MAO-ARAG: MULTI-AGENT ORCHESTRATION FOR AGENTIC RETRIEVAL-AUGMENTED GENERATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Retrieval-Augmented Generation (RAG) has become pivotal in Question Answering (QA) for enhancing accuracy and reducing hallucinations. While diverse RAG architectures—ranging from single-round to iterative and reasoning-based models—have been developed, selecting the optimal pipeline for individual queries with varying complexity remains a challenge. Existing adaptive methods typically rely on classifiers to select among a limited set of predefined workflows; however, this selection-based approach is inherently coarse-grained and restricted by the static scope of candidate workflows. Consequently, neither static workflows nor selection-based strategies can reliably balance effectiveness (answer quality) and efficiency (e.g., token or retrieval cost) in real-world scenarios. To address this, we propose **MAO-ARAG**, a fine-grained adaptive framework based on multi-agent orchestration. Conceptualizing RAG as a multi-turn decision-making process, we define a set of atomic *Executor Agents* corresponding to standard RAG modules (e.g., query reformulation, retrieval, generation). A *Planner Agent* dynamically orchestrates these executors to construct a query-specific workflow, aiming to maximize answer quality while minimizing operational costs. The Planner is optimized via Reinforcement Learning using a composite reward of F1 score and cost penalties. Extensive experiments across multiple QA benchmarks show that MAO-ARAG achieves the highest mean F1 under the reported setup and a favorable quality–cost trade-off. [1]

1 INTRODUCTION

Large Language Models (LLMs) support question answering, retrieval, reasoning, and evaluation, but their frozen parametric knowledge can be outdated or inaccurate Asai et al. (2023); Khattab et al. (2022); Zhao et al. (2023). Retrieval-Augmented Generation (RAG) addresses this limitation by incorporating external evidence through a modular architecture Gao et al. (2024b).

A naive RAG pipeline retrieves candidate texts and generates an answer from them. More advanced systems add query rewriting, document selection, self-reflection, or explicit reasoning Wang et al. (2022); Ma et al. (2023); Ke et al. (2024); Asai et al. (2023); Li et al. (2025); Jin et al. (2025).

These components suit different query complexities: naive RAG is cheap but weak on difficult questions, whereas richer workflows improve coverage at higher cost and latency. Thus, real-world QA requires heterogeneous workflows, as illustrated in Figure 2.

Consequently, a single **Static Workflow RAG** (Figure 1(a)) may struggle to balance quality and cost across queries. **Adaptive RAG** (Figure 1(b)) routes each query to one predefined pipeline Jeong et al. (2024), but remains coarse-grained. Reasoning-heavy systems such as **Search-R1** (Figure 1(c)) allow autonomous multi-turn search, yet combine reasoning, retrieval, aggregation, and generation in one monolithic process, which can introduce long-context, sparse-reward, and search-noise challenges Jin et al. (2025); Guo et al. (2025); Chen et al. (2026); Deng et al. (2025).

To bridge this gap, we propose **MAO-ARAG** (Figure 1(d)), a fine-grained multi-agent framework. It *constructs*, rather than selects, query-specific workflows by sequencing atomic RAG executors. A

[1]The anonymous code is available at `https://anonymous.4open.science/r/Agentic-RAG-0FB1`.

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

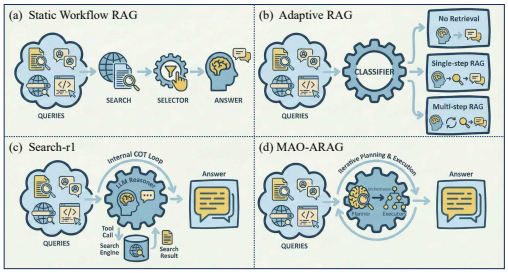


Figure 1: Comparison of different RAG paradigms. (a) Static Workflow RAG uses a static pipeline. (b) Adaptive RAG selects from predefined pipelines via a classifier. (c) Search-R1 relies on internal and monolithic reasoning loops. (d) MAO-ARAG (Ours) dynamically orchestrates atomic agents.

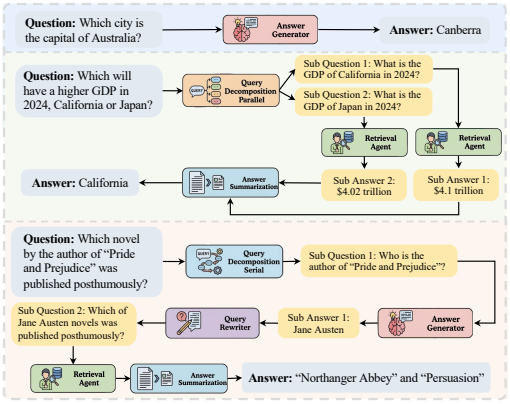


Figure 2: The appropriate workflows for different types of queries are highly heterogeneous. Fixed pipelines struggle to adapt to this variance.

Planner Agent performs this orchestration at each state and is optimized with PPO Schulman et al. (2017) using answer utility and cost penalties. This yields dynamic pipelines that trade off token, retrieval, and turn costs.

Our main contributions are as follows:

- We propose MAO-ARAG, which orchestrates atomic RAG executors into query-specific workflows instead of selecting from fixed pipelines.
- We train the Planner with PPO using answer utility and cost penalties to learn this quality–efficiency trade-off.
- Experiments on seven QA benchmarks show a favorable quality–cost frontier against fixed and classifier-based adaptive baselines.

2 Related Work

RAG systems combine retrieval, query reformulation, document selection, and answer generation. Representative methods use BM25 or dense retrievers such as Contriever, BGE, and E5; RRR and DMQR-RAG reformulate queries; BGM and RAG-DDR filter evidence; and recent work jointly optimizes multiple modules Robertson & Walker (1994); Izacard et al. (2021); Wang et al. (2022); Xiao et al. (2024); Ma et al. (2023); Li et al. (2024b); Ke et al. (2024); Li et al. (2024a); Chen et al. (2025). In the cited settings, these methods primarily optimize effectiveness and retain relatively static workflows.

Existing workflows fall into four paradigms: fixed single-round pipelines; iterative retrieval-generation loops; classifier-based Adaptive RAG; and reasoning RAG, which includes Search-o1, Search-R1, and R1-Searcher Ma et al. (2023); Jeong et al. (2024); Li et al. (2025); Jin et al. (2025); Song et al. (2025). The first two are relatively rigid, Adaptive RAG selects only among predefined pipelines, and reasoning RAG can incur high training cost and unstable optimization Deng et al. (2025); Chen et al. (2026).

Multi-agent systems extend this space. MA-RAG Nguyen et al. (2025) invokes specialized agents through prompting, while M-ASK Chen et al. (2026) separates search and knowledge roles with turn-level rewards. MAO-ARAG instead learns a cost-aware Planner that composes atomic RAG modules and can replan at every state; Appendix B gives the detailed scope comparison.

3 METHODS

3.1 MODELING RAG AS A PLANNER–EXECUTOR SEMI-MARKOV DECISION PROCESS

To capture decisions made at different time scales, we model the RAG system as a semi-Markov decision process motivated by hierarchical RL Ghavamzadeh et al. (2006); Sutton et al. (1998). The Planner makes a high-level decision by constructing a workflow, and fixed Executors carry out the resulting sequence for a variable duration rather than a single time step.

Formally, we use the semi-Markov tuple $\langle S, A, P, R, \gamma, T \rangle$. S represents the state space. For the present implementation, A denotes the Planner action set; the Executor modules are fixed operators invoked by the Planner. P denotes the transition probability. R is the reward function. Crucially, $T : S \times A \to \mathbb{R}^+$ is a function representing the *variable duration* of executing an action. In our context, a single action taken by the Planner corresponds to the generation of a specific workflow. The execution of this workflow by the Executors constitutes the duration T. The Planner only transitions to the next decision state after the entire workflow concludes, perfectly aligning with the semi-Markov property.

To implement this, MAO-ARAG employs a **Planner** to coordinate multiple **Executors**, as illustrated in Figure 3.

The Planner serves as the high-level decision-maker. It is responsible for designing an appropriate workflow for a given question (or sub-question). This workflow is an ordered sequence of executor symbols and acts as a "macro-action" that persists until execution completes.

The Executors act as the atomic workers encompassing common RAG modules:

- **Query Decomposition Serial (QDS)**: Serially decomposes a question into sequentially dependent sub-questions. The answer to a later sub-question often depends on the answers to preceding ones.
- **Query Decomposition Parallel (QDP)**: Decomposes a question into independent sub-questions for parallel search.
- **Query Rewriter (QR)**: Rewrites a question into a search-friendly version.
- **Document Selector (DS)**: Filters noisy documents from retrieved candidate documents.
- **Retrieval Agent (RA)**: Returns the top k relevant documents from the corpus.
- **Answer Generator (AG)**: Generates an answer based on context or internal knowledge.
- **Answer Summarization (AS)**: Synthesizes sub-answers into a final response.

By leveraging this semi-Markov formulation, MAO-ARAG captures the temporal abstraction of RAG tasks: the Planner optimizes the sequence of workflows (macro-actions) over variable time intervals T, while Executors handle the specific atomic operations within each interval.

3.2 SEMI-MARKOV FORMULATION FOR THE PLANNER

As illustrated in Figure 3, the architecture of MAO-ARAG is built around a Planner coordinating fixed Executor modules. The Planner functions as the central decision-maker, responsible for analyzing the input query (Observation) and orchestrating a bespoke workflow by selecting a specific combination of atomic modules (Action). The Executors then carry out this workflow to generate results. To optimize this process, the system employs a multi-objective feedback mechanism: the Planner receives a composite reward signal derived from the answer quality (Final Reward), operational efficiency (Cost Penalty), and structural validity (Format Penalty). Driven by this feedback, we utilize the PPO algorithm to refine the Planner's policy.

In the following, we will introduce the key components of the semi-Markov environment for the Planner, including the Observation, Action Space, and Reward Function.

Planner observation is defined in Eq. (1) as the Planner prompt $Prompt_{planner}$, the current question q, and accumulated context C. Here q is the initial question or a sub-question, and C is empty at the initial step.

$$O_{planner} = \{Prompt_{planner}, q, C\} \tag{1}$$

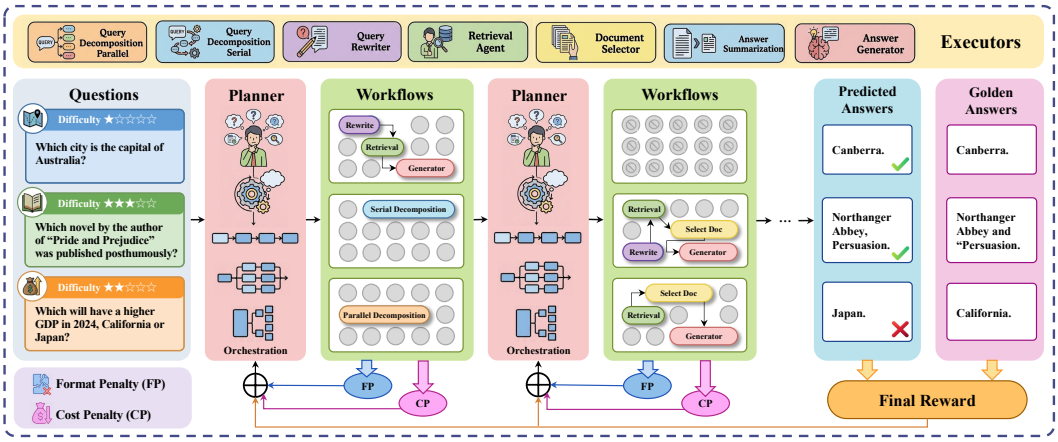


Figure 3: The overall framework of MAO-ARAG. Given queries of varying complexity, the *Planner Agent* iteratively orchestrates a set of atomic *Executor Agents* (e.g., Rewriter, Generator) to construct dynamic, query-specific workflows. The planner is optimized via Reinforcement Learning to maximize the *Final Reward* (based on answer quality) while minimizing resource consumption and ensuring executability through the *Cost Penalty (CP)* and *Format Penalty (FP)*, respectively.

The Planner action space is the set of executor symbols defined in Eq. (2). The Planner outputs a sequence of these symbols. Although Equation 2 defines the executor vocabulary, the Planner does not select only a single executor as its complete action. Instead, it autoregressively generates a workflow string composed of executor symbols. The generated string is parsed into an executor list and checked by a deterministic workflow validator. Invalid workflows receive the format penalty in Equation 5. We provide the detailed workflow validity rules in Appendix D.

$$\mathcal{A}_{planner} = \{\texttt{QDS}, \texttt{QDP}, \texttt{QR}, \texttt{RA}, \texttt{DS}, \texttt{AG}, \texttt{AS}\}. \tag{2}$$

Reward Function of the planner comprises three components. The first component, shown in Equation (3), is the F1 score calculated between the predicted answer, $a_{\text{predicted}}$, generated by the RAG, and the golden answer, a_{golden}. This F1 score serves as one of the key performance metrics we strive to optimize and is also the Final Reward illustrated in Figure 3.

$$R_{f1} = F1(a_{\text{predicted}}, a_{\text{golden}}) \tag{3}$$

The second component is a penalty term concerning the cost, denoted as Cost Penalty (CP) in Figure 3, which is defined as Equation (4).

$$R_{CP} = Token_{\text{cost}} + Turn_{\text{cost}} + \mathbb{I}(S) \tag{4}$$

In Equation (4), $Token_{\text{cost}}$ is the workflow token cost, normalized to $[0, 1]$. $Turn_{\text{cost}}$ is the normalized execution-turn cost and increases with the number of turns. The indicator $\mathbb{I}(S)$ equals 1 when the workflow invokes the Retrieval Agent (RA), and 0 otherwise. Appendix C explains the scaling choices.

The third component concerns the penalty term related to the workflow format, denoted as Format Penalty (FP) in Figure 3. A generated workflow is considered valid only if it satisfies the predefined workflow validity rules in Table 6. We define the format penalty as:

$$R_{FP} = \mathbb{I}[\text{workflow is invalid}], \tag{5}$$

where $R_{FP} = 0$ if the workflow format is valid and executable, and $R_{FP} = 1$ otherwise. Thus, invalid workflows are explicitly penalized during Planner optimization.

Finally, the total reward of the planner can be defined as Equation (6), which consists of R_{f1}, R_{CP}, and R_{FP}. α is a hyperparameter.

$$R_{planner} = R_{f1} - \alpha \cdot R_{CP} - R_{FP} \tag{6}$$

Reward Calculation and Optimization

Upon the completion of a rollout, the system transitions to reward calculation and parameter updates. As shown in Figure 3, one complete rollout may contain multiple Planner invocations, since the Planner can be repeatedly called for the initial question and the decomposed sub-questions. We therefore optimize all Planner decisions across the rollout rather than only the final one.

For a rollout τ, we collect a Planner-level experience sequence:

$$\tau = \{(O^i_{\text{planner}}, w^i, R^i_{\text{planner}})\}_{i=1}^{N_\tau}, \tag{7}$$

where i indexes the i-th Planner invocation in the rollout, N_τ denotes the total number of Planner invocations, O^i_{planner} is the Planner observation, w^i denotes the serialized workflow string generated by the Planner, and R^i_{planner} is the corresponding reward.

After the final predicted answer $a_{\text{predicted}}$ is obtained, we compare it with the ground-truth answer a_{golden} to compute the outcome reward R_{f1}. This outcome reward is shared by all Planner decisions in the same rollout, while each decision has its own format and cost penalties. Thus, the reward for the i-th Planner decision is defined as:

$$R^i_{\text{planner}} = R_{f1} - \alpha \cdot R^i_{\text{CP}} - R^i_{\text{FP}}, \quad i = 1, \ldots, N_\tau. \tag{8}$$

With these annotated multi-step trajectories, we employ PPO to update the Planner's policy. Specifically, let $w^i = (y^i_1, \ldots, y^i_{T_i})$ denote the serialized workflow string generated by the Planner at the i-th invocation, where T_i is the number of generated tokens. The actor loss is computed over all workflow tokens in the rollout:

$$\mathcal{L}_{\text{Actor}}(\theta) = -\sum_{i=1}^{N_\tau}\sum_{t=1}^{T_i} \min\left(r_{i,t}\hat{A}^{i,t}_{\pi_{\theta_{\text{old}}}}, \text{clip}(r_{i,t}, 1-\epsilon, 1+\epsilon)\hat{A}^{i,t}_{\pi_{\theta_{\text{old}}}}\right). \tag{9}$$

where

$$r_{i,t} = \frac{\pi_\theta(y^i_t \mid O^i_{\text{planner}}, y^i_{<t})}{\pi_{\theta_{\text{old}}}(y^i_t \mid O^i_{\text{planner}}, y^i_{<t})} \tag{10}$$

is the token-level probability ratio for the t-th token in the workflow string generated at the i-th Planner invocation. We estimate the corresponding advantages with Generalized Advantage Estimation (GAE) Schulman et al. (2015); the invocation-level estimate is shared by tokens from the same workflow string.

The critic is also optimized at the token level over the generated workflow tokens:

$$\mathcal{L}_{\text{Critic}}(\phi) = \sum_{i=1}^{N_\tau}\sum_{t=1}^{T_i}\left(V_\phi(O^i_{\text{planner}}, y^i_{<t}) - V^{i,t}_{\text{target}}\right)^2. \tag{11}$$

where $V^{i,t}_{\text{target}}$ denotes the token-level value target for the t-th token of the i-th Planner response.

Through this optimization over all Planner-generated workflow tokens, MAO-ARAG assigns learning signals to every workflow decision made throughout the rollout. Algorithm 1 summarizes the complete rollout and training procedure of MAO-ARAG, including workflow generation and execution, reward computation, and PPO updates.

4 Experiments

Our experiments mainly focus on the following research questions:

- **RQ.1**: How effective is the MAO-ARAG when compared to existing RAG methods and pipelines?
- **RQ.2**: Can MAO-ARAG achieve a good balance between effectiveness and efficiency?

Algorithm 1: Rollout and Training Process of MAO-ARAG

Initialize: The parameters of the Actor model θ, the Critic model ϕ, the initial model θ_{init}, and a replay buffer $\mathcal{M} = \varnothing$.
Inputs: Dataset with initial questions q and corresponding golden answers Ans_{golden}
for $batch \leftarrow 1$ **to** N_batch **do**
  `// Phase 1: Inference and Rollout`
  for *each question* $q_{init} \in batch$ **do**
    `// Rollout Multiple Turns`
    for $turn_i \leftarrow 1$ **to** *MAX_TURN* **do**
      `// Planner`
      Determine the current target question q (either q_{init} or a pending q_{sub}).
      Construct observation $O^i_{planner}$ according to Equation (1).
      Get the workflow w_i for the given question q.
      Get the Format Penalty (FP) R^i_{FP}.
      `// Execute the workflow`
      Following the workflow w_i, execute each executor involved.
      Get the Cost Penalty (CP) R^i_{CP}.
      Store tuple $\mathcal{T}_i = (O^i_{planner}, w_i, R^i_{FP}, R^i_{CP})$ in the replay buffer $\mathcal{M}$
      `// Check Termination`
      if q_{init} *is resolved (answer generated)* ***or*** *all sub-questions answered* **then**
        break
    Get the predicted answer $a_{\text{predicted}}$ for q_{init}.
    Compute the F1 score as R_{f1} for q_{init}.
    `// Update the data`
    for *each step* t *in the episode* **do**
      Calculate the total reward $R^t_{planner}$ according to Equation (6).
      Update the tuple $\mathcal{T}_t$ in $\mathcal{M}$ with the final calculated rewards.
  `// Phase 2: Reward Calculation and Optimization`
  for *each question* $q \in batch$ **do**
    Compute the advantage function $\hat{A}^t_{\pi_{\theta_{\text{old}}}}$ using GAE
    Calculate the loss of the Actor $\mathcal{L}_{\text{Actor}}(\theta)$ and Critic model $\mathcal{L}_{\text{Critic}}(\phi)$
    Update the parameters of models through the overall loss function $\mathcal{L}(\theta, \phi)$ in Equation (9) and Equation (11)
  Clear the replay buffer $\mathcal{M}$ to $\varnothing$
Output : A well-trained planner: Actor model with parameters θ_{trained}

- **RQ.3**: How does the choice of backbone LLMs for the planner and executors affect the performance and cost of MAO-ARAG?

Experimental Setup. We evaluate MAO-ARAG on seven open-domain QA benchmarks, including three single-hop datasets: NQ Kwiatkowski et al. (2019), PopQA Mallen et al. (2022), and AmbigQA Min et al. (2020); and four multi-hop datasets: HotpotQA Yang et al. (2018), 2WikiMultiHopQA Ho et al. (2020), Musique Trivedi et al. (2022), and Bamboogle Press et al. (2022). For retrieval-based methods, we use Wikipedia Karpukhin et al. (2020) as the corpus and E5 Wang et al. (2022) as the retriever. We report F1 for answer quality, and token cost, retrieval calls, and turns for efficiency. Unless otherwise specified, Qwen2.5-7B-Instruct Qwen Team (2024) serves as the Planner, and GPT-4o mini OpenAI (2024) serves as the executor backbone.

Evaluation Metrics. The F1 score is used to evaluate the quality of predicted answers $a_{\text{predicted}}$ across different methods, but the cost of generating these predictions is also important. In this study, we assessed three metrics related to prediction cost:

- **Token Cost**: Represents the average cost of tokens consumed to answer predictions (in USD per query).
- **Retrieval Calls**: Indicates the average number of retrieval calls made (calls per query).

Table 1: Performance comparison in terms of F1 score on single-hop and multi-hop open-domain QA benchmarks. We report results on each dataset, the average performance within each benchmark group, and the overall average across all datasets. The best results are **bolded** and the second-best results are underlined.

Method	Single-hop QA				Multi-hop QA					Avg
	NQ	PopQA	AmbigQA	Avg	HotpotQA	2Wiki	Musique	Bam.	Avg	All
Standard Baselines										
LLM w/o RAG	39.96	30.99	49.90	40.28	42.38	33.49	20.74	36.15	33.19	36.23
Vanilla RAG	48.02	44.23	59.04	50.43	49.54	37.62	25.66	43.45	39.07	43.94
Static Workflow RAG										
RRR Ma et al. (2023)	46.27	41.59	56.15	48.00	43.14	29.77	23.21	37.81	33.48	39.71
BGM Ke et al. (2024)	48.41	45.39	**59.25**	51.02	49.58	36.79	25.60	44.10	39.02	44.16
MMOA-RAG Chen et al. (2025)	46.88	40.26	55.88	47.67	43.19	30.40	21.78	36.53	32.98	39.28
Adaptive & Agentic RAG										
Self-RAG Asai et al. (2023)	41.60	34.25	52.06	42.64	47.94	39.53	32.88	56.33	44.17	43.51
Adaptive RAG Jeong et al. (2024)	45.25	43.44	56.25	48.31	53.00	45.34	33.99	49.82	45.54	46.72
Search-R1 Jin et al. (2025)	42.22	43.35	52.50	46.02	44.44	34.13	21.44	37.83	34.46	39.42
Search-o1 Li et al. (2025)	46.68	43.12	56.93	48.91	53.75	47.26	**39.51**	61.58	50.53	49.83
MAO-ARAG w/o train	50.57	32.73	55.15	46.15	49.68	40.75	32.36	49.41	43.05	44.38
MAO-ARAG (Ours)	**54.50**	**54.16**	57.80	**55.49**	**53.80**	**47.69**	37.33	**65.09**	**50.98**	**52.91**

- **Turns**: The average number of turns required to complete a query (turns per query).

Cost accounting scope. The reported monetary costs are inference-time API costs per query. Planner training uses one epoch over 2,400 NQ and 4,800 HotpotQA examples; only the Planner policy is updated and all Executors remain frozen. We do not fold a hardware-dependent one-time training charge into the API-cost axis because the retained experiment record does not contain GPU-hour or energy measurements. For a deployment serving N queries, the complete accounting is $C_{\text{total}}(N) = C_{\text{train}} + NC_{\text{infer}}$, where C_{train} would depend on the chosen training hardware and C_{infer} is the per-query value reported here. Appendix G records this boundary and the amortization calculation explicitly.

Baselines. We compare MAO-ARAG with three groups of baselines, as shown in Table 1: standard baselines (LLM w/o RAG and Vanilla RAG), static workflow RAG methods (RRR Ma et al. (2023), BGM Ke et al. (2024), and MMOA-RAG Chen et al. (2025)), and adaptive/agentic RAG methods (Self-RAG Asai et al. (2023), Adaptive RAG Jeong et al. (2024), Search-R1 Jin et al. (2025), and Search-o1 Li et al. (2025))[2]. Descriptions of the standard and static workflow baselines, along with implementation settings, are provided in Appendix E.

Adaptive and Agentic RAG. These methods introduce dynamic decision-making or iterative retrieval-generation processes. **Self-RAG** Asai et al. (2023) uses self-reflective retrieval and critique signals to improve the reliability of generated answers. **Adaptive RAG** Jeong et al. (2024) predicts query complexity and routes each query to one of several predefined pipelines, such as no-retrieval, single-step retrieval, or multi-step retrieval. **Search-R1** Jin et al. (2025) trains an LLM with reinforcement learning to perform reasoning and search within an agentic RAG framework. **Search-o1** Li et al. (2025) incorporates an agentic retrieval mechanism that dynamically performs search during complex reasoning.

4.1 Performance of Different Methods (RQ.1)

We evaluate all methods on seven datasets. The Planner is trained for one epoch on 2,400 NQ and 4,800 HotpotQA question–answer pairs, then tested on all seven datasets.

In Table 1, "MAO-ARAG w/o train" uses an untrained Qwen2.5-7B-Instruct Planner. "MAO-ARAG" is the RL-trained Planner with $\alpha = 0$, so this variant optimizes F1 without a cost penalty.

MAO-ARAG obtains the highest seven-dataset average (52.91). Adaptive RAG (46.72) improves over the strongest static baseline, BGM (44.16), but remains 6.19 F1 points below MAO-ARAG; Search-o1 is 3.08 points below. The untrained Planner reaches 44.38, indicating an 8.53-point gain in

[2]For a fair comparison, we re-implement all baselines with GPT-4o-mini, except Search-R1, which follows the original Search-R1 setup and is re-implemented with Qwen2.5-7B-Instruct and RL reasoning training.

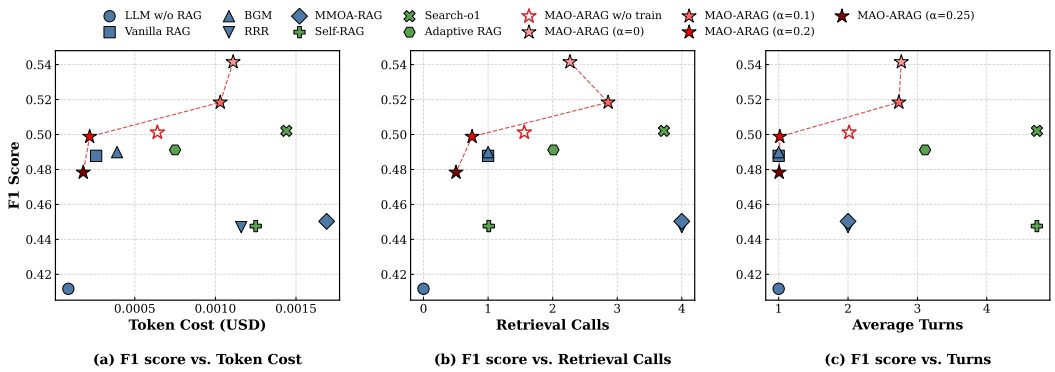


Figure 4: Trade-off analysis between performance (F1 score) and efficiency metrics: (a) token cost, (b) retrieval calls, and (c) average turns. Values are averaged over the in-domain NQ and HotpotQA workloads. Methods are categorized by workflow type: **Static Workflow**, **Adaptive & Agentic Workflow**, and **MAO-ARAG (Ours)**. The red dashed trajectory shows MAO-ARAG under different cost penalties (α); it forms a favorable frontier for this two-dataset workload.

this reported run after RL training. RL training improves the untrained Planner by 8.53 F1 points (44.38 to 52.91).

Search-R1 is included as a representative monolithic RL-based agentic-search baseline, whereas MAO-ARAG trains only the Planner and keeps the Executor backbone frozen. This comparison is therefore intended to contrast two system-level design points rather than provide a backbone-matched architectural ablation. The modular Planner–Executor interface also allows the Executor backbone to be replaced without retraining the Planner, as demonstrated in Section 4.3.

4.2 Cost-Performance Trade-Off (RQ.2)

We measure token cost, retrieval calls, and execution turns per query. Each α value is a separately trained PPO variant rather than an inference-time setting; larger α penalizes resource use more strongly. Appendix F gives the sensitivity analysis.

Figure 4 evaluates the α sweep on NQ–HotpotQA. Separately, on the seven-dataset benchmark in Table 1, MAO-ARAG (52.91 F1) improves over Search-o1 (49.83) while the reported aggregate token cost, retrieval calls, and turns change as follows:

$$\text{token cost: } 1.354 \rightarrow 1.129 \text{ mUSD}, \quad \text{retrieval calls: } 3.542 \rightarrow 2.327,$$
$$\text{turns: } 4.542 \rightarrow 2.899.$$

These are equal-weight, system-level averages; they do not establish a uniform per-query or multi-hop cost reduction.

- **vs. Static Workflow RAG**: Fixed pipelines are generally cheaper but hit a performance ceiling on multi-step questions; MAO-ARAG benefits from composing modules dynamically.
- **vs. Adaptive RAG**: Adaptive RAG improves fixed pipelines but remains below the MAO-ARAG frontier on this workload, consistent with the limits of predefined workflow selection.
- **vs. Agentic RAG (Search-o1)**: Search-o1 provides a strong agentic baseline, while MAO-ARAG is evaluated as a modular Planner–Executor system. The comparison is intended to assess system-level quality–cost behavior under the reported configurations.

Overall, MAO-ARAG gives a favorable quality–cost frontier on the evaluated workloads.

4.3 Smaller Planner and Different Executor Backbone (RQ.3)

In this section, we investigate the versatility of MAO-ARAG by exploring two critical dimensions: the scalability of the planner model size and the extensibility of executor backbones.

Lightweight Planner Deployment. We first initialize 1.5B and 0.5B Planners via Supervised Fine-Tuning (SFT), and subsequently optimize these models with PPO (+PPO). As shown in Table 2, the

Table 2: Ablation on planner size and executor backbone, averaged over NQ and HotpotQA. Avg. Cost denotes the average token cost per query (USD). Avg. Retrieval Calls and Avg. Turns denote the average numbers of retrieval calls and execution turns per query, respectively.

Configuration	F1	Avg. Cost ($)	Avg. Retrieval Calls	Avg. Turns
Reference: Qwen2.5-7B Planner + GPT-4o-mini				
7B (w/o train)	50.13	0.00064	1.56	2.02
7B (+PPO)	54.15	0.00112	2.27	2.77
Effect of planner size				
1.5B (+SFT)	53.81	0.00102	2.13	2.59
1.5B (+PPO)	53.91	0.00095	2.06	2.48
0.5B (+SFT)	53.64	0.00101	2.10	2.54
0.5B (+PPO)	53.92	0.00111	2.26	2.76
Effect of executor backbone				
w/ GPT-3.5-turbo	48.08	0.00328	1.63	2.24
w/ GPT-4.1-nano	47.43	0.00051	1.68	2.13
w/ GPT-4o	56.82	0.00644	1.64	3.23

0.5B (+PPO) Planner reaches an F1 score of 53.92, close to the 54.15 of the 7B (+PPO) configuration. These results suggest that smaller Planners can retain performance in this tested setting; latency and memory were not separately measured.

Modular Executor Adaptation. Our decoupled architecture allows executor backbones to be switched for different constraints without retraining the Planner. We compared the baseline backbone (GPT-4o-mini) against two alternatives representing different trade-offs:

- **Cost-Effective Option (GPT-4.1-nano)**: While its F1 score (47.43) is slightly lower than the baseline, it slashes the token cost by over 50% (0.00051 vs. 0.00112). This provides a cost-oriented option for budget-constrained scenarios.
- **Performance-Oriented Option (GPT-4o)**: We further integrated the more powerful GPT-4o as the executor backbone. Empirical results demonstrate that this configuration pushes the performance boundary, achieving a remarkable F1 score of 56.82—a substantial gain of 2.67 points over the GPT-4o-mini baseline (54.15). While this performance boost incurs a significantly higher token cost (0.00644, approximately $5.7\times$ that of the baseline), it illustrates the performance-oriented option when budget constraints are relaxed.
- **Legacy Comparison (GPT-3.5-turbo)**: In contrast, the outdated GPT-3.5-turbo yields a suboptimal outcome—lower F1 scores coupled with higher costs ($3\times$ that of the 7B baseline). This negative result reinforces the importance of selecting modern, efficient backbones.

These experiments show that MAO-ARAG can be configured to explore the effectiveness–cost trade-off. A lightweight Planner can be paired with different executor APIs for the tested cost-oriented and performance-oriented settings.

5 Conclusion

In this paper, we presented **MAO-ARAG**, a multi-agent orchestration framework designed to address the rigidity of fixed pipelines and the coarseness of existing adaptive RAG systems. Unlike classifier-based approaches that merely select among predefined workflows, MAO-ARAG utilizes a Planner Agent to dynamically *construct* bespoke workflows at a fine-grained level, orchestrating atomic executor agents to meet specific query needs. By optimizing the planner via Reinforcement Learning with a multi-objective reward, our framework learns to autonomously navigate the trade-off between answer quality and computational cost.

Experiments across single-hop and multi-hop QA benchmarks show a favorable quality–cost trade-off for MAO-ARAG in the evaluated settings. The decoupled planner-executor design also enables lightweight Planner deployment and flexible executor replacement. Future work will investigate heterogeneous execution, allowing the Planner to select different backend models for different workflow steps.

REFERENCES

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

Xiao Bi, Deli Chen, Guanting Chen, Shanhuang Chen, Damai Dai, Chengqi Deng, Honghui Ding, Kai Dong, Qiushi Du, Zhe Fu, et al. DeepSeek LLM: Scaling open-source language models with long-termism. *arXiv preprint arXiv:2401.02954*, 2024.

Yiqun Chen, Lingyong Yan, Weiwei Sun, Xinyu Ma, Yi Zhang, Shuaiqiang Wang, Dawei Yin, Yiming Yang, and Jiaxin Mao. Improving retrieval-augmented generation through multi-agent reinforcement learning. *arXiv preprint arXiv:2501.15228*, 2025.

Yiqun Chen, Lingyong Yan, Zixuan Yang, Erhan Zhang, Jiashu Zhao, Shuaiqiang Wang, Dawei Yin, and Jiaxin Mao. Beyond monolithic architectures: A multi-agent search and knowledge optimization framework for agentic search. *arXiv preprint arXiv:2601.04703*, 2026.

Wenlong Deng, Yushu Li, Boying Gong, Yi Ren, Christos Thrampoulidis, and Xiaoxiao Li. On grpo collapse in search-r1: The lazy likelihood-displacement death spiral. *arXiv preprint arXiv:2512.04220*, 2025.

Jingsheng Gao, Linxu Li, Weiyuan Li, Yuzhuo Fu, and Bin Dai. Smartrag: Jointly learn rag-related tasks from the environment feedback. *arXiv preprint arXiv:2410.18141*, 2024a.

Yunfan Gao, Yun Xiong, Meng Wang, and Haofen Wang. Modular rag: Transforming rag systems into lego-like reconfigurable frameworks. *arXiv preprint arXiv:2407.21059*, 2024b.

Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, Katie Millican, et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

Mohammad Ghavamzadeh, Sridhar Mahadevan, and Rajbala Makar. Hierarchical multi-agent reinforcement learning. *Autonomous Agents and Multi-Agent Systems*, 13:197–229, 2006.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. DeepSeek-R1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. Constructing a multi-hop qa dataset for comprehensive evaluation of reasoning steps. *arXiv preprint arXiv:2011.01060*, 2020.

Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. GPT-4o system card. *arXiv preprint arXiv:2410.21276*, 2024.

Gautier Izacard, Mathilde Caron, Lucas Hosseini, Sebastian Riedel, Piotr Bojanowski, Armand Joulin, and Edouard Grave. Unsupervised dense information retrieval with contrastive learning. *arXiv preprint arXiv:2112.09118*, 2021.

Soyeong Jeong, Jinheon Baek, Sukmin Cho, Sung Ju Hwang, and Jong C Park. Adaptive-RAG: Learning to adapt retrieval-augmented large language models through question complexity. *arXiv preprint arXiv:2403.14403*, 2024.

Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan Arik, Dong Wang, Hamed Zamani, and Jiawei Han. Search-R1: Training llms to reason and leverage search engines with reinforcement learning. *arXiv preprint arXiv:2503.09516*, 2025.

Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick SH Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. Dense passage retrieval for open-domain question answering. In *EMNLP (1)*, pp. 6769–6781, 2020.

Zixuan Ke, Weize Kong, Cheng Li, Mingyang Zhang, Qiaozhu Mei, and Michael Bendersky. Bridging the preference gap between retrievers and llms. *arXiv preprint arXiv:2401.06954*, 2024.

Omar Khattab and Matei Zaharia. Colbert: Efficient and effective passage search via contextualized late interaction over bert. In *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval*, pp. 39–48, 2020.

Omar Khattab, Keshav Santhanam, Xiang Lisa Li, David Hall, Percy Liang, Christopher Potts, and Matei Zaharia. Demonstrate-search-predict: Composing retrieval and language models for knowledge-intensive nlp. *arXiv preprint arXiv:2212.14024*, 2022.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, et al. Natural questions: a benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:453–466, 2019.

Xiaoxi Li, Guanting Dong, Jiajie Jin, Yuyao Zhang, Yujia Zhou, Yutao Zhu, Peitian Zhang, and Zhicheng Dou. Search-o1: Agentic search-enhanced large reasoning models. *arXiv preprint arXiv:2501.05366*, 2025.

Xinze Li, Sen Mei, Zhenghao Liu, Yukun Yan, Shuo Wang, Shi Yu, Zheni Zeng, Hao Chen, Ge Yu, Zhiyuan Liu, et al. Rag-ddr: Optimizing retrieval-augmented generation using differentiable data rewards. *arXiv preprint arXiv:2410.13509*, 2024a.

Zhicong Li, Jiahao Wang, Zhishu Jiang, Hangyu Mao, Zhongxia Chen, Jiazhen Du, Yuanxing Zhang, Fuzheng Zhang, Di Zhang, and Yong Liu. Dmqr-rag: Diverse multi-query rewriting for rag. *arXiv preprint arXiv:2411.13154*, 2024b.

Xinbei Ma, Yeyun Gong, Pengcheng He, Hai Zhao, and Nan Duan. Query rewriting for retrieval-augmented large language models. *arXiv preprint arXiv:2305.14283*, 2023.

Alex Mallen, Akari Asai, Victor Zhong, Rajarshi Das, Daniel Khashabi, and Hannaneh Hajishirzi. When not to trust language models: Investigating effectiveness and limitations of parametric and non-parametric memories. *arXiv preprint arXiv:2212.10511*, 2022.

Sewon Min, Julian Michael, Hannaneh Hajishirzi, and Luke Zettlemoyer. Ambigqa: Answering ambiguous open-domain questions. *arXiv preprint arXiv:2004.10645*, 2020.

Thang Nguyen, Peter Chin, and Yu-Wing Tai. Ma-rag: Multi-agent retrieval-augmented generation via collaborative chain-of-thought reasoning. *arXiv preprint arXiv:2505.20096*, 2025.

OpenAI. GPT-4o mini: Advancing cost-efficient intelligence. OpenAI, 2024. URL `https://openai.com/index/gpt-4o-mini-advancing-cost-efficient-intelligence/`.

Ofir Press, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah A Smith, and Mike Lewis. Measuring and narrowing the compositionality gap in language models. *arXiv preprint arXiv:2210.03350*, 2022.

Qwen Team. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024.

Stephen E Robertson and Steve Walker. Some simple effective approximations to the 2-poisson model for probabilistic weighted retrieval. In *SIGIR'94: Proceedings of the Seventeenth Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval, organised by Dublin City University*, pp. 232–241. Springer, 1994.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Zhihong Shao, Yeyun Gong, Yelong Shen, Minlie Huang, Nan Duan, and Weizhu Chen. Enhancing retrieval-augmented large language models with iterative retrieval-generation synergy. *arXiv preprint arXiv:2305.15294*, 2023.

Huatong Song, Jinhao Jiang, Yingqian Min, Jie Chen, Zhipeng Chen, Wayne Xin Zhao, Lei Fang, and Ji-Rong Wen. R1-searcher: Incentivizing the search capability in llms via reinforcement learning. *arXiv preprint arXiv:2503.05592*, 2025.

Weihang Su, Yichen Tang, Qingyao Ai, Zhijing Wu, and Yiqun Liu. Dragin: Dynamic retrieval augmented generation based on the real-time information needs of large language models. *arXiv preprint arXiv:2403.10081*, 2024.

Richard S Sutton, Andrew G Barto, et al. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. Musique: Multihop questions via single-hop question composition. *Transactions of the Association for Computational Linguistics*, 10:539–554, 2022.

Liang Wang, Nan Yang, Xiaolong Huang, Binxing Jiao, Linjun Yang, Daxin Jiang, Rangan Majumder, and Furu Wei. Text embeddings by weakly-supervised contrastive pre-training. *arXiv preprint arXiv:2212.03533*, 2022.

Shitao Xiao, Zheng Liu, Peitian Zhang, Niklas Muennighoff, Defu Lian, and Jian-Yun Nie. C-Pack: Packed resources for general chinese embeddings. In *Proceedings of the 47th international ACM SIGIR conference on research and development in information retrieval*, pp. 641–649, 2024.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

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

Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. A survey of large language models. *arXiv preprint arXiv:2303.18223*, 2023.

A Author Statements

AI use statement. During manuscript preparation, language models were used for copy editing and formatting assistance. The authors reviewed and verified the scientific claims, experimental results, citations, and final wording.

Reproducibility statement. The paper specifies the Planner action space, workflow validity rules, reward, PPO objectives, datasets, model backbones, and evaluation metrics. The anonymous code link in the abstract and the prompt and cost details in the appendix provide additional information to facilitate reproduction of the reported protocol.

B Detailed Related Work

B.1 Different Modules in RAG Systems

Retrieval Model. Retrieval models play a fundamental role in RAG systems by supplying external knowledge to the LLM-based generator for final answer generation. Among sparse retrieval methods, BM25 Robertson & Walker (1994) remains a strong and widely used baseline. In contrast, Contriever Izacard et al. (2021), BGE Xiao et al. (2024), and E5 Wang et al. (2022) produce dense embeddings and serve as representative dense retrievers. ColBERT Khattab & Zaharia (2020) further improves retrieval accuracy and efficiency through multi-vector representations and a late-interaction mechanism.

Query Reformulation. Query reformulation aims to rewrite or decompose the initial query before retrieval, thereby improving retrieval quality and downstream generation performance. Representative examples include RRR Ma et al. (2023) and DMQR-RAG Li et al. (2024b).

Document Selection. Document selection aims to identify the most useful evidence from noisy candidate documents. This module is adopted in systems such as BGM Ke et al. (2024) and RAG-DDR Li et al. (2024a).

Answer Generator. The answer generator is responsible for producing the final response given the input query and retrieved evidence. Existing RAG systems typically build on strong LLM backbones, including open-source models such as DeepSeek Bi et al. (2024), Llama Grattafiori et al. (2024), and Qwen Yang et al. (2025), as well as proprietary models such as GPT-family and Gemini models Hurst et al. (2024); Gemini Team et al. (2023).

Multi-Module Joint Optimization. Beyond optimizing individual components, recent work has explored the joint optimization of multiple RAG modules to improve overall system performance. For example, RAG-DDR Li et al. (2024a) and MMOA-RAG Chen et al. (2025) jointly optimize modules such as query rewriting, document selection, and answer generation to maximize answer quality. However, these methods mainly focus on effectiveness while paying relatively limited attention to inference efficiency and computational cost. Moreover, they typically rely on static workflows and therefore lack the flexibility to dynamically adapt the execution pipeline to queries of varying complexity.

B.2 Multi-Agent RAG and Agentic Search

MA-RAG Nguyen et al. (2025) uses a collaborative set of Planner, Step Definer, Extractor, and QA agents. Its agents are invoked on demand to support query disambiguation, evidence extraction, and answer synthesis, and the method does not require gradient-based training. M-ASK Chen et al. (2026) decouples Search Behavior Agents from Knowledge Management Agents and uses turn-level rewards to stabilize multi-turn search. Both methods are relevant multi-agent references, but their orchestration granularity differs from ours. MA-RAG coordinates specialized role agents through prompting, and M-ASK optimizes a predefined search/knowledge role structure. MAO-ARAG instead represents common RAG operations as atomic executors, generates a serialized workflow from those operations, and optimizes a cost-aware Planner with PPO. We do not insert unsupported

numerical results for MA-RAG or M-ASK into Table 1; their reported settings use different task protocols, backbones, and evaluation implementations. This scope distinction is made explicit so that the quantitative comparison remains reproducible.

Table 3: Qualitative positioning of closely related adaptive and multi-agent RAG methods.

Method	Adaptation unit	Training signal	Cost-aware atomic composition
Adaptive-RAG	Predefined pipeline	Complexity classifier	No
MA-RAG	Specialized role agents	Prompt orchestration	On-demand, no learned cost objective
M-ASK	Predefined search/knowledge roles	Turn-level RL rewards	No atomic workflow vocabulary
MAO-ARAG	Atomic RAG executors	Planner PPO with F1 and cost terms	Yes

B.3 Typical Workflows in RAG Systems

Single-Round RAG. In single-round RAG, the modules are usually organized in a linear pipeline. RRR Ma et al. (2023) proposes a rewrite-retrieve-read framework, while BGM Ke et al. (2024) introduces a selection-generation paradigm. RAG-DDR Li et al. (2024a) and MMOA-RAG Chen et al. (2025) also follow linear RAG pipelines. These methods optimize one or more modules with RL, multi-agent RL, or rollout-based objectives.

Iterative RAG. Iterative RAG adopts a loop-structured pipeline in which retrieval and generation are repeatedly interleaved. ITER-RETGEN Shao et al. (2023) improves retrieval-augmented language models through iterative integration of retrieval and generation. SELF-RAG Asai et al. (2023) enhances factuality and answer quality via self-reflective retrieval and generation. DRAGIN Su et al. (2024) dynamically addresses real-time information needs during generation, while SMARTRAG Gao et al. (2024a) uses PPO to optimize an iterative RAG framework with answer-based rewards.

Adaptive RAG. Unlike fixed pipelines, adaptive RAG dynamically selects retrieval strategies based on the requirements of the input query. Adaptive-RAG Jeong et al. (2024) uses a classifier to predict query complexity (e.g., easy, medium, or hard), and then routes the query to one of several predefined workflows, ranging from direct generation without retrieval to more complex multi-step retrieval pipelines. This design aims to balance efficiency and effectiveness.

Reasoning RAG. Reasoning RAG incorporates the reasoning capability of LLMs into the retrieval-generation process. Search-o1 Li et al. (2025) improves RAG by explicitly leveraging LLM reasoning. Following DeepSeek-R1 Guo et al. (2025), several studies further explore training reasoning-oriented LLMs for RAG. For example, Search-R1 Jin et al. (2025) and R1-Searcher Song et al. (2025) both adopt answer-based rewards to improve reasoning performance in RAG.

Limitations of Existing Paradigms. Despite these advances, important challenges remain. Single-round RAG relies on static linear pipelines, which are often insufficient for complex multi-hop queries. Iterative RAG introduces multi-turn interaction, but its workflow structure is still typically fixed and lacks flexibility in handling dynamic information needs. Adaptive RAG is more dynamic, but remains coarse-grained because it can only select from a limited set of predefined workflows. Finally, reasoning-based RAG methods often suffer from high training cost and instability during optimization Deng et al. (2025); Chen et al. (2026).

C How and why should the cost-based penalty terms be scaled?

We restate the cost-based penalty from Eq. (4):

$$R_{CP} = Token_{\text{cost}} + Turn_{\text{cost}} + \mathbb{I}(S) \tag{12}$$

We can see that R_{CP} is composed of three parts: $Token_{\text{cost}}$, $Turn_{\text{cost}}$, and $\mathbb{I}(S)$. The range of values for these three parts varies significantly. In order to effectively optimize each component, we should normalize their values to approximately between 0 and 1.

Of course, in real-world scenarios, these three components can all be converted into actual money spent, and each component will have some proportional relationship in terms of cost. However, for the purpose of this study, we consider these three components to be equally important and optimize all three.

Next, we will introduce how to scale every term in this Equation.

- $Token_{\text{cost}}$: Table 4 presents the average token cost for each executor agent. Among the executable workflows, the combination QR, DS, AG, AS has the highest listed token cost, with a token cost of approximately 6.02e-4 dollars per query. By scaling this value to 1.0, we can linearly adjust the token costs of all workflow types, denoted as $Token_{\text{cost}}$, to a range between 0 and 1.0.

Table 4: Agent token cost (per query)

Executor Name	**Token Cost**
Query Decomposition Serial (QDS)	0.91e-4
Query Decomposition Parallel (QDP)	1.00e-4
Query Rewriter (QR)	0.88e-4
Document Selector (DS)	2.08e-4
Answer Generator (AG)	1.58e-4
Answer Summarization (AS)	1.48e-4

- $Turn_{\text{cost}}$: This penalty term is related to latency. The more rounds a query requires, the longer it will take to complete, resulting in higher latency.
  Among all the executors, only QDS and QDP incur additional subsequent rounds. Since QDP decomposes the original question into multiple sub-questions that can be searched in parallel, it results in one additional round, as all sub-questions can be processed simultaneously. On the other hand, QDS decomposes the original question into sub-questions that must be searched sequentially, meaning that each subsequent sub-question requires the answer from the previous one to proceed to the next search step. Therefore, QDS results in additional rounds equal to the number of sub-questions. As we cap sequential decomposition at four sub-questions, we normalize the turn cost for all executors to a range between 0 and 1, as shown in Table 5:

Table 5: $Turn_{\text{cost}}$ for different workflows:

Workflow	**Turn Cost**
QDP	0.25
QDS	0.25, 0.5, 0.75, 1.0
QR, RA, AG	0
RA, DS, AG	0
AS	0
Other Workflows	0

- $\mathbb{I}(S)$: This term penalizes whether the workflow invokes the retrieval API. Since the cost of calling the search engine's API is relatively high, this penalty term is included to encourage the planner to minimize the expenses associated with using the search engine. Specifically, if the workflow output by the planner includes a Retrieval Agent (RA), we have $\mathbb{I}(S) = 1$; otherwise, we set $\mathbb{I}(S) = 0$.

D Action Space and Workflow Validity

In Equation 2, we define the Planner action space as:

$$\mathcal{A}_{planner} = \{\texttt{QDS}, \texttt{QDP}, \texttt{QR}, \texttt{RA}, \texttt{DS}, \texttt{AG}, \texttt{AS}\}. \tag{13}$$

This action space should be interpreted as the vocabulary of executable symbols rather than a set of single-step discrete actions. In practice, the Planner is implemented as an autoregressive language model. Given an observation, it generates a structured response string, which is parsed into a list of executor symbols. A complete Planner response therefore corresponds to a workflow, i.e., a valid combination of executor symbols. Enumerating the validator rules gives 67 valid initial-step workflow strings and 65 valid subsequent-step strings. A complete episode can still contain multiple such decisions because decomposition creates sub-questions and the Planner is invoked again for each resulting state.

For example, `QDS` denotes invoking the Serial Query Decomposition agent, while `QDP` denotes invoking the Parallel Query Decomposition agent. A workflow such as `QR, RA, DS, AG` represents a non-decomposition retrieval-augmented workflow consisting of query rewriting, retrieval, document selection, and answer generation. Similarly, `QR, DS, AG` represents a workflow that rewrites the query, selects documents from the available context, and then generates the answer. The workflow is executed only if it satisfies the validity constraints described below.

Since the Planner generates workflow strings, not arbitrary natural-language responses, we apply a deterministic parser and validity checker to each generated workflow. The validity checker maps each executor symbol to its corresponding executor agent:

$$\begin{aligned}
\texttt{QR} &\mapsto \text{QueryRewriteAgent},\\
\texttt{QDP} &\mapsto \text{QueryDecompositionAgentParallel},\\
\texttt{QDS} &\mapsto \text{QueryDecompositionAgentSerial},\\
\texttt{RA} &\mapsto \text{RetrievalAgent},\\
\texttt{DS} &\mapsto \text{DocumentSelectionAgent},\\
\texttt{AG} &\mapsto \text{AnswerGenerationAgent},\\
\texttt{AS} &\mapsto \text{AnswerSummarizationAgent}.
\end{aligned} \tag{14}$$

A generated workflow is considered valid only if it satisfies the following constraints:

- **Non-empty workflow**: the parsed workflow list must not be empty.
- **No repeated executor**: each executor symbol can appear at most once in a single Planner response.
- **Executor vocabulary constraint**: every symbol in the workflow must belong to the predefined executor vocabulary in Equation 2.
- **Decomposition constraint**: if the workflow contains `QDS` or `QDP`, then the workflow must contain only that decomposition executor. In other words, `QDS` and `QDP` are treated as high-level decomposition actions. After decomposition, the Planner is called again for each generated sub-question to decide its corresponding execution workflow.
- **Answer generation constraint**: whether `QDS` or `QDP` is used is determined over the complete case (episode), rather than only the current Planner response. If neither decomposition executor is invoked anywhere in the episode, the workflow must contain `AG`, and `AG` must be its final executor. In an episode that invokes `QDS` or `QDP`, a subsequent Planner response may consist solely of `AS` to aggregate the completed sub-question answers; this response need not contain `AG` or repeat the earlier decomposition executor.
- **No repeated decomposition after planning begins**: if the current state has already entered a subsequent planning step, i.e., $\texttt{begin_step} \geq 0$, then `QDS` and `QDP` are no longer allowed. This prevents recursive or repeated decomposition of already generated sub-questions.

If any of the above constraints is violated, the workflow is marked as invalid and receives a format penalty. Specifically, the format penalty is defined as:

$$R_{FP} = \mathbb{I}[\text{workflow is invalid}], \tag{15}$$

where $R_{FP} = 0$ if the generated workflow is valid and executable, and $R_{FP} = 1$ otherwise. This penalty encourages the Planner to generate structurally valid workflow strings during reinforcement learning. A summary of these rules is provided in Table 6.

Table 6: Workflow validity rules used to check Planner-generated workflow strings.

Rule	Validity constraint
Non-empty workflow	The parsed workflow list w must not be empty.
No repeated executor	Each executor symbol can appear at most once in a single Planner response.
Executor vocabulary	Every executor symbol $e \in w$ must belong to $\mathcal{V}_{exec} = \{\texttt{QR}, \texttt{QDP}, \texttt{QDS}, \texttt{RA}, \texttt{DS}, \texttt{AG}, \texttt{AS}\}$.
Decomposition action	If QDP or QDS appears in w, then $\lvert w\rvert = 1$. That is, QDP and QDS are treated as high-level decomposition actions.
Answer generation	If neither QDP nor QDS is invoked anywhere in the complete case (episode), then AG must appear in w and must be the final executor. This condition is evaluated over the entire episode, not only w. In a decomposed episode, $w = [\texttt{AS}]$ is allowed for final aggregation after the sub-questions have been answered.
No repeated decomposition	If the current state has already entered a subsequent planning step, i.e., $c[\texttt{begin_step}] \geq 0$, then QDP and QDS are not allowed.

E Baseline Details

We compare MAO-ARAG with a broad set of representative baselines, which are grouped into three categories according to their workflow design.

Standard Baselines. **LLM w/o RAG** directly answers the input question using the parametric knowledge of the backbone LLM, without accessing external documents. **Vanilla RAG** follows the conventional retrieve-then-generate paradigm, where retrieved documents are directly provided to the LLM for answer generation without additional intermediate modules.

Static Workflow RAG. These methods use fixed and manually designed RAG pipelines. **RRR** Ma et al. (2023) adopts a rewrite-retrieve-read workflow, where the input query is first reformulated before retrieval and answer generation. **BGM** Ke et al. (2024) introduces a document selection module to filter noisy retrieved candidates before generation. **MMOA-RAG** Chen et al. (2025) uses a fixed linear workflow consisting of query rewriting, retrieval, document selection, and answer generation. Although these methods improve the standard RAG pipeline with additional modules, their execution structures remain static across different queries.

For a fair comparison, we re-implement all baselines with GPT-4o-mini as the backbone, except Search-R1. Because both methods perform multi-turn search during reasoning, we follow the original Search-R1 setting and re-implement it using Qwen2.5-7B-Instruct with RL reasoning training.

F Effect of Different Cost Weight α (RQ.2)

Within MAO-ARAG, the reward is defined in Equation (6), where α weights the cost penalty. Each reported α value (0, 0.1, 0.2, and 0.25) was obtained from a separate PPO training job using the normalized, equally weighted cost terms in Equation (4); α was not changed after training at inference time. The exploratory coefficient settings in the development script were not used for the reported results. As α increases, the cost penalty becomes stronger, which can reduce answer quality while lowering token, retrieval, and turn costs.

Figure 5 plots F1 alongside three cost indicators under varying α values. The trend indicates that larger α values are associated with lower F1 and lower reported costs in this evaluation. When α exceeds 0.2, a sharp decline in the plotted performance metrics occurs; this may reflect the stronger relative weight of the cost penalty and a shift toward simpler workflows.

Additionally, it can be observed that the curves in Figure 5 exhibit fluctuations. This may be due to the following reasons: (1) The limited number of executors defined may mean that the optimal

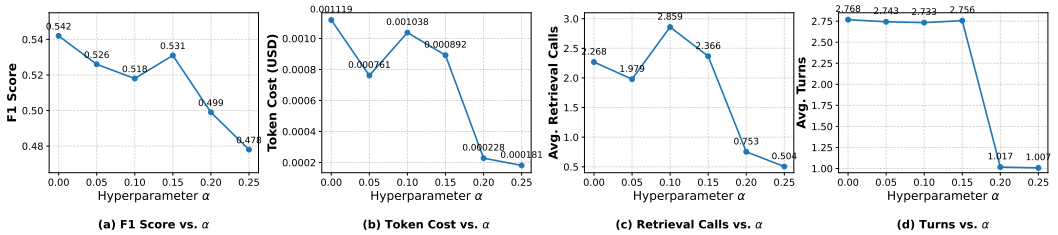


Figure 5: Impact of the cost penalty weight α on performance and efficiency metrics. Data is averaged across NQ and HotpotQA datasets. As α increases, the Planner generally uses fewer resources (Token Cost, Retrieval Calls, and Turns), suggesting that α can regulate the reported effectiveness–efficiency trade-off.

workflow is composed of fewer executors, introducing significant uncertainty and causing fluctuations. (2) To simplify the definition of the Cost Penalty term R_{CP} in Equation (4), we scaled the token cost $Token_{cost}$, turn numbers cost $Turn_{cost}$, and search engine call cost $\mathbb{I}(S)$ to a range of $[0, 1]$. However, there might be inherent weights among different cost penalty terms that ought to be considered. This coarse definition of the Cost Penalty term R_{CP} could also contribute to the observed fluctuations, indicating a potential area for future refinement.

G Training and Inference Cost Accounting

The cost numbers in the main paper measure inference-time API usage. The Planner is trained for one epoch on 2,400 NQ and 4,800 HotpotQA examples, while the Executor backbone and retriever are frozen during PPO. This separation is important: training is a one-time cost that can be amortized over a deployment workload, whereas token cost, retrieval calls, and turns are recurring per-query costs.

Table 7: Cost-accounting boundary for the reported experiments.

Component	Treatment in this paper
Planner training	One epoch; 2,400 NQ + 4,800 HotpotQA examples; policy update only
Executor training	Frozen; no Executor fine-tuning cost
Inference API cost	Reported per query in Table 2 and Figure 4
Retrieval and turn cost	Reported per query and included in the reward penalty
GPU-hour and energy cost	Not recorded in the retained experiment log; excluded rather than estimated

For a workload of N queries, the total cost is $C_{\text{total}}(N) = C_{\text{train}} + NC_{\text{infer}}$. The break-even workload against a training-free baseline with per-query cost C_{base} is $N^* = C_{\text{train}}/(C_{\text{base}} - C_{\text{infer}})$ when $C_{\text{base}} > C_{\text{infer}}$. Because C_{train} was not logged in GPU-hours, we leave this expression parameterized instead of reporting a fabricated dollar value. This also clarifies that the Pareto curves in Figure 4 compare recurring inference costs only.

H Prompt Details

Tables 8 through 13 present the prompt templates used by each agent in the MAO-ARAG framework. Specifically, Table 8 gives the prompt for the **Query Decomposition Serial (QDS)** agent; Tables 9–13 give the corresponding prompts for QDP, QR, DS, AG, and AS.

I LIMITATIONS

Since MAO-ARAG requires training the Planner with RL, the Planner backbone must follow instructions and have a basic ability to emit valid workflows. In our experiments, direct training with Qwen2.5-0.5B-Instruct and Qwen2.5-1.5B-Instruct was unsuccessful because these models struggled with instruction following.

The current experiments evaluate system-level performance but do not isolate the contribution of per-query orchestration relative to simpler routing strategies under matched settings. The executor backbone also has a large effect on the quality ceiling, as shown by the backbone ablation. Finally, the retained runs do not provide training-seed variance or significance tests; these are important follow-up evaluations. We therefore avoid claiming that the current aggregate statistics alone establish a per-query advantage.

J CASE STUDY

To illustrate representative behaviors of the proposed MAO-ARAG framework, we present qualitative case studies showing how the system generates workflows for different query types. Each case is structured as follows: we begin with a user query and its corresponding golden answer. Then, we detail each interaction turn within the MAO-ARAG framework. At each turn, the **Planner** selects a workflow by orchestrating a set of **Executors**. The selected workflow is then executed, and the resulting context—including sub-questions, intermediate answers, and retrieved documents—is accumulated. The process continues iteratively until a final answer is produced, at which point it is compared against the ground-truth answer.

We present four representative cases to illustrate workflow behavior on simple and complex queries; these qualitative examples are not intended as a systematic evaluation.

Case 1: Single-Turn Answer Generation Query: Is aluminium a ferrous or non-ferrous metal? (From NQ)

This is a straightforward factual question that can be confidently answered from the language model's internal knowledge. The case was generated by the PPO Planner used for the main results (Qwen2.5-7B-Instruct, $\alpha = 0$). The Planner correctly identifies that no retrieval or decomposition is necessary and directly selects the **AG** module in a single turn. The model produces the correct answer **non-ferrous**, which matches the ground-truth answer in this example. This case exemplifies MAO-ARAG's ability to avoid unnecessary computations and costs for simple queries.

Single-Turn Answer Generation

- **Initial question** q:
  - Is aluminium a ferrous or non-ferrous metal?
- **Golden answer** Ans_{golden}:
  - non-ferrous
- **Turn 0**:
  - **Planner: AG**
  - **Executor: AG**
  - **Context:**
    - **Question:** Is aluminium a ferrous or non-ferrous metal?
    - **Answer:** non-ferrous metal
- **Predicted answer** Ans_{predict}:
  - non-ferrous metal ✓

Case 2: Single-Turn Retrieval-Augmented Generation Query: Who was the editor of the newspaper Jugantar during the Swadeshi movement? (From NQ)

This question requires external knowledge not reliably stored in the model's parameters. The Planner selects a one-turn plan involving retrieval followed by generation (**RA** $\rightarrow$ **AG**). The retrieved documents contain relevant historical context, enabling the model to identify **Bhupendranath Dutt** in this example.

Single-Turn Retrieval-Augmented Generation

- **Initial question** q:
  - Who was the editor of the newspaper Jugantar during the Swadeshi movement?
- **Golden answer** Ans_{golden}:
  - Bhupendranath Dutt
- **Turn 0**:
  - **Planner: RA, AG**
  - **Executor: RA**
  - **Context:**
    - **Question:** Who was the editor of the newspaper Jugantar during the Swadeshi movement?
    - **Documents:**
      - Jugantar Patrika was a Bengali revolutionary newspaper founded in 1906 in Calcutta by Barindra Kumar Ghosh, Abhinash Bhattacharya, and **Bhupendranath Dutt**; it served as a political weekly and an organ of the Anushilan Samiti...
  - **Executor: AG**
  - **Context:**
    - **Question:** Who was the editor of the newspaper Jugantar during the Swadeshi movement?
    - **Documents:**
      - Jugantar Patrika was a Bengali revolutionary newspaper founded in 1906 in Calcutta by Barindra Kumar Ghosh, Abhinash Bhattacharya, and **Bhupendranath Dutt**; it served as a political weekly and an organ of the Anushilan Samiti...
    - **Answer:** Bhupendranath Dutt
- **Predicted answer** Without RAG:
  - Bepin Chandra Pal ✗
- **Predicted answer** With RAG (RA + AG) Ans_{predict}:
  - Bhupendranath Dutt ✓

Case 3: Multi-Turn Parallel Query Decomposition Query: Which performance act has a higher instrument-to-person ratio, Badly Drawn Boy or Wolf Alice? (From HotpotQA)

This is a complex comparative question requiring reasoning over multiple independent facts. The planner decomposes the main query into four focused sub-questions: (1) How many members are in the performance act Badly Drawn Boy?, (2) How many instruments are typically used in a performance by Badly Drawn Boy?, (3) How many members are in the performance act Wolf Alice?, and (4) How many instruments are typically used in a performance by Wolf Alice? These sub-queries are processed in parallel, with the planner independently determining whether external retrieval is necessary for each. Once all intermediate results are obtained, the planner invokes the **AS** module to compute the instrument-to-person ratios and synthesize a comparative answer. This case illustrates MAO-ARAG's ability to perform fine-grained parallel reasoning and adapt retrieval strategies to the needs of each sub-question.

Multi-Turn Parallel Query Decomposition

- **Initial question** q:
  - Which performance act has a higher instrument-to-person ratio, Badly Drawn Boy or Wolf Alice?
- **Golden answer** Ans_{golden}:
  - Badly Drawn Boy
- **Turn 0**:
  - **Planner: QDP**
  - **Executor: QDP**
  - **Context:**
    - **Question:** Which performance act has a higher instrument-to-person ratio, Badly Drawn Boy or Wolf Alice?
    - **Sub-questions** $subq$:
      - Sub-question 1: How many members are in the performance act Badly Drawn Boy?
      - Sub-question 2: How many instruments are typically used in a performance by Badly Drawn Boy?
      - Sub-question 3: How many members are in the performance act Wolf Alice?
      - Sub-question 4: How many instruments are typically used in a performance by Wolf Alice?
- **Turn 1**:
  - **planner: AG (parallel for sub-question 1)**
  - **Executor: AG**
  - **Context:**
    - **Question:** Which performance act has a higher instrument-to-person ratio, Badly Drawn Boy or Wolf Alice?
    - **Sub-questions** $subq$**:**

- Sub-question 1: How many members are in the performance act Badly Drawn Boy?
- Sub-question 2: How many instruments are typically used in a performance by Badly Drawn Boy?
- Sub-question 3: How many members are in the performance act Wolf Alice?
- Sub-question 4: How many instruments are typically used in a performance by Wolf Alice?

* **Documents:**
  - Documents for Sub-question 1: None
* **Sub-answers $suba$:**
  - Sub-answer 1: One member

– **planner: AG (parallel for sub-question 2)**
– **Executor: AG**
– **Context:**
* **Question:** Which performance act has a higher instrument-to-person ratio, Badly Drawn Boy or Wolf Alice?
* **Sub-questions $subq$:**
  - Sub-question 1: How many members are in the performance act Badly Drawn Boy?
  - Sub-question 2: How many instruments are typically used in a performance by Badly Drawn Boy?
  - Sub-question 3: How many members are in the performance act Wolf Alice?
  - Sub-question 4: How many instruments are typically used in a performance by Wolf Alice?
* **Documents:**
  - Documents for Sub-question 1: None
  - Documents for Sub-question 2: None
* **Sub-answers $suba$:**
  - Sub-answer 1: One member
  - Sub-answer 2: Typically four instruments are used in a performance by Badly Drawn Boy.

– **planner: AG (parallel for sub-question 3)**
– **Executor: AG**
– **Context:**
* **Question:** Which performance act has a higher instrument-to-person ratio, Badly Drawn Boy or Wolf Alice?
* **Sub-questions $subq$:**
  - Sub-question 1: How many members are in the performance act Badly Drawn Boy?
  - Sub-question 2: How many instruments are typically used in a performance by Badly Drawn Boy?
  - Sub-question 3: How many members are in the performance act Wolf Alice?
  - Sub-question 4: How many instruments are typically used in a performance by Wolf Alice?
* **Documents:**
  - Documents for Sub-question 1: None
  - Documents for Sub-question 2: None
  - Documents for Sub-question 3: None
* **Sub-answers $suba$:**
  - Sub-answer 1: One member
  - Sub-answer 2: Typically four instruments are used in a performance by Badly Drawn Boy.
  - Sub-answer 3: Four

– **planner: RA, AG (parallel for sub-question 4)**
– **Executor: RA**
– **Context:**
* **Question:** Which performance act has a higher instrument-to-person ratio, Badly Drawn Boy or Wolf Alice?
* **Sub-questions $subq$:**
  - Sub-question 1: How many members are in the performance act Badly Drawn Boy?
  - Sub-question 2: How many instruments are typically used in a performance by Badly Drawn Boy?
  - Sub-question 3: How many members are in the performance act Wolf Alice?
  - Sub-question 4: How many instruments are typically used in a performance by Wolf Alice?
* **Documents:**
  - Documents for Sub-question 1: None
  - Documents for Sub-question 2: None
  - Documents for Sub-question 3: None
  - Documents for Sub-question 4: Retrieved evidence omitted for brevity.
* **Sub-answers $suba$:**
  - Sub-answer 1: One member
  - Sub-answer 2: Typically four instruments are used in a performance by Badly Drawn Boy.
  - Sub-answer 3: Four

– **Executor: AG**
– **Context:**
* **Question:** Which performance act has a higher instrument-to-person ratio, Badly Drawn Boy or Wolf Alice?
* **Sub-questions $subq$:**
  - Sub-question 1: How many members are in the performance act Badly Drawn Boy?
  - Sub-question 2: How many instruments are typically used in a performance by Badly Drawn Boy?
  - Sub-question 3: How many members are in the performance act Wolf Alice?
  - Sub-question 4: How many instruments are typically used in a performance by Wolf Alice?
* **Documents:**

- Documents for Sub-question 1: None
- Documents for Sub-question 2: None
- Documents for Sub-question 3: None
- Documents for Sub-question 4: Retrieved evidence omitted for brevity.

* **Sub-answers $suba$:**
- Sub-answer 1: One member
- Sub-answer 2: Typically four instruments are used in a performance by Badly Drawn Boy.
- Sub-answer 3: Four
- Sub-answer 4: Four instruments

- **Turn 2**:
  - **Planner: AS**
  - **Executor: AS**
  - **Context:**
    * **Question:** Which performance act has a higher instrument-to-person ratio, Badly Drawn Boy or Wolf Alice?
    * **Sub-questions $subq$:**
      - Sub-question 1: How many members are in the performance act Badly Drawn Boy?
      - Sub-question 2: How many instruments are typically used in a performance by Badly Drawn Boy?
      - Sub-question 3: How many members are in the performance act Wolf Alice?
      - Sub-question 4: How many instruments are typically used in a performance by Wolf Alice?
    * **Documents:**
      - Documents for Sub-question 1: None
      - Documents for Sub-question 2: None
      - Documents for Sub-question 3: None
      - Documents for Sub-question 4: Retrieved evidence omitted for brevity.
    * **Sub-answers $suba$:**
      - Sub-answer 1: One member
      - Sub-answer 2: Typically four instruments are used in a performance by Badly Drawn Boy.
      - Sub-answer 3: Four
      - Sub-answer 4: Four instruments
    * **Answer:** Badly Drawn Boy
- **Predicted final answer** Ans_{predict}:
  - Badly Drawn Boy ✓

Case 4: Multi-Turn Sequential Query Decomposition Query: Ralph Hefferline was a psychology professor at a university that is located in what city? (From HotpotQA)

This is a compositional question requiring sequential reasoning. The planner decomposes the query into two dependent sub-questions: (1) At which university was Ralph Hefferline a psychology professor?, followed by (2) In what city is this university located? The second sub-question is dynamically rewritten based on the first answer. After each planning phase, the appropriate modules are invoked (**RA** and **AG** as needed), and the answers are accumulated. Once both sub-questions are resolved, the **AS** module generates the final answer. This case illustrates sequential dependencies and sub-question reformulation in one qualitative example.

Multi-Turn Sequential Query Decomposition

- **Initial question** q:
  - Ralph Hefferline was a psychology professor at a university that is located in what city?
- **Golden answer** Ans_{golden}:
  - New York City
- **Turn 0**:
  - **Planner: QDS**
  - **Executor: QDS**
  - **Context:**
    * **Question:** Ralph Hefferline was a psychology professor at a university that is located in what city?
    * **Sub-questions $subq$:**
      - Sub-question 1: At which university was Ralph Hefferline a psychology professor?
      - Sub-question 2: In what city is this university located?
- **Turn 1**:
  - **Planner: RA, AG**
  - **Executor: RA**
  - **Context:**
    * **Question:** Ralph Hefferline was a psychology professor at a university that is located in what city?
    * **Sub-questions $subq$:**
      - Sub-question 1: At which university was Ralph Hefferline a psychology professor?
      - Sub-question 2: In what city is this university located?

        - ∗ **Documents:**
            - Documents for Sub-question 1: Ralph Franklin Hefferline was a psychology professor at **Columbia University**.
    - **Executor: AG**
    - **Context:**
        - ∗ **Question:** Ralph Hefferline was a psychology professor at a university that is located in what city?
        - ∗ **Sub-questions** $subq$:
            - Sub-question 1: At which university was Ralph Hefferline a psychology professor?
            - Sub-question 2: In what city is this university located?
        - ∗ **Documents:**
            - Documents for Sub-question 1: Ralph Franklin Hefferline was a psychology professor at **Columbia University**.
        - ∗ **Sub-answers** $suba$**:**
            - Sub-answer 1: Columbia University
- **Turn 2**:
    - **Planner: AG**
    - **Executor: AG**
    - **Context:**
        - ∗ **Question:** Ralph Hefferline was a psychology professor at a university that is located in what city?
        - ∗ **Sub-questions** $subq$:
            - Sub-question 1: At which university was Ralph Hefferline a psychology professor?
            - Sub-question 2: In what city is this university located?
        - ∗ **Documents:**
            - Documents for Sub-question 1: Ralph Franklin Hefferline was a psychology professor at **Columbia University**.
            - Documents for Sub-question 2: None
        - ∗ **Sub-answers** $suba$**:**
            - Sub-answer 1: Columbia University
            - Sub-answer 2: New York City
- **Turn 3**:
    - **Planner: AS**
    - **Executor: AS**
    - **Context:**
        - ∗ **Question:** Ralph Hefferline was a psychology professor at a university that is located in what city?
        - ∗ **Sub-questions** $subq$:
            - Sub-question 1: At which university was Ralph Hefferline a psychology professor?
            - Sub-question 2: In what city is this university located?
        - ∗ **Documents:**
            - Documents for Sub-question 1: Ralph Franklin Hefferline was a psychology professor at **Columbia University**.
            - Documents for Sub-question 2: None
        - ∗ **Sub-answers** $suba$**:**
            - Sub-answer 1: Columbia University
            - Sub-answer 2: New York City
        - ∗ **Answer:** New York City
- **Predicted final answer** Ans_{predict}:
    - New York City ✓

K Detailed Values of Different Cost Metrics

The detailed per-dataset cost tables are temporarily omitted while the underlying logs are being rechecked. The aggregate values used in the main text and Figure 4 remain unchanged; the tables will be restored after verification.

Table 8: The prompt of Query Decomposition Serial Agent.

system: You are a professional assistant skilled at decomposing complex questions into a minimal sequence of logically dependent, independently searchable sub-questions. Each sub-question must:
- Be self-contained and specific
- Be suitable for direct information retrieval from search engines or structured databases
- Be strictly necessary to answer the original question
You must keep the number of sub-questions as low as possible, and never exceed 4 in total. Avoid redundancy and do not include trivial or overly granular sub-questions.

assistant: Understood. I will return only factual, retrievable sub-questions, one per line.

user: Original question is: {content of Question}.
Now decompose the original question into a logically ordered list of sub-questions.
Do not number the sub-questions, write one sub-question per line.

Table 9: The prompt of Query Decomposition Parallel Agent.

system: You are a professional assistant skilled at decomposing complex multi-entity or multi-location questions into multiple independent and searchable sub-questions. Each sub-question should be specific, logically complete, and not repeat others.

assistant: Okay, I will return the parallel sub-questions.

user: Original question is {content of Question}.
Break down this question into the minimum number of specific, logically complete, and independently searchable sub-questions needed to fully understand and answer the original question. Do not generate more than 4 sub-questions. Each sub-question should be on a separate line, avoid vague demonstratives or repetition, and ensure that each question is self-contained.

Table 10: The prompt of Query Rewriter Agent.

system: You are a professional assistant skilled at rewriting overly detailed or redundant questions into a single, concise, and searchable query. Your goal is to keep only the essential part of the question that is needed to find the answer efficiently.

assistant: Okay, I will return a concise rewritten query.

user: Original question is: {content of Question}.
Now rewrite the original question into a single, clear query that focuses only on the essential information needed to find the answer. Avoid unnecessary context, vague references, and maintain specificity. Output only the rewritten query without any extra explanation or formatting.

Table 11: The prompt of Document Selector Agent.

system: You are a helpful, respectful and honest assistant. Your task is to output the ID of the candidate Documents (0, 1, 2,..., n) which are helpful in answering the Question.

assistant: Okay, I will provide the ID of candidate Documents which are helpful in answering the Question.

user: Question is: {content of Question}
{content of Documents}

assistant: OK, I received the Question and the candidate Documents.

user: Now, output the ID of the candidate Documents (0,1,2,...,n) which are helpful in answering the Question: {content of Question}, for example, in the following format: Document0,Document4,Document6,Document7.

Table 12: The prompt of Answer Generator Agent.

system: You are a helpful, respectful and honest assistant. Your task is to predict the answer to the question based on the given documents. If you don't know the answer to a question, please don't share false information. Answer the question as accurately as possible.

assistant: Okay, I will provide the answer to the question based on the corresponding documents. Please provide the question and the corresponding documents.

user: Question is: {content of Question}
{content of Documents}
Now, answer the Question: {content of Question}, based on the above Documents.
assistant: OK, I received the Question and the corresponding Documents.

user: Given the Question and the corresponding Documents, predict the answer to the Question as briefly and accurately as possible based on the Documents. Only give the brief and accurate answer with the form of **answer**.

Table 13: The prompt of Answer Summarization Agent.

system: You are a helpful, respectful and honest assistant. Your task is to predict the final answer to the original question based on the answers to its decomposed sub-questions. If you are not sure about the final answer, do not make up information. Give the most accurate and concise answer possible based on the sub-question answers.

assistant: Okay, I will provide the final answer to the original question based on the sub-questions and their corresponding answers. Please provide the original question, the sub-questions, and their answers.

user: Original Question:{content of Question}
{content of Context}
Now, based on the above sub-questions and their answers, answer the Original Question: {content of Question}
assistant: OK, I received the Original Question, its Sub-questions, and their Answers.

user: Given the Original Question, the Sub-questions and their Answers, predict the final answer to the Original Question as briefly and accurately as possible. Only give the brief and accurate answer in the form of **answer**.