

RCR-Router: Role-Conditioned Context Routing for Resource-Efficient Multi-Agent LLM Systems

Jun Liu¹, Pu Zhao², Zhenglun Kong³, Wenbin Zhang⁴, Peiyan Dong⁵, Fan Yang⁶,
Changdi Yang², Wuzhe Fan⁷, Tianqin Li¹, Hao Tang^{1,8}, Geng Yuan⁹, Wei Niu⁹,
Joannah Nanjeyye¹, Xue Lin², Dong Huang¹, Yanzhi Wang^{2,10}

¹Carnegie Mellon University, ²Northeastern University, ³Harvard University,

⁴Florida International University ⁵MIT, ⁶Fujitsu, ⁷McGill University,

⁸National University of Singapore, ⁹University of Georgia ¹⁰Embodix

Correspondence: p.zhao@northeastern.edu, wenbin.zhang@fiu.edu

Abstract

Large language model (LLM) agents are increasingly organized into multi-agent systems for complex reasoning and decision-making tasks. However, scaling such systems can substantially increase inference cost, especially when multiple agents repeatedly process overlapping or role-irrelevant context. Many existing multi-agent frameworks focus on orchestration and role specialization, but still rely on full-context sharing or static context routing, leading to redundant token consumption, increased runtime, and inefficient memory utilization. We present *Role-Conditioned Context Routing (RCR)*, an execution-time layer that allocates role-relevant shared-memory items to agents under token budgets. Given a structured shared memory and a set of role-specialized agents, RCR selectively routes task-relevant memory items to each agent according to its role, optional task-stage metadata, and token budget. RCR integrates role-conditioned relevance scoring, token-budgeted memory selection, and iterative memory updates to reduce redundant context exposure during multi-agent execution. Experiments on multi-hop QA, embodied reasoning (ALFWorld), and web-based decision-making (WebShop) show that RCR reduces token usage by 35% on average and runtime by 38% on average relative to Full-Context, while maintaining or improving task performance.

1 Introduction

As LLM-based agents (Touvron et al., 2023; Liu et al., 2026b; Chiang et al., 2023) become more capable and widely deployed, the execution cost of multi-agent systems becomes increasingly significant. Recent multi-agent LLM frameworks such as CrewAI (crewai, 2025), Microsoft AutoGen (Wu et al., 2023), LangChain/LangGraph (Chase, 2024), OpenAI Swarm (OpenAI, 2024a), and Claude-style agent systems (Anthropic, 2025)

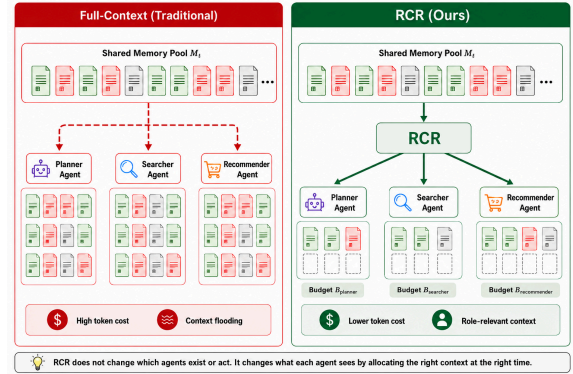


Figure 1: Motivation of RCR. Traditional Agent (Full-Context) exposes all agents to the entire shared memory, leading to redundant token usage and context flooding. RCR keeps the agent topology unchanged and allocates compact, role-relevant memory subsets under token budgets, changing what each agent sees rather than which agents are used.

adopt a role-based paradigm, where agents specialize in functions such as planning, searching, execution, or verification. While this paradigm enables modular coordination, it also makes context allocation critical: different roles require different information at different stages. However, existing frameworks primarily specify how agents are orchestrated, while often relying on full-context sharing, fixed templates, or static routing. As a result, multiple agents repeatedly process overlapping or role-irrelevant context, causing redundant token consumption, increased runtime, and inefficient memory utilization.

Despite such role specialization, existing systems often rely on *uniform context exposure*: agents receive either fixed templates (Chase, 2024), static routing, or the full interaction history. As a result, multiple agents may repeatedly process overlapping or role-irrelevant context during execution, leading to unnecessary token usage, higher runtime, and possible distraction from task-relevant information. Retrieval-based methods such as *Self-*

RAG (Asai et al., 2024) and *End-to-End Beam Retrieval* (Zhang et al., 2023) improve evidence selection, but they typically retrieve a single shared context rather than allocating different context subsets to different agents. Similarly, *Multi-Agent Debate* (Du et al., 2023) improves coordination through critique, but does not explicitly solve token-budgeted, role-conditioned context allocation over shared memory.

This mismatch becomes critical as task complexity grows. Different agents require different information at different stages: for example, in multi-hop QA, a *Planner* focuses on reasoning state, a *Searcher* on evidence retrieval, and a *Synthesizer* on answer aggregation (Chase, 2024). Providing identical context to all agents wastes tokens and can introduce role-irrelevant information (Liu et al., 2025b), which may degrade reasoning quality. This motivates our central premise: multi-agent LLM systems require execution-time context allocation that is both role-conditioned and resource-aware. Figure 1 illustrates this contrast.

To address this problem, we present *Role-Conditioned Context Routing (RCR)*, an execution-time layer that allocates role-relevant shared-memory items to agents under token budgets. Rather than introducing a new agent orchestration architecture, RCR focuses on context exposure: given an existing multi-agent structure and shared memory, it selects which information each agent should receive under token-budget constraints. As shown in Figure 1, Full-Context broadcasts the same memory pool to every agent, whereas RCR allocates compact, role-relevant memory subsets within a fixed role-based topology.

Given a structured shared memory, RCR dynamically routes relevant subsets to each agent based on its role, optional task-stage metadata, and token budget. Our method consists of three components: (1) a reward-optimized token budget allocator that selects per-agent context capacity, (2) an importance scorer that computes role- and stage-conditioned relevance, and (3) a size-aware routing mechanism that selects memory items under budget constraints. We further use an *Iterative Routing Feedback (IRF)* process, where agent outputs are incorporated into shared memory to support iterative context refinement across interaction rounds.

We report main results on HotPotQA, ALF-World, and WebShop, with additional multi-hop QA results on MuSiQue and 2WikiMultihop provided in Table 16 of Appendix G. Overall, RCR

provides a favorable efficiency–performance trade-off, reducing token usage by 35.1% and runtime by 38.0% on average relative to Full-Context, while maintaining or improving task performance.

Our contributions are as follows:

- We identify and formalize multi-agent role-conditioned context routing as a distinct execution-time resource-allocation problem: given a shared memory and fixed role-specialized agents, deciding which memory items should be exposed to which agent under token-budget constraints.
- We instantiate this formulation as RCR, a modular routing layer that integrates role- and stage-conditioned relevance scoring, adaptive budget allocation, and size-aware budgeted selection without changing the underlying agent topology.
- Under controlled conditions with matched agents, prompts, memory pools, tools, call limits, and stopping rules, we show across multi-hop QA, embodied reasoning, and web navigation benchmarks that role-conditioned context allocation improves the efficiency–performance trade-off over full-context, static-routing, retrieval-based, and debate-based baselines.

2 Related Work

Multi-Agent LLM Systems. Recent multi-agent LLM frameworks (e.g., AutoGen (Wu et al., 2023), LangGraph (Chase, 2024), CrewAI (crewai, 2025), Swarm (OpenAI, 2024a)) enable flexible role-based agent orchestration but largely rely on uniform or template-based context exposure. *LLM-MAS* (Han et al., 2024) surveys key challenges including task allocation and memory management, but does not propose a token-budgeted routing mechanism. *MasRouter* (Yue et al., 2025) addresses system-level routing and model selection, while *AgentNet* (Yang et al., 2025) focuses on decentralized coordination and evolving agent topology. *SupervisorAgent* (Lin et al., 2026) reduces token consumption through runtime supervision. These approaches are orthogonal to RCR’s pre-call context allocation and could be combined.

Memory Management for LLM Agents. *A-Mem* (Xu et al., 2025) organizes memories through indexing and linking, while *G-Memory* (Zhang et al., 2025), *H-MEM* (Sun et al., 2026) structures multi-agent histories with hierarchical memory and trajectory tracing. *HiAgent* (Hu et al.,

2024) manages working memory through subgoal-based chunks, and *Self-Notes* (Lanchantin et al., 2023) enhance reasoning with intermediate notes. *IMA* (Yuen et al., 2025) maintains role-aligned agent-specific memories, while *Collaborative Memory* (Rezazadeh et al., 2025), *MIRIX* (Wang and Chen, 2025) focuses on permission-aware memory sharing. These methods (Liu et al., 2026c; Shen et al., 2026; Fu et al., 2025; Li et al., 2026) primarily address how memories are stored (Yu et al., 2025; Fu et al., 2025; Li et al., 2026), organized (Liu et al., 2026a; Cheng et al., 2026b), or accessed (Du et al., 2026; Cheng et al., 2026a). In contrast, RCR targets execution-time context allocation: given a shared memory (Liu et al., 2025a), it selectively routes role-relevant subsets to different agents under token-budget constraints. This dimension is orthogonal to memory organization.

3 The Proposed Method

3.1 Problem Formulation

We consider a multi-agent LLM system composed of N collaborative agents $\mathcal{A} = \{A_1, A_2, \dots, A_N\}$ interacting over a shared task. Each agent A_i operates with a specific *role* R_i (e.g., Planner, Searcher, Recommender) and interacts with other agents and external tools in discrete *interaction rounds* $t = 1, 2, \dots, T$.

At each round t , agents exchange messages and perform reasoning based on a Shared Memory Store M_t , which contains: (I) *agent interaction history*, including prior communications and intermediate reasoning traces; (II) *task-relevant knowledge*, including retrieved documents, external facts, or tool outputs; and (III) *structured state representations*, including entities, subplans, and tool traces encoded in structured formats.

Each agent A_i receives a *routed context* $C_t^i \subseteq M_t$, selected by the **learned routing policy** according to the agent’s role R_i , current task stage S_t , and the token budget B_i predicted by the Token Budget Allocator. This design incorporates a mechanism for context selection and allocation.

Each agent A_i then performs an *LLM Query* based on its routed context C_t^i :

$$LLM_output_t^i = LLM(\text{Prompt}(C_t^i)), \quad (1)$$

The global objective of the system is to maximize collaborative task success while minimizing cumulative token consumption across all agents and rounds.

We formulate context routing as a budgeted relevance maximization problem. For agent A_i at round t , RCR selects a subset of shared-memory items $C_t^i \subseteq M_t$:

$$C_t^i = \arg \max_{\substack{C' \subseteq M_t \\ \sum_{m \in C'} \ell(m) \leq B_i}} \sum_{m \in C'} \alpha(m; R_i, S_t). \quad (2)$$

where $\alpha(m; R_i, S_t)$ is the role- and stage-conditioned relevance score, $\ell(m)$ is the token length of memory item m , and B_i is the allocated token budget.

Eq. (2) defines the execution-time routing interface: given role-conditioned utilities and a token budget, RCR selects a compact memory subset for each agent. The main modeling challenge is estimating $\alpha(m; R_i, S_t)$ so that selected items match the agent’s role and current stage. At inference time, we use a lightweight size-aware online selector: memory items are ordered by normalized relevance $\alpha(m; R_i, S_t)/\ell(m)$, a feasible set is constructed under B_i , and the result is compared with the best feasible singleton. The candidate with higher additive relevance is returned:

$$C_t^i = \arg \max_{C \in \{C_{\text{set}}, C_{\text{single}}\}} \sum_{m \in C} \alpha(m; R_i, S_t).$$

The normalized score is used only for size-aware ordering; the utility remains the additive objective in Eq. (2).

3.2 Role-Conditioned Context Routing with Semantic Abstraction

Context Routing Framework. We introduce **RCR**, a modular context-routing layer that learns *what information should be routed to which role* under token-budget constraints. Rather than defining a new communication protocol, RCR coordinates agents through role- and stage-conditioned routing over a shared memory store. Figure 2 illustrates the architecture. Shared memory M stores agent history, external knowledge, and task-relevant entities in structured records with role tags, stage tags, timestamps, and content fields, allowing indexing and semantic filtering.

RCR Core. Multi-agent LLM systems require role-specific access to shared memory under strict context constraints. The RCR framework introduces an importance-based structured routing policy that assigns token-efficient, role-relevant memory context to each agent at every interaction round. It consists of three modular components: (1) Token

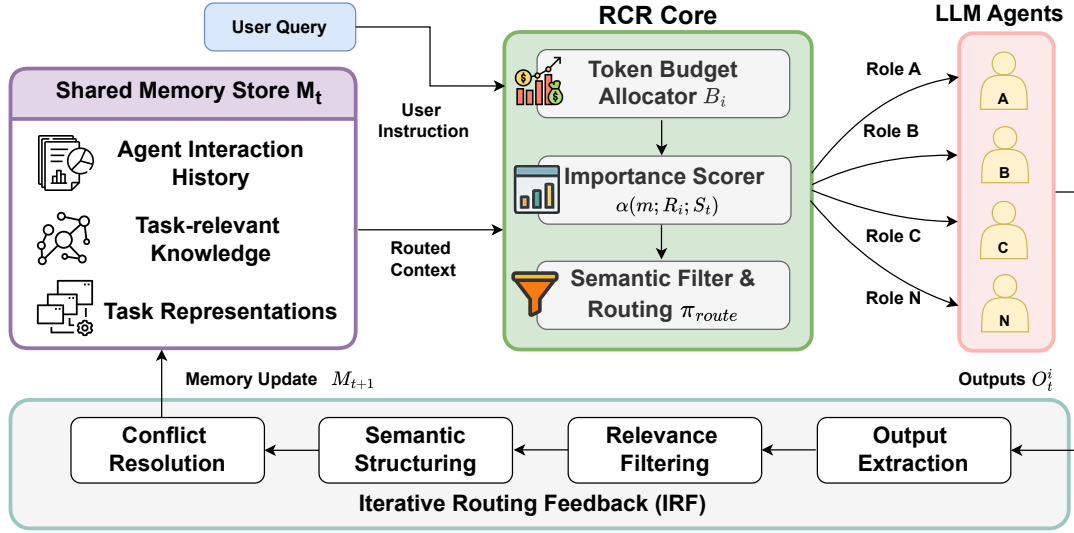


Figure 2: **Iterative RCR architecture with adaptive feedback loop.** At each interaction round t , RCR dynamically routes semantically filtered memory to each agent based on its role and current task stage. Agents process the routed contexts and produce structured outputs, which are integrated into the updated Shared Memory M_{t+1} through the Memory Update step. This iterative process forms a closed-loop cycle ($M_t \rightarrow \text{Role-Aware Routing} \rightarrow \text{Agents} \rightarrow \text{IRF} \rightarrow M_{t+1}$), enabling progressive refinement of contextual relevance and efficient multi-agent coordination.

Budget Allocator, (2) Importance Scorer, and (3) Semantic Filter with Routing Logic.

3.2.1 Reward-Optimized Token Budget Allocator

We formulate token budget allocation as a resource-aware policy optimization problem. At each interaction round t , the allocator observes the role embedding r_i , stage embedding s_t , and a usage vector u_i summarizing recent token consumption, context utilization, latency, and task feedback. It then samples a token budget for agent A_i :

$$B_i \sim \pi_\theta(\cdot \mid r_i, s_t, u_i), \quad (3)$$

where π_θ is a lightweight policy network over a discrete budget set $\mathcal{B} = \{512, 1024, 2048, 4096\}$.

After completing a multi-agent trajectory τ , the allocator receives an episode-level reward:

$$R(\tau) = Q(\tau) - \lambda_{\text{Tok}} \text{Tok}(\tau) - \lambda_{\text{Lat}} \text{Lat}(\tau), \quad (4)$$

where $Q(\tau)$ denotes the benchmark-specific primary metric (F1 for HotPotQA, success rate for ALFWorld, and reward score for WebShop), $\text{Tok}(\tau)$ denotes total token usage, and $\text{Lat}(\tau)$ denotes end-to-end latency. To make reward terms comparable across benchmarks, token and latency costs are normalized by development-set statistics before applying λ_{Tok} and λ_{Lat} , which are tuned only on the development split. Sensitivity to λ values is shown in Appendix B.6.

We optimize π_θ using a policy-gradient objective with a moving-average baseline for variance reduction and entropy regularization for exploration. During inference, we select the budget with the highest policy probability. Given B_i , RCR orders memory items by normalized relevance $\alpha(m; R_i, S_t)/\ell(m)$ and constructs a budget-feasible routed context using the size-aware online selector. Details are provided in Appendix C.

3.2.2 Importance Scorer

The importance scorer is trained with AdamW (5×10^{-4} learning rate, 10^{-4} weight decay, batch size 256, cosine decay with 10% warm-up). Because the LLM is black-box and routing is discrete, we use weak supervision from logged trajectories. Positive labels come from memory items linked to successful evidence chains, tool outputs, or action traces. Negatives are contradicted or same-role-stage hard-negative items; related unused items are down-weighted soft negatives, while ambiguous unused items are masked from the loss. The scorer is optimized with a weighted binary relevance loss conditioned on the role and task stage. Appendix D gives the full loss, examples, and label statistics. **Training and leakage control.** Both routing modules are trained only from training-split trajectories collected under the same roles, prompts, tools, memory pool, and stopping rules used in evaluation. The budget allocator observes only online

execution signals and is optimized with a PPO-style clipped policy-gradient objective; it does not use gold answers, future outcomes, or test metrics. The importance scorer uses conservative weak labels: positives are linked evidence/tool/action records, negatives are contradicted or same-role–stage hard negatives, and ambiguous unused items are discarded. Appendix C and Appendix D provide the full objectives, logging assumptions, label statistics, and discard rates.

3.2.3 Semantic Filter and Routing

The routing policy in Eq. (2) uses a budgeted size-aware online selector: it filters feasible items $M' = \{m \in M_t \mid \ell(m) \leq B_i\}$, ranks them by $\alpha(m; R_i, S_t)/\ell(m)$, forms C_{set} under B_i , and returns the higher-utility candidate between C_{set} and $C_{\text{single}} = \{\arg \max_{m \in M'} \alpha(m; R_i, S_t)\}$.

The resulting C_t^i is passed as the input context for the next LLM query of agent A_i . The relevance score is conditioned on the agent role and task stage, while the budget B_i is determined by the reward-optimized allocator. Algorithm 1 in the Appendix summarizes the routing workflow. All experiments use this deterministic size-aware online selector for controlled comparison.

3.3 Iterative Routing with Feedback

To support tasks that evolve over multiple interaction rounds, RCR incorporates **Iterative Routing Feedback (IRF)**. At round t , each agent A_i receives a budgeted context C_t^i based on its role R_i , task stage S_t , and budget B_i , then produces outputs such as messages, tool calls, retrieved evidence, or intermediate reasoning steps.

IRF converts these outputs into structured memory items, including facts, subplans, tool results, and evidence. It then updates the shared memory from M_t to M_{t+1} for the next routing step, as shown in Figure 2. This allows later rounds to use updated context without repeatedly exposing agents to stale or redundant memory. We treat IRF as an empirical refinement mechanism, not a monotonic convergence guarantee.

Memory Update. After each interaction round, RCR updates the shared memory through: (I) **Output Extraction**, which parses each O_t^i into (role, stage, key_fact, evidence, timestamp) and retains atomic records corresponding to task-relevant facts, evidence statements, tool results, and reasoning summaries; (II) **Relevance Filtering**, which scores newly extracted candidate items

by $\alpha(m; R_i, S_t)$, using the producing agent’s role and current task stage, and retains high-confidence records for insertion into M_{t+1} ; and (III) **Semantic Structuring**, which stores retained items as $m = \{\text{role_tag}, \text{stage_tag}, \text{timestamp}, \text{content}\}$. Deduplication is applied uniformly before routing and removes only near-duplicate records within the same role–stage group; it is not tuned per method.

Memory control. All controlled methods use the same atomic memory items, serialization, deduplication, and conflict-resolution rules; only the routing policy differs. Thus, token savings are not attributable to method-specific memory compression. Appendix E provides the schema, prompts, and examples used by a structured-output Memory Update Agent to convert raw agent outputs into structured memory records. Following standard interfaces such as the OpenAI Agents SDK (OpenAI, 2025b) and Google ADK (Google, 2025), RCR does not rely on task-specific hand-written parsers; the routing layer only consumes structured memory records and is agnostic to the underlying agent framework.

RCR therefore combines two processes: (1) *role-conditioned context routing*, where each agent receives a budgeted context based on role, stage, and resource constraints; and (2) *memory update*, where structured agent outputs form the memory used for subsequent routing rounds.

4 Experiments

4.1 Implementation Details

Model and Hardware Platform. All experiments were executed on a workstation with an Intel Xeon Silver CPU (32 cores), 128GB RAM, and one NVIDIA A6000 GPU. We use OpenAI’s text-embedding-3-small for memory-item embeddings and retrieval features (OpenAI, 2024b). For agent LLM backbones, we evaluate GPT-5-mini (OpenAI, 2025a), Qwen3-Max (Qwen, 2025), and Gemini 2.5 Pro (Google Cloud, 2025).

Benchmarks and Metrics. We report main results on HotPotQA (Yang et al., 2018), ALF-World (Shridhar et al., 2021), and WebShop (Yao et al., 2022), covering multi-hop reasoning, embodied decision-making, and web navigation. Additional multi-hop QA results on MuSiQue (Trivedi et al., 2022) and 2WikiMultiHop (Ho et al., 2020) are provided in Appendix G. All reported results are averaged over 5 independent runs.

Baselines. We compare RCR with **Full-Context**

Table 1: Main results on HotPotQA, ALFWorld, and WebShop. Each benchmark is evaluated with its *official* primary metric: F1 score for HotPotQA, task success rate (%) for ALFWorld, and reward score (0–1) for WebShop. HotPotQA uses three roles, while ALFWorld and WebShop use four-role configurations as detailed in Appendix F. Runtime and token usage are mean values over five independent runs; primary metrics are mean $\pm$ std. Full Precision/Recall breakdowns for HotPotQA are provided in Appendix G.2.

Benchmark	Method	Avg Runtime (s)↓	Token (K)↓	Primary Metric↑
HotPotQA (F1, 0–100)	Full-Context	150.65	5.10	74.6 $\pm$ 0.6
	Static Routing	128.29	3.85	76.1 $\pm$ 0.6
	Self-RAG	118.44	3.92	78.4 $\pm$ 0.5
	End-to-End Retrieval	110.12	3.80	79.0 $\pm$ 0.5
	Multi-Agent Debate	105.33	4.06	80.0 $\pm$ 0.4
	RCR (Ours)	91.58	3.63	83.6 $\pm$0.5
ALFWorld (Success Rate, %)	Full-Context	145.3	6.82	59.1 $\pm$ 0.8
	Static Routing	122.8	5.12	62.2 $\pm$ 0.7
	Self-RAG	117.6	5.40	63.7 $\pm$ 0.7
	End-to-End Retrieval	114.2	5.18	65.5 $\pm$ 0.6
	Multi-Agent Debate	136.7	5.95	60.2 $\pm$ 0.9
	RCR (Ours)	94.5	4.30	67.3 $\pm$0.8
WebShop (Reward, 0–1)	Full-Context	180.2	9.78	0.565 $\pm$ 0.009
	Static Routing	142.6	7.12	0.616 $\pm$ 0.008
	Self-RAG	134.3	6.85	0.642 $\pm$ 0.007
	End-to-End Retrieval	120.7	6.47	0.639 $\pm$ 0.006
	Multi-Agent Debate	128.9	6.58	0.627 $\pm$ 0.007
	RCR (Ours)	108.7	5.91	0.653 $\pm$0.005

Routing (crewai, 2025), **Static Routing** (Chase, 2024), **Self-RAG** (Asai et al., 2024), **End-to-End Beam Retrieval** (Zhang et al., 2023), and **Multi-Agent Debate** (Du et al., 2023). Full-Context and Multi-Agent Debate share M_t without role-specific routing; Static Routing uses fixed role templates; Self-RAG and End-to-End Beam Retrieval are retrieval-based baselines using the same memory pool and comparable prompt budgets.

Training. The two routing modules are trained on the training split and tuned on the validation split, while all reported results are computed on the held-out test split. The allocator is optimized at the trajectory level for the efficiency–accuracy trade-off, whereas the scorer is trained at the item level for role-conditioned memory relevance. Full training objectives, logging assumptions, and weak-label statistics are provided in Appendix C and Appendix D. For more details, see Appendix B.

4.2 Results

Table 1 reports the main results under the default single-pass setting with fixed role configurations. Each benchmark is evaluated with its official primary metric: F1 for HotPotQA, success rate for ALFWorld, and reward score for WebShop. RCR

achieves a favorable efficiency–performance trade-off, reducing token usage and runtime while preserving or improving the primary task metric.

Backbone-Transfer Robustness. RCR transfers to Qwen3-Max and Gemini 2.5 Pro without backbone-specific retraining. Across all three benchmarks, it consistently reduces token usage and runtime over Full-Context and Static Routing while preserving or improving the primary task metric (Appendix G.1).

4.3 Controlled Routing Comparison

To isolate the routing policy, we compare RCR with alternatives using the same agents, prompts, rounds, memory construction, tool interfaces, and stopping criteria; only the memory selection policy differs. Table 2 shows that RCR achieves the strongest cost–quality trade-off among lexical, semantic, role-only, stage-only, and fixed-budget routing variants under this controlled setup, suggesting that the observed differences are not merely due to agent topology, prompt protocol, or memory construction. Sensitivity to the token budget and routing iterations is reported in Appendix G.5.

Table 2: Controlled routing comparison under identical agents, prompts, interaction rounds, memory construction, and stopping criteria. Only the memory selection policy differs. We report mean $\pm$ standard deviation over five independent runs. RCR improves the cost–quality trade-off over controlled routing variants; gains over Fixed Budget are strongest on HotPotQA and WebShop, and moderate on ALFWorld.

Method	HotPotQA			ALFWorld			WebShop		
	Tok.↓	Run.↓	F1↑	Tok.↓	Run.↓	Succ.↑	Tok.↓	Run.↓	Reward↑
BM25 Retrieval	3.79 $\pm$ 0.05	103.4 $\pm$ 1.8	78.2 $\pm$ 0.5	4.44 $\pm$ 0.06	108.2 $\pm$ 2.1	63.1 $\pm$ 0.7	6.12 $\pm$ 0.07	123.6 $\pm$ 2.4	0.617 $\pm$ 0.007
Semantic Retrieval	3.76 $\pm$ 0.04	99.7 $\pm$ 1.6	79.2 $\pm$ 0.5	4.40 $\pm$ 0.05	104.7 $\pm$ 1.9	64.0 $\pm$ 0.6	6.05 $\pm$ 0.06	119.4 $\pm$ 2.1	0.624 $\pm$ 0.006
Role-only Routing	3.71 $\pm$ 0.04	96.8 $\pm$ 1.5	80.5 $\pm$ 0.4	4.36 $\pm$ 0.05	100.3 $\pm$ 1.7	65.9 $\pm$ 0.6	5.99 $\pm$ 0.06	115.6 $\pm$ 1.9	0.640 $\pm$ 0.006
Stage-only Routing	3.74 $\pm$ 0.04	98.5 $\pm$ 1.6	79.7 $\pm$ 0.5	4.38 $\pm$ 0.05	101.4 $\pm$ 1.8	65.2 $\pm$ 0.7	6.01 $\pm$ 0.06	117.1 $\pm$ 2.0	0.633 $\pm$ 0.006
Fixed Budget	3.69 $\pm$ 0.03	95.4 $\pm$ 1.4	81.2 $\pm$ 0.5	4.34 $\pm$ 0.04	98.9 $\pm$ 1.6	66.3 $\pm$ 0.6	6.16 $\pm$ 0.07	112.8 $\pm$ 1.8	0.629 $\pm$ 0.006
RCR	3.63$\pm$0.03	91.6$\pm$1.3	83.6$\pm$0.4	4.30$\pm$0.04	94.5$\pm$1.5	67.3$\pm$0.5	5.91$\pm$0.05	108.7$\pm$1.7	0.653$\pm$0.005

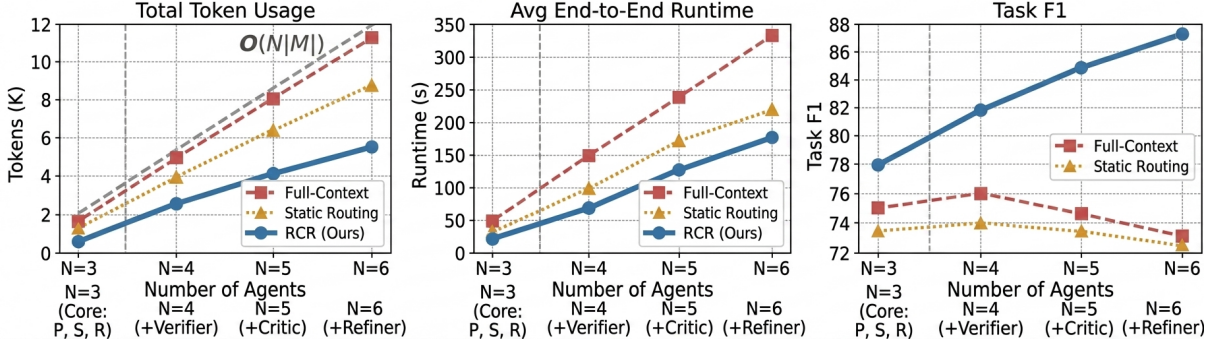


Figure 3: Agent-count sensitivity on HotPotQA as agent count increases ($N \in \{3, 4, 5, 6\}$). All points are measured under the same evaluation split, prompts, stopping rules, and routing implementation as the main experiments; the dashed vertical line marks the default $N=3$ configuration from Table 1. **(Left)** Token usage increases with N ; the gray dashed $O(N|M|)$ line is a reference trend, and RCR grows more slowly than Full-Context. **(Center)** Runtime follows a similar pattern. **(Right)** RCR maintains stable or improved F1 as auxiliary roles are added, while Full-Context shows no consistent gain under uniform context exposure.

Table 3: Effect of Iterative Routing Feedback (IRF) across **HotPotQA**, **ALFWorld**, and **WebShop**. Each benchmark is evaluated using its official primary metric. Increasing routing iterations T improves task performance but increases runtime and token usage.

Benchmark	T	Avg Rt (s)↓	Tok (K)↓	Metric↑
HotPotQA (F1, 0–100)	1	91.6	3.73	83.6
	2	115.8	5.92	85.0
	3	141.7	7.68	85.8
ALFWorld (Success Rate, %)	1	94.5	4.30	67.3
	2	118.5	6.28	70.9
	3	143.9	7.85	71.7
WebShop (Reward, 0–1)	1	108.7	5.91	0.653
	2	132.7	7.54	0.680
	3	157.6	9.11	0.687

4.4 Impact of Iterative Feedback Routing (IRF)

Table 3 studies routing iterations $T \in \{1, 2, 3\}$. More iterations improve the primary metric but increase token usage and runtime. While $T=3$ gives the best performance, its gain over $T=2$ is modest relative to the added cost. Thus, $T=2$ offers a practical efficiency–performance trade-off,

Table 4: Controlled comparison of iterative routing feedback and additional computation on HotPotQA. All $T=2$ variants use comparable model-call budgets.

Variant	Rt (s)↓	Tok (K)↓	F1↑
Single-pass RCR ($T=1$)	91.6	3.73	83.6
Static memory rerouting ($T=2$)	109.4	5.48	84.1
Extra rounds w/o update ($T=2$)	112.7	5.61	83.5
Full-Context same rounds ($T=2$)	196.4	8.74	81.2
IRF w/ memory update ($T=2$)	115.8	5.92	85.0

whereas $T=1$ remains the default low-overhead setting. To disentangle feedback from merely spending more compute, Table 4 compares IRF with static-memory rerouting and extra rounds without structured memory updates. IRF yields the strongest F1 among these variants under comparable additional-call settings, suggesting that structured memory updates contribute beyond additional computation alone.

4.5 Ablation Studies

We ablate the Token Budget Allocator (TBA), Importance Scorer (IS), and Semantic Filter & Routing (SFR) under the same single-pass setting ($T=1$)

Table 5: Module-wise ablations under the default single-pass setting ($T=1$). TBA = Token Budget Allocator, IS = Importance Scorer, and SFR = Semantic Filter and Routing. Each benchmark is evaluated using its official primary metric. Removing any module degrades the efficiency–performance trade-off.

Benchmark	Configuration	Avg Runtime (s)↓	Token (K)↓	Primary Metric↑
HotPotQA (F1)	Full (TBA + IS + SFR)	91.58	3.63	83.6
	w/o TBA	101.8	3.92	79.4
	w/o IS (Recency only)	96.9	3.84	78.0
	w/o SFR (FC-in-B)	108.7	3.77	76.5
ALFWorld (Success %)	Full (TBA + IS + SFR)	94.5	4.30	67.3
	w/o TBA	107.3	4.56	64.4
	w/o IS (Recency only)	102.1	4.47	63.3
	w/o SFR (FC-in-B)	111.5	4.39	62.3
WebShop (Reward 0–1)	Full (TBA + IS + SFR)	108.7	5.91	0.653
	w/o TBA	122.4	6.28	0.627
	w/o IS (Recency only)	118.2	6.14	0.619
	w/o SFR (FC-in-B)	126.9	6.03	0.612

Table 6: Comparison with generic context optimization strategies under identical settings. Runtime/token usage are mean $\pm$ std over five runs; * denotes significant improvement over the strongest baseline ($p < 0.05$).

Benchmark	Strategy	Avg Runtime (s)↓	Token (K)↓	Primary Metric↑
HotPotQA (F1)	MMR Retrieval	94.8 $\pm$ 1.2	3.72 $\pm$ 0.06	81.8 $\pm$ 0.6
	Learned Reranker	93.6 $\pm$ 1.1	3.70 $\pm$ 0.05	82.5 $\pm$ 0.5
	Prompt Compression	89.7 $\pm$ 1.0	3.10 $\pm$ 0.04	81.5 $\pm$ 0.7
	RCR	91.6$\pm$1.3	3.63$\pm$0.05	83.6$\pm$0.5*
ALFWorld (Success %)	MMR Retrieval	101.2 $\pm$ 1.8	4.40 $\pm$ 0.08	66.7 $\pm$ 0.8
	Learned Reranker	99.8 $\pm$ 1.6	4.37 $\pm$ 0.07	67.0 $\pm$ 0.7
	Prompt Compression	95.1 $\pm$ 1.4	3.75 $\pm$ 0.06	65.8 $\pm$ 0.9
	RCR	94.5$\pm$1.5	4.30$\pm$0.07	67.3$\pm$0.8*
WebShop (Reward 0–1)	MMR Retrieval	110.9 $\pm$ 2.0	6.05 $\pm$ 0.10	0.638 $\pm$ 0.008
	Learned Reranker	109.5 $\pm$ 1.9	6.02 $\pm$ 0.09	0.636 $\pm$ 0.007
	Prompt Compression	106.2 $\pm$ 1.7	5.35 $\pm$ 0.08	0.612 $\pm$ 0.009
	RCR	108.7$\pm$1.7	5.91$\pm$0.09	0.653$\pm$0.005*

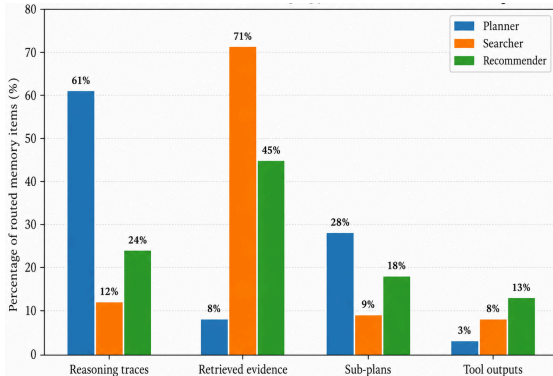


Figure 4: Distribution of selected memory types across roles on HotPotQA. This pattern is consistent with role-conditioned routing: the Planner receives predominantly reasoning traces and sub-plans, while the Searcher receives mostly retrieved evidence, suggesting that RCR allocates functionally distinct memory subsets rather than merely shortening prompts uniformly.

and role configuration. **w/o TBA** uses a fixed uniform budget, **w/o IS** uses recency-only scoring, and

Table 7: Failure-recovery analysis on HotPotQA.

Setting	Traj. Len	Mem Gro	Tok (K)↓	Rt (s)↓
Normal	1x	1x	3.63	91.6
+Retry-1	1.25x	1.25x	4.12	103.5
+Retry-2	1.5x	1.5x	4.67	116.8

Table 8: RCR’s routing overhead on HotPotQA. .

Component	Latency (s)	Percentage
Scorer	1.2	1.3%
TBA + Memory Selection	0.8	0.9%
Agent LLM Inference	89.6	97.8%
Total	91.6	100%

w/o SFR selects memory items by insertion order within the budget. Appendix G.3, Table 17 shows that learned TBA improves over both uniform and development-tuned static per-role budgets across all three benchmarks.

4.6 Mechanism analysis.

RCR’s gains are not merely efficiency artifacts. As quantified in Figure 4, RCR exhibits clear role-wise separation: the Planner mainly receives reasoning traces (61%) and sub-plans (28%), while the Searcher mainly receives retrieved evidence (71%). The Recommender receives a more balanced mixture of evidence and answer-related signals. This functional filtering reduces irrelevant context exposure and helps explain the quality gains in Table 1 while accounting for the reduced token cost. A concrete routing example is provided in Appendix G.4.

4.7 Agent-Count Sensitivity Analysis

To examine how routing cost changes as the number of agents increases, we conduct a controlled HotPotQA agent-count analysis. Starting from the default three-agent setup, we incrementally add a *Verifier* ($N=4$), *Critic* ($N=5$), and *Refiner* ($N=6$), while keeping the shared memory pool, prompts, routing implementation, and stopping criteria fixed. The $N=3$ baseline is anchored to Table 1.

Token and Runtime Scaling. Full-Context token usage increases from 5.10 K at $N=3$ to 10.18 K at $N=6$, reflecting repeated full-memory exposure as agents are added. RCR grows more slowly (3.63 K $\rightarrow$ 5.98 K), as the Token Budget Allocator assigns smaller budgets to narrowly scoped agents, consistent with the $\mathcal{O}(\sum_i B_i)$ routing-cost bound. Runtime shows the same trend: 301.8 s for Full-Context versus 137.6 s for RCR at $N=6$ (Figure 3, Appendix Table 19).

Task Performance. RCR maintains stable or improved F1 as auxiliary roles are added (83.6 $\rightarrow$ 85.7), while Full-Context peaks at $N=4$ and degrades, preserving RCR’s advantage at scale.

4.8 Failure Recovery Scenario

In Table 7, we constructed a failure-recovery variant on HotPotQA: after injecting a failure at a random intermediate step, we trigger 1 or 2 additional retry cycles (re-planning and re-execution) and measure the resulting cost. As shown below, token usage and runtime increase roughly proportionally with the number of retries, while RCR continues to route context under its existing budget and relevance mechanism without requiring architectural changes.

4.9 Computational Overhead.

In Table 8, we isolate the routing layer’s own overhead on HotPotQA: the Importance Scorer ac-

Table 9: Role granularity sensitivity on HotPotQA.

Level	Role Decom	Rol	Rt (s)↓	Tok ↓	F1↑
Coarse	Reasoner; Executor	2	93.1	3.75	82.4
Default	Planner; Searcher; Rec	3	91.6	3.63	83.6
Fine	Planner; Searcher; Checker; Composer; Verifier	5	96.2	4.02	83.8

counts for 1.2s (1.3%), and the Token Budget Allocator plus size-aware selection account for 0.8s (0.9%), together contributing 2.2% of RCR’s total 91.6s runtime. The remaining 97.8% (89.6s) is agent LLM inference. This confirms that the routing components are lightweight relative to LLM calls, and that RCR’s net runtime reduction over Full-Context (91.6s vs. 150.65s) comes primarily from shorter, more relevant LLM contexts rather than from routing computation itself. We will add this breakdown to the revised paper.

4.10 Role Configuration Sensitivity.

Role configurations vary across benchmarks: HotPotQA uses three roles, while ALFWorld and WebShop use four. RCR remains robust across role taxonomies, with role decomposition generated by an LLM-assisted agent (Appendix F.1) and shared across methods. The importance scorer uses free-form role descriptions. Table 9 shows only 1.4-point F1 variation (82.4–83.8) from 2-role coarse to 5-role fine decompositions, with gradual cost increase. Together with the agent-count study (Section 4.7), these results demonstrate robustness to role count and decomposition.

5 Conclusion

We presented RCR, an execution-time context allocation layer that formulates memory routing as a role-conditioned, token-budgeted selection problem. By combining budget allocation, relevance scoring, size-aware routing, and iterative memory updates, RCR reduces redundant context exposure while maintaining task performance across multi-hop QA, ALFWorld, and WebShop, outperforming full-context, static-routing, retrieval-based, debate-based, and controlled routing baselines.

Limitations

RCR does not optimize the agent topology or design the roles themselves. Instead, it focuses on deciding which shared-memory items should be exposed to each existing agent under resource constraints. Accordingly, RCR assumes a given multi-agent topology and uses available agent-level meta-data, such as role identifiers or task-stage tags, for context allocation.

Our experiments use fixed multi-agent topologies, with agent-count sensitivity studied for $N \in \{3, 4, 5, 6\}$. These results suggest that RCR can preserve its efficiency advantage within the evaluated range of agent counts, but they do not fully characterize behavior in much larger systems with more roles, longer interaction horizons, or more complex coordination patterns.

Our evaluation covers multi-hop reasoning on HotPotQA (Yang et al., 2018), embodied decision-making on ALFWorld (Shridhar et al., 2021), and web navigation on WebShop (Yao et al., 2022), with additional multi-hop QA results on MuSiQue (Trivedi et al., 2022) and 2WikiMulti-Hop (Ho et al., 2020) reported in Appendix G.2. This provides preliminary evidence of applicability beyond a single task family, but future work should evaluate RCR in additional domains such as long-horizon tool use, collaborative coding, and larger-scale multi-agent workflows.

Finally, our main experiments use GPT-5 mini (OpenAI, 2025a) as the primary agent backbone. Appendix G.1 reports backbone-transfer experiments on Gemini 2.5 Pro (Google Cloud, 2025) and Qwen3-Max (Qwen, 2025). These results reduce, but do not eliminate, the concern that RCR is tied to a specific LLM backbone. Future work could further test broader open-weight, closed-source, multilingual, and domain-specialized backbones.

Acknowledgments

This work was supported in part by the National Science Foundation under Grant No. SaTC-2414652.

References

Anthropic. 2025. How we built our multi-agent research system. <https://www.anthropic.com/engineering/multi-agent-research-system>. Accessed: 2026-5-8.

Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. 2024. Self-rag: Learning to retrieve, generate, and critique through self-reflection. *ICLR*.

Harrison Chase. 2024. langchain: Balance agent control with agency. <https://github.com/langchain-ai>. Accessed: 2026-05-08.

Zhiyuan Cheng, Longying Lai, Yue Liu, Kai Cheng, and Xiaoxi Qi. 2026a. Enhancing financial report question-answering: A retrieval-augmented generation system with reranking analysis. In *2026 6th International Conference on Electrical, Computer and Energy Technologies (ICECET)*, pages 1–7.

Zhiyuan Cheng, Longying Lai, Yue Liu, and Yu Sun. 2026b. Toward sustainable on-device intelligence: A survey on energy-efficient rag systems with small language models. *Available at SSRN 6698538*.

Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E. Gonzalez, Ion Stoica, and Eric P. Xing. 2023. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality.

crewai. 2025. CrewAI: Ai agents for collaborative automation. <https://www.crewai.com/>. Accessed: 2026-05-08.

Shiyi Du, Jiayuan Liu, Weihua Du, Yue Huang, Jiayi Li, Yingtao Luo, Xiangliang Zhang, Vincent Conitzer, and Carl Kingsford. 2026. Why search when you can transfer? amortized agentic workflow design from structural priors. *arXiv preprint arXiv:2604.25012*.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B Tenenbaum, and Igor Mordatch. 2023. Improving factuality and reasoning in language models through multiagent debate. In *Forty-first International Conference on Machine Learning*.

Xingyu Fu, Siyi Liu, Yinuo Xu, Pan Lu, Guangqiuse Hu, Tianbo Yang, Taran Anantasagar, Christopher Shen, Yikai Mao, Yuanzhe Liu, and 1 others. 2025. Learning human-perceived fakeness in ai-generated videos via multimodal llms. *arXiv preprint arXiv:2509.22646*.

Google. 2025. Llm agents – agent development kit (adk). <https://adk.dev/agents/llm-agents/>. Accessed 2026-05-12.

Google Cloud. 2025. Gemini 2.5 Pro. <https://docs.cloud.google.com/gemini-enterprise-agent-platform/models/gemini/2-5-pro>. Google Cloud documentation. Accessed: 2026-05-23.

Shanshan Han, Qifan Zhang, Yuhang Yao, Weizhao Jin, Zhaozhuo Xu, and Chaoyang He. 2024. Llm multi-agent systems: Challenges and open problems. *arXiv preprint arXiv:2402.03578*.

- Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. 2020. [Constructing a multi-hop QA dataset for comprehensive evaluation of reasoning steps](#). In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 6609–6625, Barcelona, Spain (Online). International Committee on Computational Linguistics.
- Mengkang Hu, Tianxing Chen, Qiguang Chen, Yao Mu, Wenqi Shao, and Ping Luo. 2024. Hiagent: Hierarchical working memory management for solving long-horizon agent tasks with large language model. *arXiv preprint arXiv:2408.09559*.
- Jack Lanchantin, Shubham Toshniwal, Jason Weston, Arthur Szlam, and Sainbayar Sukhbaatar. 2023. Learning to reason and memorize with self-notes (2023). URL <https://arxiv.org/abs/2305.00833>.
- Boyi Li, Yifan Shen, Yuanzhe Liu, Yifan Xu, Jiateng Liu, Xinzhuo Li, Zhengyuan Li, Jingyuan Zhu, Yunhan Zhong, Fangzhou Lan, and 1 others. 2026. Toward cognitive supersensing in multimodal large language model. *arXiv preprint arXiv:2602.01541*.
- Fulin Lin, Shaowen Chen, Ruishan Fang, Hongwei Wang, and Tao Lin. 2026. [Stop wasting your tokens: Towards efficient runtime multi-agent systems](#). In *The Fourteenth International Conference on Learning Representations*.
- Jiayuan Liu, Shiyi Du, Weihua Du, Mingyu Guo, and Vincent Conitzer. 2026a. The consensus trap: Rescuing multi-agent llms from adversarial majorities via token-level collaboration. *arXiv preprint arXiv:2604.17139*.
- Jun Liu, Zhenglun Kong, Peiyan Dong, Xuan Shen, Pu Zhao, Hao Tang, Geng Yuan, Wei Niu, Wenbin Zhang, Xue Lin, Dong Huang, and Yanzhi Wang. 2025a. [Rora: Efficient fine-tuning of llm with reliability optimization for rank adaptation](#). In *ICASSP 2025 - 2025 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1–5.
- Jun Liu, Zhenglun Kong, Peiyan Dong, Changdi Yang, Tianqin Li, Yanyue Xie, Yifan Gong, Xuan Shen, Pu Zhao, Hao Tang, Geng Yuan, Wei Niu, Wenbin Zhang, Xue Lin, Dong Huang, and Yanzhi Wang. 2026b. [Structured agent distillation for large language model agents](#). In *Proceedings of the 25th International Conference on Autonomous Agents and Multiagent Systems, AAMAS '26*, page 3676–3685, Richland, SC. International Foundation for Autonomous Agents and Multiagent Systems.
- Jun Liu, Zhenglun Kong, Pu Zhao, Changdi Yang, Xuan Shen, Hao Tang, Geng Yuan, Wei Niu, Wenbin Zhang, Xue Lin, Dong Huang, and Yanzhi Wang. 2025b. Toward adaptive large language models structured pruning via hybrid-grained weight importance assessment. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 18879–18887.
- Yuanzhe Liu, Jingyuan Zhu, Yuchen Mo, Gen Li, Xu Cao, Jin Jin, Yifan Shen, Zhengyuan Li, Tianjiao Yu, Wenzhen Yuan, and 1 others. 2026c. Palm: Progress-aware policy learning via affordance reasoning for long-horizon robotic manipulation. *arXiv preprint arXiv:2601.07060*.
- OpenAI. 2024a. Openai swarm: Lightweight multi-agent framework. <https://github.com/openai/swarm>. Accessed: 2026-5-8.
- OpenAI. 2024b. Openai text-embedding-3-small and text-embedding-3-large models. <https://platform.openai.com/docs/guides/embeddings>. Accessed: 2026-05-01.
- OpenAI. 2025a. GPT-5 mini Model. <https://developers.openai.com/api/docs/models/gpt-5-mini>. OpenAI API documentation. Accessed: 2026-05-23.
- OpenAI. 2025b. Openai agents sdk: Agents. <https://openai.github.io/openai-agents-python/agents/>. Accessed 2026-05-10.
- Qwen. 2025. Qwen3-Max. <https://qwen.ai/blog?id=qwen3-max>. Qwen blog. Accessed: 2026-05-23.
- Alireza Rezazadeh, Zichao Li, Ange Lou, Yuying Zhao, Wei Wei, and Yujia Bao. 2025. Collaborative memory: Multi-user memory sharing in llm agents with dynamic access control. *arXiv preprint arXiv:2505.18279*.
- Yifan Shen, Yuanzhe Liu, Jingyuan Zhu, Xu Cao, Xiaofeng Zhang, Yixiao He, Wenming Ye, James Rehg, and Imini Lourentzou. 2026. Fine-grained preference optimization improves spatial reasoning in vlms. *Advances in Neural Information Processing Systems*, 38:17929–17960.
- Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. 2021. [ALFWorld: Aligning Text and Embodied Environments for Interactive Learning](#). In *Proceedings of the International Conference on Learning Representations (ICLR)*.
- Haoran Sun, Shaoning Zeng, and Bob Zhang. 2026. [H-MEM: Hierarchical memory for high-efficiency long-term reasoning in LLM agents](#). In *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 341–350, Rabat, Morocco. Association for Computational Linguistics.
- Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, and 1 others. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.
- Harsh Trivedi, Niranjana Balasubramanian, Tushar Khot, and Ashish Sabharwal. 2022. Musique: Multi-hop questions via single-hop question composition.

Transactions of the Association for Computational Linguistics, 10:539–554.

Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. 2020. Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. *Advances in neural information processing systems*, 33:5776–5788.

Yu Wang and Xi Chen. 2025. Mirix: Multi-agent memory system for llm-based agents. *arXiv preprint arXiv:2507.07957*.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, and 1 others. 2023. Autogen: Enabling next-gen llm applications via multi-agent conversation, 2023. URL <https://arxiv.org/abs/2308.08155>, 2308.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. 2025. A-mem: Agentic memory for llm agents. *arXiv preprint arXiv:2502.12110*.

Yingxuan Yang, Huacan Chai, Shuai Shao, Yuanyi Song, Siyuan Qi, Renting Rui, and Weinan Zhang. 2025. Agentnet: Decentralized evolutionary coordination for llm-based multi-agent systems. *arXiv preprint arXiv:2504.00587*.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W Cohen, Ruslan Salakhutdinov, and Christopher D Manning. 2018. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. *arXiv preprint arXiv:1809.09600*.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. 2022. Webshop: Towards scalable real-world web interaction with grounded language agents. *Advances in Neural Information Processing Systems*, 35:20744–20757.

Tianjiao Yu, Xinzhuo Li, Yifan Shen, Yuanzhe Liu, and Ismini Lourentzou. 2025. Core3d: Collaborative reasoning as a foundation for 3d intelligence. *arXiv preprint arXiv:2512.12768*.

Yanwei Yue, Guibin Zhang, Boyang Liu, , and 1 others. 2025. Masrouter: Learning to route llms for multi-agent systems. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15549–15572.

Sizhe Yuen, Francisco Gomez Medina, Ting Su, Yali Du, and Adam J Sobey. 2025. Intrinsic memory agents: Heterogeneous multi-agent llm systems through structured contextual memory. *arXiv preprint arXiv:2508.08997*.

Guibin Zhang, Muxin Fu, Guancheng Wan, and 1 others. 2025. G-memory: Tracing hierarchical memory for multi-agent systems. *arXiv preprint arXiv:2506.07398*.

Jiahao Zhang, Haiyang Zhang, Dongmei Zhang, Yong Liu, and Shen Huang. 2023. End-to-end beam retrieval for multi-hop question answering. *arXiv preprint arXiv:2308.08973*.

Appendix

A Algorithm

Algorithm 1 summarizes the routing workflow.

Algorithm 1 RCR Context Routing

Input: Memory M , Role R_i , Stage S_t , Budget B_i

Output: Routed context C_t^i

$C_{\text{set}} \leftarrow \emptyset$, $rem \leftarrow B_i$

$M' \leftarrow \{m \in M \mid \text{Len}(m) \leq B_i\}$

if $M' = \emptyset$ **then**

return $\emptyset$

end if

for each $m \in M'$ **do**

$\alpha(m) \leftarrow \text{Score}(m, R_i, S_t)$

end for

Sort M' by descending $\alpha(m)/\text{Len}(m)$

for each $m \in M'$ **do**

if $\text{Len}(m) \leq rem$ **then**

$C_{\text{set}} \leftarrow C_{\text{set}} \cup \{m\}$; $rem \leftarrow rem - \text{Len}(m)$

end if

end for

$C_{\text{single}} \leftarrow \{\arg \max_{m \in M'} \alpha(m)\}$

$C_t^i \leftarrow \arg \max_{C \in \{C_{\text{set}}, C_{\text{single}}\}} \sum_{m \in C} \alpha(m)$

return C_t^i

B Additional Implementation Details

B.1 Benchmarks and Metrics.

We evaluate our system on three multi-hop QA benchmarks: *HotPotQA* (Yang et al., 2018), *MuSiQue* (Trivedi et al., 2022), and *2WikiMulti-hopQA* (Ho et al., 2020); one embodied reasoning benchmark, *ALFWorld* (Shridhar et al., 2021); and one web navigation benchmark, *WebShop* (Yao et al., 2022). All reported results are averaged over 5 independent runs.

B.2 Experimental Setup

To ensure comparability, all controlled multi-agent methods use the same agent roles, role prompts, task prompts, tool interfaces, maximum interaction rounds, and stopping conditions within each benchmark. The only varied component is the context routing policy. Full-Context routes the entire

shared memory to every agent, Static Routing uses fixed role-specific context templates, and Multi-Agent Debate uses the same roles and round limit but shares identical context among debating agents. Retrieval-based baselines use the same memory pool and token budget, but retrieve a single shared context rather than role-conditioned contexts.

Dataset Statistics. We follow the official standard settings released with each benchmark. The number of trajectories used for router training is already reported in Appendix D, Table 8 ("Weak-label statistics for training the importance scorer"): 1,000 for HotPotQA, 800 for ALFWorld, and 1,000 for WebShop. We consolidate this information alongside dataset split sizes and test instance counts in a single summary table, Table 10, for easier reference, as suggested by the reviewer.

B.3 Training.

Budget Allocator Training. The token budget allocator is trained with a PPO-style policy-gradient update over $\mathcal{B} = \{512, 1024, 2048, 4096\}$. Its reward combines the benchmark-specific primary metric $Q(\tau)$ with token and latency penalties. We use a moving-average baseline and entropy regularization; budgets are sampled during training and chosen by maximum policy probability at inference. Full optimization details are provided in Appendix C.

Importance Scorer Training. The importance scorer is trained with AdamW (5×10^{-4} learning rate, 10^{-4} weight decay, batch size 256, cosine decay with 10% warm-up). Because the LLM is black-box and routing is discrete, we use weak supervision from logged trajectories. Positives are explicitly supporting memory items; negatives are contradicted items or same-role-stage hard negatives. Ambiguous unused items are discarded, and labels use only information available up to the current round, without gold evidence or future outcomes. The scorer is optimized with a binary relevance loss conditioned on role and task stage. Details are provided in Appendix D.

B.4 Baseline Normalization.

For controlled multi-agent baselines, including Full-Context, Static Routing, Multi-Agent Debate, and RCR, we use the same agent topology, role prompts, task prompts, shared memory pool, tool interface, maximum number of model calls, and stopping condition within each benchmark. The only varied component is the context routing policy.

Full-Context sends the entire shared memory to every agent, Static Routing uses fixed role-specific context allocation, Multi-Agent Debate uses the same agent set but shares identical context during critique, and RCR performs role-conditioned budgeted routing.

For these methods, we use the same underlying memory/evidence pool and match the total prompt budget and number of model calls whenever applicable. They retrieve a single shared context for generation.

B.5 Split Protocol and Leakage Prevention.

For each benchmark, we construct disjoint train, validation, and test splits at the task-instance level. The allocator and importance scorer are trained only on trajectories collected from the training split. Hyperparameters, including the default budget and routing thresholds, are selected on the validation split. All numbers in Tables are reported on the held-out test split. No test trajectories, final answers, gold evidence labels, or post-hoc task outcomes are used for training the scorer or allocator. During evaluation, the shared memory is initialized only with information available to the agents before or during execution, and memory updates are generated online from agent outputs, preventing leakage from future rounds or gold annotations.

B.6 Reward Coefficient Tuning

The reward coefficients are tuned on the development split via grid search over $\lambda_{\text{Tok}}, \lambda_{\text{Lat}} \in \{0.01, 0.05, 0.1\}$, yielding $\lambda_{\text{Tok}} = 0.05$ and $\lambda_{\text{Lat}} = 0.01$ for HotPotQA and ALFWorld, and $\lambda_{\text{Tok}} = 0.05$ and $\lambda_{\text{Lat}} = 0.05$ for WebShop. To make all reward terms comparable, $Q(\tau)$ is normalized to $[0, 1]$ across benchmarks, and $\text{Tok}(\tau)$ and $\text{Lat}(\tau)$ are standardized by development-set mean and standard deviation before applying the coefficients.

Table 11 reports a small sensitivity study on the HotPotQA development split. Larger token or latency penalties reduce cost but can slightly lower F1. We select $(\lambda_{\text{Tok}}, \lambda_{\text{Lat}}) = (0.05, 0.01)$ because it gives the best development-set efficiency-performance trade-off, and keep this setting fixed for test evaluation.

B.7 Role Configuration.

Each benchmark adopts a task-specific cooperative-agent configuration following the instruction-based agent abstraction used in the OpenAI Agents

Table 10: Dataset statistics and routing data configuration.

Benchmark	Train/Val/Test	Router Traj.	Test Inst.	Mem Items	Mem Tokens
HotPotQA	90,447 / 7,405 / 7,405 (fullwiki)	1,000	7,405	18.6	3.2K
ALFWorld	3,553 / 140 / 134	800	134	24.3	4.1K
WebShop	10,587 / 1,000 / 500	1,000	500	31.7	5.6K

Table 11: Sensitivity to reward coefficients on the HotPotQA development split. The selected setting is marked in bold. Token and runtime are measured from actual execution.

λ_{Tok}	λ_{Lat}	F1 $\uparrow$	Token (K) $\downarrow$	Runtime (s) $\downarrow$
0.01	0.01	83.4	3.91	96.8
0.05	0.01	83.6	3.63	91.6
0.10	0.01	83.1	3.41	88.9
0.05	0.05	83.2	3.58	89.7
0.05	0.10	82.7	3.49	87.5

SDK (OpenAI, 2025b). For multi-hop QA tasks (HotPotQA, MuSiQue, 2WikiMultihop), we employ three functional roles: Planner, Searcher, and Recommender, responsible for query decomposition, evidence retrieval, and answer synthesis, respectively.

For embodied reasoning (ALFWorld), we extend to a four-role setup: Planner, Navigator, Interactor, and Verifier, corresponding to high-level planning, environment navigation, object manipulation, and goal verification phases during textual interaction with the environment.

For the *multi-turn recommendation* benchmark (WebShop), we adopt a conversational four-agent configuration consisting of a **Buyer** (expresses preferences and constraints such as price, style, or brand), a **Retriever** (searches the catalog for matching products), a **Rater** (evaluates candidate items based on metadata and user alignment), and a **Recommender** (produces the final ranked list with concise justifications).

All roles share a global memory store M_t and communicate through role-conditioned context routing in the RCR framework. RCR uses these fixed role and stage annotations only as routing metadata and does not optimize the agent topology. In RCR, the trained Token Budget Allocator selects each agent’s budget B_i from $\{512, 1024, 2048, 4096\}$, while $B_i=2048$ serves as the reference budget for controlled baseline comparisons. Detailed prompt examples are provided in Appendix F.

B.8 Reproducibility Checklist

See Table 12.

C Reward-Optimized Budget Allocation

Policy Inputs and Actions. At interaction round t , for agent A_i , the budget allocator observes

$$o_t^i = [r_i; s_t; u_i], \quad (5)$$

where r_i is the role embedding, s_t is the stage embedding, and u_i summarizes recent token consumption, context utilization, latency, and task feedback. The allocator outputs a distribution over a discrete budget set

$$\mathcal{B} = \{512, 1024, 2048, 4096\}. \quad (6)$$

A budget is sampled as

$$B_i \sim \pi_\theta(\cdot \mid o_t^i). \quad (7)$$

Given B_i , RCR orders memory items by normalized utility $\alpha(m; R_i, S_t)/\ell(m)$ and applies the size-aware online selector.

Reward Function. For each benchmark, $Q(\tau)$ is instantiated as the official primary metric. We normalize $\text{Tok}(\tau)$ and $\text{Lat}(\tau)$ using development-set statistics before reward aggregation, so the task-quality and cost terms remain on comparable scales. The coefficients λ_{Tok} and λ_{Lat} are selected on the development split and fixed for test evaluation.

Trajectory Logs. Allocator training uses logged trajectories collected under the same agent roles, prompts, tool interfaces, memory pool, and stopping rules as the main experiments. To avoid self-labeling from RCR, we initialize training with logs generated by non-RCR policies, including Full-Context and Static Routing. Each trajectory records the shared memory state, routed contexts, agent outputs, final task metric, token usage, and latency. These logs provide the initial state-action-outcome tuples for policy optimization; the allocator is then refined using the reward in Eq. (4).

Table 12: Reproducibility checklist for controlled experiments.

Component	Specification
Agent interface	Same role descriptions, task prompts, tool interfaces, round limits, and stopping rules are used across controlled multi-agent methods.
Role metadata	Roles are treated as metadata provided by the underlying agent framework or decomposition module. RCR does not require a fixed role taxonomy.
Stage metadata	Stage descriptors are used as optional routing metadata. Baselines that do not use stage-conditioned routing ignore this metadata.
Memory pool	All controlled methods use the same shared-memory pool within each benchmark.
Routing policy	The only varied component is how memory items are selected and routed to agents.
Evaluation	QA benchmarks use Precision/Recall/F1. ALFWorld and WebShop use their benchmark-specific success or task-score metrics. Runtime and token usage are reported across all benchmarks.

Training Objective. We train the allocator with the following policy-gradient objective:

$$\mathcal{L}_{\text{PG}} = - \sum_{t,i} \log \pi_{\theta}(B_i \mid o_t^i) (R(\tau) - b), \quad (8)$$

where b is a moving-average baseline used for variance reduction. The baseline is updated as

$$b \leftarrow \beta b + (1 - \beta) R(\tau), \quad (9)$$

where β is a smoothing coefficient.

PPO-Style Stabilization. For stable optimization, we use a PPO-style clipped update. Let θ_{old} denote the policy parameters before the update. The importance ratio is defined as

$$\rho_{t,i} = \frac{\pi_{\theta}(B_i \mid o_t^i)}{\pi_{\theta_{\text{old}}}(B_i \mid o_t^i)}. \quad (10)$$

The trajectory-level advantage is

$$A(\tau) = R(\tau) - b. \quad (11)$$

The clipped PPO loss is

$$\mathcal{L}_{\text{PPO}} = - \sum_{t,i} \min \left(\rho_{t,i} A(\tau), \right. \\ \left. \text{clip}(\rho_{t,i}, 1 - \epsilon, 1 + \epsilon) A(\tau) \right). \quad (12)$$

We additionally use entropy regularization to encourage exploration:

$$\mathcal{L}_{\text{alloc}} = \mathcal{L}_{\text{PPO}} - \eta \sum_{t,i} \mathcal{H}(\pi_{\theta}(\cdot \mid o_t^i)), \quad (13)$$

where η is the entropy coefficient.

Inference. During inference, we use deterministic budget selection:

$$B_i = \arg \max_{B \in \mathcal{B}} \pi_{\theta}(B \mid o_t^i). \quad (14)$$

The selected budget is then passed to the routing module. Thus, the allocator controls how much context each agent receives, while the importance scorer determines which memory items are selected under that budget.

D Importance Scorer Architecture

Inputs. At interaction round t , for a memory item m and agent A_i with role R_i and stage S_t , we construct three textual inputs: the memory text x_m , the role description x_{R_i} , and the stage description x_{S_t} . We also compute the memory age Δt , defined as the number of interaction rounds since m was generated.

Shared Text Encoder. We use a compact Transformer encoder initialized from MiniLM-L6 (Wang et al., 2020) to encode memory, role, and stage inputs:

$$\begin{aligned} e_m &= \text{Enc}(x_m), \\ e_R &= \text{Enc}(x_{R_i}), \\ e_S &= \text{Enc}(x_{S_t}). \end{aligned} \quad (15)$$

The encoder is shared across the three inputs to reduce parameters and improve training stability. To distinguish input types, we prepend lightweight type prefixes such as [MEM], [ROLE], and [STAGE] before encoding. Unless otherwise stated, the encoder uses 3 Transformer layers, 8 attention heads, and hidden size 512.

Temporal Encoding. To model recency, we encode the memory age Δt as a temporal feature:

$$e_T = \text{TimeEmb}(\Delta t). \quad (16)$$

In our implementation, $\text{TimeEmb}(\cdot)$ is a learnable embedding table over discretized interaction ages, with a fallback bucket for memories older than the maximum tracked age. We use a temporal embedding dimension of 32 and project it to the same dimension as the text embeddings before fusion.

Role- and Stage-Conditioned Fusion. The scorer combines memory, role, stage, and temporal features through gated conditioning:

$$\tilde{e}_m = e_m \odot \sigma(W_R e_R) \odot \sigma(W_S e_S) \odot \sigma(W_T e_T), \quad (17)$$

where $\odot$ denotes elementwise multiplication and $\sigma(\cdot)$ is the sigmoid function. This gating mechanism conditions the memory representation on both the agent role and the current task stage.

We then form the scoring representation:

$$z_m = [\tilde{e}_m; e_R; e_S; e_T], \quad (18)$$

where $[\cdot; \cdot]$ denotes concatenation.

Scoring Head. The final relevance score is computed by a two-layer MLP:

$$\alpha(m; R_i, S_t) = \sigma \left(w_2^\top \text{ReLU}(W_1 z_m + b_1) + b_2 \right), \quad (19)$$

where $\alpha(m; R_i, S_t) \in (0, 1)$ represents the relevance of memory item m for role R_i and stage S_t . During routing, memory items are ranked by normalized utility:

$$u(m; R_i, S_t) = \frac{\alpha(m; R_i, S_t)}{\ell(m)}, \quad (20)$$

where $\ell(m)$ is the token length of m .

Importance Scorer Label Construction. The scorer is trained as a role- and stage-conditioned relevance ranker over memory items using logged training trajectories. Positive examples are memory items explicitly linked to successful outputs, evidence chains, tool outputs, or action traces. High-confidence negatives are restricted to contradicted items and hard negatives sampled from the same role-stage memory pool. We do not treat all unused items as negatives, since unused memory items may still be potentially relevant but omitted due to budget limits or because an equivalent item was selected. Instead, unused items are triaged into soft negatives or unlabeled masked items according to confidence.

Labels are constructed only from training-split trajectory records. Gold answers, gold evidence

annotations, test trajectories, and future-round information unavailable at the current routing step are not used for scorer training.

We use weighted BCE to account for weak-label confidence:

$$\mathcal{L}_{\text{score}} = - \sum_{i,t,m} w_m^{i,t} b_m^{i,t}, \quad (21)$$

where

$$b_m^{i,t} = y_m^{i,t} \log \alpha_m^{i,t} + (1 - y_m^{i,t}) \log(1 - \alpha_m^{i,t}). \quad (22)$$

where $y_m^{i,t} \in \{0, 1\}$ is the weak relevance label, $\alpha_m^{i,t} = \alpha(m; R_i, S_t)$, and $w_m^{i,t}$ is the label-confidence weight:

$$(y_m^{i,t}, w_m^{i,t}) = \begin{cases} (1, 1), & \text{high-confidence positive,} \\ (0, 1), & \text{high-confidence negative,} \\ (0, \eta), & \text{soft negative,} \\ (\cdot, 0), & \text{unlabeled (ambiguous).} \end{cases} \quad (23)$$

We select $\eta = 0.3$ on the development split. Soft negatives are unused items whose role-stage mismatch or low similarity to successful evidence suggests likely irrelevance, but which are not contradicted by trajectory evidence with full confidence. Fully ambiguous items are masked out from the scorer loss rather than treated as negatives.

Weak-Label Triage. Table 13 summarizes the resulting weak-label statistics. Items explicitly supporting successful evidence chains, tool outputs, or action traces are labeled as positives. Contradicted items and same-role-stage hard negatives are labeled as hard negatives. Soft negatives are identified by role-stage tag mismatch or cosine similarity below 0.3 to any positive item in the same trajectory round; they are retained in training with reduced loss weight η . Items whose relevance cannot be reliably inferred from the logged trajectory are left unlabeled and assigned zero loss weight. Thus, the ‘‘Unlabeled’’ column in Table 13 does not indicate that these items are treated as irrelevant; rather, they are excluded from the hard-label loss to avoid introducing false-negative noise.

Table 14 gives concrete examples of weak-label construction across benchmarks. These examples

Table 13: Weak-label statistics for training the importance scorer. We use a three-way triage: high-confidence positives, high-confidence negatives, soft negatives (reduced loss weight η), and unlabeled ambiguous items (zero loss weight).

Benchmark	Traj.	Items	Pos.	Hard Neg.	Soft Neg.	Unlabeled
HotPotQA	1,000	48.2K	18.6%	31.4%	17.3%	32.7%
ALFWorld	800	39.5K	16.9%	29.8%	16.6%	36.7%
WebShop	1,000	56.7K	15.4%	33.1%	17.0%	34.5%

illustrate that negative labels are not arbitrary unused items: high-confidence negatives are contradicted records or hard negatives from the same role-stage pool, while lower-confidence unused records are treated as soft negatives or left unlabeled.

E Memory Update Agent Example

Rather than manually writing task-specific parsers, we implement memory construction as a dedicated structured-output agent following the standard agent abstraction with instructions and output types (OpenAI, 2025b). The Memory Update Agent converts raw agent outputs into lightweight records:

$$m = \{\text{role_tag}, \text{stage_tag}, \text{timestamp}, \text{content}\}.$$

RCR itself is agnostic to how these records are produced; they may come from a dedicated structured-output agent, a rule-based parser, or an external memory system.

Update Example. Given a raw Searcher output: The evidence states that Dataset Y contains 12K labeled samples.

the Memory Update Agent produces the structured memory item:

```
{
  "role_tag": "Searcher",
  "stage_tag": "Evidence",
  "timestamp": 3,
  "content": "Dataset Y contains 12K
              labeled samples."
}
```

At round t , the Memory Update Agent reads the outputs of all agents and generates candidate memory items ΔM_t . The shared memory is updated as

$$M_{t+1} = \text{Dedup}(M_t \cup \Delta M_t).$$

In our implementation, near-duplicate entries within the same role-stage group are removed using embedding similarity, retaining the more recent

item. This step controls memory growth and reduces redundant entries, but is not a core assumption of the routing algorithm.

Given the updated memory, RCR scores each item by $\alpha(m; R_i, S_t)$ and routes items according to normalized utility $\alpha(m; R_i, S_t)/\ell(m)$ under the allocated token budget B_i .

Deduplication Example. Suppose the shared memory already contains:

```
{
  "role_tag": "Searcher",
  "stage_tag": "Evidence",
  "timestamp": 3,
  "content": "Dataset Y contains 12K
              labeled samples."
}
```

and the Memory Update Agent generates a new item:

```
{
  "role_tag": "Searcher",
  "stage_tag": "Evidence",
  "timestamp": 5,
  "content": "Dataset Y has 12,000 labeled
              examples."
}
```

Since the two items share the same role-stage tag and their embedding similarity exceeds the duplicate threshold, RCR keeps the more recent item:

```
{
  "role_tag": "Searcher",
  "stage_tag": "Evidence",
  "timestamp": 5,
  "content": "Dataset Y has 12,000
              labeled examples."
}
```

This deduplication step reduces redundant memory growth but does not assume perfect semantic conflict resolution.

F Role Configuration and Prompt Examples

F.1 LLM-assisted role decomposition.

We use an external LLM-based decomposition agent implemented with OpenAI Agents SDK (OpenAI, 2025b) to generate candidate role configurations for each benchmark. The decomposition agent is prompted with the benchmark description, task objective, available tool interface, and general role-specialization principles

Table 14: Examples of weak-label construction for the importance scorer. Labels are derived from logged training trajectories without gold evidence annotations, test trajectories, or future-round leakage.

Benchmark	Memory item	Label type	Reason
HotPotQA	Retrieved passage cited by the Searcher and later used by the Synthesizer in a successful answer trajectory.	Positive	Linked to the logged evidence chain used in the successful trajectory.
HotPotQA	Entity-overlapping passage retrieved for the same question but contradicted by the later accepted evidence.	Hard negative	Same-role-stage candidate whose content conflicts with the logged successful evidence.
HotPotQA	Unused passage with partial entity overlap but no clear support or contradiction in the trajectory.	Unlabeled	Relevance cannot be reliably inferred; masked out from the scorer loss to avoid false-negative noise.
ALFWorld	Action/state trace: open fridge; put apple in fridge.	Positive	Appears in a successful logged action trajectory satisfying the goal condition.
ALFWorld	Action/state trace: put apple on counter when the target goal requires storing the apple in the fridge.	Hard negative	Contradicts the target state observed in the successful trajectory.
ALFWorld	Unused navigation observation from a different room with weak lexical overlap to the goal.	Soft negative	Likely irrelevant due to role-stage mismatch, but not contradicted with full confidence; included with reduced loss weight.
WebShop	Candidate product metadata matching logged user constraints, e.g., white blackout shade, correct size, and cordless option.	Positive	Used by the Recommender in the final selected product rationale.
WebShop	Candidate product metadata violating a hard user constraint, e.g., wrong size or non-blackout material.	Hard negative	Same-stage candidate inconsistent with logged user constraints.
WebShop	Unselected candidate with similar category but incomplete evidence about size or material.	Soft negative	Likely less useful than selected evidence but not confidently irrelevant; included with reduced loss weight.

commonly used in LLM-agent systems, such as separating planning, retrieval/search, interaction, verification, memory management, and synthesis. The generated roles are fixed before evaluation and shared by all compared methods. We apply only minimal post-processing to merge duplicate roles and normalize role names; no role is selected or modified based on test-set performance.

Role configurations follow a standard instruction-based agent abstraction with structured prompts, as supported by the OpenAI Agents SDK (OpenAI, 2025b). Each agent is defined by a role description, task instruction, available tools, and output format. RCR uses role descriptions only as routing metadata. In our experiments, roles are generated by an LLM-assisted decomposition agent, fixed before evaluation, and shared by all compared methods. The same generated topology is used for Full-Context, Static Routing, Multi-Agent Debate, and RCR, so differences in performance and efficiency are attributed to the routing policy rather than to role design.

F.2 Multi-hop QA (HotPotQA).

These benchmarks require decomposing complex questions into sub-queries, retrieving supporting evidence, and synthesizing final answers. The LLM-assisted decomposition agent produces three cooperative roles, which are instantiated as role-specific prompt templates:

- **Planner:** parses the original question into reasoning steps and generates intermediate sub-queries.
- **Searcher:** retrieves relevant passages or facts from the shared memory or corpus.
- **Recommender:** integrates evidence and formulates the final answer with natural-language justification.

Prompt Template Example:

For the **user** query “Which film featuring Tom Hanks won the Oscar for Best Picture?”, The **Planner** decomposes it into “list films starring Tom Hanks” and “find which of them won Best Picture,”

The **Searcher** retrieves the relevant Wikipedia entries, and the **Recommender** concludes “*Forrest Gump*.”

The **Recommender** outputs: “*Nike Revolution 5—lightweight and under \$90, highly rated for comfort.*”

F.3 Embodied Reasoning (ALFWorld).

ALFWorld involves multi-step textual interaction with a simulated environment. The decomposition agent maps the task pipeline into functional roles for planning, navigation, object interaction, and verification:

- **Planner:** decomposes the task goal into sequential subtasks.
- **Navigator:** determines movement actions to reach the target location.
- **Interactor:** executes object-level actions such as pick, open, and place.
- **Verifier:** checks whether the goal condition is satisfied.

Prompt Template Example:

Given the **instruction** “Put the soap in the sink,”
The **Planner** formulates a plan, the **Navigator** moves to the bathroom,
The **Interactor** picks up the soap and places it in the sink,
and the **Verifier** confirms completion.

F.4 Multi-turn Recommendation (WebShop).

In WebShop, the system engages in goal-oriented shopping interaction. The decomposition agent instantiates four functional prompt templates corresponding to user-intent reformulation, retrieval, candidate evaluation, and final recommendation:

- **Buyer:** reformulates user preferences and constraints.
- **Retriever:** searches the online catalog for matching products.
- **Rater:** evaluates retrieved candidates using product metadata and user preference alignment.
- **Recommender:** selects and presents the final ranked list with concise justifications.

Prompt Template Example:

When the user says “I need lightweight running shoes under \$100,”
The **Buyer** reformulates the intent and constraints,
The **Retriever** fetches candidate items from the product database,
The **Rater** scores them based on utility and user fit,

These configurations reflect domain-specific functional decompositions rather than arbitrary design choices. They align with each task’s reasoning or interaction pipeline and were validated empirically in ablation studies (see Table 5). Future work may explore automated role discovery or hierarchical role refinement to further generalize RCR.

G Additional Experiment Results

G.1 Backbone-Transfer Robustness.

The main experiments use GPT-5-mini as the agent backbone. To test whether RCR is tied to a single model family or deployment setting, we conduct a backbone-transfer study using Qwen3-Max and Gemini 2.5 Pro. Gemini 2.5 Pro is used as a stronger reasoning-oriented closed-source API backbone. Qwen3-Max is used as a trillion-parameter Qwen-family model and represents a non-English-origin large-model family. Importantly, the Importance Scorer and Token Budget Allocator are trained only once using GPT-5-mini trajectories and are frozen when transferred to the other backbones. This frozen-router setting evaluates whether RCR acts as a model-agnostic context allocation layer rather than a backbone-specific tuning strategy.

As shown in Table 15, absolute task scores vary across agent LLMs. This is expected because the backbones differ in reasoning ability, instruction following, and action-format reliability. Nevertheless, the relative trend remains consistent across HotPotQA, ALFWorld, and WebShop: RCR reduces token usage and runtime compared with Full-Context and Static Routing while preserving or improving the primary task metric. These results provide initial evidence that RCR transfers across model families, open/closed deployment settings, and lightweight versus stronger agent backbones.

G.2 Additional multi-hop QA benchmarks

Table 16 reports additional results on two multi-hop QA benchmarks, MuSiQue and 2WikiMulti-hop, which cover compositional and open-domain reasoning settings. All experiments use the same three-role configuration (Planner, Searcher, Recommender). In RCR, the trained Token Budget Allocator selects the per-agent budget B_i from $\{512, 1024, 2048, 4096\}$ at inference time; $B_i=2048$ is used only as the reference budget

level for normalizing baseline prompt budgets. We report standard QA evaluation metrics (Precision, Recall, and F1) following the official benchmark protocols. Token usage differs across benchmarks due to dataset-specific context characteristics: MuSiQue instances typically require longer compositional evidence chains, while 2WikiMultihop often involves shorter supporting passages, leading to lower prompt token counts under the same evaluation and tokenization settings.

G.3 Diagnostic Analysis of Budget Allocation

Dev-oracle static per-role is obtained by grid search on the development split. For each role, we choose one fixed budget from $\{512, 1024, 2048, 4096\}$ that maximizes the development-set efficiency–performance objective. The selected role-specific budgets are then fixed and applied to all test instances.

Table 17 compares learned TBA with a development-set oracle-tuned static per-role budget. Learned TBA reaches comparable quality while using fewer tokens and less runtime. The dev-oracle baseline can slightly improve raw task quality by assigning larger fixed budgets to individual roles, whereas learned TBA achieves comparable quality through instance-level budget selection at lower token and runtime cost.

G.4 Mechanism Analysis: Routed Memory Case Study

To inspect whether RCR routes role-relevant memory rather than merely shortening prompts, we analyze a HotPotQA-style multi-hop example. The question is: *“Which city is home to both the oldest university in the English-speaking world and a river that shares its name with a US state?”* Table 20 shows the shared memory pool, while Tables 21 and 22 compare a successful routing case with an imperfect routing case. The examples show that RCR can improve reasoning by routing evidence, planning traces, and synthesis notes to different roles, but can still fail when stale or misleading memory receives a high relevance score.

G.5 Sensitivity Analysis

Table 18 reports a fixed-budget diagnostic with the learned allocator disabled. Increasing B_i improves quality at small budgets, but the gains diminish at larger budgets while token usage and runtime continue to increase. We therefore use $B_i=2048$ as a reference budget for controlled comparisons, not

as an upper bound on possible quality. Table 3 reports the cost–performance trade-off of additional routing iterations.

G.6 Compared with the Full-Context to RCR

G.6.1 Full-Context vs. RCR: HotPotQA Case Study

Tables 24 and 25 compare the same HotPotQA instance under Full-Context and RCR. Under Full-Context, each agent receives a mixed context window containing useful information together with stale, duplicate, and role-irrelevant items. RCR suppresses these distracting items and routes compact role-relevant subsets to each agent: the Planner receives decomposition and constraint-tracking information, the Searcher receives evidence-heavy items, and the Recommender receives verified evidence and synthesis cues.

G.6.2 Full-Context vs. RCR: Webshop Case Study

Tables 26 and 27 compare the same WebShop instance under Full-Context and RCR. Full-Context mixes useful product information with stale, duplicate, and role-irrelevant items, whereas RCR filters these distractors and routes compact role-relevant subsets to each agent.

G.6.3 Full-Context vs. RCR: ALFWorld Case Study

Tables 28 and 29 compare the same ALFWorld instance under Full-Context and RCR. Full-Context mixes useful information with stale, duplicate, and role-irrelevant items, whereas RCR filters these distractors and routes compact role-relevant subsets to each agent.

Table 15: Backbone-transfer robustness across model families and deployment settings. The Importance Scorer and Token Budget Allocator are trained once using GPT-5-mini trajectories and then frozen when evaluated with Qwen3-Max and Gemini 2.5 Pro agents. Qwen3-Max provides an open-weight 7B backbone, while Gemini 2.5 Pro provides a stronger closed-source API backbone. This experiment tests whether RCR transfers across model families, capability levels, and open/closed deployment settings without backbone-specific retraining. Results are intended as a robustness check rather than a full re-evaluation of all baselines.

Benchmark	Backbone	Method	Avg Runtime (s)↓	Token (K)↓	Primary Metric↑
HotPotQA (F1, 0–100)	GPT-5-mini	Full-Context	150.65	5.10	74.6 ±0.6
	GPT-5-mini	Static Routing	128.29	3.85	76.1 ±0.6
	GPT-5-mini	RCR	91.58	3.63	83.6 ±0.5
	Qwen3-Max	Full-Context	162.8	5.07	70.9 ±0.8
	Qwen3-Max	Static Routing	137.6	3.86	72.4 ±0.7
	Qwen3-Max	RCR	101.7	3.64	78.6 ±0.7
	Gemini 2.5 Pro	Full-Context	172.5	5.12	78.9 ±0.6
	Gemini 2.5 Pro	Static Routing	146.8	3.88	80.3 ±0.5
	Gemini 2.5 Pro	RCR	109.4	3.66	85.1 ±0.5
ALFWorld (Success Rate, %)	GPT-5-mini	Full-Context	145.3	6.82	59.1 ±0.8
	GPT-5-mini	Static Routing	122.8	5.12	62.2 ±0.7
	GPT-5-mini	RCR	94.5	4.30	67.3 ±0.8
	Qwen3-Max	Full-Context	156.2	6.76	54.8 ±1.0
	Qwen3-Max	Static Routing	131.9	5.08	57.6 ±0.9
	Qwen3-Max	RCR	101.3	4.25	62.4 ±0.9
	Gemini 2.5 Pro	Full-Context	169.8	6.86	63.5 ±0.8
	Gemini 2.5 Pro	Static Routing	142.1	5.16	66.1 ±0.7
	Gemini 2.5 Pro	RCR	111.6	4.34	71.8 ±0.7
WebShop (Reward, 0–1)	GPT-5-mini	Full-Context	180.2	9.78	0.565 ±0.009
	GPT-5-mini	Static Routing	142.6	7.12	0.616 ±0.008
	GPT-5-mini	RCR	108.7	5.91	0.653 ±0.005
	Qwen3-Max	Full-Context	188.9	9.70	0.526 ±0.012
	Qwen3-Max	Static Routing	151.2	7.08	0.572 ±0.010
	Qwen3-Max	RCR	116.8	5.86	0.611 ±0.009
	Gemini 2.5 Pro	Full-Context	205.6	9.82	0.603 ±0.010
	Gemini 2.5 Pro	Static Routing	163.7	7.16	0.646 ±0.008
	Gemini 2.5 Pro	RCR	124.5	5.95	0.681 ±0.006

Table 16: Additional results on **HotPotQA**, **MuSiQue**, and **2WikiMultihop**. Token usage reflects dataset-specific memory pool sizes: MuSiQue instances require longer compositional evidence chains (avg. 4-hop), resulting in larger shared memory; 2WikiMultihop involves shorter bridge passages (avg. 2-hop) with a smaller memory footprint. All results are averaged over five independent runs.

Benchmark	Method	Avg Runtime (s)↓	Token (K)↓	Precision↑	Recall↑	F1↑
HotPotQA	Full-Context	150.65	5.10	72.3	75.1	74.6
	Static Routing	128.29	3.85	74.8	77.5	76.1
	Self-RAG	118.44	3.92	77.5	79.3	78.4
	End-to-End Retrieval	110.12	3.80	78.0	80.1	79.0
	Multi-Agent Debate	105.33	4.06	79.1	81.0	80.0
	RCR (Ours)	91.58	3.63	81.8	84.4	83.6
MuSiQue	Full-Context	231.8	8.24	69.7	70.5	70.1
	Static Routing	197.5	6.21	72.6	73.9	73.2
	Self-RAG	182.2	6.33	74.0	75.2	74.6
	End-to-End Retrieval	169.3	6.13	76.1	77.3	76.7
	Multi-Agent Debate	162.0	6.55	76.9	78.0	77.6
	RCR (Ours)	140.7	5.34	80.0	81.1	80.6
2WikiMultihop	Full-Context	122.4	4.38	70.5	72.1	71.3
	Static Routing	104.3	3.31	73.2	74.8	74.0
	Self-RAG	96.2	3.37	75.6	77.0	76.3
	End-to-End Retrieval	89.5	3.26	78.7	80.0	79.4
	Multi-Agent Debate	85.6	3.49	79.3	80.6	79.9
	RCR (Ours)	74.4	2.84	82.1	83.2	82.4

Table 17: Diagnostic comparison of learned TBA against a development-set oracle-tuned static per-role budget. Uniform static $B_i=2048$ is included as a reference. The oracle baseline selects one fixed budget per role on the development split and applies it to all test instances.

Benchmark	Budget Strategy	Runtime↓	Token (K)↓	Metric↑
HotPotQA (F1, 0–100)	Uniform static $B_i=2048$	95.4	3.69	81.2
	Dev-oracle static per-role	99.8	4.21	83.8
	Learned TBA (RCR)	91.6	3.63	83.6
ALFWorld (Success Rate, %)	Uniform static $B_i=2048$	98.9	4.34	66.3
	Dev-oracle static per-role	103.6	4.86	67.6
	Learned TBA (RCR)	94.5	4.30	67.3
WebShop (Reward, 0–1)	Uniform static $B_i=2048$	112.8	6.16	0.629
	Dev-oracle static per-role	118.5	6.42	0.656
	Learned TBA (RCR)	108.7	5.91	0.653

Table 18: Fixed-budget sensitivity analysis for per-agent budget B_i . RCR is evaluated with $B_i \in \{512, 1024, 2048, 4096\}$ while keeping the routing policy fixed. Each benchmark is evaluated using its official primary metric. Performance improves up to $B_i=2048$ and then saturates, supporting $B_i=2048$ as the reference budget level.

Benchmark	B_i	Ave Runtime (s)↓	Token (K)↓	Primary Metric↑
HotPotQA (F1)	512	75.1	2.42	71.1
	1024	86.7	3.12	76.8
	2048	91.6	3.63	83.6
	4096	126.4	5.85	82.0
ALFWorld (Success %)	512	71.3	2.21	60.5
	1024	83.8	3.13	64.6
	2048	94.5	4.30	67.3
	4096	121.7	6.64	67.1
WebShop (Reward 0–1)	512	85.9	2.58	0.603
	1024	98.3	4.15	0.632
	2048	108.7	5.91	0.653
	4096	142.8	9.74	0.649

Table 19: Scalability on HotPotQA as agent count grows from $N=3$ to $N=6$ by adding Verifier, Critic, and Refiner roles. $N=3$ (Planner / Searcher / Recommender) is identical to the main experiment (Table 1). Token in thousands; runtime in seconds.

Method	Token (K)↓				Runtime (s)↓				F1↑			
	$N=3$	$N=4$	$N=5$	$N=6$	$N=3$	$N=4$	$N=5$	$N=6$	$N=3$	$N=4$	$N=5$	$N=6$
Full-Context	5.10	6.82	8.49	10.18	150.7	201.2	251.3	301.8	74.6	75.8	75.2	74.3
Static Routing	3.85	5.14	6.40	7.65	128.3	171.4	213.8	256.2	76.1	77.2	76.9	76.5
RCR (Ours)	3.63	4.53	5.31	5.98	91.6	109.5	124.3	137.6	83.6	84.8	85.3	85.7

Table 20: Shared memory pool for the qualitative routing example. All items are visible to Full-Context agents, while RCR selects role-specific subsets under each agent’s token budget.

Role	Stage	Time	Memory item
Planner	Decompose	$t=1$	Sub-query 1: find the oldest university in the English-speaking world.
Planner	Decompose	$t=1$	Sub-query 2: find rivers sharing a name with a U.S. state.
Searcher	Evidence	$t=2$	Oxford University was founded around 1096–1167 and is located in Oxford, England.
Searcher	Evidence	$t=2$	River Thames flows through Oxford; River Mississippi shares a name with a U.S. state.
Searcher	Evidence	$t=2$	River Ohio flows through Oxford, Ohio, giving a candidate river–state match.
Searcher	Stale	$t=1$	[Stale] Cambridge was also considered; this is contradicted by later evidence.
Recommender	Synthesis	$t=2$	Oxford satisfies the university condition, but the river-name condition needs revision.

Table 21: Case A: successful role-specific routing. RCR selects planning traces for the Planner, evidence-heavy items for the Searcher, and compact evidence plus synthesis notes for the Recommender.

Planner receives	Searcher receives	Recommender receives
Sub-query 1: oldest English-speaking university.	Oxford University was founded around 1096–1167 and is located in Oxford, England.	Oxford University was founded around 1096–1167 and is located in Oxford, England.
Sub-query 2: river sharing a U.S. state name.	River Thames flows through Oxford; River Mississippi shares a name with a U.S. state.	River Ohio flows through Oxford, Ohio, giving a candidate match.
Reasoning trace: verify whether the candidate city satisfies both constraints.	River Ohio flows through Oxford, Ohio, giving a candidate match.	Synthesis note: the Oxford candidate needs revision on the river-name condition.
Filtered item: the stale Cambridge item is suppressed. The Planner receives reasoning traces without raw evidence overload; the Searcher receives evidence-heavy memory; and the Recommender receives evidence plus synthesis signals rather than all planning traces.		
Resulting outputs: The Planner decomposes the task into university identification and river–state verification. The Searcher flags that Oxford, England satisfies the university condition, while Oxford, Ohio provides the river–state match. The Recommender revises the initial candidate and avoids relying on the stale Cambridge memory.		

Table 22: Case B: imperfect routing under stale-memory interference. The scorer assigns high relevance to a stale item, causing key river evidence to be dropped under the Searcher’s budget.

Planner receives	Searcher receives	Recommender receives
Sub-query 1: oldest English-speaking university.	Oxford University was founded around 1096–1167 and is located in Oxford, England.	Oxford University was founded around 1096–1167 and is located in Oxford, England.
Sub-query 2: river sharing a U.S. state name.	[Stale] Cambridge was also considered.	Synthesis note: the river condition needs revision.
Reasoning trace: verify both constraints.	River Thames flows through Oxford.	Missing key evidence: River Ohio match is absent.
Dropped item: the River Ohio evidence is not routed to the Searcher because the stale Cambridge item consumes part of the budget. The Recommender therefore lacks the key river–state evidence needed to complete verification.		
Resulting outputs: The Planner still produces the correct decomposition, but the Searcher reports Oxford/Cambridge as competing university candidates and fails to retrieve the River Ohio evidence. The Recommender returns an incomplete or incorrect answer because the river condition remains unresolved.		

Table 23: Distribution of selected memory types across roles on HotPotQA. Values denote the percentage of routed memory items assigned to each role.

Memory Type	Planner	Searcher	Recommender
Reasoning traces	61%	12%	24%
Retrieved evidence	8%	71%	45%
Sub-plans	28%	9%	18%
Tool outputs	3%	8%	13%

Table 24: Illustrative HotPotQA Full-Context examples showing context flooding in Figure 1. Black spans denote function-relevant content, while red spans denote redundant, stale, or role-irrelevant content interleaved within the same context window.

Agent	Mixed context window
Planner	Current goal: decompose the question into two sub-queries ; earlier brainstorming suggested Cambridge but was later contradicted ; identify the oldest English-speaking university ; duplicate raw evidence from a previous search ; verify whether the candidate city also satisfies the river-name condition ; unrelated notes about university rankings and old search attempts.
Searcher	Oxford University was founded around 1096–1167 and is located in Oxford, England ; stale snippet about Cambridge as a possible oldest university ; River Ohio flows through Oxford, Ohio and shares its name with a U.S. state ; duplicate search results about the River Thames ; retrieve evidence linking each candidate city to the relevant constraint ; broken links, unrelated pages, and outdated retrieval keywords.
Recommender	The answer must satisfy both the university condition and the river-name condition ; unused draft answer: Oxford, England ; verified evidence supports revising the initial candidate ; repeated evidence fragments and obsolete summaries ; final synthesis should use only verified evidence ; irrelevant meeting notes and superseded reasoning traces.

Table 25: RCR routed-context example on the same HotPotQA instance. RCR suppresses stale, duplicate, and role-irrelevant items, and routes compact role-relevant subsets to each agent under token budgets.

Agent	RCR routed context
Planner	Current goal: decompose the question into two sub-queries. Identify the oldest English-speaking university. Verify whether the candidate city also satisfies the river-name condition.
Searcher	Oxford University was founded around 1096–1167 and is located in Oxford, England. River Ohio flows through Oxford, Ohio and shares its name with a U.S. state. Retrieve evidence linking each candidate city to the relevant constraint.
Recommender	The answer must satisfy both the university condition and the river-name condition. Verified evidence supports revising the initial candidate. Final synthesis should use only verified evidence.

Table 26: Illustrative WebShop Full-Context examples showing context flooding in Figure 1. Black spans denote function-relevant content, while red spans denote redundant, stale, or role-irrelevant content interleaved within the same context window.

Agent	Mixed context window
Planner	Current goal: find a lightweight running shoe under \$100 ; earlier plan suggested formal leather shoes but was abandoned ; decompose the task into price, category, size, and review constraints ; duplicate notes from previous searches for hiking boots ; prioritize products matching the user’s stated budget and use case ; unrelated browsing history about office chairs and phone cases.
Searcher	Product result: lightweight running shoe, size 10, price \$79.99, average rating 4.5 ; stale result: out-of-stock running shoe from an earlier page ; retrieved attribute: breathable mesh upper and suitable for daily running ; duplicate search result from the same seller with a different thumbnail ; compare price, availability, rating, and shipping constraints ; unrelated product snippets for dress shoes, sandals, and backpacks.
Recommender	The selected item should satisfy category, price, availability, and user preference constraints ; unused draft recommendation for a leather loafer ; verified evidence supports the lightweight running shoe as the best match ; repeated product descriptions and obsolete cart candidates ; final recommendation should cite price, rating, and matching attributes ; irrelevant notes about previously rejected products and unrelated categories.

Table 27: RCR routed-context example on the same WebShop instance. RCR removes stale, duplicate, and role-irrelevant items, and routes compact role-relevant subsets to each agent under token budgets.

Agent	RCR routed context
Planner	Current goal: find a lightweight running shoe under \$100. Extract constraints: category = running shoe, price < \$100, size = 10, preference = lightweight and well-reviewed. Prioritize candidates matching the user’s stated budget and use case.
Searcher	Candidate product: lightweight running shoe, size 10, price \$79.99, average rating 4.5. Retrieved attributes: breathable mesh upper, daily running use, available stock. Compare price, availability, rating, and shipping constraints.
Recommender	Verified evidence supports the lightweight running shoe as the best match. The item satisfies category, price, size, availability, and user preference constraints. Final recommendation should cite price, rating, and matching attributes.

Table 28: Illustrative ALFWorld Full-Context examples showing context flooding in Figure 1. Black spans denote function-relevant content, while red spans denote redundant, stale, or role-irrelevant content interleaved within the same context window.

Agent	Mixed context window
Planner	Current goal: put a clean mug on the coffee table ; earlier plan searched the bathroom sink but was abandoned ; decompose the task into locating a mug, cleaning it, and moving it to the target receptacle ; duplicate notes about opening already-checked drawers ; track unresolved constraints: mug must be clean and placed on the coffee table ; unrelated observations about a knife, apple, and bedroom lamp.
Searcher	Observation: a dirty mug is on the kitchen counter ; stale observation: mug might be in the cabinet, contradicted by later observation ; useful receptacles: sink can clean the mug, coffee table is the target location ; repeated action log: open drawer, close drawer, open drawer ; retrieve next actionable locations for washing and placing the mug ; unrelated room-state traces from the bedroom and pantry.
Recommender	The next action should move toward satisfying both conditions: clean the mug and place it on the coffee table ; unused draft action: put mug in cabinet ; verified trace supports the sequence: pick up mug, go to sink, clean mug, go to coffee table ; obsolete action candidates from failed drawer exploration ; final action proposal should avoid revisiting checked locations ; irrelevant object notes about the knife, apple, and lamp.

Table 29: RCR routed-context example on the same ALFWorld instance. RCR removes stale, duplicate, and role-irrelevant items, and routes compact role-relevant subsets to each agent under token budgets.

Agent	Routed context type	RCR routed context
Planner	Reasoning state and sub-plan	Current goal: put a clean mug on the coffee table. Decompose the task into: locate mug, clean mug, and move it to the coffee table. Track constraints: the mug must be clean and the final receptacle must be the coffee table.
Searcher	Evidence and actionable observations	Observation: a dirty mug is on the kitchen counter. Useful receptacle: the sink can be used to clean the mug. Target location: the coffee table. Next actionable locations: kitchen counter, sink, and coffee table.
Recommender	Verified action trace and next action	Verified trace: pick up mug, go to sink, clean mug, go to coffee table. Avoid revisiting checked drawers or unrelated rooms. Recommended next step: continue the action sequence that satisfies both the clean-mug and target-placement constraints.