depth	Total	Train	Test	Test Partitions
2	291	88	203	5 partitions (45, 45, 45, 45, 23 seeds)
3	117	35	82	2 partitions (45, 37 seeds)
4	11	0	11	1 partition (11 seeds)

Seeds are shuffled deterministically before splitting. The train set comprises the first 30% of shuffled seeds, with the remainder allocated to the test set. Depth 4 contains only 11 seeds due to the limited number of recipes requiring four crafting steps; all are used for evaluation.

B.2.3 ALFWorld

ALFWorld provides pre-defined dataset splits through its environment configuration, derived from the ALFRED dataset Shridhar et al. (2020). The benchmark uses TextWorld-based game files generated from ALFRED trajectories.

Table 6: ALFWorld dataset configurations.

Dataset	Games	Description
`train`	3,553	Training set
`eval_in_distribution`	140	In-distribution evaluation
`eval_out_of_distribution`	134	Out-of-distribution evaluation

The primary evaluation uses the `eval_out_of_distribution` dataset (also known as `valid_unseen`) containing 134 test cases with novel scene and object configurations not seen during training.

The `eval_in_distribution` dataset (`valid_seen`) contains 140 games with scene configurations similar to the training set, used for ablation studies.

B.2.4 Finance-Agent

Finance-Agent consists of the 50 publicly available financial question-answering tasks requiring retrieval and analysis of SEC filings and financial documents.

Table 7: Finance-Agent dataset split.

Split	Seeds	Percentage
Train	15	30%
Test	35	70%
Total	50	100%

B.2.5 Reproducibility

All random shuffling operations use Python's `random` module with a fixed seed of 42:

```
random.seed(42)
seeds = list(range(total_seeds))
random.shuffle(seeds)
train_seeds = seeds[:train_size]
test_seeds = seeds[train_size:]
```

This ensures identical train/test splits across all experimental runs.

C Shared-win token-cost comparison

Table 8: Relative token usage of cyclic workflows against ReAct. Each comparison block reports average total tokens per test case on the shared-win intersection (primary comparison) and on all won cases (secondary context). 'Excess' and 'Rel. diff.' are computed with respect to the shared-win row unless separately populated for the all-win row.

Benchmark	Compared pair	Scope	ReAct avg total	Cyclic avg total	Excess (C−R)	Rel. diff. (%)
TextCraft-2	ReAct (`4o-mini`) vs Spec-Cyc, generalist, P/E/R=4o/4o/4o	Shared wins	81084.08	93208.69	12124.61	14.95
		All wins	83360.6	93835.68	10475.08	12.57
TextCraft-2	ReAct (`4o-mini`) vs Gen-Cyc, generalist, P/E/R=4o/4o/4o	Shared wins	81820.76	77694.46	-4126.3	-5.04
		All wins	83360.6	78741.56	-4619.04	-5.54
TextCraft-2	ReAct (`5-mini`) vs Spec-Cyc, generalist, P/E/R=4o/5/5	Shared wins	22341.6	97919.85	75578.25	338.28
		All wins	22490.85	97919.85	75429.0	335.38
TextCraft-2	ReAct (`5-mini`) vs Gen-Cyc, generalist, P/E/R=4o/5/5	Shared wins	22490.85	108678.92	86188.07	383.21
		All wins	22490.85	108678.92	86188.07	383.21
TextCraft-3	ReAct (`4o-mini`) vs Spec-Cyc, generalist, P/E/R=4o/4o/4o	Shared wins	109047.05	245365.14	136318.09	125.01
		All wins	197838.72	252334.26	54495.54	27.55
TextCraft-3	ReAct (`4o-mini`) vs Gen-Cyc, generalist, P/E/R=4o/4o/4o	Shared wins	172396.49	210395.97	37999.48	22.04
		All wins	197838.72	209554.94	11716.22	5.92
TextCraft-3	ReAct (`5-mini`) vs Spec-Cyc, generalist, P/E/R=4o/5/5	Shared wins	57238.43	226266.16	169027.73	295.3

Continued on next page

Benchmark	Compared pair	Scope	ReAct avg total	Cyclic avg total	Excess (C−R)	Rel. diff. (%)
		All wins	61405.6	226266.16	164860.56	268.48
TextCraft-3	ReAct (`5-mini`) vs Gen-Cyc, generalist, P/E/R=4o/5/5	Shared wins	61405.6	180870.26	119464.66	194.55
		All wins	61405.6	180870.26	119464.66	194.55
TextCraft-4	ReAct (`4o-mini`) vs Spec-Cyc, generalist, P/E/R=4o/4o/4o	Shared wins	275556.0	423724.0	148168.0	53.77
		All wins	264396.0	582049.0	317653.0	120.14
TextCraft-4	ReAct (`4o-mini`) vs Gen-Cyc, generalist, P/E/R=4o/4o/4o	Shared wins	243655.4	558693.6	315038.2	129.3
		All wins	264396.0	690016.0	425620.0	160.98
TextCraft-4	ReAct (`5-mini`) vs Spec-Cyc, generalist, P/E/R=4o/5/5	Shared wins	122673.18	485882.27	363209.09	296.08
		All wins	122673.18	485882.27	363209.09	296.08
TextCraft-4	ReAct (`5-mini`) vs Gen-Cyc, generalist, P/E/R=4o/5/5	Shared wins	122673.18	344174.82	221501.64	180.56
		All wins	122673.18	344174.82	221501.64	180.56
ALFWorld	ReAct (`4o-mini`) vs Spec-Cyc, generalist, P/E/R=4o/4o/4o	Shared wins	45454.23	177099.37	131645.14	289.62
		All wins	44948.19	212515.74	167567.55	372.8
ALFWorld	ReAct (`4o-mini`) vs Gen-Cyc, generalist, P/E/R=4o/4o/4o	Shared wins	45693.79	173272.44	127578.65	279.2
		All wins	44948.19	228945.04	183996.85	409.35
ALFWorld	ReAct (`5-mini`) vs Spec-Cyc, generalist, P/E/R=4o/5/5	Shared wins	94583.33	174485.97	79902.64	84.48
		All wins	94315.32	189729.14	95413.82	101.16
ALFWorld	ReAct (`5-mini`) vs Gen-Cyc, generalist, P/E/R=4o/5/5	Shared wins	94315.32	185859.83	91544.51	97.06
		All wins	94315.32	207151.84	112836.52	119.64
Finance-Agent	ReAct (`4o-mini`) vs Spec-Cyc, generalist, P/E/R=4o/4o/4o	Shared wins	97959.4	120534.0	22574.6	23.04
		All wins	83090.1	134142.86	51052.76	61.44
Finance-Agent	ReAct (`4o-mini`) vs Gen-Cyc, generalist, P/E/R=4o/4o/4o	Shared wins	61555.12	140562.12	79007.0	128.35
		All wins	83090.1	148646.0	65555.9	78.9
Finance-Agent	ReAct (`5-mini`) vs Spec-Cyc, generalist, P/E/R=4o/5/5	Shared wins	157094.82	258076.09	100981.27	64.28
		All wins	144014.85	275482.0	131467.15	91.29
Finance-Agent	ReAct (`5-mini`) vs Gen-Cyc, generalist, P/E/R=4o/5/5	Shared wins	146905.5	383115.0	236209.5	160.79
		All wins	144014.85	394334.5	250319.65	173.82

D Key ablation tables

Table 9: Planner–executor–router model-tier sweep with n-shot successful-trajectory summaries and generalist executors. We compare Spec-Cyc vs. Gen-Cyc across role allocations and report SR, TC (on successes), STR, and subtask-visitation metrics (ATS/AUS/ASR).

Benchmark	Graph	Planner	Executor	Router	SR (%)	TC	STR	ATS	AUS	ASR
TextCraft-2	Spec-Cyc	5-mini	4o-mini	4o-mini	$94.9\% \pm 1.0\%$	10.3 ± 5.6	9.2	1.8 ± 1.4	1.5 ± 1.4	0.3 ± 0.6
TextCraft-2	Spec-Cyc	4o-mini	5-mini	5-mini	$92.5\% \pm 2.0\%$	10.9 ± 4.9	8.5	3.5 ± 2.2	2.5 ± 1.0	1.0 ± 1.9
TextCraft-2	Spec-Cyc	4o-mini	4o-mini	4o-mini	$83.6\% \pm 1.3\%$	12.0 ± 6.7	7.0	2.5 ± 1.8	1.8 ± 1.1	0.7 ± 1.1
TextCraft-2	Gen-Cyc	5-mini	4o-mini	4o-mini	$94.4\% \pm 2.2\%$	9.4 ± 5.8	10.0	1.20 ± 1.3	1.0 ± 1.0	0.2 ± 0.5
TextCraft-2	Gen-Cyc	4o-mini	5-mini	5-mini	$100\% \pm 0.0\%$	9.9 ± 3.1	10.1	2.3 ± 3.3	1.0 ± 1.1	1.3 ± 2.7
TextCraft-2	Gen-Cyc	4o-mini	5-mini	4o-mini	$99.7\% \pm 0.5\%$	11.1 ± 3.8	9.0	6.9 ± 1.7	3.6 ± 0.6	3.3 ± 1.5
TextCraft-2	Gen-Cyc	4o-mini	4o-mini	5-mini	$98.9\% \pm 0.2\%$	7.9 ± 5.2	12.5	1.9 ± 1.2	1.7 ± 0.9	0.2 ± 0.5
TextCraft-2	Gen-Cyc	4o-mini	4o-mini	4o-mini	$93.8\% \pm 0.9\%$	9.9 ± 5.7	9.5	1.3 ± 1.2	1.1 ± 0.9	0.2 ± 0.5
TextCraft-3	Spec-Cyc	5-mini	4o-mini	4o-mini	$72.4\% \pm 2.5\%$	23.2 ± 12.1	3.1	4.5 ± 2.5	2.9 ± 1.2	1.6 ± 1.9
TextCraft-3	Spec-Cyc	4o-mini	5-mini	5-mini	$79.7\% \pm 2.5\%$	22.1 ± 11.1	3.6	7.5 ± 4.7	3.4 ± 1.2	4.1 ± 4.4
TextCraft-3	Spec-Cyc	4o-mini	4o-mini	4o-mini	$57.3\% \pm 1.0\%$	24.3 ± 11.2	2.4	5.71 ± 3.0	3.1 ± 1.2	2.6 ± 2.4
TextCraft-3	Gen-Cyc	5-mini	4o-mini	4o-mini	$68.7 \pm 2.1\%$	23.5 ± 13.5	2.9	4.2 ± 2.9	2.3 ± 1.0	1.9 ± 2.3
TextCraft-3	Gen-Cyc	4o-mini	5-mini	5-mini	$100\% \pm 0.0\%$	16.6 ± 8.5	6.0	6.1 ± 3.7	3.1 ± 0.5	3.0 ± 3.5
TextCraft-3	Gen-Cyc	4o-mini	5-mini	4o-mini	$95.5\% \pm 1.5\%$	19.3 ± 8.7	4.9	9.7 ± 4.2	3.8 ± 0.5	5.9 ± 4.0
TextCraft-3	Gen-Cyc	4o-mini	4o-mini	5-mini	$91.1\% \pm 2.1\%$	21.6 ± 10.0	4.2	4.9 ± 2.1	3.0 ± 0.7	1.9 ± 1.9
TextCraft-3	Gen-Cyc	4o-mini	4o-mini	4o-mini	$69.5\% \pm 1.0\%$	21.8 ± 12.4	3.2	4.6 ± 3.2	2.4 ± 1.0	2.2 ± 2.7
TextCraft-4	Spec-Cyc	5-mini	4o-mini	4o-mini	$42.4 \pm 8.6\%$	54.1 ± 19.5	0.8	10.7 ± 4.1	4.4 ± 1.1	6.3 ± 3.9
TextCraft-4	Spec-Cyc	4o-mini	5-mini	5-mini	$78.8\% \pm 4.3\%$	42.5 ± 22.3	1.9	15.8 ± 11.7	4.0 ± 1.6	11.8 ± 11.1
TextCraft-4	Spec-Cyc	4o-mini	4o-mini	4o-mini	$18.2\% \pm 0.0\%$	71.0 ± 18.6	0.3	17.2 ± 3.6	4.2 ± 0.7	13.0 ± 3.7
TextCraft-4	Gen-Cyc	5-mini	4o-mini	4o-mini	$45.6 \pm 7.4\%$	58.1 ± 22.5	0.8	11.5 ± 4.9	3.9 ± 1.2	7.6 ± 4.2
TextCraft-4	Gen-Cyc	4o-mini	5-mini	5-mini	$100\% \pm 0.0\%$	29.5 ± 9.8	3.4	12.1 ± 5.5	3.3 ± 0.6	8.8 ± 5.4
TextCraft-4	Gen-Cyc	4o-mini	5-mini	4o-mini	$100\% \pm 0.0\%$	38.4 ± 18.3	2.6	16.4 ± 8.9	4.2 ± 0.7	12.2 ± 8.6
TextCraft-4	Gen-Cyc	4o-mini	4o-mini	5-mini	$84.9\% \pm 8.6\%$	44.6 ± 22.5	1.9	9.6 ± 4.6	3.3 ± 0.4	6.4 ± 4.4
TextCraft-4	Gen-Cyc	4o-mini	4o-mini	4o-mini	$48.5 \pm 4.3\%$	63.2 ± 23.9	0.8	12.3 ± 4.9	3.8 ± 0.9	8.5 ± 4.7
ALFWorld	Spec-Cyc	5-mini	5-mini	5-mini	$96.3\% \pm 0.6\%$	17.6 ± 13.5	5.5	5.2 ± 3.8	3.6 ± 0.9	1.7 ± 3.4
ALFWorld	Spec-Cyc	5-mini	4o-mini	4o-mini	$76.9\% \pm 2.7\%$	29.7 ± 18.5	2.6	2.6 ± 1.7	2.0 ± 1.1	0.6 ± 1.0
ALFWorld	Spec-Cyc	4o-mini	5-mini	5-mini	$85.8\% \pm 1.6\%$	22.0 ± 16.5	3.9	8.4 ± 8.4	3.6 ± 1.2	4.8 ± 8.1
ALFWorld	Spec-Cyc	4o-mini	4o-mini	4o-mini	$71.9\% \pm 2.1\%$	31.0 ± 18.9	2.3	3.1 ± 3.5	2.1 ± 1.1	1.0 ± 3.0
ALFWorld	Gen-Cyc	5-mini	4o-mini	4o-mini	$62.9\% \pm 1.5\%$	28.0 ± 19.1	2.2	1.7 ± 2.2	1.1 ± 1.0	0.6 ± 1.6
ALFWorld	Gen-Cyc	4o-mini	5-mini	5-mini	$91.5\% \pm 0.4\%$	19.6 ± 14.8	4.7	11.8 ± 9.3	5.1 ± 1.4	6.6 ± 8.5
ALFWorld	Gen-Cyc	4o-mini	4o-mini	4o-mini	$65.7\% \pm 3.7\%$	31.5 ± 20.5	2.1	2.8 ± 3.0	1.9 ± 1.3	0.9 ± 2.1
Finance-Agent	Spec-Cyc	5-mini	4o-mini	4o-mini	$15.2\% \pm 5.4\%$	6.5 ± 4.3	2.3	0.5 ± 0.8	0.5 ± 0.8	0.0 ± 0.0
Finance-Agent	Spec-Cyc	4o-mini	5-mini	5-mini	$30.5\% \pm 3.6\%$	7.1 ± 3.8	4.3	2.3 ± 2.3	1.8 ± 1.1	0.5 ± 1.6
Finance-Agent	Spec-Cyc	4o-mini	4o-mini	4o-mini	$17.1\% \pm 4.7$	4.6 ± 2.3	3.7	0.1 ± 0.2	0.1 ± 0.2	0.0 ± 0.0
Finance-Agent	Gen-Cyc	5-mini	4o-mini	4o-mini	$11.4\% \pm 2.3\%$	5.3 ± 3.5	2.2	0.4 ± 0.9	0.4 ± 0.9	0.0 ± 0.0
Finance-Agent	Gen-Cyc	4o-mini	5-mini	5-mini	$20.0\% \pm 0.0\%$	9.2 ± 4.4	2.2	2.6 ± 1.9	1.9 ± 1.5	0.7 ± 1.5
Finance-Agent	Gen-Cyc	4o-mini	4o-mini	4o-mini	$17.1\% \pm 2.3\%$	5.1 ± 2.6	3.4	0.4 ± 0.6	0.4 ± 0.6	0.0 ± 0.0

Table 10: Executor–router model-tier sweep with n-shot summaries and tool-restricted specialist executors. We vary router and executor model tiers (planner fixed) and report SR, TC (on successes), STR, and subtask-visitation metrics (ATS/AUS/ASR).

Benchmark	Graph	LLM Router	LLM Executor	SR (%)	TC	STR	ATS	AUS	ASR
TextCraft-2	Spec-Cyc	gpt-5-mini	gpt-4o-mini	$49.1\% \pm 2.9\%$	21.0 ± 4.8	2.3	4.3 ± 1.3	2.6 ± 0.7	1.7 ± 0.8

Continued on next page

Benchmark	Graph	LLM Router	LLM Executor	SR (%)	TC	STR	ATS	AUS	ASR
TextCraft-2	Spec-Cyc	gpt-4o-mini	gpt-5-mini	54.2% ± 1.1%	23.2 ± 5.0	2.3	6.3 ± 1.7	3.4 ± 0.8	2.9 ± 1.4
TextCraft-2	Spec-Cyc	gpt-4o-mini	gpt-4o-mini	39.7% ± 4.2%	21.9 ± 5.5	1.8	4.8 ± 1.5	3.0 ± 0.8	1.8 ± 1.1
TextCraft-2	Gen-Cyc	gpt-5-mini	gpt-4o-mini	86.7% ± 3.6%	19.6 ± 4.6	4.4	3.6 ± 0.9	2.3 ± 0.5	1.3 ± 0.6
TextCraft-2	Gen-Cyc	gpt-4o-mini	gpt-5-mini	89.5% ± 1.5%	21.1 ± 4.2	4.6	4.8 ± 1.2	2.7 ± 0.6	2.2 ± 0.9
TextCraft-2	Gen-Cyc	gpt-4o-mini	gpt-4o-mini	70.6% ± 0.9%	20.0 ± 4.8	3.5	3.6 ± 0.9	2.2 ± 0.4	1.4 ± 0.7
TextCraft-3	Spec-Cyc	gpt-5-mini	gpt-4o-mini	28.9% ± 2.1%	36.2 ±8.7	0.8	7.6 ± 2.1	3.7 ± 1.1	3.9 ± 1.5
TextCraft-3	Spec-Cyc	gpt-4o-mini	gpt-5-mini	29.3% ± 6.2%	38.3 ± 8.2	0.8	10.1 ± 2.4	4.3 ± 1.2	5.8 ± 2.0
TextCraft-3	Spec-Cyc	gpt-4o-mini	gpt-4o-mini	17.1% ± 3.4%	35.7 ± 7.8	0.5	7.7 ± 1.9	3.7 ± 1.1	4.0 ± 1.4
TextCraft-3	Gen-Cyc	gpt-5-mini	gpt-4o-mini	72.8% ± 4.5%	33.9 ± 8.3	2.1	6.2 ± 1.6	2.8 ± 0.7	3.4 ± 1.4
TextCraft-3	Gen-Cyc	gpt-4o-mini	gpt-5-mini	72.8% ± 3.2%	34.5 ± 8.5	2.1	8.1 ± 2.5	3.1 ± 0.7	5.0 ± 2.3
TextCraft-3	Gen-Cyc	gpt-4o-mini	gpt-4o-mini	38.6% ± 3.2%	33.8 ± 9.0	1.1	6.1 ± 1.7	2.7 ± 0.7	3.4 ± 1.5
TextCraft-4	Spec-Cyc	gpt-5-mini	gpt-4o-mini	27.3% ± 7.4%	68.0 ± 20.0	0.4	13.6 ± 5.0	3.9 ± 0.9	9.7 ± 4.7
TextCraft-4	Spec-Cyc	gpt-4o-mini	gpt-5-mini	27.3% ± 7.4%	79.3 ± 20.4	0.3	22.0 ± 7.0	4.4 ± 1.2	17.6 ± 7.0
TextCraft-4	Spec-Cyc	gpt-4o-mini	gpt-4o-mini	0.0% ± 0.0%	D/A	D/A	D/A	D/A	D/A
TextCraft-4	Gen-Cyc	gpt-5-mini	gpt-4o-mini	63.6% ± 0.0%	57.3 ± 22.1	1.1	10.3 ± 4.1	3.1 ± 0.8	7.3 ± 3.7
TextCraft-4	Gen-Cyc	gpt-4o-mini	gpt-5-mini	66.7% ± 11.3%	57.6 ± 17.6	1.2	13.5 ± 4.8	3.4 ± 0.7	10.1 ± 4.3
TextCraft-4	Gen-Cyc	gpt-4o-mini	gpt-4o-mini	15.2% ± 4.2%	84.6 ± 8.2	0.2	27.0 ± 2.8	5.0 ± 0.0	22.0 ± 2.8
Finance-Agent	Spec-Cyc	gpt-5-mini	gpt-4o-mini	3.8% ± 1.3%	4.5 ± 4.3	0.8	1.5 ± 0.9	1.3 ± 0.4	0.3 ± 0.4
Finance-Agent	Spec-Cyc	gpt-4o-mini	gpt-5-mini	3.8% ± 1.3%	8.0 ± 6.4	0.6	2.0 ± 1.2	1.5 ± 0.5	0.5 ± 0.9
Finance-Agent	Spec-Cyc	gpt-4o-mini	gpt-4o-mini	5.7% ± 4.0%	4.7 ± 2.4	1.2	1.5 ± 0.5	1.5 ± 0.5	0.0 ± 0.0
Finance-Agent	Gen-Cyc	gpt-5-mini	gpt-4o-mini	16.2% ± 3.6%	3.3 ± 3.6	4.9	1.2 ± 0.7	1.1 ± 0.2	0.2 ± 0.5
Finance-Agent	Gen-Cyc	gpt-4o-mini	gpt-5-mini	14.3% ± 2.3%	7.2 ± 4.5	2.0	2.5 ± 1.7	1.5 ± 0.7	1.0 ± 1.1
Finance-Agent	Gen-Cyc	gpt-4o-mini	gpt-4o-mini	10.5% ± 1.3%	2.0 ± 0.0	5.3	1.0 ± 0.0	1.0 ± 0.0	0.0 ± 0.0

Table 11: Robustness under control-flow fault injection for specialized executors (all components gpt-4o-mini). We compare unperturbed routing to perturbed routing (random subtask redirection) and report SR/TC along with absolute and relative performance deltas.

Benchmark	Graph	Unperturbed		Perturbed		Δ (Pert - Unpert)		Rel. Diff.
		SR (%)	TC	SR (%)	TC	SR (pp)	TC	
TextCraft-2	Spec-Cyc	39.7% ± 4.2%	21.9 ± 5.5	41.7% ± 2.5%	20.9 ± 6.1	2.0%	−1.0	4.8%
	Gen-Cyc	70.6% ± 0.9%	20.0 ± 4.8	64.0% ± 1.8%	17.8 ± 5.9	−6.6%	−2.2	9.3%
TextCraft-3	Spec-Cyc	17.1% ± 3.4%	35.7 ± 7.8	16.7% ± 3.2%	37.5 ± 9.6	−0.4%	+1.8	2.3%
	Gen-Cyc	38.6% ± 3.2%	33.8 ± 9.0	35.0% ± 4.1%	35.5 ± 10.1	−3.6%	+1.7	9.3%
TextCraft-4	Spec-Cyc	0.0% ± 0.0%	D/A	0.0% ± 0.0%	D/A	0%	D/A	D/A
	Gen-Cyc	15.2% ± 4.2%	84.6 ± 8.2	9.1% ± 7.4%	77.7 ± 5.4	−6.1%	−6.9	40.1%

Table 12: Robustness under control-flow fault injection for generalist executors (all components gpt-4o-mini). We compare unperturbed vs. perturbed routing and report SR/TC along with absolute and relative deltas, characterizing recovery under disrupted control flow.

Benchmark	Graph	Unperturbed		Perturbed		Δ (Pert - Unpert)		Rel. Diff.
		SR (%)	TC	SR (%)	TC	SR (pp)	TC	
TextCraft-2	Spec-Cyc	83.6% ± 1.3%	12.0 ± 6.7	49.4% ± 0.5%	19.5 ± 6.6	−34.2	+7.5	40.9%
	Gen-Cyc	93.8% ± 0.9%	9.9 ± 5.7	90.1% ± 0.7%	11.1 ± 5.6	−3.7%	+1.2	3.9%

Continued on next page

Benchmark	Graph	Unperturbed		Perturbed		Δ (Pert - Unpert)		Rel. Diff.
		SR (%)	TC	SR (%)	TC	SR (pp)	TC	
TextCraft-3	Spec-Cyc	$57.3\% \pm 1.0\%$	24.3 ± 11.2	$19.5\% \pm 4.6\%$	36.6 ± 9.6	-37.8%	$+12.3$	66.0%
	Gen-Cyc	$69.5\% \pm 1.0\%$	21.8 ± 12.4	$65.4\% \pm 2.3\%$	25.2 ± 11.9	-4.1%	$+3.4$	5.9%
TextCraft-4	Spec-Cyc	$18.2\% \pm 0.0\%$	71.0 ± 18.6	$0.0\% \pm 0.0\%$	D/A	-18.2%	D/A	D/A
	Gen-Cyc	$48.5\% \pm 4.3\%$	63.2 ± 23.9	$42.4\% \pm 4.3\%$	57.0 ± 24.0	-6.1%	-6.2	12.6%
ALFWorld	Spec-Cyc	$71.9\% \pm 2.1\%$	31.0 ± 18.9	$62.2\% \pm 0.4\%$	29.5 ± 18.7	-9.7%	-1.5	13.5%
	Gen-Cyc	$65.7\% \pm 3.7\%$	31.5 ± 20.5	$50.5\% \pm 2.0\%$	29.9 ± 20.1	-15.2%	-1.6	23.1%

Table 13: Unique transition counts (UT) for the generalist sweep with n-shot summaries. For each configuration, we report UT over all runs, UT restricted to successful episodes, and UT that never appears in any successful episode (indicative of transitions associated with failure modes or unproductive exploration).

Benchmark	Graph	Planner	Executor	Router	SR (%)	UT (all)	UT (won)	UT (never won)
TextCraft-2	Gen-Cyc	5-mini	4o-mini	4o-mini	$94.4\% \pm 2.2\%$	30	27	3
TextCraft-2	Gen-Cyc	4o-mini	5-mini	5-mini	$100\% \pm 0.0\%$	38	38	0
TextCraft-2	Gen-Cyc	4o-mini	5-mini	4o-mini	$99.7\% \pm 0.5\%$	25	23	2
TextCraft-2	Gen-Cyc	4o-mini	4o-mini	5-mini	$98.9\% \pm 0.2\%$	22	18	4
TextCraft-2	Gen-Cyc	4o-mini	4o-mini	4o-mini	$93.8\% \pm 0.9\%$	22	20	2
TextCraft-3	Gen-Cyc	5-mini	4o-mini	4o-mini	$68.7 \pm 2.1\%$	31	26	5
TextCraft-3	Gen-Cyc	4o-mini	5-mini	5-mini	$100\% \pm 0.0\%$	20	20	0
TextCraft-3	Gen-Cyc	4o-mini	5-mini	4o-mini	$95.5\% \pm 1.5\%$	26	25	1
TextCraft-3	Gen-Cyc	4o-mini	4o-mini	5-mini	$91.1\% \pm 2.1\%$	21	20	1
TextCraft-3	Gen-Cyc	4o-mini	4o-mini	4o-mini	$69.5\% \pm 1.0\%$	20	18	2
TextCraft-4	Gen-Cyc	5-mini	4o-mini	4o-mini	$45.6 \pm 7.4\%$	26	23	3
TextCraft-4	Gen-Cyc	4o-mini	5-mini	5-mini	$100\% \pm 0.0\%$	19	19	0
TextCraft-4	Gen-Cyc	4o-mini	5-mini	4o-mini	$100\% \pm 0.0\%$	21	21	0
TextCraft-4	Gen-Cyc	4o-mini	4o-mini	5-mini	$84.9\% \pm 8.6\%$	18	16	2
TextCraft-4	Gen-Cyc	4o-mini	4o-mini	4o-mini	$48.5 \pm 4.3\%$	33	26	7
ALFWorld	Gen-Cyc	5-mini	4o-mini	4o-mini	$62.9\% \pm 1.5\%$	35	30	5
ALFWorld	Gen-Cyc	4o-mini	5-mini	5-mini	$91.5\% \pm 0.4\%$	77	72	5
ALFWorld	Gen-Cyc	4o-mini	4o-mini	4o-mini	$65.7\% \pm 3.7\%$	69	59	10
Finance-Agent	Gen-Cyc	5-mini	4o-mini	4o-mini	$11.4\% \pm 2.3\%$	13	5	8
Finance-Agent	Gen-Cyc	4o-mini	5-mini	5-mini	$20.0\% \pm 0.0\%$	21	15	6
Finance-Agent	Gen-Cyc	4o-mini	4o-mini	4o-mini	$17.1\% \pm 2.3\%$	12	3	9

Table 14: Unique transition counts (UT) for the specialized-executor sweep with n-shot summaries. We report UT over all runs, UT in successful episodes, and UT never observed in successful episodes, enabling comparison of transition diversity and "never-won" behaviors under tool restriction.

Benchmark	Graph	LLM Router	LLM Executor	SR (%)	UT (all)	UT (won)	UT (never won)
TextCraft-2	Gen-Cyc	gpt-5-mini	gpt-4o-mini	$86.7\% \pm 3.6\%$	24	22	2
TextCraft-2	Gen-Cyc	gpt-4o-mini	gpt-5-mini	$89.5\% \pm 1.5\%$	24	22	2
TextCraft-2	Gen-Cyc	gpt-4o-mini	gpt-4o-mini	$70.6\% \pm 0.9\%$	25	24	1
TextCraft-3	Gen-Cyc	gpt-5-mini	gpt-4o-mini	$72.8\% \pm 4.5\%$	22	21	1
TextCraft-3	Gen-Cyc	gpt-4o-mini	gpt-5-mini	$72.8\% \pm 3.2\%$	25	25	0

Continued on next page

Benchmark	Graph	LLM Router	LLM Executor	SR (%)	UT (all)	UT (won)	UT (never won)
TextCraft-3	Gen-Cyc	gpt-4o-mini	gpt-4o-mini	$38.6\% \pm 3.2\%$	25	23	2
TextCraft-4	Gen-Cyc	gpt-5-mini	gpt-4o-mini	$63.6\% \pm 0.0\%$	24	21	3
TextCraft-4	Gen-Cyc	gpt-4o-mini	gpt-5-mini	$66.7\% \pm 11.3\%$	22	21	1
TextCraft-4	Gen-Cyc	gpt-4o-mini	gpt-4o-mini	$15.2\% \pm 4.2\%$	22	17	5
Finance-Agent	Gen-Cyc	gpt-5-mini	gpt-4o-mini	$16.2\% \pm 3.6\%$	4	3	1
Finance-Agent	Gen-Cyc	gpt-4o-mini	gpt-5-mini	$14.3\% \pm 2.3\%$	4	2	2
Finance-Agent	Gen-Cyc	gpt-4o-mini	gpt-4o-mini	$10.5\% \pm 1.3\%$	7	2	5

Table 15: Observed Gen-Cyc graph size by benchmark. Reported ranges summarize the extracted benchmark-generic graphs across observed configurations. Criterion length is measured in words per edge-transition criterion.

Benchmark	Average Nodes (observed range)	Avg. criterion length (words)
ALFWorld	6–9	7.2–30.2
Finance-Agent	3–7	8.3–28.6
TextCraft-2	5.5–7	9.2–24.1
TextCraft-3	6–7	9.2–24.1
TextCraft-4	5	11.0

Table 16: Observed Gen-Cyc routing behavior by benchmark. Values are averaged per test case and summarized as ranges across observed Gen-Cyc configurations. Self-loops correspond to repeating the current subtask; inter-node transitions correspond to routing to a different subtask.

Benchmark	Avg. self-loops / test case	Avg. inter-node transitions / test case
ALFWorld	1.0–3.3	1.5–11.5
Finance-Agent	0.0–1.6	0.2–3.6
TextCraft-2	0.2–3.8	1.1–3.1
TextCraft-3	2.1–6.4	2.6–3.9
TextCraft-4	4.4–11.9	5.2–7.7

Table 17: Observed Spec-Cyc graph size by benchmark. Reported ranges summarize the extracted task-specific graphs across observed configurations. Criterion length is measured in words per edge-transition criterion.

Benchmark	Average Nodes (observed range)	Avg. criterion length (words)
ALFWorld	4.7–5.2	8.0–24.0
Finance-Agent	4.0–5.6	12.3–32.7
TextCraft-2	3.9–5.1	9.4–21.2
TextCraft-3	4.7–4.8	8.8–8.9
TextCraft-4	5.0–6.4	8.7–22.1

Table 18: Observed Spec-Cyc routing behavior by benchmark. Values are averaged per test case and summarized as ranges across extracted Spec-Cyc configurations with complete transition statistics. Self-loops correspond to repeating the current subtask; inter-node transitions correspond to routing to a different subtask.

Benchmark	Avg. self-loops / test case	Avg. inter-node transitions / test case
ALFWorld	0.9–2.9	1.4–9.4
Finance-Agent	0.0–1.5	0.2–2.2
TextCraft-2	0.3–0.7	1.6–3.5
TextCraft-3	2.7–2.8	5.2–7.4
TextCraft-4	6.3–11.0	10.6–14.5

E Environment-level exploration on ALFWorld

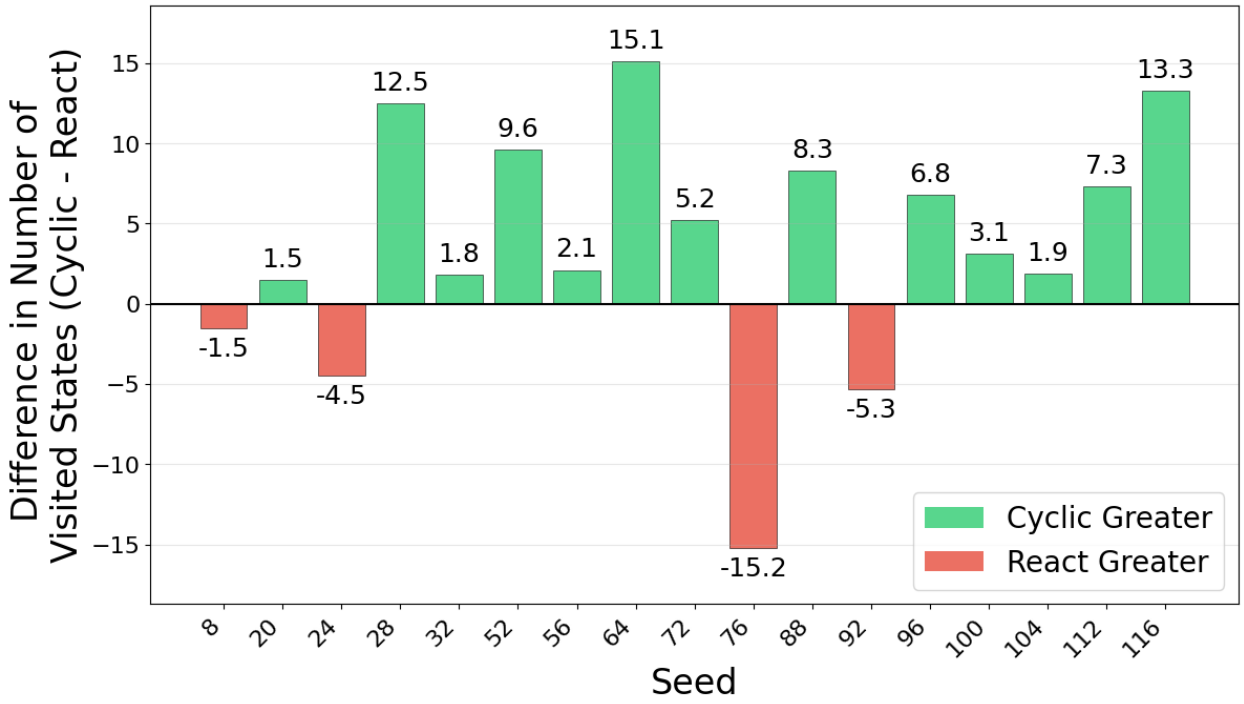

Figure 4: ALFWorld environment-level exploration study: difference in unique environment states visited per episode (cyclic minus ReAct) across the 30-task subsample.

F Routing complexity

Routing requires at most $O(|T|)$ criterion-evaluation work per subtask segment, assuming the router state is truncated/summarized to a bounded context window. Let $T = \{t_1, \ldots, t_n\}$ with $n = |T|$. At the end of segment m, the router is at node t_{i_m} and considers an admissible set of next-node indices $\mathcal{A}_m \subseteq \{1, \ldots, n\}$. The routing decision is made by evaluating the outgoing transition criteria

$$\{c_{i_m \to j}\}_{j \in \mathcal{A}_m}, \tag{9}$$

i.e., one criterion per admissible candidate j.

Counting argument. Define the routing work at segment m as the number of criterion evaluations performed:

$$W_m \coloneqq |\mathcal{A}_m| \,. \tag{10}$$

Since $\mathcal{A}_m \subseteq \{1, \ldots, n\}$, it follows immediately that

$$W_m = |\mathcal{A}_m| \leq n = |T|. \tag{11}$$

Therefore, the number of criterion evaluations per segment is at most $|T|$, i.e.,

$$W_m \in O(|T|). \tag{12}$$

Relating evaluations to LLM cost. Let $C_{\text{eval}}(m)$ denote the cost of evaluating a single criterion $c_{i_m \to j}$ against the router memory (e.g., measured in tokens processed or bounded wall-clock time under fixed generation limits). If each criterion is evaluated independently, then the routing cost at segment m satisfies

$$C_{\text{route}}(m) = |\mathcal{A}_m| \cdot C_{\text{eval}}(m) \leq |T| \cdot C_{\text{eval}}(m). \tag{13}$$

If $C_{\text{eval}}(m)$ is bounded by a fixed prompt/template and fixed decoding limits, then $C_{\text{route}}(m) \in O(|T|)$ as a function of $|T|$.

One-call implementation. If routing is implemented as a single LLM call that reads all criteria $\{c_{i_m \to j}\}_{j \in \mathcal{A}_m}$ at once and outputs the next index, then the number of LLM calls per segment is 1, but the input length grows linearly with $|\mathcal{A}_m| \leq |T|$. In this case, the routing cost remains linear in the number of criteria inspected, i.e., $O(|T|)$ in the size of the candidate set.

G Qualitative case studies

G.1 Qualitative routing pathology case study (ALFWorld) on successful trajectory

```
Spec-Cyc trajectory excerpt (ALFWorld, OOD): task = "put a pencil in shelf"; seed=14; won; steps=2; tool
calls=34.

Key pattern:  successful error correction via revisitation (backtracking to recover misplaced object).

Segment 1:  Locate_pencil (misplacement → backtrack)
go desk 1 → see {...  pencil 1}; take pencil 1
error: move pencil 1 to desk 1 (instead of shelf)
go shelf 1 → see {keychain 2} (pencil absent)
go desk 1 → pencil 1 visible again
go shelf 1 → ...
take keychain 2 from shelf 1; distractor: move keychain 2 to shelf 1
go desk 1 → see pencil 1; take pencil 1
repeat error: move pencil 1 to desk 1 ...
...  (repeats desk↔shelf to re-acquire pencil after each misplacement) ...
Router analysis:  identifies the loop (repeatedly returning pencil to desk) and transitions to PLACE_PENCIL.

Segment 2:  Place_pencil (loop detection → targeted revisitation)
go shelf 1 → see {keychain 2}; take keychain 2; move keychain 2 to shelf 1
navigation drift: go shelf 2 → see {alarmclock 1} → go shelf 1
...  (repeats keychain take/move and shelf1↔shelf2 oscillation) ...
Router analysis:  detects the distraction loop and revisits LOCATE_PENCIL to re-anchor on the target (desk
1) rather than continuing the oscillation.
terminates successfully within budget.
```

Figure 5: Illustrative Spec-Cyc ALFWorld success showing error correction through revisitation: after misplacing the pencil (moving it back to the desk), the controller backtracks to the prior locating context to re-acquire the target and re-attempt placement, rather than committing to a forward-only plan. All agents here are `gpt-4o-mini` with generalist executors.

G.2 Qualitative routing pathology case study (ALFWorld) on unsuccessful trajectory

```
Spec-Cyc trajectory excerpt (ALFWorld, OOD, seed=2): task = "put a clean egg in microwave"; lost; steps=7;
tool calls=75.

Step 1: Locate_clean_egg
... go/open cabinets ...
open fridge 1 → see {... egg 1 ...}
take egg 1; examine → "cold egg 1"
error: move egg 1 to fridge 1 (instead of progressing to microwave)
Router analysis: "mistakenly moved the egg back to the fridge ... stuck in a loop ..."
Next subtask → TAKE_CLEAN_EGG.

Step 2: Take_clean_egg
go microwave 1; go fridge 1; inventory → "carrying: egg 1"
error: move egg 1 to fridge 1 ... take egg 1 ... move egg 1 to fridge 1 ... (repeats)
Router analysis: "caught in a repetitive loop of moving egg back to fridge ..."
Next subtask → PLACE_CLEAN_EGG_IN_MICROWAVE.

Step 3: Place_clean_egg_in_microwave
take egg 1 from fridge 1
error: move egg 1 to fridge 1
go/open microwave 1 → contains apple 1, potato 1
distractor loop: take apple 1; move apple 1 to microwave 1; take potato 1; move potato 1 to microwave 1; ...
(repeats)
Router analysis: "moved egg back to fridge ... stuck in a loop ..."
...

Step 5: Place_clean_egg_in_microwave (later succeeds)
go microwave 1; move egg 1 to microwave 1 (subtask success)
Next subtask → CLEAN_EGG.

Step 6: Clean_egg
take egg 1 from microwave 1; go sinkbasin 1; clean egg 1 with sinkbasin 1 (confirmed)
error: move egg 1 to sinkbasin 1 ... take egg 1 ... move egg 1 to sinkbasin 1 ... (repeats)
Router analysis: "cleaning confirmed, but objective still not met ... examine egg to verify ..."
Next subtask → TAKE_CLEAN_EGG.

Step 7: Take_clean_egg
examine egg 1 → "cool and clean egg 1"
Router analysis: "egg confirmed clean, yet objective still not met ..."
Next subtask → PLACE_CLEAN_EGG_IN_MICROWAVE.
```

Figure 6: Illustrative Spec-Cyc ALFWorld failure trace showing oscillation between task-specific nodes (TAKE/PLACE/CLEAN) and feedback-entangled criteria ("objective not met"), consistent with the discussion claim that higher expressivity can increase criterion ambiguity and brittleness. All agents here are `gpt-4o-mini` with generalist executors. This trace also highlights a task-specific ambiguity: node labels encode high-precision predicates (e.g., "LOCATE *CLEAN* EGG") while the environment initially only reports "cold"; this can bias routing toward believing the node intent over the actual observations, yielding premature transitions into "clean-egg" subtasks and brittle oscillation when executor actions contradict the intended predicate.

G.3 Qualitative executor-style pathology case studies (TextCraft-3) on unsuccessful trajectory

```
Gen-Cyc trajectory excerpt (TextCraft-3, specialized executor): task = "craft polished granite"; seed=85;
lost; steps=10; tool calls=51.

Key pattern: repeated self-loop on Gather_materials under tool-restricted specialization.
The specialized "raw-materials" executor repeatedly queries for diorite/granite as if they are directly
gatherable, leading to unproductive retries and budget burn rather than progressing to prerequisite
crafting.

Step 1: Gather_materials
try get diorite → Could not find diorite
try get granite → Could not find granite
get quartz → Got 4 quartz
Router: "attempted to gather materials but did not collect any" ... Next → GATHER_MATERIALS (self-loop).

Step 2: Gather_materials (self-loop)
get granite → Could not find granite
get diorite → Could not find diorite
get quartz → Got 4 quartz
Router: "has 4 quartz, 0 diorite, 0 granite ... try gathering diorite again" ... Next →
GATHER_MATERIALS.

Steps 3-6: Gather_materials (repeated self-loops)
... repeated attempts: get diorite → Could not find diorite; get granite → Could not find granite ...
Router repeatedly concludes: "still needs 4 diorite ... attempt to gather again" ...

Step 7: Gather_materials (partial prerequisite attempt, still fails)
get cobblestone → Got 8 cobblestone
get quartz → Got 4 quartz (now 8 quartz total)
attempt get/craft diorite → Could not find diorite
Router: "attempted to gather 4 diorite ... unsuccessful" ... Next → GATHER_MATERIALS.

Steps 8-10: Gather_materials (terminal budget burn)
... repeated Could not find diorite (and occasional Could not find granite) ...
End: still has quartz (4-8) and cobblestone (8), but never reaches the intermediate crafting needed to
obtain granite and thus polished granite.
```

Figure 7: Illustrative Gen-Cyc TextCraft-3 failure under a specialized raw-materials executor: the controller repeatedly self-loops on GATHER_MATERIALS, issuing direct "get" queries for non-gatherable intermediates (`diorite`/`granite`) and consuming budget with retries. This qualitative trace supports the coordination-tax view that tool-restricted specialization can underperform generalists when intermediate prerequisites must be inferred and executed via cross-step coordination. All agents here are `gpt-4o-mini`.

```
Gen-Cyc trajectory excerpt (TextCraft-3, generalist executor): task = "craft polished granite"; seed=85;
lost; steps=10; tool calls=50.

Key pattern: the generalist can craft intermediates, but still fails under action-selection drift and
goal/recipe misalignment.
Unlike the specialized raw-materials executor, the generalist occasionally performs prerequisite crafting
(e.g., craft diorite), but repeatedly issues wrong craft commands (unrelated items) and repeatedly attempts
terminal goals (craft polished_granite) before crafting granite.

Step 1: Identify_target_item
attempt craft polished_granite → Could not find enough items ...
get quartz → Got 4 quartz; try get diorite → Could not find diorite
Next → ACQUIRE_RAW_MATERIALS.

Step 2: Acquire_raw_materials
get granite → Could not find granite; get diorite → Could not find diorite
inventory → [quartz] (4); get quartz → Got 4 quartz (now 8); get cobblestone → Got 4 cobblestone
Next → ACQUIRE_RAW_MATERIALS.

Step 3: Acquire_raw_materials (drift to oversized requests)
... get granite → Could not find granite; get diorite → Could not find diorite ...
get 100 cobblestone → Got 100 cobblestone; get 100 quartz → Got 100 quartz.
Next → DETERMINE_RAW_MATERIALS.

Steps 4-5: Determine_raw_materials (still treats intermediates as gatherable)
... repeated get diorite/get granite → Could not find ...
inventory → large quartz/cobblestone counts.
...

Step 6: Determine_raw_materials (partial progress, then wrong crafts)
craft 2 diorite using 2 quartz, 2 cobblestone → Crafted 2 diorite
wrong craft: craft dark oak planks → fails
premature terminal craft: craft polished_granite → fails (no granite)
... get diorite → Could not find diorite; get quartz → Got 2 quartz
Next → ACQUIRE_RAW_MATERIALS.

Step 9: Determine_raw_materials (repeat: crafts diorite, skips granite)
inventory → ... [diorite] (2); craft 2 diorite → Crafted 2 diorite (now 4)
still skips granite: craft polished_granite → fails
Next → ACQUIRE_RAW_MATERIALS.

Step 10: Acquire_raw_materials (terminal drift persists)
get granite → Could not find granite; inventory → [diorite] (4)
wrong craft (repeats): craft dark oak planks → fails
premature terminal craft (repeats): craft polished_granite → fails
End: substantial quartz/cobblestone and some diorite, but never crafts granite → cannot craft
polished_granite.
```

Figure 8: Illustrative Gen-Cyc TextCraft-3 failure under a generalist executor. The generalist occasionally makes prerequisite progress (e.g., successfully crafting `diorite`), but still exhibits action-selection drift: it issues unrelated craft commands (e.g., `dark_oak_planks`) and repeatedly attempts the terminal goal (`polished_granite`) without executing the missing intermediate (`granite`). Compared to the specialized trace (Appendix Fig. 7), the failure mode shifts from tool-restriction-induced dead-ends to mis-executed command selection, underscoring that generalist access is necessary but not sufficient when routing/execution is weak. All agents here are `gpt-4o-mini`.