WHEN AGENTS SHOULD NOT REMEMBER: HABITUAL EXECUTION FOR RETRIEVAL-EFFICIENT LONG-RUNNING LLM AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Long-running LLM agents often retrieve long-term memory even on stable, repeated workflows, which adds retrieval cost, stale context, and operational complexity. Reducing retrieval is not simple caching: the agent must decide, before retrieval, which workflows are SOP-stable, and must interrupt an automatic routine when policy, tool, or user-goal signals show that the context has changed. We propose habitual execution, a complementary control path. A Habitual Memory Layer compiles verified successful trajectories into guarded routines that can bypass explicit retrieval when the context is stable, and return control to retrieval when the routine no longer applies. On tool-using airline and retail customer-service tasks, HM-Gated RAG matches always-retrieve RAG in mean success over four trials (airline 0.667 vs. 0.675; retail 0.497 vs. 0.506; 95% CIs on the paired gap are $[-8.3, +6.7]$ and $[-6.9, +5.0]$ percentage points, both crossing zero) while cutting retrieval calls by about 59% and 51% (95% CIs $[40.7\%, 76.1\%]$ and $[40.0\%, 61.9\%]$). The saving requires offline task-type tags; keyword and no-tag routers do not produce it. The method is not a general way to reduce local tokens or latency.

1 INTRODUCTION

Many deployed LLM agents are long-running tool-using systems: they handle related user requests over time, call external tools, and carry operational context across sessions. In customer service, an agent may repeatedly handle booking changes, returns, refunds, and account updates while consulting domain policies and transactional tools. Such agents need memory because later decisions can depend on user state, prior actions, preferences, and policy context.

Existing systems emphasize what to store and retrieve: working, episodic, semantic, and procedural memories, vector stores, reflective lessons, and virtual-context mechanisms (Zhang et al., 2024; Du, 2026; Sumers et al., 2024; Park et al., 2023; Shinn et al., 2023; Zhao et al., 2024; Packer et al., 2023; Chhikara et al., 2025). In production workflows, many tasks are stable and repetitive, so repeated retrieval adds remote-store latency, policy-chunk injection, and exposure to stale or irrelevant context. The decision must be made before retrieval, and bypassing memory is unsafe when policies, tools, or user goals shift. This raises a complementary question: when should an agent not pay the full cost of remembering the way it did last time? Existing memory systems largely optimize how agents store, retrieve, or compress memory, but rarely treat retrieval bypass itself as a control decision.

Human habit offers a useful engineering analogy: repeated routines can run with weak explicit recall, but automaticity must stop when context changes (Wood & Neal, 2007). We propose habitual execution: a memory control layer that compiles successful trajectories into guarded routines with triggers, confidence, fallback, and lifecycle metadata. At inference time, a router treats the artifact as an interruptible routine: it may bypass retrieval only while the trigger and guards hold, and otherwise restores retrieval-based reasoning. The goal is controlled memory bypass, not forgetting. An airline "add passenger" habit, for example, stores a trigger, a short tool routine, guards such as "policy unchanged" and "user goal unchanged," a confidence score, and a fallback to policy retrieval and replanning.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

We evaluate an HDB-style adaptation of τ^2 airline and retail tasks with routine/disrupted pairs and habit-formation episodes. The efficiency target is the number of explicit retrieval calls, and whether mean task success matches always-retrieve RAG when the workflow type is observable. Reported gains are on the retrieval side, not a general drop in latency or tokens.

Paper focus. We study whether a tool-using agent can treat memory access as a pre-retrieval control decision and thereby reduce explicit memory retrieval. HM-Gated RAG routes each episode to habitual execution or RAG. On the habit path, it falls back to retrieval when the model declares that the routine no longer applies, or when a tool message returns an error signal. When the workflow type is observable, this control reduces retrieval calls and matches RAG in mean success.

Contributions.

- We formulate memory access as a pre-retrieval control decision: retrieve, bypass via a routine, or restore retrieval when disruption signals make automaticity unsafe.
- We propose a Habitual Memory Layer whose habit artifacts expose trigger, routine, guards, confidence, fallback, and lifecycle metadata.
- We introduce HM-Gated RAG and show that tag-based routing matches RAG in mean success while reducing retrieval calls, whereas keyword and no-tag routers do not yield that saving.

2 Background and Related Work

Agent-memory surveys and CoALA organize memory by source, representation, and read/write operation, including working, episodic, semantic, and procedural memory (Zhang et al., 2024; Du, 2026; Sumers et al., 2024). Retrieval systems such as Mem0 and Zep optimize long-term fact and preference lookup (Chhikara et al., 2025; Rasmussen et al., 2025). Reflective systems such as Generative Agents, Reflexion, and ExpeL consolidate experience into future prompts (Park et al., 2023; Shinn et al., 2023; Zhao et al., 2024). MemGPT-style virtual context pages relevant memory into a finite window (Packer et al., 2023). These methods improve how agents read memory. We ask when a mature routine can bypass that read.

Reusable skill libraries such as Voyager store executable artifacts (Wang et al., 2023). Habitual execution adds a control policy around such artifacts: explicit triggers, guard checks, confidence, and fallback. Adaptive retrievers such as Self-RAG and CRAG decide at each step whether to retrieve and what to retrieve (Asai et al., 2023; Yan et al., 2024). Habitual execution asks a complementary question: when the workflow is stable, can the agent skip retrieval entirely and run a guarded routine, re-enabling retrieval only on interruption? LongMemEval and MemoryArena move memory evaluation toward long-horizon agent settings (Wu et al., 2024; He et al., 2026). Our HDB-style evaluation asks a narrower operational question: retrieval efficiency under stable repetition, and whether mean success matches always-retrieve RAG, when the workflow type is observable. The cognitive analogy is habit as context-triggered action (Wood & Neal, 2007), operationalized as an auditable engineering abstraction.

3 Problem Formulation

We consider a long-running tool-using agent that observes input x_t, maintains context c_t, acts under goal g_t, and can access long-term memory M_t and a habit store $\mathcal{H}_t$. The agent may follow context-only reasoning, retrieval-conditioned acting, or habitual execution from a compiled routine. A habit stores a trigger, a routine, guards, a confidence, and a fallback: the trigger defines applicability, the routine encodes a structured action sequence, guards check safety and disruption, confidence estimates reliability, and fallback restores retrieval-based reasoning.

The decision is which path to take before paying for long-term retrieval. Let $h \in \mathcal{H}_t$ be a candidate habit. Habitual execution is eligible only if the trigger matches, all guards pass, confidence exceeds a threshold τ, and estimated risk stays within a budget ρ:

$$\begin{aligned} \mathrm{elig}(h, x_t, g_t) = \; & \mathrm{match}(h.\mathrm{trig}, x_t, g_t) \wedge \mathrm{guards}(h, x_t) \\ & \wedge\, h.\mathrm{conf} \geq \tau \wedge \mathrm{risk}(x_t, h) \leq \rho. \end{aligned} \tag{1}$$

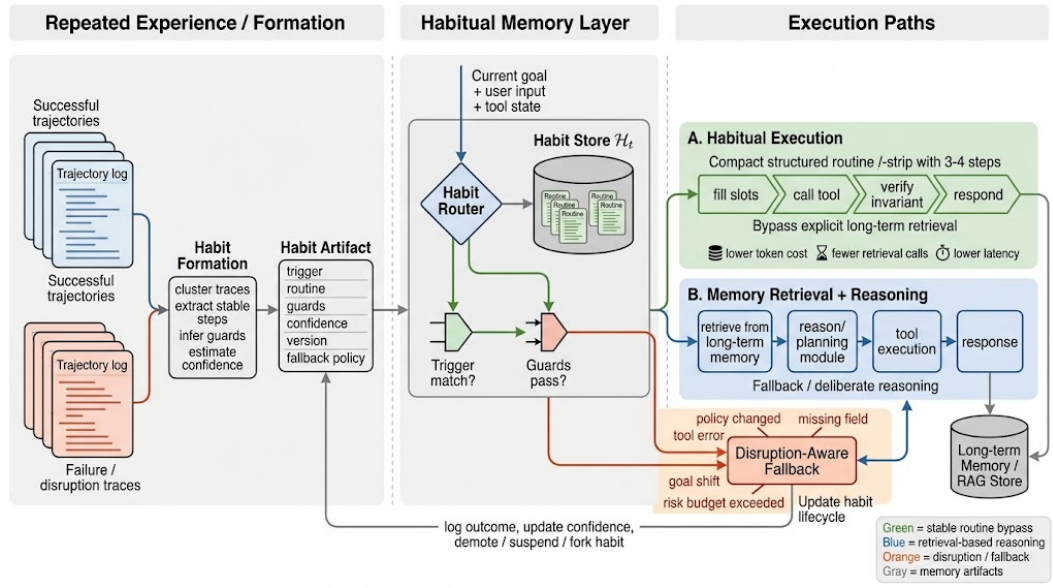

Figure 1: Habitual Memory Layer. Repeated trajectories form guarded habit artifacts. At inference time the router either executes an eligible routine or returns to retrieval. In the reported experiments, routing uses offline `eval_tags` rather than a learned trigger matcher; habits are compiled from τ^2 `reference_actions`; guards are prompt self-checks; runtime fallback uses only the two signals in Section 4.3. Lifecycle updates are not run online.

If at least one habit is eligible, the agent may execute the highest-scoring routine without retrieving M_t; otherwise it retrieves. During execution, guard failure, a tool error, a missing signal, a user-goal change, or a risk-budget violation interrupts the routine and triggers fallback. Habitual memory is thus a control layer over long-term memory: it reduces retrieval on stable workflows while keeping a path back to retrieval.

4 Method: Habitual Memory Layer

Figure 1 places a habit router before long-term retrieval. Given goal g_t, input x_t, tool state, and habit store $\mathcal{H}_t$, the router executes a compiled routine, uses context-only reasoning, or falls back to retrieval and planning. Routines are structured action templates. Triggers are lightweight predicates. Guards check whether execution is still inside the routine's validity region.

4.1 Habit Formation

In the abstract layer, habits are formed by clustering repeated trajectories, extracting stable tool-call sequences, inferring guards from disruptions, and estimating confidence from success rate and formation count k. New evidence can promote, demote, suspend, or fork a version. In the experiments, formation uses successful τ^2 `reference_actions` through `GoldSyntheticProvider`, with k=3 identical synthetic trajectories. Confidence is $\min(0.95, 0.55 + 0.08k)$, the same heuristic as `formation.py`, and is not a calibrated probability. Clustering, guard inference from disruptions, and online version updates are not run. Procedural Prompt uses the same demonstrations; the contrast is static prompt injection versus a compiled routine with guard text and a fallback path.

4.2 Habit Execution

At inference time the agent attempts a habit match before retrieval. A matched habit is executed as a structured routine rather than as a long natural-language memory. We measure retrieval calls. End-to-end tokens and latency are diagnostics, not the primary result. The reported experiments do

not run a multi-habit scorer: the initial path is the tag route in Section 4.4. Context-only reasoning is the no-memory baseline, not a branch of HM-Gated RAG.

4.3 Disruption-Aware Fallback

In the abstract layer, interruption returns control to retrieval when the trigger no longer holds, a required signal is missing, a routine step fails, the user goal changes, or the risk budget is exceeded. The reported system does not wire all of these conditions, and it does not update the habit lifecycle online. Fallback is checked only when the initial route is `habit_only` and the episode has not already fallen back. It fires if the model writes `fallback=true` or `eligible=false` inside `<hm_router>`, or if the latest tool message contains a risk word such as error, failed, unavailable, mismatch, invalid, or denied. Later turns then use the RAG prompt. Tool calls already executed are not rolled back.

4.4 Instantiation

Routing uses offline `metadata.eval_tags`: `sop_heavy` maps to `habit_only` and `policy_heavy` to `rag_first`, unless an ablation sets `HDB_ROUTER_MODE` to `no_tag` or `keyword`. On `habit_only`, the system prompt contains the habit text and that episode does not retrieve. `rag_first` is pure RAG and does not overlay habit text. Guards are prompt self-checks. Runtime fallback uses only the two signals above. RAG retrieves τ^2 policy chunks at top-k=4. Under `no_tag`, a habit forces the initial route to `rag_first`. Under `keyword`, policy keywords in the instruction or policy text select `rag_first`; otherwise the route is `habit_only`.

5 Experimental Setup

5.1 Protocol

We use an HDB-style protocol on τ^2-bench customer-service domains. It defines paired routine and disrupted episodes, habit formation from k successful trajectories, and metrics for retrieval efficiency and disruption robustness. Primary experiments use τ^2 airline (30 episodes) and retail (80 episodes) and report the mean of four trials (seeds 42, 43, 44, and 45). We do not release a standalone HDB beyond these adapters. The underlying domains, tools, policies, and tasks are unchanged. Each episode has three phases: formation with $k \in \{1, 3, 5, 10\}$, routine evaluation, and disruption evaluation. The main experiment uses k=3. Disruptions include user-goal shifts, payment restrictions, and return-policy exceptions.

5.2 Baselines and Routing

We compare No Memory, Full Context, RAG, Procedural Prompt (a handwritten SOP), Habitual Memory (habit-first, with guards), and HM-Gated RAG. No Memory and Full Context run on airline only. Procedural Prompt injects τ^2 `reference_actions` and policy excerpts on every turn. Habit formation uses the same references at k=3.

The HM-Gated router is a heuristic tag policy, not a learned classifier. `sop_heavy` starts at `habit_only` and `policy_heavy` at `rag_first`. Routine and disrupted variants of the same task share tags, so the route reflects task type rather than the condition label. Guards are prompt self-checks (Table 5). Runtime `rag_fallback` uses the two signals in Section 4.3. Router telemetry is logged in `<hm_router>` blocks.

The main HM-Gated results use the tag router. On retail we ablate `HDB_ROUTER_MODE` over four trials: `tag`, `no_tag` (always initial `rag_first` when a habit exists), and `keyword` (route from instruction or policy keywords, without tags). All variants share habits, guards, and the RAG store.

5.3 Models and Metrics

The four-trial results use `qwen-plus` via DashScope/LiteLLM. Table 4 also reports one airline run each of `qwen-turbo` and `qwen2.5-7b-instruct`. RAG embeds domain policy chunks from the τ^2 manifests, filters by task id, and retrieves top-k=4.

Table 1: Main results: means over four trials (seeds 42–45). Procedural Prompt is a handwritten-SOP upper bound. Habitual Memory is a zero-retrieval control. No Memory and Full Context run on airline only.

	Airline		Retail	
Method	**SR**	**Retr.**	**SR**	**Retr.**
No Memory	0.533	0	—	—
Full Context	0.625	0	—	—
Procedural Prompt	0.700	0	0.566	0
RAG	0.675	280.8	0.506	847.2
Habitual Memory	0.633	0	0.506	0
HM-Gated RAG	**0.667**	**113.8**	**0.497**	**417.0**

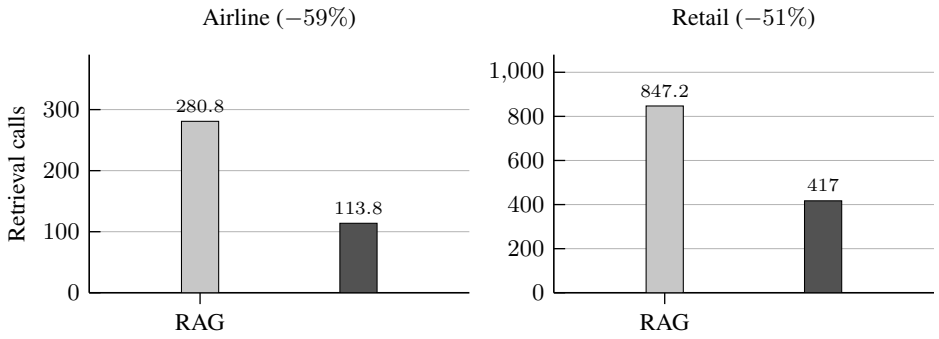

Figure 2: Mean retrieval calls over four trials. Panels use separate vertical scales. Airline: RAG 280.8, HM-Gated RAG 113.8. Retail: RAG 847.2, HM-Gated RAG 417.0.

Primary numbers are mean success and mean retrieval calls over four trials, plus 95% confidence intervals for the paired success gap and the retrieval reduction. Intervals resample episodes and keep the four paired trials inside each episode. We also report end-to-end tokens and latency, wrong habit activation, false-negative fallback, disruption recovery, and habit-to-RAG fallback. These describe what happens besides the retrieval saving. They are not evidence that success is higher or that an interruption was caught. Wrong habit activation means a disrupted episode succeeds and its trace covers every routine `reference_action`. False-negative fallback means a disrupted `habit_only` episode fails without reaching `rag_fallback`. Disruption recovery is success on a disrupted episode. Safety counts come from the rule grader and were not manually audited.

6 Results and Analysis

Table 1 reports the four-trial means. Higher success is better and fewer retrieval calls are better. HM-Gated RAG matches RAG (airline 0.667 vs. 0.675; retail 0.497 vs. 0.506). The 95% CIs on the paired gap (HM-Gated minus RAG) are $[-8.3, +6.7]$ and $[-6.9, +5.0]$ percentage points; both cross zero. Retrieval calls fall by about 59% and 51%.

Habitual Memory matches RAG on retail (0.506) at zero retrieval and is lower on airline (0.633 vs. 0.675). HM-Gated keeps both domains near RAG while still retrieving on policy-heavy tasks. Table 2 is one seed-42 run, not the four-trial mean. On that run, retail routine and disrupted success are both below the paired RAG run. Wrong activation is zero. Fallbacks are 2/16 on airline and 14/52 on retail. The largest same-route gap versus RAG is $+14.3$pp on airline and $+7.1$pp on retail, both on `rag_first`. These counts show whether the Section 4.3 signals fired. They do not show that fallback raised success. Among disrupted retail episodes that start at `habit_only`, false-negative fallback is 9/26.

Table 2: Compact diagnostics from one seed-42 run, not the four-trial mean in Table 1. Best route gain is versus RAG on the same route subset.

Diagnostic (HM-Gated)	Airline	Retail
Routine SR vs. RAG	0.667 / 0.600	0.425 / 0.500
Disrupted SR vs. RAG	0.667 / 0.733	0.525 / 0.575
Wrong activation	0/15	0/40
Disruption recovery	10/15	21/40
Fallback (habit→rag)	2/16	14/52
Best route gain	+14.3pp	+7.1pp

Table 3: Retail router ablation, four-trial means (n=80 per trial). %RAG is retrieval calls relative to RAG.

Router	SR	Retr.	%RAG
RAG (reference)	0.506	847.2	100%
Tag (main)	0.497	417.0	49.2%
No tag	0.478	863.8	102.0%
Keyword	0.512	855.0	100.9%

Table 3 shows that only the tag router cuts retrieval below RAG while matching its success. No-tag success is 0.478, below RAG. Keyword success is 0.512, only slightly above RAG, and neither variant retrieves less than RAG.

Table 4 is one airline run per model. On that run, HM-Gated helps `qwen-turbo` (+10pp success, -13.5% latency) and matches `qwen-plus` at 43.4% of RAG's retrieval calls. On `qwen2.5-7b-instruct` it is 3.3pp below RAG and increases retrieval by 8.7%. We would not enable the habit path on that weaker backbone without tighter guards. The four-trial `qwen-plus` comparison remains Table 1.

Cost. The measured gain is fewer embedding searches and policy-chunk injections (four-trial bootstrap 95% CIs $[40.7\%, 76.1\%]$ airline and $[40.0\%, 61.9\%]$ retail). Local end-to-end cost does not follow that drop. Airline tokens and latency rise ($+16.2\%$ / $+12.4\%$) and model calls rise by about 6%. Retail tokens fall by 5.1% and latency is $+0.1\%$, with almost unchanged model calls. The split is in episodes that finish the habit at zero retrieval. Those airline episodes take about 1.1 more model calls than the paired RAG episode, with a larger prompt each call, so the omitted policy chunks do not offset the longer history. The corresponding retail episodes keep about the same number of calls (10.7 vs. 10.6) but a smaller prompt, because the policy chunks are not injected. Episodes that start on RAG stay close to the RAG baseline. At an assumed 500 ms per remote retrieval, estimated remote time is 1.9 s vs. 4.7 s per airline episode and 2.6 s vs. 5.3 s per retail episode. That estimate scales with call count. It is not a measured remote deployment. HM-Gated RAG is preferable when retrieval is remote, billed per call, or operationally expensive. It is not a general method for cutting local tokens or latency.

Appendix B separates fallback from success on four seed-42 trajectories. Airline task 32 falls back and still fails, while RAG succeeds. Retail task 5 succeeds on the disrupted episode at zero retrieval; its routine episode fails without fallback. Retail task 12 succeeds after fallback, and RAG also succeeds. Retail task 62 succeeds on both sides under `rag_first`.

7 Discussion

The results support a limited operating range: SOP-stable tasks skip retrieval, and policy-heavy tasks stay on RAG. Habitual Memory equals RAG on retail at zero retrieval and is lower on airline. HM-Gated uses some retrieval to keep both domains near RAG. It is not required for success on every domain.

Table 4: Airline HM-Gated versus RAG, one run of 30 episodes per model. The four-trial `qwen-plus` means are in Table 1, not in this row.

Model	ΔSR vs. RAG	Retr. %RAG	Deploy?
qwen-turbo	+10.0pp	79.6%	Yes
qwen-plus	0.0pp	43.4%	Yes
qwen2.5-7B	−3.3pp	108.7%	No

The router ablation shows that the saving is the tag split, not the habit text alone. Without tags, retrieval calls are at least as high as always-retrieve RAG. Procedural Prompt is higher on airline (0.700 vs. 0.667) because it injects a full handwritten SOP. Fifteen airline SOPs are on the order of 30 expert hours to write, and a static SOP does not track policy drift or new variants. HM-Gated reaches 95.3% of that airline upper bound without placing the full SOP in the prompt.

When retrieval is operationally expensive, the four-trial `qwen-plus` result is the relevant one: about 59% fewer retrieval calls at matched mean success. The single-run cross-model comparison is separate. The main risk is continuing a habit after the situation has changed. Lifecycle updates were not run.

8 Conclusion

Habitual execution reframes memory as control: stable routines can bypass retrieval, and interruption returns the agent to deliberation. Over four trials, HM-Gated RAG matches RAG in mean success when task-type tags are available, and cuts retrieval calls by about 59% and 51%. The gain is retrieval efficiency, not a general reduction in latency or tokens.

9 Limitations

Habitual execution needs repeated trajectories and explicit guards. Our experiments use offline `eval_tags`, routines from τ^2 `reference_actions` via `GoldSyntheticProvider`, prompt-level guard checks, and a rule-based safety grader. We do not evaluate a learned router, online formation, or a guard ablation. The benchmark is synthetic (30 airline and 80 retail episodes, four trials) and has no production validation. The confidence interval on the success gap crosses zero. `qwen2.5-7B` regresses. Local tokens and latency rise on airline and do not on retail, so the gain depends on the cost of the retrieval infrastructure.

AI use statement

We used generative AI to translate and edit this manuscript and to help interpret logged results. We did not use it to create data, design or run experiments, implement the method, or derive mathematical claims. The authors checked the text and every reported number against the experiment logs and take responsibility for the final content.

Reproducibility statement

Implementation details, routing rules, habit formation settings, fallback conditions, model configurations, and additional experimental results are provided in the appendix. The main experiments report results over four random seeds (42–45), and the corresponding evaluation protocol and hyperparameters are described in Sections 5 and Appendix A.

References

Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. Self-RAG: Learning to retrieve, generate, and critique through self-reflection. *arXiv preprint arXiv:2310.11511*, 2023. URL https://arxiv.org/abs/2310.11511.

Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. Mem0: Building production-ready AI agents with scalable long-term memory. *arXiv preprint arXiv:2504.19413*, 2025. URL https://arxiv.org/abs/2504.19413.

Pengfei Du. Memory for autonomous LLM agents: Mechanisms, evaluation, and emerging frontiers. *arXiv preprint arXiv:2603.07670*, 2026. URL https://arxiv.org/abs/2603.07670.

Zexue He, Yu Wang, Churan Zhi, Yuanzhe Hu, Tzu-Ping Chen, Lang Yin, Ze Chen, Tong Arthur Wu, Siru Ouyang, Zihan Wang, Jiaxin Pei, Julian McAuley, Yejin Choi, and Alex Pentland. MemoryArena: Benchmarking agent memory in interdependent multi-session agentic tasks. *arXiv preprint arXiv:2602.16313*, 2026. URL https://arxiv.org/abs/2602.16313.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, and Joseph E. Gonzalez. MemGPT: Towards LLMs as operating systems. *arXiv preprint arXiv:2310.08560*, 2023. URL https://arxiv.org/abs/2310.08560.

Joon Sung Park, Joseph C. O'Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, 2023. URL https://arxiv.org/abs/2304.03442.

Preston Rasmussen, Pavlo Paliychuk, Travis Beauvais, Jack Ryan, and Daniel Chalef. Zep: A temporal knowledge graph architecture for agent memory. *arXiv preprint arXiv:2501.13956*, 2025. URL https://arxiv.org/abs/2501.13956.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, 2023. URL https://arxiv.org/abs/2303.11366.

Theodore R. Sumers, Shunyu Yao, Karthik Narasimhan, and Thomas L. Griffiths. Cognitive architectures for language agents. *Transactions on Machine Learning Research*, 2024. URL https://arxiv.org/abs/2309.02427.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023. URL https://arxiv.org/abs/2305.16291.

Wendy Wood and David T. Neal. A new look at habits and the habit-goal interface. *Psychological Review*, 114(4):843–863, 2007. doi: 10.1037/0033-295X.114.4.843.

Di Wu, Hongwei Wang, Wenhao Yu, Yuwei Zhang, Kai-Wei Chang, and Dong Yu. LongMemEval: Benchmarking chat assistants on long-term interactive memory. *arXiv preprint arXiv:2410.10813*, 2024. URL https://arxiv.org/abs/2410.10813. Published at ICLR 2025.

Shi-Qi Yan, Jia-Chen Gu, Yun Zhu, and Zhen-Hua Ling. Corrective retrieval augmented generation. *arXiv preprint arXiv:2401.15884*, 2024. URL https://arxiv.org/abs/2401.15884.

Zeyu Zhang, Xiaohe Bo, Chen Ma, Rui Li, Xu Chen, Quanyu Dai, Jieming Zhu, Zhenhua Dong, and Ji-Rong Wen. A survey on the memory mechanism of large language model based agents. *arXiv preprint arXiv:2404.13501*, 2024. URL https://arxiv.org/abs/2404.13501.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. ExpeL: LLM agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 19632–19642, 2024. doi: 10.1609/aaai.v38i17.29936.

A Implementation Details

This appendix aligns the paper with `experiments/hdb/`. `formation.py` implements five steps: collect successful trajectories, cluster them, extract a stable tool sequence, infer guards from disruptions, and estimate confidence from success and k. The reported runs do not execute clustering, disruption-time guard inference, or online version updates. Habits come from `GoldSyntheticProvider` at k=3.

The tag router reads `metadata.eval_tags`. `policy_heavy` selects `rag_first`. `sop_heavy` or `transfer_ok` selects `habit_only`. Missing tags select `rag_first`. A disrupted episode whose expected guard is absent from the habit guards also selects `rag_first`; that rule did not change the initial routes in the seed-42 retail run. `no_tag` selects `rag_first` whenever a habit exists. `keyword` selects `rag_first` on policy keywords such as refund, cancel, or exception, and `habit_only` otherwise.

Table 5: Default guards and runtime fallback signals.

Guard or signal	Role
`policy_unchanged`	Prompt self-check before tool use
`tool_outputs_match_expected`	Prompt self-check before tool use
`user_goal_unchanged`	Prompt self-check before tool use
Tool-error keywords	Runtime `rag_fallback`
Stalled progress ($\geq$ 2 assistant turns)	Not used for runtime fallback

Runtime fallback applies only to episodes that start at `habit_only` and have not yet fallen back. It switches to `rag_fallback` on an `<hm_router>` declaration or on a tool-message risk word. It does not use a stalled-progress signal. The RAG store is τ^2 policy text, embedded with DashScope, indexed in memory, filtered by `tau_task_id`, and retrieved at top-k=4. Primary numbers are means over seeds 42–45. Seed-42 diagnostics use `airline_hdb_seed42_test15/` and `retail_hdb_seed42_test40/`.

B Additional Results

Table 6: Airline cross-model results, the same single run as Table 4. Four-trial `qwen-plus` means are in Table 1.

Model	Method	SR	Ret.	%RAG	Latency
qwen-plus	RAG	0.667	251	100%	1,096
qwen-plus	HM-Gated	0.667	109	43.4%	1,417
qwen-turbo	RAG	0.133	1,266	100%	1,500
qwen-turbo	HM-Gated	**0.233**	**1,008**	79.6%	**1,298**
qwen2.5-7B	RAG	**0.267**	564	100%	**2,624**
qwen2.5-7B	HM-Gated	0.233	613	108.7%	3,443

Case A (airline task 32). Both `32:routine` and `32:disrupted` route `habit_only` $\rightarrow$ `rag_fallback` and fail. Retrieval after fallback is 1 and 8. Paired RAG succeeds, with 14 and 13 retrievals. Fallback fires and the task still fails.

Case B (retail task 62, disrupted). The episode routes `rag_first`. HM-Gated succeeds with 10 retrievals and RAG succeeds with 13. This episode is not a win for HM-Gated over RAG.

Case C (retail task 5). The disrupted episode stays on `habit_only` and succeeds with zero retrievals. The routine episode also stays on `habit_only` and fails with zero retrievals, so fallback does not fire.

Case D (retail task 12, disrupted). The episode routes `habit_only` $\rightarrow$ `rag_fallback`, retrieves 4 times, and succeeds, with no wrong-habit activation. RAG also succeeds, with 9 retrievals.

Table 7 is one manual audit of failure episodes. It is not part of the four-trial main comparison.

Table 7: Manual failure audit (10 airline failures and 18 sampled retail failures). Not part of Table 1.

Category	Airline	Retail	Total
Missed disruption	4	9	13
Auth / user state	2	3	5
Tool precondition	2	4	6
Missing policy detail	1	1	2
Over-conservative fallback	1	0	1
Simulator ambiguity	0	1	1
Total	**10**	**18**	**28**

Table 8: Disrupted-only success on the seed-42 run (n=15 airline, n=40 retail).

Subset	Domain	RAG	HM-Gated	Procedural
Disrupted	Airline	0.733	0.667	0.733
Disrupted	Retail	0.575	0.525	0.575

C Habit Artifact and Overlay

The overlay below is a representative airline habit rendered into the system prompt. It adds on the order of 200–400 tokens per episode, less than a full retrieved policy chunk (about 1,000–2,000 tokens per document). Confidence for the same four-step booking habit is 0.63, 0.79, 0.95, and 0.95 at $k \in \{1, 3, 5, 10\}$, saturating at 0.95. The reported experiments use k=3.

```
<habitual_memory>
Trigger: airline booking with extra passenger.
Routine: get_user_details -> get_reservation_details
  -> search_direct_flight -> book_reservation
Confidence: 0.79
Guards: policy_unchanged; tool_outputs_match_expected;
  user_goal_unchanged
Fallback: fall_back_and_replan
</habitual_memory>
```

Table 9: Routine versus disrupted success on the seed-42 run.

Condition	Domain	RAG	HM-Gated
Routine	Airline	0.600	0.667
Disrupted	Airline	0.733	0.667
Routine	Retail	0.500	0.425
Disrupted	Retail	0.575	0.525

Table 10: Safety and routing diagnostics on the seed-42 run.

Metric	Airline	Retail
Wrong activation (disrupted)	0/15	0/40
Disruption recovery	10/15 (67%)	21/40 (53%)
Fallback (habit→rag)	2/16 (12.5%)	14/52 (26.9%)
Mean retrieval (disrupted)	3.8	5.2

Table 11: HM-Gated by final route on the seed-42 run. Paired RAG success is 0.714 on airline episodes that stay in `habit_only` and 0.571 on airline `rag_first`; 0.596 on retail episodes that start in `habit_only` and 0.429 on retail `rag_first`.

Initial	Final	Air n	Air SR	Ret. n	Ret. SR
`rag_first`	`rag_first`	14	0.714	28	0.500
`habit_only`	`habit_only`	14	0.714	38	0.447
`habit_only`	`rag_fallback`	2	0.000	14	0.500

Table 12: Retrieval-side versus end-to-end cost. Remote times multiply mean retrieval calls by 500 ms. They are not a measured remote deployment.

Setting	Airline	Retail	Note
Retrieval reduction	59.5%	50.8%	Four-trial mean
End-to-end tokens	+16.2%	−5.1%	Four-trial mean
End-to-end latency	+12.4%	+0.1%	Four-trial mean
Remote p95=500 ms	1.9 s / 4.7 s	2.6 s / 5.3 s	HM-Gated / RAG, estimated