

# RCR-Router: Efficient Role-Aware Context Routing for Multi-Agent LLM Systems with Structured Memory

Jun Liu<sup>1,2</sup>, Zhenglun Kong<sup>3</sup>, Changdi Yang<sup>2</sup>, Fan Yang<sup>4</sup>, Tianqin Li<sup>1</sup>, Peiyan Dong<sup>5</sup>, Joannah Nanjekye<sup>1</sup>, Hao Tang<sup>6</sup>, Geng Yuan<sup>7</sup>, Wei Niu<sup>7</sup>, Wenbin Zhang<sup>8</sup>, Pu Zhao<sup>2</sup>, Xue Lin<sup>2</sup>, Dong Huang<sup>1</sup>, Yanzhi Wang<sup>2</sup>

<sup>1</sup>Carnegie Mellon University, <sup>2</sup>Northeastern University, <sup>3</sup>Harvard University, <sup>4</sup>Fujitsu Research of America, <sup>5</sup>MIT, <sup>6</sup>Peking University, <sup>7</sup>University of Georgia, <sup>8</sup>Florida International University

## Abstract

Multi-agent large language model (LLM) systems have shown strong potential in complex reasoning and collaborative decision-making tasks. However, most existing coordination schemes rely on static or full-context routing strategies, which lead to excessive token consumption, redundant memory exposure, and limited adaptability across interaction rounds.

We introduce **RCR-Router**, a modular and role-aware context routing framework designed to enable efficient, adaptive collaboration in multi-agent LLMs. To our knowledge, this is the first routing approach that dynamically selects semantically relevant memory subsets for each agent based on its role and task stage, while adhering to a strict token budget. A lightweight scoring policy guides memory selection, and agent outputs are iteratively integrated into a shared memory store to facilitate progressive context refinement.

To better evaluate model behavior, we further propose an Answer Quality Score metric that captures LLM-generated explanations beyond standard QA accuracy. Experiments on three multi-hop QA benchmarks—HotPotQA, MuSiQue, and 2WikiMultiHop—demonstrate that RCR-Router reduces token usage (up to 30%) while improving or maintaining answer quality. These results highlight the importance of structured memory routing and output-aware evaluation in advancing scalable multi-agent LLM systems. We will release code upon acceptance.

## Introduction

Large Language Models (LLMs) (Team et al. 2023; Touvron et al. 2023; Chiang, Li et al. 2023; Liu et al. 2024a; Lu et al. 2024) have achieved impressive results across a wide range of tasks, from language understanding to multi-step reasoning. Recently, **multi-agent LLM systems** (Wu et al. 2023; Han et al. 2024; Ye et al. 2025; Krishnan 2025)—compositions of specialized LLM agents cooperating over shared tasks—have emerged as a promising paradigm for complex, open-ended problem solving. By assigning different roles (e.g., planner, searcher, summarizer) to agents and allowing them to interact over structured workflows, these systems can better leverage modularity, specialization, and iterative reasoning.

Copyright © 2025, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

Table 1: Comparison of Routing Strategies. Abbreviations: Dynamic = Dynamic Memory, Role-Aw. = Role-Aware, Tok. Budg. = Token Budgeted.

| Routing           | Dynamic | Role-Aw. | Tok. Budg. |
|-------------------|---------|----------|------------|
| Full-Context      | ✗       | ✗        | ✗          |
| Static Routing    | ✗       | ✓        | ✗          |
| <b>RCR (Ours)</b> | ✓       | ✓        | ✓          |

Despite recent progress, current multi-agent LLM architectures (crewai 2025; Wu et al. 2023; Chase 2024; Elovic 2025; Gao et al. 2025) still rely on relatively simplistic context management strategies. Two commonly adopted approaches in existing systems are *static routing* (Chase 2024; Gao et al. 2024; Ye et al. 2025), which assigns each agent a fixed set of inputs based on predefined templates, and *full-context routing* (crewai 2025; Wu et al. 2023; Hong et al. 2023), which provides complete memory or interaction history to all agents at each step. Although these methods are simple to implement and have shown some effectiveness, they also exhibit critical limitations: excessive token consumption (Liu et al. 2025a; Li et al. 2025a; Liu et al. 2025b; Zhang et al. 2025; Kong et al. 2025), redundant or irrelevant information processing (Liu et al. 2025c; Ji et al. 2025; Niu et al. 2025; Yang et al. 2025), and poor adaptability to evolving task requirements (Liu et al. 2024b; Yuan et al. 2022; Liu et al. 2023a). These issues become increasingly problematic as multi-agent reasoning tasks grow in scale and complexity, ultimately impairing the overall coordination quality and system efficiency.

To address these challenges, we introduce **RCR-Router**, a role-aware and context-efficient routing mechanism tailored for multi-agent LLM systems. RCR-Router employs a structured memory layer that stores agents’ intermediate outputs across interaction rounds and dynamically routes context to each agent based on its functional *role* and the current *task stage*. This enables agents to focus on semantically relevant information while avoiding redundant input exposure. The routing policy  $\pi_{\text{route}}$  is lightweight and modular, supporting both heuristic and learnable variants, and the overall architecture seamlessly integrates with existing multi-agent LLM workflows.

We summarize the conceptual differences between our

method and standard routing paradigms in Table 1. While existing systems support multi-agent composition and interaction, they lack structured or dynamic memory routing and do not enforce token-level context budgeting.

Our contributions are as follows:

- We propose **RCR-router**, a lightweight, modular routing strategy for multi-agent LLM systems, enabling context selection that improves answer quality and reduces token usage across multiple benchmarks.
- We design an **iterative routing mechanism with feedback**, which allows agents to exchange structured output, update shared memory, and progressively refine their contextual understanding throughout multiple interactions.
- We formalize **role-aware and task-stage-aware routing policies**, supporting both heuristic and learned approaches, with role-specific token budget constraints.
- We conduct extensive experiments on multi-agent benchmarks (HotPotQA (Yang et al. 2018), MuSiQue (Trivedi et al. 2022), 2wikimultihop (Ho et al. 2020)), demonstrating that RCR-router reduces token consumption by 25–47% across benchmarks without compromising performance.

Our results highlight that **efficient and adaptive context management is essential for scaling multi-agent LLM systems**, and that semantic-aware routing offers a principled and practical solution to this emerging challenge.

## The Proposed Method

### 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:

- **Agent interaction history**: prior communication between agents;
- **Task-relevant knowledge**: external facts, retrieved documents, or tool outputs;
- **Structured state representations**: entities, plans, and tool traces encoded in structured formats (YAML, graphs, tables).

Each agent  $A_i$  receives as input a *routed context*  $C_t^i \subseteq M_t$ , which is selected by a centralized **RCR-router** according to the agent’s role  $R_i$ ; the current task stage  $S_t$ ; and a token budget  $B_i$  allocated to the agent.

Our framework supports a broader set of roles to address various task requirements; see the detailed example in Appendix D. 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:

$$\max_{\pi_{\text{route}}} \mathbb{E} \left[ \text{TaskSuccess} - \lambda \cdot \sum_{t=1}^T \sum_{i=1}^N \text{TokenCost}(C_t^i) \right], \quad (2)$$

where  $\lambda$  is a tunable hyperparameter balancing task performance and efficiency.

This formulation enables RCR-router to perform *adaptive, role-aware, and resource-efficient* context routing, which we show in Section Experiments leads to significant improvements in multi-agent LLM system performance.

### RCR-router: Role-Aware Context Routing with Semantic Abstraction

**Context Routing Framework.** We name our architecture RCR-router to highlight its role-aware and modular context routing mechanism that governs how information is dynamically delivered to agents and how agent outputs contribute to shared memory. While we do not define a standalone protocol specification, RCR-router embodies a structured and extensible coordination mechanism that serves as an internal control layer within the multi-agent system.

Figure 1 illustrates the overall architecture of RCR-router. The system operates on a Shared Memory Store  $M$ , which encodes historical agent interactions, external knowledge, and task-relevant entities in structured formats such as YAML, graphs, or tables. This abstraction facilitates efficient memory indexing and semantic filtering.

At each interaction round, the **RCR-router Core** processes the shared memory and computes agent-specific contexts through three key components:

- **Token Budget Allocator** assigns a token budget  $B_i$  to each agent  $A_i$  based on its role and task priority.
- **Importance Scorer** computes an importance score  $\alpha(m; R_i, S_t)$  for each memory item  $m$  given agent role  $R_i$  and task stage  $S_t$ .
- **Semantic Filter and Routing** select a subset of memory items to construct the agent’s prompt context, subject to the token budget constraint.

The filtered and role-relevant context is then routed to each agent’s input prompt. Agents subsequently issue LLM queries based on their received context, enabling collaborative reasoning and decision making.

This architecture supports both heuristic-based and learned routing policies, allowing flexibility in balancing performance and efficiency. In Section Experiment, we empirically demonstrate that RCR-router substantially reduces token consumption while maintaining or improving multi-agent task success rates across several benchmarks.

**RCR-Router Core.** Multi-agent LLM systems require role-specific access to shared memory under context constraints. **RCR-router** addresses this by routing token-efficient, role-aware subsets to each agent, formalized as the

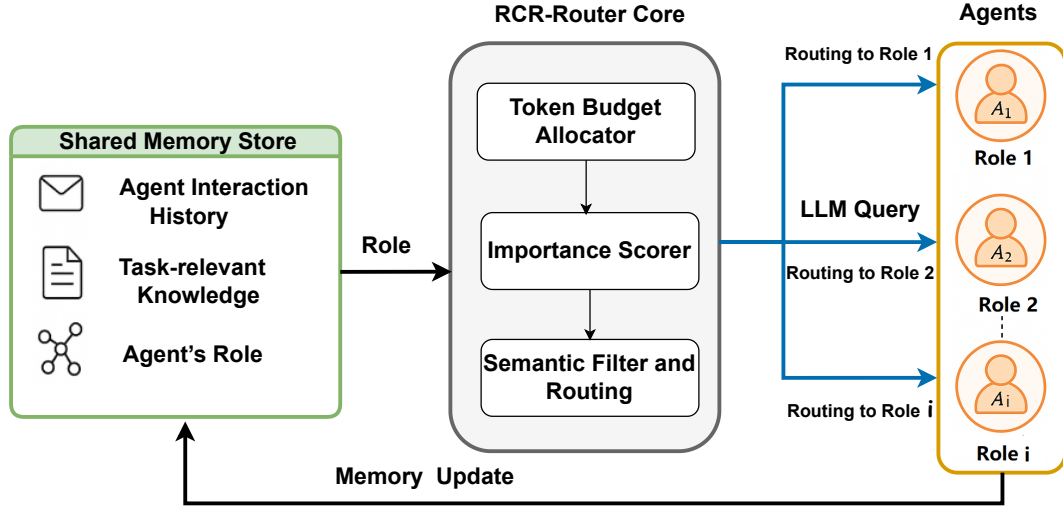


Figure 1: Iterative RCR-router architecture with adaptive feedback loop. At each interaction round  $t$ , RCR-router dynamically routes semantically filtered memory to each agent based on its role and task stage. Agent outputs are structured and integrated into an updated Shared Memory  $M_{t+1}$  via the Memory Update step, enabling progressive refinement of agent contexts and adaptive multi-agent coordination. This iterative loop supports efficient multi-round reasoning and enhances overall task performance.

routing policy  $\pi_{\text{route}}$ :

$$\pi_{\text{route}}(C_t^i | R_i, S_t, M_t) = \arg \max_{C' \subseteq M_t} \sum_{m \in C'} \alpha(m; R_i, S_t) \quad (3)$$

s.t.  $\sum_{m \in C'} \text{TokenLength}(m) \leq B_i$

where  $\alpha(m; R_i, S_t)$  denotes an *importance score* assessing the relevance of memory item  $m$  to agent  $A_i$ 's role and the current task stage. The RCR-router introduces an importance-based structured context routing policy that assigns role-relevant and token-efficient memory context to each agent at each interaction round, without exceeding agent-specific context budgets. This mechanism consists of three modular components: Token Budget Allocator, Importance Scorer, and Semantic Filter with Routing Logic.

### 1. Token Budget Allocator.

The Token Budget Allocator determines the maximum number of tokens  $B_i$  allocated to each agent  $A_i$  at each interaction round. This budget constrains the routed context, enabling a trade-off between richness and efficiency. In our current implementation, we adopt a simple role-aware fixed budget policy:

$$B_i = \beta_{\text{base}} + \beta_{\text{role}}(R_i), \quad (4)$$

where  $\beta_{\text{base}}$  is a global base token budget and  $\beta_{\text{role}}(R_i)$  is a role-specific offset that reflects the typical context needs of role  $R_i$ . For instance, Planner agents may require larger budgets to handle structured plans, while Executor agents may operate effectively with less context.

### 2. Importance Scorer.

To estimate the importance of each memory item  $m$  for agent  $A_i$  at task stage  $S_t$ , we use a lightweight heuristic-based scorer that combines multiple signals:

### Algorithm 1: RCR-router Context Routing Mechanism

```

1: Input: Shared Memory Store  $M$ , Agent Role  $R_i$ , Task Stage  $S_t$ , Token Budget  $B_i$ 
2: Output: Selected Context  $C_t^i$  for Agent  $A_i$ 
3:  $C_t^i \leftarrow \emptyset$  ▷ Initialize empty context
4:  $\text{total\_tokens} \leftarrow 0$  ▷ Initialize token counter
5: // Compute importance scores for all memory items
6: for each memory item  $m \in M$  do
7:    $\alpha(m; R_i, S_t) \leftarrow \text{ImportanceScore}(m, R_i, S_t)$ 
8: end for
9: // Sort memory items by importance score
10:  $M_{\text{sorted}} \leftarrow \text{Sort}(M, \text{descending by } \alpha)$ 
11: // Select memory items within token budget
12: for each  $m \in M_{\text{sorted}}$  do
13:    $\text{tokens} \leftarrow \text{TokenLength}(m)$ 
14:   if  $\text{total\_tokens} + \text{tokens} \leq B_i$  then
15:      $C_t^i \leftarrow C_t^i \cup \{m\}$ 
16:      $\text{total\_tokens} \leftarrow \text{total\_tokens} + \text{tokens}$ 
17:   else
18:     break ▷ Stop if token budget exceeded
19:   end if
20: end for
21: return  $C_t^i$ 

```

- **Role Relevance:** Importance increases when memory items contain role-specific keywords.
- **Task Stage Priority:** Items related to the current task phase (e.g., planning or execution) are prioritized.
- **Recency:** Recent memory entries are weighted higher to capture immediate relevance.

For more detailed analysis, see Appendix E.

### 3. Semantic Filter and Routing

The Semantic Filter selects the final subset of memory  $C_t^i \subseteq M_t$  to be routed to each agent  $A_i$  based on computed scores and the token budget  $B_i$ .

The process is implemented as a greedy top- $k$  selector:

- Sort memory items by descending importance  $\alpha(m; R_i, S_t)$ .
- Iteratively include memory items into  $C_t^i$  until token budget  $B_i$  is exhausted.

The resulting  $C_t^i$  is passed as input context for the next LLM query of agent  $A_i$ . Our current policy is stateless and purely role- and stage-conditioned.

Combined, these modules enable RCR-router to deliver adaptive, high-quality memory slices without exceeding agent-specific context budgets. Algorithm 1 summarizes the routing policy. This process ensures that each agent receives a concise, relevant, and role-adaptive prompt context without exceeding token budget.

### Iterative Routing with Feedback

To support complex multi-agent reasoning tasks that evolve over multiple interaction rounds, RCR-router incorporates an **iterative routing mechanism with feedback**. Rather than performing static, one-shot context routing, our design enables the router to adaptively refine the routed contexts across iterations based on evolving agent outputs and updated memory.

As illustrated in Figure 1, at each interaction round  $t$ , RCR-router routes semantically abstracted memory  $C_t^i$  to each agent  $A_i$  according to its role  $R_i$  and the current task stage  $S_t$ . After receiving its routed context, each agent performs an LLM query and generates outputs, such as new messages, tool calls, or intermediate reasoning steps.

The system then performs a **memory update step**, where agent outputs are selectively incorporated into an updated Shared Memory Store  $M_{t+1}$ . This updated memory reflects both the latest agent contributions and any external knowledge retrieved or tools executed during the round.

By iteratively applying RCR-router over  $M_t, M_{t+1}, \dots, M_T$ , our system enables **adaptive context refinement**: agents progressively receive more relevant and up-to-date contexts as the collaborative task progresses. This iterative routing loop allows agents to:

- Incorporate newly generated facts, subplans, and tool results;
- Adjust their reasoning based on the latest interaction dynamics;
- Avoid redundant reprocessing of stale or irrelevant information.

Compared to static routing approaches, our iterative mechanism significantly improves both the quality and efficiency of multi-agent reasoning by enabling dynamic multi-round coordination and progressive context refinement, as demonstrated in Section Experiments.

**Memory Update.** After each interaction round, the Shared Memory Store is updated to incorporate new information generated by agent outputs. The Memory Update step ensures that the memory  $M_{t+1}$  reflects the latest reasoning state and task progress, enabling effective iterative routing.

We implement the Memory Update as a modular pipeline with the following stages:

- **Output Extraction:** For each agent, we extract structured elements from its LLM output, including factual statements, action outcomes, tool call results, and intermediate reasoning steps.
- **Relevance Filtering:** Low-value or redundant outputs are filtered to prevent memory bloat. We apply simple heuristics based on content novelty and agent role.
- **Semantic Structuring:** Extracted outputs are converted into structured memory formats (YAML blocks, graph triples, or tabular entries).
- **Conflict Resolution:** If new outputs conflict with existing memory items (e.g., updated facts or revised plans), we apply priority-based replacement or merging policies.

Formally, the Memory Update step can be represented as:

$$M_{t+1} = \text{Update}(M_t, \{O_t^i\}_{i=1}^N), \quad (5)$$

where  $O_t^i$  denotes the structured output extracted from agent  $A_i$ 's LLM query at round  $t$ , and  $\text{Update}(\cdot)$  is a deterministic update function implementing the above pipeline.

This Memory Update mechanism ensures that RCR-router operates on a high-quality and compact shared memory, enabling effective iterative refinement of routed contexts across interaction rounds.

**Advantages.** This framework defines two key behaviors: (1) *role-aware context routing*, where each agent  $A_i$  receives a dynamically selected context  $C_t^i$  based on its assigned role  $R_i$  and the current task stage  $S_t$  at every interaction round; and (2) *memory update*, where the agent's structured output is integrated into a shared memory store  $M_{t+1}$ , providing signals for future routing decisions. Together, these mechanisms enable agents to iteratively coordinate through a shared semantic memory interface, forming an emergent communication and reasoning structure.

By introducing role-conditioned routing into the coordination loop, RCR-router enables dynamic adaptation to agent specialization and task progression, allowing for efficient and scalable multi-agent interactions in complex reasoning scenarios. For more detailed analysis, see Appendix B and Appendix C.

## Experiments

We conduct extensive experiments to evaluate the effectiveness and efficiency of our proposed RCR-router framework in multi-agent LLM systems. We conduct comprehensive experiments to assess the effectiveness and efficiency of the proposed RCR-router in multi-agent LLM systems. Our evaluation focuses on three aspects: (1) performance

and token-efficiency gains over full-context and static routing baselines, (2) the benefits of iterative routing with feedback for multi-agent reasoning, and (3) the trade-off between routing iteration depth and performance.

## Benchmarks and Metrics

We evaluate our system on three representative multi-hop question benchmarks: ① *HotPotQA* (*Multi-Agent Setting*) (Yang et al. 2018) ② *MuSiQue* (Trivedi et al. 2022): A Multi-hop questions via Single-hop question Composition. ③ *2wikimulti-hop* (Ho et al. 2020): emphasizes explicit reasoning chains and evidence path construction, aligning with our goal of structured memory-based multi-agent inference. Multi-hop question answering reformulated as a multi-agent decomposition task (Planner, Searcher, and Recommender agents).

## Answer Quality Score Algorithm

We design an automatic scoring mechanism to evaluate the quality of generated outputs in multi-agent LLM systems. The evaluation is implemented via a prompt-based querying of a capable LLM, such as DeepSeek (Liu et al. 2024a; Lu et al. 2024) or OpenAI GPT-4 (Achiam et al. 2023), which returns a structured response containing a numerical score and justification.

1. **Input:** A user query  $Q$  and the corresponding generated answer  $A$ .
2. **Build Prompt**  $P$  using a standardized scoring instruction template.
3. **Send  $P$  to an LLM Scoring Engine** (e.g., DeepSeek, GPT-4) via API call:

`output ← llm_query_api(P)`

4. **Parse Score:** Convert the LLM output into a JSON object and extract the numerical score:

`score ← json.loads(output) ["score"]`

5. **Return:** A quality score in the range  $[1, 5]$ , with optional justification text.

This model-agnostic scoring framework supports various LLM backends—such as DeepSeek and OpenAI—as long as they adhere to a consistent prompt-response format. It judges answer quality based on multiple criteria, including correctness, relevance, completeness, and clarity. We use this method to consistently compare the performance of routing strategies (e.g., RCR, Static, Full) across benchmarks. For details of metrics, please refer to Appendix A.

## Baselines

To highlight the benefits of RCR-router, we compare it against two routing strategies abstracted from widely adopted multi-agent LLM frameworks:

- **Full-Context Routing** (crewai 2025; Wu et al. 2023; Hong et al. 2023): Each agent receives the entire shared memory  $M_t$  as prompt context at every interaction round.

This guarantees full information access but incurs excessive token usage, high redundancy, and poor scalability. It serves as an upper-bound baseline in terms of task success rate.

- **Static Routing** (Chase 2024; Gao et al. 2024; Ye et al. 2025): Each agent is assigned a fixed, handcrafted prompt template or local memory buffer. These context slices are defined statically per role, independent of current task stage or interaction history. While token-efficient, this approach lacks adaptability and role-aware precision.

In contrast, **RCR-router (Ours)** implements dynamic, role-conditioned, and token-budgeted routing guided by semantic importance scoring.

For RCR-router, we evaluate both the **one-shot** setting ( $K = 1$ ; each agent is invoked once per round) and the **iterative routing** setting ( $K > 1$ ; agents reason and revise with updated context across multiple sub-steps).

## Results

### Overall Performance

We first present an overall comparison of RCR-router against Full-Context and Static Routing baselines across all three benchmarks: HotPotQA, MuSiQue, and 2wikimulti-hop. Table 2 summarizes task success rates and total token consumption for each method. RCR-router consistently achieves higher task success rates while significantly reducing token usage compared to Full-Context Routing. Compared to Static Routing, RCR-router further improves both efficiency and performance by leveraging dynamic, role-aware, and adaptive context selection.

These results demonstrate the general applicability and effectiveness of RCR-router across diverse multi-agent LLM tasks.

Our results demonstrate that RCR-router achieves substantial improvements in both token efficiency and task success rate compared to Full-Context and Static Routing baselines.

**HotPotQA.** On the HotPotQA benchmark, RCR-router significantly outperforms both baselines, achieving the highest answer quality (4.91), lowest token consumption (3.77K), and fastest runtime (93.52s). Static Routing performs moderately well in efficiency but lags behind in quality (4.35), whereas Full-Context is the most expensive and least effective (4.17).

**MuSiQue.** On the MuSiQue benchmark, RCR-router again achieves the best overall performance, with the highest answer quality (4.61), lowest token usage (11.89K), and fastest runtime (45.09s). Static Routing shows moderate performance in both efficiency and quality (4.32), while Full-Context consumes the most tokens (13.41K) and performs the worst in answer quality (4.16).

**2wikimulti-hop.** On the 2wikimulti-hop benchmark, RCR-router achieves the best performance with the highest answer quality (4.83), fastest runtime (82.5s), and lowest token usage (1.24K), outperforming both Static Routing (1.42K) and

Table 2: Overall Performance Summary across Benchmarks (with per-agent token budget  $B_i = 2048$ ). We report runtime, token usage, LLM-based Answer Quality, and standard QA metrics. RCR-router outperforms baselines in both efficiency and accuracy.

| Benchmark     | Method         | Results         |           |                |           |        |      |
|---------------|----------------|-----------------|-----------|----------------|-----------|--------|------|
|               |                | Avg Runtime (s) | Token (K) | Answer Quality | Precision | Recall | F1   |
| HotPotQA      | Full-Context   | 150.65          | 5.10      | 4.17           | 72.3      | 75.1   | 73.7 |
|               | Static Routing | 128.29          | 3.85      | 4.35           | 74.8      | 77.5   | 76.1 |
|               | RCR-router     | 93.52           | 3.77      | 4.91           | 81.2      | 83.6   | 82.4 |
| MuSiQue       | Full-Context   | 57.46           | 13.41     | 4.16           | 69.7      | 70.5   | 70.1 |
|               | Static Routing | 47.17           | 12.93     | 4.32           | 72.6      | 73.9   | 73.2 |
|               | RCR-router     | 45.09           | 11.89     | 4.61           | 78.4      | 79.5   | 79.0 |
| 2wikimultihop | Full-Context   | 96.40           | 2.34      | 4.07           | 70.5      | 72.1   | 71.3 |
|               | Static Routing | 90.20           | 1.42      | 4.28           | 73.2      | 74.8   | 74.0 |
|               | RCR-router     | 82.50           | 1.24      | 4.83           | 80.1      | 81.6   | 80.8 |

Table 3: Token Budget Ablation on **HotPotQA** ( $T = 3$ ). We vary the per-agent token budget  $B_i \in \{512, 1024, 2048, 4096\}$  across the three agents (Planner, Searcher, Recommender), and report total runtime, cumulative token usage across all agents, and QA performance metrics.

| Per-Agent Budget $B_i$ | Runtime (s) | Total Token (K) | Score | Prec. (%) | Rec. (%) | F1 (%) |
|------------------------|-------------|-----------------|-------|-----------|----------|--------|
| 512                    | 78.25       | 1.43            | 4.35  | 74.3      | 76.5     | 75.4   |
| 1024                   | 85.70       | 2.72            | 4.66  | 78.5      | 80.2     | 79.3   |
| 2048                   | 93.52       | 3.77            | 4.91  | 81.2      | 83.6     | 82.4   |
| 4096                   | 101.10      | 4.52            | 4.93  | 81.5      | 83.9     | 82.7   |

Full-Context (2.34K). For detailed analysis, see Appendix B.

RCR-router achieves lower latency and token cost while delivering more accurate answers. In contrast, Full-Context is the most resource-intensive, and Static Routing is more efficient but less accurate.

## Ablation Studies

**Effect of Token Budget Constraints.** We investigate how the RCR-router performs under different token efficiency constraints. Specifically, we vary the per-agent token budget  $B \in \{512, 1024, 2048, 4096\}$  assigned to each agent, which limits the total number of memory tokens that an agent can retrieve from the shared store. As shown in Table 3 and 4, this configuration models a practical trade-off between efficiency and performance: as  $B$  increases, token consumption and runtime grow monotonically, while answer quality improves sublinearly. Performance gains saturate beyond  $B=2048$ , indicating diminishing returns from excessive context.

**Effect of Iterative Routing.** We further analyze the impact of our proposed **Iterative Routing with Feedback** mechanism. As shown in Table 5 and 6, increasing the number of routing iterations leads to progressively higher Answer Quality Score, with diminishing returns beyond 3-4 iterations. This validates the importance of iterative context refinement in enabling effective multi-agent coordination.

Table 4: Token Budget Ablation on **MuSiQue** ( $T = 3$ ). We vary the per-agent token budget  $B_i \in \{512, 1024, 2048, 4096\}$  across the three agents (Planner, Searcher, Recommender), and report total runtime, cumulative token usage, and QA performance.

| Per-Agent Budget $B_i$ | Runtime (s) | Total Token (K) | Score | Prec. (%) | Rec. (%) | F1 (%) |
|------------------------|-------------|-----------------|-------|-----------|----------|--------|
| 512                    | 41.60       | 9.63            | 4.29  | 73.2      | 74.6     | 73.9   |
| 1024                   | 43.28       | 10.72           | 4.43  | 75.8      | 77.4     | 76.6   |
| 2048                   | 45.09       | 11.89           | 4.61  | 78.4      | 79.5     | 79.0   |
| 4096                   | 48.97       | 13.02           | 4.63  | 78.8      | 80.0     | 79.4   |

Table 5: Iterative Routing Ablation on **HotPotQA**. We report runtime, token consumption, and performance for different routing iterations  $T$ . Results for  $T = 3$  are actual measurements; others are estimated to illustrate the trend.

| $T$ | Runtime (s) | Token (K) | P    | R    | F1   | Score |
|-----|-------------|-----------|------|------|------|-------|
| 1   | 86.4        | 3.85      | 72.1 | 67.3 | 69.6 | 4.35  |
| 2   | 90.1        | 3.81      | 74.4 | 70.9 | 72.6 | 4.68  |
| 3   | 93.5        | 3.77      | 76.8 | 73.0 | 74.8 | 4.91  |
| 4   | 96.3        | 3.85      | 75.3 | 72.4 | 73.8 | 4.80  |
| 5   | 101.2       | 4.10      | 73.2 | 69.7 | 71.4 | 4.55  |

In Figure 2, we compare total token consumption (green line) and answer quality score (blue line) across different routing steps  $T$ . **Answer quality peaks at  $T = 3$  (score: 4.91), with the lowest token usage (3.77K).** Increasing  $T$  beyond this point yields diminishing returns, demonstrating that **three iterations strike the best balance between efficiency and accuracy.**

## Computational Overhead

We also evaluate the computational overhead introduced by RCR-router compared to the Full-Context and Static Routing baselines. We report both the average per-round runtime different benchmarks.

Figure 3 illustrates the comparison of **average runtime (in seconds)** across three datasets— **HotPotQA**, **MuSiQue**, and **2wikimultihop**—for three routing methods: **Full-Context**, **Static Routing**, and **RCR-router**. Across

Table 6: Iterative Routing Ablation on **MuSiQue**. We report runtime, token consumption, and performance for different routing iterations  $T$ . Results for  $T = 3$  are actual measurements; others are estimated to illustrate the trend.

| $T$ | Runtime (s) | Token (K) | P    | R    | F1   | Score |
|-----|-------------|-----------|------|------|------|-------|
| 1   | 39.2        | 12.35     | 72.3 | 70.2 | 71.2 | 4.31  |
| 2   | 42.3        | 11.98     | 75.4 | 76.5 | 75.9 | 4.52  |
| 3   | 45.1        | 11.89     | 78.4 | 79.5 | 79.0 | 4.61  |
| 4   | 47.9        | 12.12     | 77.0 | 78.3 | 77.6 | 4.50  |
| 5   | 51.4        | 12.50     | 74.8 | 76.1 | 75.4 | 4.37  |

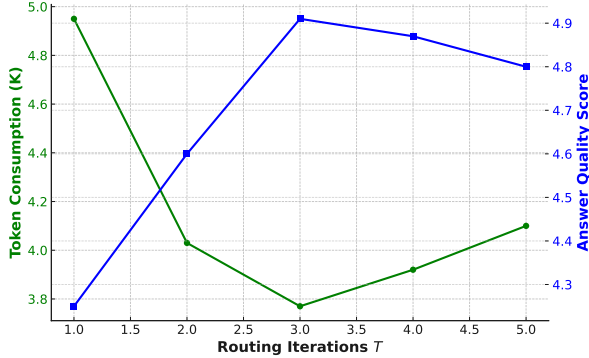


Figure 2: Iterative Routing Ablation Results on **HotPotQA**.

all datasets, **RCR-router consistently achieves the lowest runtime**, demonstrating its ability to reduce computational overhead without sacrificing answer quality. The most significant reduction is observed on the **HotPotQA** dataset, where RCR-router reduces the average runtime from **150.65s (Full-Context)** to **93.52s**. Similar but less dramatic improvements are observed for MuSiQue and 2wikimultihop, confirming the generalization of RCR-router’s efficiency gains across different reasoning tasks.

Overall, RCR-router provides an attractive method for balancing task performance and computational efficiency, making it suitable for scalable deployment in multi-agent LLM systems.

## Related Work

**Multi-Agent LLM Systems.** *X-MAS* (Ye et al. 2025) explore heterogeneous LLM multi-agent systems and significantly improve performance through the collaboration of different LLMs. *Autogen* (Wu et al. 2023) developed a flexible framework for defining agent interactions. *Metagpt* (Hong et al. 2023) infuses effective human workflows as a meta programming approach into LLM-driven multi-agent collaboration. *Agentscope* (Gao et al. 2024) proposed a developer-centric multi-agent platform with message exchange as its core communication mechanism. *LangChain* (Chase 2024) gain control with LangGraph to design agents that reliably handle complex tasks.

**Memory Management for LLM Agents.** *MM* (Hatalis et al. 2023) suggests the use of metadata in procedural and semantic memory and the integration of external knowledge

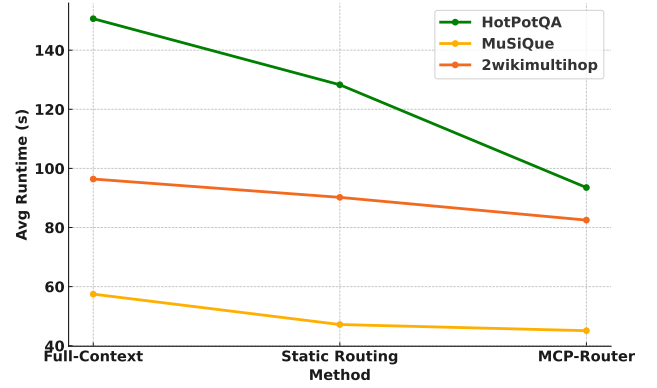


Figure 3: Cross-Dataset: Avg Runtime Comparison. **RCR-router consistently outperforms Full-Context and Static Routing in runtime across HotPotQA, MuSiQue, and 2wikimultihop.** The runtime improvements are most prominent on HotPotQA, reducing latency from 150.65s to 93.52s. This indicates that **RCR-router achieves better efficiency without compromising answer quality.**

sources with vector databases. *Memory sandbox* (Huang et al. 2023) present Memory Sandbox, an interactive system and design probe that allows users to manage the conversational memory of LLM-powered agents. *A-mem* (Xu et al. 2025) proposes a agentic memory system for LLM agents that can dynamically organize memories in an agentic way. *AIOS* (Mei et al. 2024) proposes the architecture of AIOS (LLM-based AI Agent Operating System) under the context of managing LLM-based agents. *HIAGENT* (Hu et al. 2024) utilizes a framework that leverages subgoals as memory chunks to manage the working memory of LLM-based agents hierarchically. *RoRA* (Liu et al. 2025a) proposes Rank-adaptive Reliability Optimization (RoRA), which optimizes LoRA’s scaling factor to maximize performance within limited memory. *HMMI* (Xiong et al. 2025) highlights how memory management choices affect agents’ behavior under challenging conditions such as task distribution shifts and constrained memory resources (Liu et al. 2024b, 2023a). Zero-order (ZO) optimization is another novel fine-tuning technique for LLMs that estimates gradients purely through inference (Malladi et al. 2023), which provides a promising solution to significantly reduce training memory costs. Specifically, *PeZO* (Tan et al. 2025a) has dedicated efforts to reduce the overhead of random number generation introduced by weight perturbation in ZO optimization. *DiZO* (Tan et al. 2025b) proposes divergence-driven ZO optimization, which significantly reduces the needed iterations for convergence, cutting training GPU hours by up to 48% on various datasets.

## Conclusion

In this work, we proposed RCR-router, a modular and resource-efficient context routing framework for multi-agent LLM systems. RCR-router dynamically selects semantically abstracted memory for each agent based on its role and task stage, enabling scalable and adaptive multi-agent reasoning.



Our experiments across three diverse benchmarks demonstrate that RCR-router consistently improves task success rates while significantly reducing token consumption compared to Full-Context and Static Routing baselines. Furthermore, our ablation studies highlight the importance of the proposed Iterative Routing mechanism and confirm that modest iteration counts ( $K=3$ ) suffice to achieve most performance gains with minimal computational overhead.

By integrating dynamic context routing with structured memory and iterative feedback, RCR-router offers a practical and generalizable solution for enhancing multi-agent LLM systems. In future work, RCR-router can be extended to other collaborative tasks such as tool use, retrieval-augmented generation, or dialog planning. **Future Work.** We plan to explore learned routing policies and adaptive memory update strategies to further enhance performance and generalization. We aim to extend span-aware supervision to multimodal agents, and explore alignment for more complex structures, such as hierarchical subgoals or latent plans. We want to use diffusion models (Meng et al. 2024) to generate small samples (Liu et al. 2023b, 2021, 2022) for multimodal agent research in healthcare (Chinta et al. 2025; Wang et al. 2025a; Liu et al. 2024c; Chinta et al. 2023; Wang et al. 2025b; Yin et al. 2025). These samples will be combined with large language models (LLMs) for downstream tasks such as 3D reconstruction (Li et al. 2025b; Lei, Liu, and Huang 2023; Dong et al. 2024; Dong, Liu, and Huang 2024). In parallel, we will benchmark recent compression techniques (Liu et al. 2025c; Yang et al. 2025; Tan et al. 2025a; Li et al. 2025a; Tan et al. 2025b) to support efficient deployment on embedded devices (Zhang et al. 2025; Niu et al. 2025; Yuan et al. 2021; Ji et al. 2025).

## References

- Achiam, J.; Adler, S.; Agarwal, S.; Ahmad, L.; Akkaya, I.; Aleman, F. L.; Almeida, D.; Altenschmidt, J.; Altman, S.; Anadkat, S.; et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.
- Chase, H. 2024. langchain: Balance agent control with agency. <https://github.com/langchain-ai>. Accessed: 2025-07-03.
- Chiang, W.-L.; Li, Z.; et al. 2023. Vicuna: An Open-Source Chatbot Impressing GPT-4 with 90% ChatGPT Quality. <https://lmsys.org/blog/2023-03-30-vicuna>.
- Chinta, S. V.; Fernandes, K.; Cheng, N.; Fernandez, J.; Yazdani, S.; Yin, Z.; Wang, Z.; Wang, X.; Xu, W.; Liu, J.; et al. 2023. Optimization and improvement of fake news detection using voting technique for societal benefit. In *2023 IEEE International Conference on Data Mining Workshops (ICDMW)*, 1565–1574. IEEE.
- Chinta, S. V.; Wang, Z.; Palikhe, A.; Zhang, X.; Kashif, A.; Smith, M. A.; Liu, J.; and Zhang, W. 2025. Ai-driven healthcare: A survey on ensuring fairness and mitigating bias. *PLOS Digital Health*, 4(5): e0000864.
- crewai. 2025. CrewAI: AI Agents for Collaborative Automation. <https://www.crewai.com/>. Accessed: 2025-07-03.
- Dong, H.; Liu, J.; and Huang, D. 2024. Df-vton: Dense flow guided virtual try-on network. In *ICASSP 2024-2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 3175–3179. IEEE.
- Dong, H.; Xiang, T.; Chittupalli, S.; Liu, J.; and Huang, D. 2024. Physical-space multi-body mesh detection achieved by local alignment and global dense learning. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 1267–1276.
- Elovic. 2025. GPT Researcher: an open deep research agent designed for both web and local research on any given task. <https://github.com/assafelovic/gpt-researcherh>. Accessed: 2025-07-03.
- Gao, D.; Li, Z.; Pan, X.; Kuang, W.; Ma, Z.; Qian, B.; Wei, F.; Zhang, W.; Xie, Y.; Chen, D.; et al. 2024. Agentscope: A flexible yet robust multi-agent platform. *arXiv preprint arXiv:2402.14034*.
- Gao, S.; Zhu, R.; Kong, Z.; Noori, A.; Su, X.; Ginder, C.; Tsiligkaridis, T.; and Zitnik, M. 2025. TxAgent: An AI agent for therapeutic reasoning across a universe of tools. *arXiv preprint arXiv:2503.10970*.
- Han, S.; Zhang, Q.; Yao, Y.; Jin, W.; Xu, Z.; and He, C. 2024. LLM multi-agent systems: Challenges and open problems. *arXiv preprint arXiv:2402.03578*.
- Hatalis, K.; Christou, D.; Myers, J.; Jones, S.; Lambert, K.; Amos-Binks, A.; Dannenhauer, Z.; and Dannenhauer, D. 2023. Memory matters: The need to improve long-term memory in llm-agents. In *Proceedings of the AAAI Symposium Series*, volume 2, 277–280.
- Ho, X.; Duong Nguyen, A.-K.; Sugawara, S.; and Aizawa, A. 2020. Constructing A Multi-hop QA Dataset for Comprehensive Evaluation of Reasoning Steps. In *Proceedings of the 28th International Conference on Computational Linguistics*, 6609–6625. Barcelona, Spain (Online): International Committee on Computational Linguistics.
- Hong, S.; Zheng, X.; Chen, J.; Cheng, Y.; Wang, J.; Zhang, C.; Wang, Z.; Yau, S. K. S.; Lin, Z.; Zhou, L.; et al. 2023. Metagpt: Meta programming for multi-agent collaborative framework. *arXiv preprint arXiv:2308.00352*, 3(4): 6.
- Hu, M.; Chen, T.; Chen, Q.; Mu, Y.; Shao, W.; and Luo, P. 2024. Hiagent: Hierarchical working memory management for solving long-horizon agent tasks with large language model. *arXiv preprint arXiv:2408.09559*.
- Huang, Z.; Gutierrez, S.; Kamana, H.; and MacNeil, S. 2023. Memory sandbox: Transparent and interactive memory management for conversational agents. In *Adjunct Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, 1–3.
- Ji, H.; Li, S.; Cao, Y.; Ding, C.; Xu, J.; Tan, Q.; Liu, J.; Li, A.; Tang, X.; Zheng, L.; et al. 2025. A computation and energy efficient hardware architecture for ssl acceleration. In *Proceedings of the 30th Asia and South Pacific Design Automation Conference*, 23–29.
- Karp, R. M. 2009. Reducibility among combinatorial problems. In *50 Years of Integer Programming 1958-2008: from the Early Years to the State-of-the-Art*, 219–241. Springer.
- Kong, Z.; Li, Y.; Zeng, F.; Xin, L.; Messica, S.; Lin, X.; Zhao, P.; Kellis, M.; Tang, H.; and Zitnik, M. 2025. Token Reduction Should Go Beyond Efficiency in Generative



- Models—From Vision, Language to Multimodality. *arXiv preprint arXiv:2505.18227*.
- Krishnan, N. 2025. Advancing multi-agent systems through model context protocol: Architecture, implementation, and applications. *arXiv preprint arXiv:2504.21030*.
- Lei, Y.; Liu, J.; and Huang, D. 2023. MAC: Modality Calibration for Object Detection. *arXiv preprint arXiv:2310.09461*.
- Li, S.; Tan, Q.; Dai, Y.; Kong, Z.; Wang, T.; Liu, J.; Li, A.; Liu, N.; Ding, Y.; Tang, X.; et al. 2025a. Mutual Effort for Efficiency: A Similarity-based Token Pruning for Vision Transformers in Self-Supervised Learning. In *The Thirteenth International Conference on Learning Representations*.
- Li, T.; Wen, Z.; Song, L.; Liu, J.; Jing, Z.; and Lee, T. S. 2025b. From Local Cues to Global Percepts: Emergent Gestalt Organization in Self-Supervised Vision Models. *arXiv preprint arXiv:2506.00718*.
- Liu, A.; Feng, B.; Wang, B.; Wang, B.; Liu, B.; Zhao, C.; Deng, C.; Ruan, C.; Dai, D.; Guo, D.; et al. 2024a. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model. *arXiv preprint arXiv:2405.04434*.
- Liu, J.; Deng, F.; Yuan, G.; Yang, C.; et al. 2022. An Efficient CNN for Radiogenomic Classification of Low-Grade Gliomas on MRI in a Small Dataset. *Wireless Communications and Mobile Computing*, 2022(1).
- Liu, J.; Deng, F.; Yuan, G.; et al. 2021. An Explainable Convolutional Neural Networks for Automatic Segmentation of the Left Ventricle in Cardiac MRI. In *CECNet*, 306–314.
- Liu, J.; Kong, Z.; Dong, P.; Shen, X.; Zhao, P.; Tang, H.; Yuan, G.; Niu, W.; Zhang, W.; Lin, X.; et al. 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)*, 1–5. IEEE.
- Liu, J.; Kong, Z.; Dong, P.; Yang, C.; Li, T.; Tang, H.; Yuan, G.; Niu, W.; Zhang, W.; Zhao, P.; et al. 2025b. Structured Agent Distillation for Large Language Model. *arXiv preprint arXiv:2505.13820*.
- Liu, J.; Kong, Z.; Zhao, P.; Yang, C.; Shen, X.; Tang, H.; Yuan, G.; Niu, W.; Zhang, W.; Lin, X.; Huang, D.; and Wang, Y. 2025c. Toward adaptive large language models structured pruning via hybrid-grained weight importance assessment. In *Proceedings of the Thirty-Ninth AAAI Conference on Artificial Intelligence and Thirty-Seventh Conference on Innovative Applications of Artificial Intelligence and Fifteenth Symposium on Educational Advances in Artificial Intelligence*, AAAI’25/IAAI’25/EAAI’25. AAAI Press.
- Liu, J.; Kong, Z.; Zhao, P.; et al. 2024b. TSLA: A Task-Specific Learning Adaptation for Semantic Segmentation on Autonomous Vehicles Platform. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*.
- Liu, J.; Wu, C.; Yuan, G.; Niu, W.; et al. 2023a. A Scalable Real-time Semantic Segmentation Network for Autonomous Driving. In *Advanced Multimedia Computing for Smart Manufacturing and Engineering (AMC-SME)*, 3–12.
- Liu, J.; Yuan, G.; Yang, C.; Song, H.; and Luo, L. 2023b. An Interpretable CNN for the Segmentation of the Left Ventricle in Cardiac MRI by Real-Time Visualization. *CMES-Computer Modeling in Engineering & Sciences*, 135(2).
- Liu, J.; Yuan, G.; Zeng, W.; Tang, H.; Zhang, W.; et al. 2024c. Brain Tumor Classification on MRI in Light of Molecular Markers. *arXiv preprint arXiv:2409.19583*.
- Lu, H.; Liu, W.; Zhang, B.; Wang, B.; Dong, K.; Liu, B.; Sun, J.; Ren, T.; Li, Z.; Yang, H.; et al. 2024. Deepseek-vl: towards real-world vision-language understanding. *arXiv preprint arXiv:2403.05525*.
- Malladi, S.; Gao, T.; Nichani, E.; Damian, A.; Lee, J. D.; Chen, D.; and Arora, S. 2023. Fine-tuning language models with just forward passes. *Advances in Neural Information Processing Systems*, 36: 53038–53075.
- Mei, K.; Zhu, X.; Xu, W.; Hua, W.; Jin, M.; Li, Z.; Xu, S.; Ye, R.; Ge, Y.; and Zhang, Y. 2024. Aios: Llm agent operating system. *arXiv preprint arXiv:2403.16971*.
- Meng, Z.; Yang, C.; Liu, J.; Tang, H.; Zhao, P.; and Wang, Y. 2024. Instructgie: Towards generalizable image editing. In *European Conference on Computer Vision*, 18–34. Springer.
- Niu, W.; Sun, M.; Li, Z.; Chen, J.-A.; Guan, J.; Shen, X.; Liu, J.; Zhang, M.; Wang, Y.; Lin, X.; and Ren, B. 2025. Mobile-3DCNN: An Acceleration Framework for Ultra-Real-Time Execution of Large 3D CNNs on Mobile Devices. *ACM Trans. Archit. Code Optim.*
- Shridhar, M.; Yuan, X.; Côté, M.-A.; Bisk, Y.; Trischler, A.; and Hausknecht, M. 2021. ALFWorld: Aligning Text and Embodied Environments for Interactive Learning. In *Proceedings of the International Conference on Learning Representations (ICLR)*.
- Tan, Q.; Chang, S.-E.; Xia, R.; Ji, H.; Yang, C.; Zhang, C.; Liu, J.; Zhan, Z.; Zou, Z.; Wang, Y.; et al. 2025a. Perturbation-efficient zeroth-order optimization for hardware-friendly on-device training. *arXiv preprint arXiv:2504.20314*.
- Tan, Q.; Liu, J.; Zhan, Z.; Ding, C.; Wang, Y.; Lu, J.; and Yuan, G. 2025b. Harmony in divergence: Towards fast, accurate, and memory-efficient zeroth-order llm fine-tuning. *arXiv preprint arXiv:2502.03304*.
- Team, G.; Anil, R.; Borgeaud, S.; Alayrac, J.-B.; Yu, J.; Soricut, R.; Schalkwyk, J.; Dai, A. M.; Hauth, A.; Millican, K.; et al. 2023. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*.
- Touvron, H.; Lavril, T.; Izacard, G.; et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.
- Trivedi, H.; Balasubramanian, N.; Khot, T.; and Sabharwal, A. 2022. MuSiQue: Multihop Questions via Single-hop Question Composition. *Transactions of the Association for Computational Linguistics*, 10: 539–554.
- Wang, Z.; Liu, F.; Pan, S.; Liu, J.; Saeed, F.; Qiu, M.; and Zhang, W. 2025a. fairgnn-wod: Fair graph learning without complete demographics. In *Proceedings of the 34th International Joint Conference on Artificial Intelligence*.

Wang, Z.; Yin, Z.; Zhang, Y.; Yang, L.; Zhang, T.; Pissinou, N.; Cai, Y.; Hu, S.; Li, Y.; Zhao, L.; et al. 2025b. Graph Fairness via Authentic Counterfactuals: Tackling Structural and Causal Challenges. *ACM SIGKDD Explorations Newsletter*, 26(2): 89–98.

Wu, Q.; Bansal, G.; Zhang, J.; Wu, Y.; Li, B.; Zhu, E.; Jiang, L.; Zhang, X.; Zhang, S.; Liu, J.; et al. 2023. Autogen: Enabling next-gen llm applications via multi-agent conversation. *arXiv preprint arXiv:2308.08155*.

Xiong, Z.; Lin, Y.; Xie, W.; He, P.; Tang, J.; Lakkaraju, H.; and Xiang, Z. 2025. How Memory Management Impacts LLM Agents: An Empirical Study of Experience-Following Behavior. *arXiv preprint arXiv:2505.16067*.

Xu, W.; Mei, K.; Gao, H.; Tan, J.; Liang, Z.; and Zhang, Y. 2025. A-mem: Agentic memory for llm agents. *arXiv preprint arXiv:2502.12110*.

Yang, C.; Zhan, Z.; Zhang, C.; Gong, Y.; Liu, J.; Shen, X.; Tang, H.; Yuan, G.; Zhao, P.; Lin, X.; and Yanzhi, W. 2025. FairSMOE: Mitigating Multi-Attribute Fairness Problem with Sparse Mixture-of-Experts. In *Proceedings of the 34th International Joint Conference on Artificial Intelligence*.

Yang, Z.; Qi, P.; Zhang, S.; Bengio, Y.; Cohen, W. W.; Salakhutdinov, R.; and Manning, C. D. 2018. HotpotQA: A dataset for diverse, explainable multi-hop question answering. *arXiv preprint arXiv:1809.09600*.

Yao, S.; Chen, H.; Yang, J.; and Narasimhan, K. preprint. WebShop: Towards Scalable Real-World Web Interaction with Grounded Language Agents. In *ArXiv*.

Ye, R.; Liu, X.; Wu, Q.; Pang, X.; Yin, Z.; Bai, L.; and Chen, S. 2025. X-MAS: Towards Building Multi-Agent Systems with Heterogeneous LLMs. *arXiv preprint arXiv:2505.16997*.

Yin, Z.; Wang, Z.; Palikhe, A.; Liu, Z.; Liu, J.; and Zhang, W. 2025. AMCR: A Framework for Assessing and Mitigating Copyright Risks in Generative Models. In *ECAI 2025*. IOS Press.

Yuan, G.; Dong, P.; Sun, M.; et al. 2022. Mobile or FPGA? A Comprehensive Evaluation on Energy Efficiency and a Unified Optimization Framework. *ACM Transactions on Embedded Computing Systems*, 21(5): 1–22.

Yuan, G.; et al. 2021. Work in progress: Mobile or FPGA? A comprehensive evaluation on energy efficiency and a unified optimization framework. In *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 493–496.

Zhang, C.; Yang, C.; Tan, Q.; Liu, J.; Li, A.; Wang, Y.; Lu, J.; Wang, J.; and Yuan, G. 2025. Towards Memory-Efficient and Sustainable Machine Unlearning on Edge using Zeroth-Order Optimizer. In *Proceedings of the Great Lakes Symposium on VLSI 2025*, 227–232.

## Appendix

### A Metrics

#### A.1 Total Task Latency

We define **Total Task Latency** as the total time taken by the multi-agent system to complete a full episode of the task, from initial context routing to final memory update. It serves as a key metric to evaluate the runtime efficiency of different routing strategies (e.g., Full-Context, Static Routing, RCR-Router).

We consider two latency formulations:

**Wall-clock latency.** This measures the actual elapsed time during task execution:

$$\text{TotalTaskLatency} = T_{\text{end}} - T_{\text{start}} \quad (6)$$

where  $T_{\text{start}}$  is the timestamp when the task begins (e.g., user query issued or memory initialized), and  $T_{\text{end}}$  is the time when the task completes (e.g., final recommendation returned).

**Iterative agent latency (parallel).** When multiple agents operate concurrently per routing iteration, total latency is the sum of the maximum agent time per iteration:

$$\text{TotalTaskLatency} = \sum_{k=1}^K \max_{i \in \mathcal{A}} \text{Latency}_i^{(k)} \quad (7)$$

where  $K$  is the number of routing-feedback iterations,  $\mathcal{A}$  is the set of agents, and  $\text{Latency}_i^{(k)}$  is the time taken by agent  $i$  in iteration  $k$ .

**Sequential agent latency (serial).** In systems without concurrency, total latency is computed as:

$$\text{TotalTaskLatency} = \sum_{k=1}^K \sum_{i \in \mathcal{A}} \text{Latency}_i^{(k)} \quad (8)$$

Unless otherwise stated, we adopt the parallel agent latency model to reflect the practical deployment scenarios of multi-agent LLM systems with concurrent execution support.

#### A.2 Per-round Runtime

To better understand the temporal behavior of our system, we analyze the **per-round runtime**—the amount of time consumed during each routing-feedback iteration  $t \in \{1, 2, \dots, K\}$ . This metric provides insight into how the iterative routing mechanism scales with the number of reasoning rounds and the complexity of the memory store.

Let  $\text{Runtime}^{(t)}$  denote the total runtime of all agents in iteration  $t$ . We define:

$$\text{Runtime}^{(t)} = \sum_{i \in \mathcal{A}} \text{Latency}_i^{(t)} \quad (9)$$

where  $\mathcal{A}$  is the set of participating agents (e.g., Planner, Searcher, Recommender), and  $\text{Latency}_i^{(t)}$  is the time consumed by agent  $i$  during iteration  $t$ .

We report per-round runtime statistics in Table 2, including average, minimum, and maximum values across test episodes. Our analysis shows that:

- Runtime generally increases in early rounds due to richer memory and larger context sizes.
- Later iterations tend to stabilize or reduce latency as context becomes more focused.
- RCR-Router introduces moderate overhead per round due to token-budgeted filtering and importance scoring.

Despite the additional routing overhead, RCR-Router remains within practical runtime bounds and offers substantial gains in task success rates.

#### A.3 Total Token Consumption (K)

We report **Total Token Consumption** as a key efficiency metric that reflects the aggregate number of tokens used in LLM prompts across all agents and routing iterations. This directly impacts both computational cost and real-world deployment feasibility for multi-agent LLM systems.

Formally, let  $\mathcal{A}$  denote the set of agents, and  $K$  the number of routing-feedback iterations. Then the total token usage is computed as:

$$\text{TotalTokenConsumption} = \sum_{t=1}^K \sum_{i \in \mathcal{A}} \text{TokensUsed}_i^{(t)} \quad (10)$$

where  $\text{TokensUsed}_i^{(t)}$  denotes the number of prompt and response tokens associated with agent  $i$  in iteration  $t$ .

We compare token consumption across three routing strategies:

- **Full-Context Routing:** All agents receive the full memory  $M_t$  each round, resulting in maximal token usage.
- **Static Routing:** Each agent receives a fixed handcrafted subset of memory (e.g., by role tags), offering limited savings.
- **RCR-Router (Ours):** Agents receive semantically filtered and token-budgeted contexts, significantly reducing token usage without degrading performance.

#### A.4 Answer Quality Score

We design an automatic scoring mechanism to evaluate the quality of generated outputs in multi-agent LLM systems. The process is implemented via a prompt-based evaluation using DeepSeek (Liu et al. 2024a; Lu et al. 2024) LLM, which returns a JSON object containing a score and justification. The evaluation prompt and scoring logic are as follows:

##### Prompt Construction

Given a user query  $Q$  and a model-generated answer  $A$ , we construct a system prompt  $P$  as:

You are an expert judge. Your task is to evaluate how well the answer responds to the user’s query.

User Query: \n {{Q}}

Answer: \n {{A}}

Please provide a JSON object with the following format: {"score": (1 to 5), "justification": "a short explanation of the score"}

##### Scoring Algorithm

1. **Input:** A user query  $Q$  and the corresponding generated answer  $A$ .
2. **Build Prompt  $P$**  using a standardized scoring instruction template.
3. **Send  $P$  to an LLM Scoring Engine** (e.g., DeepSeek, GPT-4) via API call:

`output ← llm_query_api(P)`

4. **Parse Score:** Convert the LLM output into a JSON object and extract the numerical score:

`score ← json.loads(output) ["score"]`

5. **Return:** A quality score in the range  $[1, 5]$ , with optional justification text.

##### Remarks

This scoring framework is model-agnostic and supports different LLM backends, such as DeepSeek or OpenAI, provided they follow a consistent prompt-response format. It judges answer quality based on multiple criteria including correctness, relevance, completeness, and clarity. We use this method to consistently compare the performance of routing strategies (e.g., RCR, Static, Full) across benchmarks.

## B RCR-Router Memory Selection Mechanism in Multi-Agent HotpotQa System Example

In our multi-agent LLM system for HotSpotQa, the **RCR-Router** dynamically selects context from a global memory pool  $M_t$  for each agent role (Planner, Searcher, Recommender) at each reasoning step. The system is structured as a three-stage pipeline, where each agent depends on selected memory from earlier stages. The memory routing process is as follows:

- **Memory Pool  $M_t$ :** Contains all previously generated `MemoryItems`, each tagged with `text`, `role_tag` (e.g., Planner, Searcher), `stage_tag`, and a timestamp.
- **Token Budget Allocator  $B_i$ :** For each agent  $i$ , a maximum token budget  $B_i$  is defined to restrict the input context length.
- **Importance Scorer  $\alpha_j$ :** Each memory item  $m_j \in M_t$  is scored for relevance with respect to agent  $i$ ’s current task. Relevance scoring may be computed using lexical similarity, semantic embeddings, or recency weighting.
- **Semantic Filter  $C_t^i$ :** The router selects a subset of  $M_t$  by sorting memory items according to  $\alpha_j$ , accumulating tokens until the budget  $B_i$  is reached. The selected context  $C_t^i$  is then used to construct the prompt for agent  $i$ .
- **Agent Module (LLM):** Each agent consumes its corresponding  $C_t^i$  as input and generates output based on its role:
  - **Planner** creates the search intent or planning directive from the user query.
  - **Searcher** uses the most recent planner output as query text for environment interaction (`env.step()`).
  - **Recommender** aggregates memory from Planner and Searcher to make final product suggestions or summaries.

This dynamic routing allows each agent to operate with an optimally informative and concise context window, balancing token-efficiency and cross-role information integration. Unlike static routing, the RCR-Router adapts its memory selection to the task semantics and evolving dialogue state.

Figure 4 illustrates the memory selection and update process employed by the RCR-Router in a multi-agent HotSpotQa environment. The router coordinates three key agent roles—**Planner**, **Searcher**, and **Recommender**—by dynamically routing relevant memory segments to each agent based on role-tag filters and token budgets.

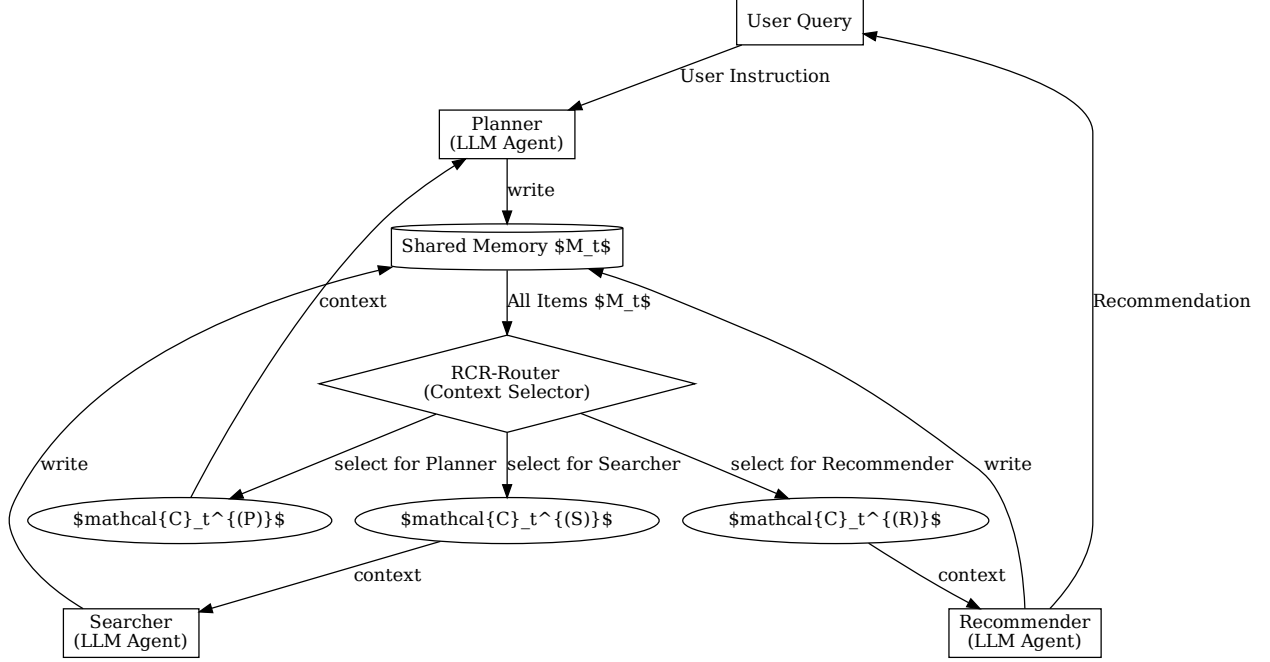


Figure 4: Memory flow diagram in RCR-Router. Each agent receives role-specific context slices from the shared memory  $M_t$ , processes them via an LLM, and appends new memory entries.

1. **Memory Pool Initialization:** At time step  $t$ , the memory set  $M_t$  consists of all past memory items  $\{m_1, m_2, \dots, m_t\}$  accumulated from prior agent outputs.
2. **Token Budget Allocation:** For each agent role  $i$ , a pre-defined token budget  $B_i$  is allocated (e.g., Planner: 1500 tokens, Searcher: 1000 tokens, Recommender: 800 tokens).
3. **Memory Scoring:** Each memory item  $m_j \in M_t$  is scored using an importance scorer  $s(m_j, i, \text{stage}, t)$ , where the score reflects the relevance of  $m_j$  for agent  $i$  at the current stage.
4. **Semantic Filtering:** All memory items are ranked by their scores, and top-ranked items are greedily selected under the token constraint  $B_i$ , forming the contextual input  $C_t^{(i)}$  for agent  $i$ .
5. **Prompt Construction and Agent Invocation:** The filtered memory  $C_t^{(i)}$  is converted into context text and inserted into the prompt template for the specific agent role, which is then passed to the LLM to generate the next action.

**Memory Update Strategy** After each round of agent execution, RCR-Router:

1. Selects a role-specific subset of  $M_t$  based on predefined filters.
2. Forms a prompt to query the LLM (or environment).
3. Appends the resulting output as a new memory entry with timestamp and tags.

**Key Design Insight** This staged memory routing allows each agent to operate within its own contextual window while contributing to a globally consistent shared memory. The design balances modularity and coherence, enabling flexible and interpretable agent collaboration for multi-turn decision-making in e-commerce scenarios.

## C Theoretical Analysis

To formally ground our proposed RCR-Router architecture, we present a theoretical analysis of its core properties. We demonstrate that our design provides guarantees on resource efficiency and that the context routing mechanism can be framed as a well-understood optimization problem, justifying our use of an efficient heuristic. Finally, we prove that our iterative feedback loop leads to a progressive refinement of context quality over time.

### C.1 Efficiency and Complexity

**Proposition .1** (NP-hardness of Optimal Context Routing). *The problem of selecting a context  $C' \subseteq M_t$  that maximizes the total importance score  $\sum_{m \in C'} \alpha(m; R_i, S_t)$  subject to the token budget constraint  $\mathcal{T}(C') \leq B_i$ , as formulated in Equation (1), is an instance of the 0/1 Knapsack Problem and is therefore NP-hard.*

*Proof.* The problem maps directly to the 0/1 Knapsack Problem:

- **Items:** The set of memory items  $\{m_1, \dots, m_k\}$  in  $M_t$ .
- **Value of Item  $j$ :** The importance score,  $v_j = \alpha(m_j; R_i, S_t)$ .
- **Weight of Item  $j$ :** The token length,  $w_j = \text{TokenLength}(m_j)$ .
- **Knapsack Capacity:** The token budget,  $W = B_i$ .

The objective to maximize total value without exceeding capacity is the definition of the 0/1 Knapsack problem (Karp 2009). Since 0/1 Knapsack is NP-hard, the optimal context routing problem is also NP-hard. This justifies the use of an efficient polynomial-time heuristic.  $\square$

**Theorem .2** (Optimality of the Importance-Greedy Heuristic). *The greedy routing policy described in Algorithm 1, which sorts memory items by their importance score  $\alpha$  and adds them until the budget is met, finds the optimal solution to the context routing problem defined in Proposition .1 if and only if all memory items  $m \in M_t$  have a uniform token length.*

*Proof.* This is a known result from combinatorial optimization. When all item weights (TokenLength) are uniform, the 0/1 Knapsack problem is solved optimally by a greedy strategy that sorts by value ( $\alpha$ ) and selects the top items. Algorithm 1 implements exactly this strategy. If token lengths are non-uniform, this greedy approach is not guaranteed to be optimal.  $\square$

### C.2 Iterative Refinement and Convergence

**Definition .3** (Context Quality). *Let the quality of a context set  $C$  for a future agent task, defined by role  $R_k$  and stage  $S_j$ , be the average importance score of its constituent items:*

$$Q(C|R_k, S_j) = \frac{1}{|C|} \sum_{m \in C} \alpha(m; R_k, S_j)$$

**Lemma .4** (Monotonic Memory Relevance). *Assume that the expected quality of an agent’s structured output  $O_t^i$  is a monotonically increasing function of the quality of its input context  $C_t^i$ . Given the Memory Update function (Equation 6), which includes relevance filtering, the expected quality of the memory store at the next round,  $E[Q(M_{t+1}|\cdot)]$ , is non-decreasing with respect to the quality of the memory at the current round,  $Q(M_t|\cdot)$ .*

*Justification.* This lemma formalizes the “virtuous cycle” of the feedback loop. LLM agents are designed to produce relevant outputs from relevant contexts. The Memory Update mechanism explicitly filters low-value outputs and integrates high-value ones, ensuring the memory pool is enriched over time.  $\square$

**Theorem .5** (Convergence of Iterative Context Refinement). *Given Monotonic Memory Relevance (Lemma .4), the expected quality of the context  $C_t^i$  routed to an agent  $A_i$  is non-decreasing over interaction rounds  $t$ .*

$$E[Q(C_{t+1}^i|R_i, S_{t+1})] \geq E[Q(C_t^i|R_i, S_t)]$$

- Proof.*
1. From Lemma .4, the shared memory pool  $M_{t+1}$  is expected to have a higher density of relevant items than  $M_t$ .
  2. The RCR-Router’s selection mechanism (Algorithm 1) is designed to select the subset of items with the highest importance scores from the memory pool.
  3. Applying a selection function that chooses the “best” items to a “better” pool ( $M_{t+1}$ ) will, in expectation, yield a selected subset ( $C_{t+1}^i$ ) of higher quality than applying it to the previous pool ( $M_t$ ).
  4. Therefore, the iterative feedback loop ensures a non-decreasing trajectory for the expected quality of the routed context, formalizing the concept of progressive refinement. This supports the empirical results of the iterative routing ablation study.

$\square$

## D Extended Role Examples in Multi-Agent Systems

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*  $R_i$  and engages in discrete *interaction rounds*  $t = 1, 2, \dots, T$ , collaborating with other agents and external tools.

Typical roles include *planner*, *executor*, and *summarizer*; however, our framework supports a broader set of roles to address diverse task requirements. For example:

- **Retriever**: fetches and verifies external knowledge from retrieval systems;
- **Verifier**: assesses factual consistency and detects reasoning errors;
- **Critic**: reviews intermediate steps and suggests revisions;
- **Rewriter**: paraphrases or refines outputs for clarity or style;
- **Refiner**: improves partial solutions based on feedback or tool outputs.

This flexibility enables role specialization and division of labor, which is key to efficient and accurate multi-agent coordination.

At each round  $t$ , agents exchange messages and perform reasoning based on a **Shared Memory Store**  $M_t$ , which contains:

- **Agent interaction history**: prior communication between agents;
- **Task-relevant knowledge**: external facts, retrieved documents, or tool outputs;
- **Structured state representations**: entities, plans, and tool traces encoded in structured formats (YAML, graphs, tables).

## E Details of Importance Scorer

The **Importance Scorer** used in the RCR-router is designed to estimate the utility of each memory item  $m \in M_t$  for a specific agent  $A_i$  at interaction time  $t$ , given its role  $R_i$  and task stage  $S_t$ . To keep routing efficient and interpretable, we adopt a lightweight heuristic-based scoring mechanism, which combines the following three components:

- **Role Relevance**. Each agent is assigned a semantic role (e.g., Planner, Executor, Summarizer). We maintain a role-specific keyword list  $\mathcal{K}_i$  for each role  $R_i$ , manually curated from the task schema. A memory item  $m$  is scored higher if it contains any keywords from  $\mathcal{K}_i$ . Formally:

$$\text{Score}_{\text{role}}(m) = \mathbb{1}[\exists k \in \mathcal{K}_i \text{ such that } k \in m]$$

- **Task Stage Priority**. Certain memory items are more relevant to specific task stages (e.g., context planning, candidate filtering, final decision). For each stage  $S_t$ , we define a preferred item type (e.g., query history, tool results, prior plans). We assign higher scores to items matching the preferred type for the current stage. Let  $\mathcal{T}_t$  be the set of relevant memory types at stage  $S_t$ :

$$\text{Score}_{\text{stage}}(m) = \mathbb{1}[\text{Type}(m) \in \mathcal{T}_t]$$

- **Recency**. Memory items generated more recently (i.e., closer to  $t$ ) are typically more relevant. We compute recency score based on their relative position in the memory buffer, using a decaying weight:

$$\text{Score}_{\text{recency}}(m) = \exp(-\lambda \cdot (t - t_m))$$

where  $t_m$  is the timestamp or round index when  $m$  was created, and  $\lambda$  is a tunable decay factor.

The final importance score is computed as a weighted combination:

$$\alpha(m; R_i, S_t) = w_1 \cdot \text{Score}_{\text{role}}(m) + w_2 \cdot \text{Score}_{\text{stage}}(m) + w_3 \cdot \text{Score}_{\text{recency}}(m)$$

where  $w_1, w_2, w_3$  are hyperparameters (e.g., set to 1.0 by default).

This design balances interpretability, extensibility, and efficiency, and allows plug-in of learned components if desired.

## F Modular Multi-Agent System Design for ALFWorld

ALFWorld (Shridhar et al. 2021) is an embodied instruction-following environment where agents must complete household tasks through navigation and object manipulation.

We define a structured multi-agent framework for ALFWorld, where each agent specializes in a specific sub-task and interacts via a shared memory interface. This architecture supports effective division of labor and token-efficient context routing.

The agents communicate via a shared semantic memory store  $\mathcal{M}_t$ , with memory slices dynamically routed at each timestep  $t$  using a token-budgeted context router (e.g., RCR-Router). Each agent  $A_i$  receives a context  $C_t^i \subseteq \mathcal{M}_t$  based on its *role* and *task stage*.

### Interfaces and Coordination



Table 7: Agent Roles in ALFWorld for RCR-Router

| Agent Role                  | Description                                                                                                                                                                                                                                                                                                         |
|-----------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Planner</b>              | <ul style="list-style-type: none"> <li>• Parses natural language task instructions.</li> <li>• Decomposes them into symbolic subgoals (e.g., <code>Find(bottle)</code>).</li> <li>• Integrates feedback from memory and updates plan iteratively.</li> </ul>                                                        |
| <b>Searcher / Navigator</b> | <ul style="list-style-type: none"> <li>• Explores the simulated environment to discover goal-relevant objects.</li> <li>• Navigates the agent to specified locations using spatial reasoning.</li> <li>• Supports multi-hop search via memory-informed object-location mapping.</li> </ul>                          |
| <b>Interactor</b>           | <ul style="list-style-type: none"> <li>• Performs object-level actions (e.g., <code>Pickup</code>, <code>PutIn</code>, <code>Open</code>).</li> <li>• Executes low-level manipulations in response to subgoal execution.</li> <li>• Verifies interaction success and provides outcomes to shared memory.</li> </ul> |

### Agent Interfaces and Inputs

- **Planner Input:** Natural language task instruction.
- **Planner Output:** High-level sub-goals  $\mathcal{P} = [g_1, g_2, \dots, g_k]$ .
- **Searcher Input:** Sub-goal type + prior memory state; queries environment for candidate object locations.
- **Navigator Input:** Sub-goal location + current agent position.
- **Interactor Input:** Goal object/location + interaction command (`Pickup`, `Open`, etc.).
- **Memory Access:** All agents interact with shared memory  $\mathcal{M}_t$  using read/write APIs, managed by RCR-router.

### Execution Flow

1. **Planner** interprets the instruction and generates sub-goals.
2. **Searcher** queries the environment (via **Perception**) to identify objects satisfying current sub-goal.
3. **Navigator** moves the agent toward target objects or locations.
4. **Interactor** performs the necessary environment interaction.
5. **Evaluator** verifies task completion and optionally signals **Planner** for replanning.

### Design Benefits

- **Role Modularity and Separation of Concerns:** Each agent is assigned a distinct functional role—such as planning, navigation, or interaction—enabling clean abstraction of responsibilities. This separation simplifies debugging, benchmarking, and targeted model improvements.
- **Token-Efficient Context Routing:** The integration of RCR-router ensures that each agent receives only task-relevant memory slices filtered by semantic importance, role identity, and token budget. This significantly reduces redundant communication and enhances inference efficiency.
- **Structured Execution with Iterative Feedback:** Agents operate in a recurrent loop, reading from and writing to a centralized memory store  $\mathcal{M}_t$ . This allows for dynamic plan revisions, success verification, and coordination between planning and execution modules.
- **Perceptual Grounding and Semantic Sharing:** Agents share a structured memory that encodes grounded observations (e.g., object types, spatial positions, interaction outcomes), facilitating semantic consistency and spatial reasoning across modules.
- **Scalability and Extensibility:** The architecture accommodates additional roles such as `Reasoner`, `Memory Summarizer`, or `Language Explainer` without changes to the routing pipeline. It generalizes across different task settings in embodied environments.
- **Backend Compatibility and Reusability:** Each agent can be instantiated using different backbone LLMs or decision models (e.g., DeepSeek, GPT-4, distilled variants), supporting plug-and-play experimentation without modifying upstream pipeline logic.

Table 8: Overall Performance Summary across Benchmarks (with per-agent token budget  $B_i = 2048$ ). We report runtime, token usage, LLM-based Answer Quality, and standard QA metrics. RCR-router outperforms baselines in both efficiency and accuracy.

| Benchmark | Method         | Results         |           |                |             |             |             |
|-----------|----------------|-----------------|-----------|----------------|-------------|-------------|-------------|
|           |                | Avg Runtime (s) | Token (K) | Answer Quality | Precision   | Recall      | F1          |
| ALFWorld  | Full-Context   | 145.3           | 6.82      | 3.91           | 58.2        | 60.1        | 59.1        |
|           | Static Routing | 122.8           | 5.12      | 4.07           | 61.5        | 63.0        | 62.2        |
|           | RCR-router     | 96.4            | 4.39      | <b>4.42</b>    | <b>66.7</b> | <b>67.9</b> | <b>67.3</b> |

**Evaluation on ALFWorld.** Table 8 reports the overall performance comparison on the ALFWorld benchmark under a fixed per-agent token budget of  $B_i = 2048$ . We evaluate three routing strategies: *Full-Context*, *Static Routing*, and our proposed *RCR-router*. RCR-router achieves the best performance across all metrics. Specifically, it reduces the average runtime from 145.3s (Full-Context) and 122.8s (Static) to 96.4s, while also lowering token usage to 4.39K—representing a 35.6% reduction over Full-Context.

In terms of answer quality, RCR-router attains a score of 4.42, significantly higher than both baselines (3.91 and 4.07). Standard QA metrics also improve: RCR-router achieves 66.7% precision, 67.9% recall, and 67.3% F1, compared to 58.2/60.1/59.1 for Full-Context and 61.5/63.0/62.2 for Static Routing. These results confirm that our method improves both efficiency and accuracy in multi-agent embodied environments.

## G Modular for Multi-Agent System Design in WebShop

WebShop (Yao et al. preprint) is a text-based e-commerce environment where agents must fulfill shopping goals through search, click, and buy tool invocations.

We define the following modular roles for WebShop agent collaboration:

- **Planner (Query Decomposer):**
  - Interprets the user’s natural language instruction and extracts structured constraints.
  - Identifies product attributes (e.g., size, color, category, quality keywords).
  - Constructs a canonical query or subgoals to guide retrieval.
- **Searcher (Retriever):**
  - Uses the structured query from the Planner to search a product corpus (e.g., via Pyserini or ScraperAPI).
  - Retrieves top- $k$  relevant product candidates.
  - May refine results using user-specific filters or historical memory.
- **Recommender (Evaluator):**
  - Analyzes product candidates returned by the Searcher.
  - Matches them against instruction attributes (both explicit and implicit).
  - Selects and ranks products and generates a natural language justification or recommendation.

We design a modular multi-agent system for WebShop consisting of three specialized roles:

**Execution Flow** Given a user instruction, our WebShop multi-agent system proceeds as follows:

1. **Planner** receives the instruction and performs constraint extraction (e.g., attributes, options), forming a structured query.
2. **Searcher** takes the query and retrieves candidate products using a retrieval engine (e.g., Pyserini + indexed corpus, or live ScraperAPI).
3. **Recommender** evaluates the retrieved products, matching them against user constraints and preferences, and generates a ranked shortlist with textual rationales.
4. **(Optional)** The final product or product list is returned to the user, or used to simulate downstream actions (e.g., “choose [Product Title]”, “choose [Option]”, etc.).

Agents interact via a shared structured memory  $M_t$ , with context routed at each iteration based on role and task stage.

Table 9: Agent Roles and Responsibilities in WebShop Multi-Agent System

| Agent Role         | Responsibilities                                                                                                                                                                                                                                                                         |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Planner</b>     | <ul style="list-style-type: none"> <li>Parses user instructions (e.g., “I need a high-quality storage case for my Infinitipro”).</li> <li>Extracts key attributes: category, brand, size, quality, etc.</li> <li>Generates structured queries or sub-goals for the Searcher.</li> </ul>  |
| <b>Searcher</b>    | <ul style="list-style-type: none"> <li>Retrieves relevant products using structured queries and a product index (e.g., Pyserini).</li> <li>Ranks results based on semantic similarity and attribute match.</li> <li>Refines query based on contextual or memory feedback.</li> </ul>     |
| <b>Recommender</b> | <ul style="list-style-type: none"> <li>Evaluates product candidates against user preferences and soft constraints.</li> <li>Filters or re-ranks items based on implicit signals (e.g., “GMO-free”).</li> <li>Generates natural language justifications and final suggestions.</li> </ul> |

Table 10: Overall Performance Summary on WebShop Benchmark (with per-agent token budget  $B_i = 2048$ ). We report run-time, token usage, LLM-based Answer Quality, and standard QA metrics. RCR-router achieves superior efficiency and recommendation quality.

| Benchmark | Method         | Results         |           |                |             |             |             |
|-----------|----------------|-----------------|-----------|----------------|-------------|-------------|-------------|
|           |                | Avg Runtime (s) | Token (K) | Answer Quality | Precision   | Recall      | F1          |
| WebShop   | Full-Context   | 180.2           | 9.78      | 3.85           | 55.4        | 57.6        | 56.5        |
|           | Static Routing | 142.6           | 7.12      | 4.01           | 60.7        | 62.5        | 61.6        |
|           | RCR-router     | 110.9           | 6.03      | <b>4.36</b>    | <b>64.8</b> | <b>65.9</b> | <b>65.3</b> |

### Design Benefits

- **Scalability:** The architecture supports the inclusion of additional specialized roles (e.g., Critic, Explainer, Dialogue Handler) without modifying the core system, allowing for progressive enhancement of capabilities.
- **Context-Aware Interaction:** Through memory abstraction and structured routing, agents can share and retrieve relevant semantic information, enhancing coordination and reducing context redundancy.
- **Token Efficiency:** Leveraging selective routing (e.g., via RCR-Router), agents only receive context slices pertinent to their role and current task state, minimizing unnecessary token usage under LLM constraints.
- **Backend-Agnostic Compatibility:** The modular design is compatible with diverse LLM backends (e.g., OpenAI, DeepSeek, ChatGLM), as long as agents operate under a consistent prompt-response protocol.
- **Interpretable Reasoning:** Role-specific outputs (e.g., structured plans, ranked product lists, natural language recommendations) improve system interpretability and facilitate human-in-the-loop evaluation.

**Evaluation on WebShop.** We evaluate our method on the WebShop benchmark, a realistic multi-agent shopping assistant task involving semantic retrieval, attribute grounding, and decision recommendation. As shown in Table 10, our proposed RCR-router achieves the best overall performance across all evaluation metrics. Specifically, it reduces average runtime by 22% compared to Static Routing and by 38% compared to the Full-Context baseline. It also achieves the lowest token consumption (6.03K) while improving answer quality to 4.36. Moreover, RCR-router significantly outperforms other methods in standard QA metrics, achieving an F1 score of 65.3, representing a +3.7 improvement over Static Routing and a +8.8 gain over Full-Context. These results highlight the effectiveness of our role- and stage-aware context routing framework in reducing overhead and enhancing decision accuracy in complex multi-agent interactions.