SPARC-RAG: Adaptive Sequential–Parallel Scaling with Context Management for Retrieval-Augmented Generation

Anonymous ACL submission

Abstract

Retrieval-Augmented Generation (RAG) grounds large language model outputs in external evidence, but remains challenged on multi-hop question answering that requires long reasoning. Recent works scale RAG at inference time along two complementary dimensions: sequential depth for iterative refinement and parallel width for coverage expansion. However, naive scaling causes context contamination and scaling inefficiency, leading to diminishing or negative returns despite increased computation. To address these limitations, we propose SPARC-RAG, a framework that coordinates joint depth–width scaling around a *unified context management mechanism*—a shared global context state on which specialized control agents regulate parallel exploration and sequential stopping. To optimize scaling behavior, we further introduce a lightweight fine-tuning method with process-level verifiable preferences, which improves the efficiency of sequential scaling and effectiveness of parallel scaling. Across single- and multi-hop QA benchmarks, SPARC-RAG consistently outperforms previous RAG baselines, yielding an average +6.2 F1 improvement under lower inference cost.

1 Introduction

Large Language Models (LLMs) (Yang et al., 2025; OpenAI, 2023) augmented with retrieval have become a dominant approach for knowledge-intensive tasks. Retrieval-Augmented Generation (RAG) (Lewis et al., 2020) grounds LLM outputs in external evidence, providing access to current and factual information. Despite these advantages, standard RAG systems with a single retrieval and question answering step often struggle on complex queries, particularly multi-hop queries (Gao et al., 2023; Tang and Yang, 2024) which demand deep reasoning and hypothesis exploration. To handle these intricate demands, recent works focus on scaling RAG at inference time along two dimensions: deepening the reasoning chain (sequential scaling or depth) (Trivedi et al., 2023; Wang et al., 2025b; Lee et al., 2024) and broadening hypothesis exploration (parallel scaling or width) (Wang et al., 2025d; Li et al., 2024).

However, scaling is not straightforward: mechanically increasing depth and width results in redundant computation and thus *Scaling Inefficiency*, while uncontrolled context growth introduces noise and leads to *Context Contamination*; together, these effects cause performance to plateau and place the system on a suboptimal cost–accuracy trade-off curve (Leng et al., 2024). More concretely, sequential scaling iteratively refines reasoning but suffers from context contamination as evidence accumulates indiscriminately, overwhelming the model's attention capacity and deteriorating performance (Wu et al., 2025; Inoue et al., 2025). Parallel scaling increases coverage by exploring multiple reasoning paths, but it produces redundant branches and relies on simple aggregation schemes (e.g., majority voting) that fail to synthesize complementary evidence (Wang et al., 2025c; Xiao et al., 2025).

Although sequential depth and parallel width are intrinsically complementary — the former enables iterative refinement while the latter expands evidence coverage — this complementarity is difficult to realize. Without explicit regulation of scaling behavior and information consolidation, their interaction amplifies redundancy and noise, undermining both efficiency and answer quality. This highlights the need for a framework that coordinates sequential refinement and parallel exploration under a unified context management mechanism.

Building on these insights, we propose SPARC-RAG (Adaptive **S**equential–**Par**allel Scaling with **C**ontext Management for Retrieval-Augmented Generation), a multi-agent framework that explicitly controls reasoning width, depth, and context

during inference-time scaling. Instead of indiscriminately increasing computation, SPARC-RAG organizes scaling around a *unified context management mechanism*: a CONTEXT MANAGER maintains a shared global context across parallel branches and sequential rounds, on which two specialized control agents (QUERY REWRITER, ANSWER EVALUATOR) regulate parallel exploration and sequential stopping. Together, these agents enable SPARC-RAG to achieve better accuracy–cost trade-offs than prior sequential-only or parallel-only approaches.

Furthermore, we design a lightweight fine-tuning strategy that leverages process-level preference pairs derived from scaling behaviors. These preference pairs guide the model toward more effective and efficient scaling by reducing redundant expansions and learning a more accurate stopping policy. We summarize our contributions as follows:

❶ **Agent design principles for efficient scaling.** We identify *Context Contamination* and *Scaling Inefficiency* as the key challenges in inference-time RAG scaling and formulate agent-level design principles for mitigating them, including enhanced exploration diversity and context management across sequential depth and parallel width.

❷ **Unified context management for joint depth–width scaling.** We introduce SPARC-RAG, which organizes depth–width scaling around a shared global context state maintained by a CONTEXT MANAGER, on which specialized control agents regulate parallel exploration and sequential stopping—generalizing prior sequential-only and parallel-only methods while achieving superior accuracy–cost trade-offs.

❸ **Scaling-oriented fine-tuning.** We design a scaling-oriented fine-tuning strategy that supervises the scaling process. Preference pairs are constructed from verifiable intermediate metrics (evidence coverage and stopping accuracy), allowing the model to internalize when scaling is beneficial. This process-level calibration enhances both effectiveness and efficiency without heavy fine-tuning.

2 Background and Related Work

We review sequential scaling and parallel scaling in RAG, along with recent work that introduces limited adaptive control over these processes.

Sequential Scaling in RAG. Sequential RAG methods iteratively refine reasoning through multi-round retrieval and generation. IRCoT (Trivedi et al., 2023) interleaves retrieval and chain-of-thought reasoning via iterative query reformulation. Iter-RetGen (Shao et al., 2023) alternates retrieval and generation to gradually surface evidence. Chain-of-RAG (Wang et al., 2025a) constructs stepwise retrieval chains with explicit supervision. These methods demonstrate the value of deeper reasoning but typically *rely on fixed iteration counts and accumulate evidence across steps*, which can lead to noisy context and deteriorated performance (context contamination) (Wu et al., 2025; Inoue et al., 2025).

Parallel Scaling in RAG. Parallel approaches expand coverage by exploring multiple reasoning paths in parallel. DMQR-RAG (Li et al., 2024) and MCTS-RAG (Hu et al., 2025) generate diverse query rewrites or reasoning branches to retrieve complementary evidence, while Speculative-RAG (Wang et al., 2025d) drafts multiple candidate answers for verification. However, these methods typically *use fixed branching factors and simple aggregation strategies* (e.g., majority voting), which limits cross-branch evidence synthesis (Wang et al., 2025c; Xiao et al., 2025).

Adaptive and Planning-Based RAG. Recent work introduces adaptive behaviors and planning into RAG pipelines. Self-RAG (Asai et al., 2024) and following works (Yan et al., 2024; Xu et al., 2025) evaluate retrieval quality to dynamically revise outputs. Planning-based approaches (Lee et al., 2024; Verma et al., 2024; Gu et al., 2025) generate structured plans before retrieval, while question-decomposition methods (Ammann et al., 2025) split complex queries into sub-questions and plan the following execution. Autonomous web-search agents such as InfoAgent (Zhang et al., 2025) integrate retrieval with long-horizon planning through task-specific post-training. DeepNote (Wang et al., 2025b) maintains a persistent note state across steps and employs a heuristic stopping rule. However, these approaches do not treat sequential depth and parallel width as *jointly controllable inference-time resources*, making scaling implicit and inefficient. This gap motivates a mechanism that coordinates sequential refinement, parallel exploration, and context evolution in a unified manner.

3 Methods

To address the *Context Contamination* and *Scaling Inefficiency* challenges identified in Section 1, we

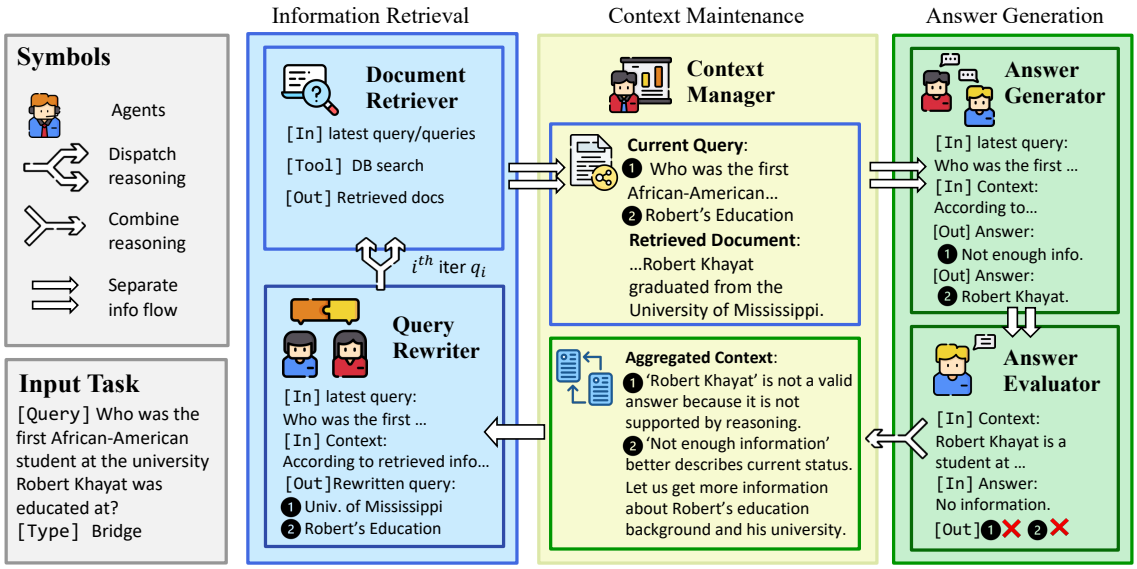

Figure 1: Overall SPARC-RAG framework. The figure illustrates the interactions, information flow, and decision signals across agents.

introduce SPARC-RAG, a multi-agent framework that provides explicit control over depth, width, and information consolidation during inference-time scaling. We begin by formally characterizing scaling in RAG.

3.1 Task Formulation & System Overview

Problem Formulation. We formulate the multi-hop QA task as a joint optimization process over Sequential Depth (D) and Parallel Width (W). Given an initial query $\boldsymbol{q}_0$ and an external corpus $\mathcal{C}$, SPARC-RAG models the reasoning process as a hierarchical composition of three core operators, denoted as $\Pi = S \circ P \circ I$ (formal formulation in Appendix B.1):

(i) **Single-path Inference Operator** (I)**:** Executes a standard "retrieve-reason-generate" operation, as in naive RAG.

(ii) **Parallel Expansion Operator** (P)**:** Dynamically generates $W^{(t)}$ parallel branches at the t-th inference step and aggregates information upon completion.

(iii) **Sequential Refinement Operator** (S)**:** Connects multiple parallel rounds in sequence, determining based on the current state whether to deepen the reasoning or terminate with the current answer.

Moving from the abstract formulation to its system-level realization, we provide overview of system in Figure 1 and rollout in 2. At round t, the QUERY REWRITER generates parallel rewritten queries that form independent branches, where the RETRIEVER gathers evidence, the CONTEXT MANAGER updates the shared context, and the ANSWER GENERATOR produces a candidate answer that is evaluated by the ANSWER EVALUATOR. The CONTEXT MANAGER then selects the best branch, merges evidence into a unified state, and decides whether to continue or stop reasoning based on the selected answer's evaluation. Together with standard RAG components, the CONTEXT MANAGER, QUERY REWRITER, and ANSWER EVALUATOR enable coordinated depth–width scaling with improved accuracy–cost trade-offs over prior sequential-only or parallel-only approaches.

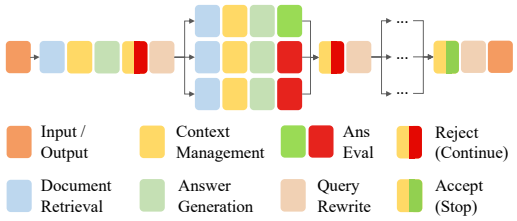

Figure 2: Single left-to-right rollout of the SPARC-RAG reasoning process. The Accept (Stop) and Reject (Continue) decisions are produced by the CONTEXT MANAGER after merging the branches.

3.2 Agents

We highlight that the architectural core of SPARC-RAG is a *shared global context* $m^{(t)}$, updated by the CONTEXT MANAGER, that is maintained across parallel branches within round t and carried forward across sequential rounds. The remaining agents operate as modular control signals on this state. Figure 2 illustrates the resulting select $\rightarrow$ retrieve/rewrite $\rightarrow$ merge $\rightarrow$ update loop.

RETRIEVER. Given the current query $\boldsymbol{q}^{(t)}$, the retriever selects a small subset of passages from the corpus $\mathcal{C}$ using a similarity-based retrieval function followed by top-k ranking. $\sigma^{(t)}$ denotes the retrieval strategy used at this round.

$$r^{(t)} \sim \mathsf{Retrieve}_{\sigma^{(t)}}\big(\boldsymbol{q}^{(t)},\, C\big).$$

ANSWER GENERATOR. Conditioned on the global query $\boldsymbol{q}_0$ and current memory $m^{(t)}$, the generator produces a candidate answer for this round using a language model.

$$a^{(t)} \sim \mathsf{AnsGenerate}_{\mathrm{LLM}}\big(\boldsymbol{q}_0,\, m^{(t)}\big).$$

ANSWER EVALUATOR. The evaluator evaluates the quality of the generated answer based on the query and current memory, generating a decision signal $e^{(t)}$ (e.g., `CONTINUE` or `STOP`).

$$e^{(t)} \sim \mathsf{EvaluateAns}_{\mathrm{LLM}}\big(\boldsymbol{q}_0,\, \boldsymbol{q}^{(t)},\, a^{(t)},\, m^{(t)}\big).$$

Crucially, this prevents redundant iterations on simple queries, dynamically allocating the computational budget only to complex instances, reducing *Scaling Inefficiency*.

CONTEXT MANAGER. This module lies in the center of the Context Management functionality. It maintains the evolving reasoning state by consolidating newly retrieved documents and generated candidate answers with the existing memory during scaling, mitigating *Context Contamination*. Within each branch, it performs a query-aware update that attends to the current question and retains only the relevant evidence.

$$m^{(t)} \sim \mathsf{MemUpdate}_{\mathrm{LLM}}\big(\boldsymbol{q}_0,\, \boldsymbol{q}^{(t)},\, m^{(t-1)},\, r^{(t)}, a^{(t)}\big).$$

This mechanism ensures the memory update is both **incremental** and **minimal**, thereby maintaining a consistent context window with low noise. After parallel branches complete, the CONTEXT MANAGER orchestrates the aggregation of branches: it selects the most promising reasoning path based on answer quality and evidence support, then consolidates complementary information from all branches into a unified context.

$$\begin{aligned} k^* &\leftarrow \mathsf{SelectBest}_{\mathrm{LLM}}\big(\{(a_k, m'_k)\}_{k=1}^{W}\big), \\ m^{(t)} &\leftarrow \mathsf{ContextMerge}_{\mathrm{LLM}}\big(m'_{k^*}, \{m'_k\}_{k\neq k^*}\big). \end{aligned}$$

These operations alleviate *Context Contamination* by maintaining a unified, query-focused memory state rather than collecting evidence without focus.

QUERY REWRITER. To operationalize the Parallel Expansion Operator (P) efficiently, this agent functions as a *one-to-many generator*. Distinct from standard rewriters that sequentially refine a single query, it synthesizes $W^{(t)}$ intent-specific queries in a *single inference pass*. Conditioned on the global question $\boldsymbol{q}_0$, current memory $m^{(t)}$, and the target width $W^{(t)}$, it explores orthogonal retrieval directions to maximize information coverage, alleviating the *Scaling Inefficiency*.

$$\begin{aligned} &\big(\{\sigma^{(t+1,k)}, q^{(t+1,k)}\}_{k=1}^{W^{(t)}}\big) \\ &\qquad \sim \mathsf{RewriteQuery}_{\mathrm{LLM}}\big(\boldsymbol{q}_0,\, \boldsymbol{q}^{(t)},\, m^{(t)}, W^{(t)}\big). \end{aligned}$$

3.3 Execution Protocol

We formally describe the execution flow as a dynamic execution loop (summarized in Algorithm 1). The process begins with the user query $\boldsymbol{q}_0$ and an empty memory state. SPARC-RAG regulates the consumption of computational resources ($\mathcal{B}$) through two mechanisms:

Parallel Distribution & Branch Execution. To explore the search space (Width W), the system first triggers the QUERY REWRITER to generate a set of intent-specific rewritten queries $\{q_k\}_{k=1}^{W}$. Crucially, each rewritten query spawns an independent **parallel execution branch**. Within each branch k, the agents perform a full update cycle:

(i) Retrieve: The branch retrieves raw evidence r_k relevant to the rewritten query.

(ii) Context Update: The CONTEXT MANAGER compresses r_k and integrates it with the previous history $m^{(t-1)}$ to form a branch-local memory m'_k.

(iii) Generate & Evaluate: Conditioned on updated state m'_k, answer a_k is generated and evaluated.

This phase yields W fully processed hypothesis tuples $\{(m'_k, a_k, e_k)\}_{k=1}^{W}$.

Aggregation & Global Control. Once the parallel branches complete, the system performs a select-and-decide operation:

(i) Selection and Consolidation: The CONTEXT MANAGER selects the most promising branch k^* from the candidate paths $\{(a_k, m'_k)\}_{k=1}^{W}$ and consolidates evidence into a unified memory, preserving informative signals while reducing noise.

(ii) Termination Decision: The system checks the decision signal of the selected winner (e_{k^*}). If e_{k^*} == `STOP` or the budget D_{max} is reached, the process terminates returning a_{k^*}. Otherwise, the loop continues to $t \leftarrow t + 1$.

Algorithm 1 SPARC-RAG Inference Algorithm

Require: Query q_0, Corpus $\mathcal{C}$, Max Depth $D_{\max}$

$m^{(0)} \leftarrow \emptyset,\ t \leftarrow 1$
while $t \leq D_{\max}$ **do**
  // Phase 1: Parallel Distribution
  $\{q_k\}_{k=1}^{W} \leftarrow \mathsf{RewriteQuery}(q_0, m^{(t-1)})$
  // Phase 2: Sequential Execution
  for all $k \in \{1, \ldots, W\}$ **in parallel do**
    $r_k \leftarrow \mathsf{Retrieve}(q_k, \mathcal{C})$
    $m'_k \leftarrow \mathsf{MemUpdate}(m^{(t-1)}, r_k)$
    $a_k \leftarrow \mathsf{AnsGenerate}(q_0, m'_k)$
    $e_k \leftarrow \mathsf{EvaluateAns}(a_k, m'_k)$
  // Phase 3: Aggregation & Control
  $k^* \leftarrow \mathsf{SelectBest}_{\mathrm{LLM}}\big(\{(a_k, e_k, m'_k)\}_{k=1}^{W}\big)$
  $m^{(t)} \leftarrow \mathsf{ContextMerge}(m'_{k^*}, \{m'_k\}_{k \neq k^*})$
  if $e_{k^*} = \mathsf{STOP}$ **then**
    return a_{k^*}
  $t \leftarrow t + 1$

3.4 Scaling-oriented Fine-tuning

To improve system performance without incurring high cost, we adopt a scaling-oriented, preference-based fine-tuning strategy. We construct preference pairs that reward higher combined paragraph recall for parallel scaling and more accurate stopping decisions for sequential scaling. These process-level rewards encourage the model to achieve broader evidence coverage and stop more reliably, improving scaling effectiveness and efficiency.

Preference Data Collection for the Query Rewriter. The QUERY REWRITER is supervised using paragraph-level retrieval recall. Rewrites that retrieve more gold-supporting evidence are preferred over those with lower recall. Although this supervision signal is purely recall-based, it implicitly encourages *diverse yet focused* sub-queries: improving recall typically requires accessing complementary evidence rather than duplicating the same retrieval trajectory. As a result, the trained rewriter reduces redundancy across parallel branches(Section 5.2), making width expansion more effective.

Preference Data Collection for the Answer Evaluator. We construct preference data by sampling multiple STOP/CONTINUE decisions of ANSWER EVALUATOR and assigning preferences based on the correctness of the corresponding answers.

A key challenge in supervising this decision is the asymmetry of error costs: a *wrong stop* (accepting an incorrect answer, exiting prematurely) is typically far more damaging than a *wrong continue* (rejecting a correct answer, continuing reasoning), as it eliminates the possibility of further corrections. To reflect this asymmetry, we introduce a weighting function $w(y^+, y^-)$ over preference pairs. For early-stop–critical cases in which the preferred action is continue but the rejected action is stop, we assign $w(y^+, y^-) = \lambda > 1$; all other preference pairs retain unit weight. The weighted DPO loss for ANSWER EVALUATOR is therefore

$$\mathcal{L}_{\mathrm{AE}}(x, y^+, y^-) = w(y^+, y^-)\Big[-\log \sigma\big(\beta[\log p_\theta(y^+ \mid x) - \log p_\theta(y^- \mid x)]\big)\Big],$$

where $\lambda > 1$ emphasizes penalties on incorrect early stopping while still allowing early termination on easy instances.

As we show in Section 5.2, targeted fine-tuning calibrates stopping behaviour while encouraging diverse parallel exploration.

4 Experiments

4.1 Datasets and Metrics

We evaluate on **Natural Questions** (NQ; Kwiatkowski et al. 2019) for single-hop QA, and on four multi-hop QA benchmarks: **HotpotQA** (Yang et al., 2018), **2WikiMultiHopQA** (**2WikiMQA**) (Ho et al., 2020), **Bamboogle** (Press et al., 2023), and **MuSiQue** (Trivedi et al., 2022). HotpotQA, 2WikiMultiHopQA, and Bamboogle require 2-hop reasoning, while MuSiQue covers 2–4 hops. The QUERY REWRITER and ANSWER EVALUATOR are trained on the corresponding training splits, and results are reported on 500-example subsampled test splits; see Appendix A.1.

We report Exact Match (EM), F1, and Accuracy following Vu and Moschitti (2021) and Asai et al. (2024), and additionally use paragraph recall to assess retrieval effectiveness. Cost is measured by total token usage (input + output) and latency.

4.2 Backbone Models

All agent components are instantiated using Qwen2.5-7B-Instruct (Qwen et al., 2025) and Qwen3-32B (Yang et al., 2025). These two backbones enable us to evaluate SPARC-RAG under different levels of reasoning capacity.

4.3 Retrievers

We use a hybrid retrieval stack for SPARC-RAG across all experiments, following common prac-

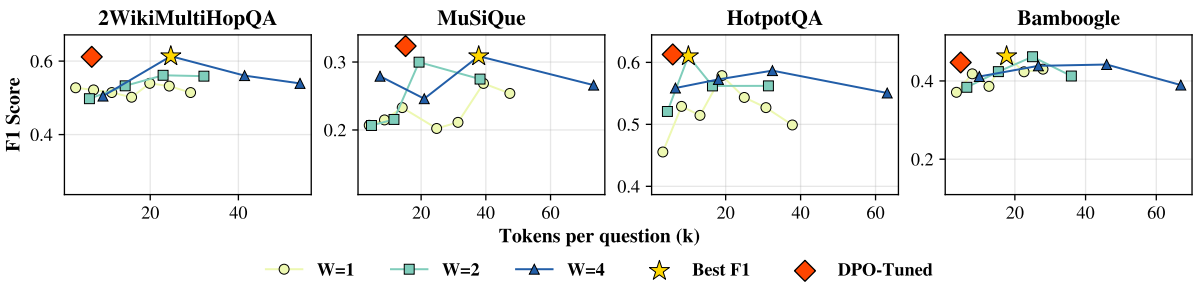

Figure 3: F1–cost tradeoff across (W, D) configurations with Qwen2.5-7B-Instruct model. Each panel plots F1 versus tokens per question for one dataset. Curves correspond to widths $W{=}1, 2, 4$. For $W{=}1$, the sequential depths are $D{=}2, 4, 6, 8, 10, 12, 14$. For $W{=}2$ and $W{=}4$, the depths are $D{=}2, 4, 6, 8$. Along each curve, points are ordered from left to right by increasing D. The gold star marks the best untuned (W, D) configuration, and the red diamond denotes our fine-tuned model, which achieves comparable or better F1 at substantially lower token cost.

tice in RAG systems (Thakur et al., 2021; Seo et al., 2019; Zhao et al., 2026). By default, SPARC-RAG agentically selects between sparse and dense retrievers; details are provided in Appendix A.3. Since most baselines support only a single retrieval method, Appendix C.8 reports single-retriever ablations, showing that this restriction does not degrade performance and preserves fair comparison with the baselines.

4.4 Baselines

We compare SPARC-RAG against strong inference-time scaling paradigms in RAG: (i) sequential refinement methods IRCoT (Trivedi et al., 2023), Iter-RetGen (Shao et al., 2023), and DeepNote (Wang et al., 2025b); (ii) sequential self-reflective correction method Self-RAG (Asai et al., 2024); and (iii) parallel drafting–verification method Speculative-RAG (Wang et al., 2025d). See Appendix A.5 for more details.

4.5 Implementation Details

All experiments are run on a multi-GPU server with four NVIDIA RTX PRO 6000 Blackwell GPUs and dual Intel Xeon 6730P CPUs. LLM inference uses `vLLM` (v0.10.2) with continuous batching and tensor parallelism.

For all results, the sequential controller is capped at a maximum depth of 8. We use fixed $W = 2$ for all reported results. For SPARC-RAG without fine-tuning, we use $D = 6$ for Qwen-2.5-7B-Instruct and $D = 8$ for Qwen3-32B to avoid confounding performance with noisy termination signals.

We finetune the base LLM using combined training data from single-step generation of QUERY REWRITER and ANSWER EVALUATOR, trained with LoRA (Hu et al., 2022); details in Appendix A.4.

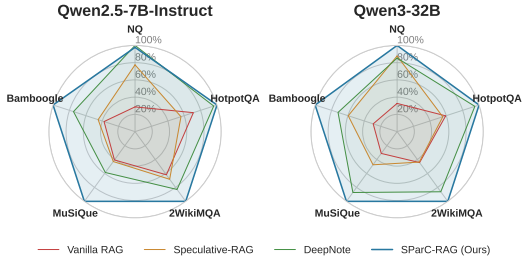

Figure 4: Relative performance of different RAG systems using two backbone LLMs across five datasets. Each plot compares Vanilla RAG, the representative parallel (Speculative-RAG) and sequential (DeepNote) RAG scaling methods, and SPARC-RAG (ours).

5 Results

Figure 4 and Table 1 summarize end-to-end performance across QA datasets.

Without fine-tuning, SPARC-RAG with joint depth-width scaling consistently matches or outperforms strong sequential- and parallel-only baselines, with the largest gains on multi-hop QA. On MuSiQue (2–4 hops) with Qwen2.5-7B-Instruct, SPARC-RAG achieves 24.3 F1 versus 19.8 for DeepNote, showing the benefit of unified depth-width control on complex queries even for base models. Appendix C.10 provides an inference-cost breakdown across $W \times D$ settings.

After fine-tuning, SPARC-RAG achieves matched or higher accuracy with significantly lower cost. SPARC-RAG+DPO with Qwen2.5-7B-Instruct backbone improves average F1 score on multihop QA datasets by 5.1 and reduces the cost by ~60%. For Qwen3-32B, SPARC-RAG+DPO slightly decreases the performance but reduces the average token cost by ~90% (Figure 3, details in Table 5); this is best characterized as a cost–accuracy operating-point shift under cross-model preference transfer (token cost -86 to -98% at a 3–4 point F1 drop) rather than a loss of transferability. Compared with the previous best RAG scaling baseline,

Table 1: End-to-end performance across QA benchmarks. Best and second-best results are highlighted. For some datasets, we report DeepNote* results using the numbers provided in the original paper.

Methods & LLMs	Single-hop				Multi-hop															
	NQ				HotpotQA				2WikiMQA				MusiQue				Bamboogle			
	acc.	f1	em	avg.	acc.	f1	em	avg.	acc.	f1	em	avg.	acc.	f1	em	avg.	acc.	f1	em	avg.
Vanilla RAG																				
Qwen2.5-7B-Instruct	26.2	13.3	2.4	14.0	42.4	42.5	32.0	39.0	35.6	38.5	32.6	35.6	11.4	13.8	6.6	10.6	14.4	17.7	12.0	14.7
Qwen3-32B	28.2	16.0	4.2	16.1	51.2	37.3	21.8	36.8	48.4	32.2	14.2	31.6	15.8	10.6	2.4	9.6	25.6	17.2	6.4	16.4
Baselines using Qwen2.5-7B-Instruct																				
Parallel (Naive)	27.6	13.0	3.0	14.5	41.0	42.4	32.6	38.7	35.6	38.2	32.0	35.3	11.0	13.8	6.4	10.4	19.2	24.0	15.2	19.5
Sequential (Naive)	28.6	13.6	3.2	15.1	43.2	41.3	30.2	38.2	35.6	37.1	30.0	34.2	10.8	12.1	5.2	9.4	19.2	22.5	15.2	19.0
IRCoT	39.6	17.3	4.6	20.5	40.4	33.8	23.2	32.5	40.0	33.2	23.6	32.3	11.8	12.4	4.8	9.7	19.2	16.7	12.0	16.0
Iter-RetGen	55.2	46.5	35.2	45.6	48.0	50.2	38.6	45.6	43.4	37.9	29.4	36.9	19.8	21.4	13.0	18.1	32.0	36.6	28.0	32.2
Speculative-RAG	40.6	41.5	31.4	37.8	29.2	34.6	28.2	30.7	39.6	41.6	36.8	39.3	9.8	15.7	8.0	11.2	16.8	19.8	16.0	17.5
Self-RAG	37.6	37.2	26.6	33.8	42.6	46.0	35.4	41.3	40.8	41.3	35.2	39.1	11.6	15.8	8.0	11.8	37.6	41.1	29.6	36.1
DeepNote	52.8	53.3	40.2	48.8	50.6*	59.2*	48.0*	52.6*	50.0*	51.4*	41.8*	47.7*	14.6*	19.8*	11.6*	15.3*	26.4	36.3	24.8	29.2
SPARC-RAG (Ours)	45.4	46.9	35.6	42.6	48.4	55.9	45.4	49.9	56.2	56.2	45.2	52.5	17.8	24.3	16.2	19.4	36.8	42.7	36.0	38.5
SPARC-RAG + DPO (Ours)	51.4	51.8	40.0	47.7	53.8	61.3	49.4	54.8	63.0	61.2	48.4	57.5	25.2	32.4	20.8	26.1	38.4	44.7	33.6	38.9
Baselines using Qwen3-32B																				
Parallel (Naive)	31.2	16.4	4.4	17.3	51.0	38.1	23.6	37.6	44.6	32.9	16.6	31.4	14.8	10.9	2.6	9.4	27.2	16.6	6.4	16.7
Sequential (Naive)	31.4	15.6	4.0	17.0	51.2	37.4	22.2	36.9	45.2	31.5	14.8	30.5	14.6	10.3	1.8	8.9	28.8	15.4	4.0	16.1
IRCoT	33.2	22.7	11.2	22.4	34.4	36.1	27.2	32.6	24.4	19.6	12.0	18.7	6.6	10.1	5.4	7.4	25.6	12.9	7.2	15.2
Iter-RetGen	54.4	47.4	34.6	45.5	52.4	61.4	48.6	54.1	53.8	54.7	42.2	50.2	23.2	29.4	18.6	23.7	40.8	47.3	33.6	40.6
Speculative-RAG	45.6	48.0	36.0	43.2	33.6	39.0	31.2	34.6	31.2	32.7	29.4	31.1	13.8	19.1	10.8	14.6	32.0	37.3	31.2	33.5
Self-RAG	45.4	44.9	31.8	40.7	53.0	57.9	45.0	52.0	55.8	55.3	46.2	52.4	12.0	20.7	10.4	14.4	49.6	54.6	42.4	48.9
DeepNote	58.4	41.3	26.4	42.0	60.2	65.2	51.2	58.9	69.2	64.4	50.6	61.4	30.8	30.9	18.6	26.8	40.8	47.0	33.6	40.5
SPARC-RAG (Ours)	53.6	54.7	39.4	49.2	62.6	68.7	55.0	62.1	78.4	74.1	61.2	71.2	31.8	36.6	23.8	30.7	56.0	63.0	49.6	56.2
SPARC-RAG + DPO (Ours)	56.6	51.1	38.0	48.6	58.2	65.0	51.0	58.1	75.2	69.8	57.6	67.5	28.8	33.9	21.4	28.0	53.6	59.5	44.8	52.6

SPARC-RAG achieves +6.19 F1 score with 52.2% less token cost on average (see Appendix C.2).

These results validate our two main claims: (i) joint sequential-parallel scaling substantially strengthens multi-hop reasoning compared to existing single-direction scaling, and (ii) targeted lightweight fine-tuning improves both answer quality and computational efficiency.

All three agents contribute to the gains. Controlled ablations show distinct contributions from each agent: without the QUERY REWRITER's feedback, accuracy drops by up to 9.0 points; naive concatenation in place of the CONTEXT MANAGER hurts accuracy and uses 4–6× more tokens; and removing the ANSWER EVALUATOR's adaptive stopping raises compute by 74–291% with negligible quality change (details in Appendix C.5).

5.1 Analysis of Sequential Depth and Parallel Width

To better understand the gains in joint scaling, we first analyze how sequential depth and parallel width individually and jointly contribute to the performance and cost of SPARC-RAG.

Depth and width provide complementary benefits under scaling. Figure 5 reports accuracy curves on 2WikiMQA and MuSiQue for $W=1,2,4$. Accuracy generally improves from $D=1$ to $D=4$, then plateaus or slightly declines, while increasing width continues to improve quality after depth scaling saturates. Across both datasets,

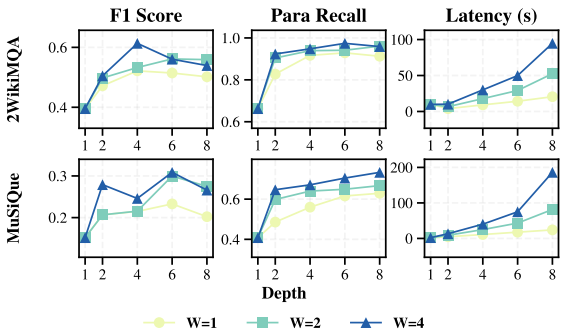

Figure 5: Answer quality (F1 scores, top row) and retrieval performance (paragraph recall, bottom row) on 2WikiMQA and MuSiQue for $W=1,2,4$ using Qwen2.5-7B-Instruct model.

paragraph recall rises consistently with width, suggesting that parallel branches retrieve more diverse evidence and translate into higher F1. Since latency grows roughly linearly with both depth and width, larger scaling levels incur rapidly increasing cost, motivating a closer analysis of the cost–quality tradeoff.

Cost–quality tradeoffs. Figure 3 provides a direct comparison of performance at matched compute budgets. Across all configurations, $W=2$ strikes the best balance between answer quality and cost. It outperforms $W=1$ configuration by introducing diversity in reasoning and in retrieval, while avoiding the sharp cost increases associated with $W=4$. We observe that the best performance–cost trade-off emerges at moderate depth and width rather than extreme scaling in either direction con-

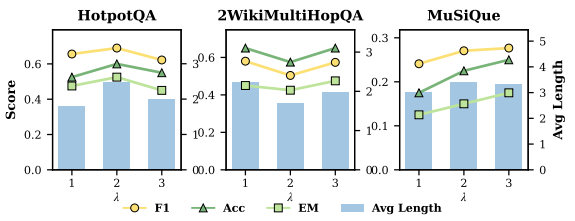

Figure 6: Effect of the weighting parameter λ on ANSWER EVALUATOR behavior evaluated on a 100-sample test subset.

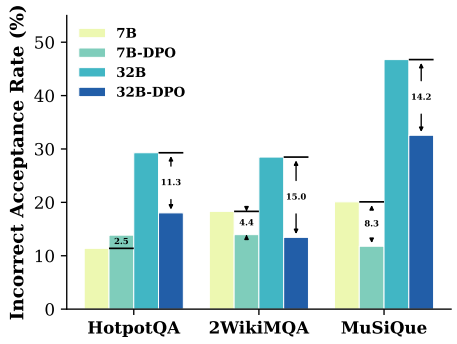

Figure 7: Incorrect acceptance rates across datasets for base and DPO-finetuned models.

sistently on all datasets. For the results reported in Table 1, we uniformly employed $W{=}2$ based on this observation. Moreover, the fact that the optimal depth varies across datasets justifies the adaptive, query-dependent depth controller that adjusts the reasoning length according to the difficulty of each query.

Finally, fine-tuning (Section 5.2) yields a more favorable F1–cost operating point: it achieves matching or higher F1 scores to the $W \times D$ grid search at substantially lower token costs by diversifying parallel exploration (QUERY REWRITER) and avoiding wasteful continuation (ANSWER EVALUATOR). For example, on 2WikiMultiHopQA, the fine-tuned model at $W{=}2$ matches the base model's best performance across all $W \times D$ configurations while using only ~30% tokens.

5.2 Fine-Tuning for Improved Sequential and Parallel Scaling

In this subsection, we provide an in-depth analysis of how targeted fine-tuning of the base LLM enhances the scaling behavior of SPARC-RAG.

Larger document coverage in Query Rewriting. Fine-tuning improves query rewriting by increasing retrieval recall and reducing redundancy. In a controlled comparison on samples at the second retrieval iteration, recall improves consistently across datasets: by +2.3% to +36.1% for the 7B model, with an average gain of +13.1%, and by +0.5% to +5.2% for the 32B model (Appendix C.3). We further measure the average pairwise Jaccard overlap between document sets retrieved by rewritten queries (Appendix B.2) and find that fine-tuning reduces overlap by up to 27.4% on NQ, indicating less redundant and more complementary retrieval.

Weighted DPO leads to more conservative stopping. Figure 6 shows that increasing λ, which up-weights *wrong-stop* errors, penalizes premature termination and makes the fine-tuned evaluator more conservative. This yields deeper rollouts and higher F1 on the more complex MuSiQue dataset (2–4 hops), where additional reasoning helps aggregate multi-step evidence. However, a large λ can slightly hurt 2-hop 2WikiMQA by causing unnecessary continuation and introducing distractors. We therefore use $\lambda{=}2$, which balances robustness on harder datasets with avoiding over-conservative stopping on simpler ones.

Reduced incorrect acceptance rates during Answer Evaluation. Fine-tuning substantially reduces the tendency to accept incorrect intermediate answers. As shown in Figure 7, larger base models (32B) exhibit higher over-acceptance rates than the 7B model, reflecting stronger overconfidence, while DPO fine-tuning consistently mitigates this failure—reducing incorrect acceptance by 11–15 points on 32B and 3–8 points on 7B across datasets. These behaviors translate into higher end-to-end QA performance (Table 1).

6 Conclusions

We introduced SPARC-RAG, a multi-agent framework that addresses *Context Contamination* and *Scaling Inefficiency* in RAG scaling. Through specialized control agents that jointly regulate scaling along sequential depth and parallel width, SPARC-RAG coordinates how computation is expanded across branches and rounds while consolidating complementary evidence into a unified context state, yielding substantial gains across single-hop and multi-hop QA benchmarks. We further introduced a scaling-targeted, preference-based fine-tuning strategy that encourages complementary parallel queries and improves the stability of stopping decisions, thereby reducing unnecessary expansions and achieving better cost–performance characteristics for SPARC-RAG.

7 Limitations

Our approach focuses on improving retrieval-augmented generation through agentic test-time scaling and structured control over depth and width of reasoning. As a result, the proposed method introduces additional inference-time overhead compared to standard single-pass RAG pipelines, which may limit its applicability in latency-critical settings. While our design aims to make this overhead cost-aware, the trade-off between performance and efficiency depends on the quality of the stopping and aggregation mechanisms. Moreover, our experiments are conducted on established open-domain question answering benchmarks with retrieval preferences. In particular, all five evaluation datasets share a Wikipedia-based retrieval corpus; we have not evaluated SPARC-RAG on cross-corpus settings such as biomedical, legal, or scientific-literature retrieval, which we leave to future work. Performance may vary under different retrieval systems, corpus characteristics, or domain-specific knowledge bases. Finally, while the proposed framework improves robustness to incomplete retrieval by exploring multiple reasoning paths, it does not guarantee correctness when retrieved evidence is systematically misleading or missing.

8 Ethical considerations

This work uses publicly available datasets and pretrained language models, and does not involve the collection of personal or sensitive data. We use all third-party datasets, pretrained models, and software libraries in accordance with their respective licenses, terms of use, and citation requirements. Risks are limited to those already associated with LLMs and RAG systems: generated outputs may reflect biases in the underlying models or databases, and the method should be deployed with appropriate safeguards in high-stakes settings. AI-assisted writing tools were used to refine clarity and presentation, and minor code edits were conducted with AI coding assistance under human supervision; all technical ideas, experimental design, and final implementations were created and verified by the authors.

References

Paul J. L. Ammann, Jonas Golde, and Alan Akbik. 2025. Question decomposition for retrieval-augmented generation. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 4: Student Research Workshop)*, pages 497–507, Vienna, Austria. Association for Computational Linguistics.

Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. 2024. Self-RAG: Learning to retrieve, generate, and critique through self-reflection. In *Proceedings of the Twelfth International Conference on Learning Representations*.

Mingyang Chen, Linzhuang Sun, Tianpeng Li, Haoze Sun, Yijie Zhou, Chenzheng Zhu, Haofen Wang, Jeff Z. Pan, Wen Zhang, Huajun Chen, Fan Yang, Zenan Zhou, and Weipeng Chen. 2025. ReSearch: Learning to reason with search for LLMs via reinforcement learning. In *Advances in Neural Information Processing Systems*.

Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Meng Wang, and Haofen Wang. 2023. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*.

Hongchao Gu, Dexun Li, Kuicai Dong, Hao Zhang, Hang Lv, Hao Wang, Defu Lian, Yong Liu, and Enhong Chen. 2025. RAPID: Efficient retrieval-augmented long text generation with writing planning and information discovery. *arXiv preprint arXiv:2503.00751*.

Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. 2020. Constructing a multi-hop QA dataset for comprehensive evaluation of reasoning steps. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 6609–6625, Barcelona, Spain (Online). International Committee on Computational Linguistics.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.

Yunhai Hu, Yilun Zhao, Chen Zhao, and Arman Cohan. 2025. MCTS-RAG: Enhancing retrieval-augmented generation with Monte Carlo tree search. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 12581–12597, Suzhou, China. Association for Computational Linguistics.

Yuichi Inoue, Kou Misaki, Yuki Imajuku, So Kuroki, Taishi Nakamura, and Takuya Akiba. 2025. Wider or deeper? scaling LLM inference-time compute with adaptive branching tree search. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*. Spotlight.

Soyeong Jeong, Jinheon Baek, Sukmin Cho, Sung Ju Hwang, and Jong Park. 2024. Adaptive-RAG: Learning to adapt retrieval-augmented large language models through question complexity. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 7036–7050, Mexico City, Mexico. Association for Computational Linguistics.

Jiajie Jin, Yutao Zhu, Zhicheng Dou, Guanting Dong, Xinyu Yang, Chenghao Zhang, Tong Zhao, Zhao Yang, and Ji-Rong Wen. 2025. FlashRAG: A modular toolkit for efficient retrieval-augmented generation research. In *Companion Proceedings of the ACM on Web Conference 2025*, pages 737–740. Association for Computing Machinery.

Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2021. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, 7(3):535–547.

Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6769–6781, Online. Association for Computational Linguistics.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, Kristina Toutanova, Llion Jones, Matthew Kelcey, Ming-Wei Chang, Andrew M. Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. 2019. Natural questions: A benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:452–466.

Myeonghwa Lee, Seonho An, and Min-Soo Kim. 2024. PlanRAG: A plan-then-retrieval augmented generation for generative large language models as decision makers. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 6537–6555, Mexico City, Mexico. Association for Computational Linguistics.

Quinn Leng, Jacob Portes, Sam Havens, Matei Zaharia, and Michael Carbin. 2024. Long context RAG performance of large language models. *arXiv preprint arXiv:2411.03538*. NeurIPS 2024 Workshop on Adaptive Foundation Models.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems*, volume 33, pages 9459–9474. Curran Associates, Inc.

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

OpenAI. 2023. GPT-4 technical report. *arXiv preprint arXiv:2303.08774*.

Ofir Press, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah Smith, and Mike Lewis. 2023. Measuring and narrowing the compositionality gap in language models. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 5687–5711, Singapore. Association for Computational Linguistics.

Qwen, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, and 24 others. 2025. Qwen2.5 technical report. *Preprint*, arXiv:2412.15115.

Minjoon Seo, Jinhyuk Lee, Tom Kwiatkowski, Ankur Parikh, Ali Farhadi, and Hannaneh Hajishirzi. 2019. Real-time open-domain question answering with dense-sparse phrase index. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4430–4441, Florence, Italy. Association for Computational Linguistics.

Zhihong Shao, Yeyun Gong, Yelong Shen, Minlie Huang, Nan Duan, and Weizhu Chen. 2023. Enhancing retrieval-augmented large language models with iterative retrieval-generation synergy. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 9248–9274, Singapore. Association for Computational Linguistics.

Yixuan Tang and Yi Yang. 2024. MultiHop-RAG: Benchmarking retrieval-augmented generation for multi-hop queries. In *First Conference on Language Modeling*.

Nandan Thakur, Nils Reimers, Andreas Rücklé, Abhishek Srivastava, and Iryna Gurevych. 2021. BEIR: A heterogeneous benchmark for zero-shot evaluation of information retrieval models. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*.

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

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. 2023. Interleaving retrieval with chain-of-thought reasoning for knowledge-intensive multi-step questions. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 10014–10037, Toronto, Canada. Association for Computational Linguistics.

Prakhar Verma, Sukruta Prakash Midigeshi, Gaurav Sinha, Arno Solin, Nagarajan Natarajan, and Amit Sharma. 2024. Plan*RAG: Efficient test-time planning for retrieval augmented generation. *arXiv preprint arXiv:2410.20753*.

Thuy Vu and Alessandro Moschitti. 2021. AVA: an automatic eValuation approach for question answering systems. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 5223–5233, Online. Association for Computational Linguistics.

Liang Wang, Haonan Chen, Nan Yang, Xiaolong Huang, Zhicheng Dou, and Furu Wei. 2025a. Chain-of-retrieval augmented generation. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Ruobing Wang, Qingfei Zhao, Yukun Yan, Daren Zha, Yuxuan Chen, Shi Yu, Zhenghao Liu, Yixuan Wang, Shuo Wang, Xu Han, Zhiyuan Liu, and Maosong Sun. 2025b. DeepNote: Note-centric deep retrieval-augmented generation. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 19688–19715, Suzhou, China. Association for Computational Linguistics.

Yiming Wang, Pei Zhang, Siyuan Huang, Baosong Yang, Zhuosheng Zhang, Fei Huang, and Rui Wang. 2025c. Sampling-efficient test-time scaling: Self-estimating the best-of-N sampling in early decoding. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*. Spotlight.

Zilong Wang, Zifeng Wang, Long Le, Steven Zheng, Swaroop Mishra, Vincent Perot, Yuwei Zhang, Anush Mattapalli, Ankur Taly, Jingbo Shang, Chen-Yu Lee, and Tomas Pfister. 2025d. Speculative RAG: Enhancing retrieval augmented generation through drafting. In *The Thirteenth International Conference on Learning Representations*.

Zongqian Wu, Tianyu Li, Baoduo Xu, Jiaying Yang, Mengmeng Zhan, Xiaofeng Zhu, and Lei Feng. 2025. Is depth all you need? an exploration of iterative reasoning in LLMs. *arXiv preprint arXiv:2502.10858*.

Yang Xiao, Jessie Wang, Ruifeng Yuan, Chunpu Xu, Kaishuai Xu, Wenjie Li, and Pengfei Liu. 2025. LIMOPro: Reasoning refinement for efficient and effective test-time scaling. In *Advances in Neural Information Processing Systems*.

Ran Xu, Hui Liu, Sreyashi Nag, Zhenwei Dai, Yaochen Xie, Xianfeng Tang, Chen Luo, Yang Li, Joyce C. Ho, Carl Yang, and Qi He. 2025. SimRAG: Self-improving retrieval-augmented generation for adapting large language models to specialized domains. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 11534–11550, Albuquerque, New Mexico. Association for Computational Linguistics.

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

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, and 41 others. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. 2018. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2369–2380, Brussels, Belgium. Association for Computational Linguistics.

Gongrui Zhang, Jialiang Zhu, Ruiqi Yang, Kai Qiu, Miaosen Zhang, Zhirong Wu, Qi Dai, Bei Liu, Chong Luo, Zhengyuan Yang, Linjie Li, Lijuan Wang, Weizhu Chen, Yuan Zhang, Xin Li, Zhaoyi Liu, Xin Geng, and Baining Guo. 2025. InfoAgent: Advancing autonomous information-seeking agents. *arXiv preprint arXiv:2509.25189*.

Penghao Zhao, Hailin Zhang, Qinhan Yu, Zhengren Wang, Yunteng Geng, Fangcheng Fu, Ling Yang, Wentao Zhang, Jie Jiang, and Bin Cui. 2026. Retrieval-augmented generation for AI-generated content: A survey. *Data Science and Engineering*, 11:1–29.

Appendix

A Experiment details

A.1 Datasets

We evaluate our method on five knowledge-intensive QA benchmarks, covering both single-hop and multi-hop reasoning.

Natural Questions (NQ). Natural Questions (Kwiatkowski et al., 2019) is a single-hop open-domain QA benchmark consisting of real anonymized Google search queries. In our experiments, we follow the AdaptiveRAG evaluation setup (Jeong et al., 2024), which introduces a 500-query test subset. The retrieval corpus is the standard DPR Wikipedia passage collection (Karpukhin et al., 2020), chunked into 100-word passages. Indexing follows the dense retrieval pipeline described in the AdaptiveRAG documentation.

HotpotQA. HotpotQA (Yang et al., 2018) is a 2-hop QA benchmark with supporting-fact supervision. Following IRCoT (Trivedi et al., 2023), we use the 500-query subsampled test set provided by the IRCoT repository.[1] The corpus is constructed using the IRCoT Wikipedia passage-building procedure.

2WikiMultiHopQA. 2WikiMultiHopQA (Ho et al., 2020) contains 2-hop questions requiring cross-entity bridging inference based on Wikipedia pages. We again follow IRCoT (Trivedi et al., 2023) and use the 500-query subsampled test set distributed in the IRCoT processed-data package. The corpus is the same IRCoT Wikipedia passage corpus.

MuSiQue. MuSiQue (Trivedi et al., 2022) is a challenging 2–4 hop QA benchmark created by compositional reasoning over multiple Wikipedia paragraphs. As with the previous datasets, we use the 500-query subsampled test set following IRCoT (Trivedi et al., 2023), and apply the same corpus construction procedure as IRCoT.

Bamboogle. Bamboogle (Press et al., 2023) is a 2-hop QA dataset targeting adversarial or low-resource settings, with a small official test set. We use the official test split from the HuggingFace distribution.[2] Following the recent open-ended Bamboogle evaluation in ReSearch (Chen et al., 2025), we evaluate Bamboogle using the DPR Wikipedia passage corpus (Karpukhin et al., 2020; Jin et al., 2025).

[1] https://github.com/StonyBrookNLP/ircot

[2] https://huggingface.co/datasets/chiayewken/bamboogle

A.2 Indexing

We build separate retrieval indices for each dataset corpus. The indexing pipeline consists of the following components:

Document Chunking. In the retrieval corpus, documents are organized at the paragraph level, where each paragraph is associated with a title and treated as an independent retrieval unit. For datasets that use Wikipedia as the corpus (i.e., Natural Questions, Bamboogle), we use the DPR Wikipedia passages[3], which splits Wikipedia articles into 100-word passages.

Sparse Indexing. We use Elasticsearch 7.10.2 to build BM25 indices. Each document is indexed with the following fields: `title`, `paragraph_text`, `paragraph_index`, `url`, and `is_abstract`. The `title` and `paragraph_text` fields are analyzed using the English analyzer for stemming and stopword removal.

Dense Indexing. We use BGE-large-en-v1.5 (`BAAI/bge-large-en-v1.5` hosted on Huggingface) as the embedding model to encode all documents into dense vectors. The embeddings are indexed using FAISS (Johnson et al., 2021) with a flat inner product (FlatIP) index for exact nearest neighbor search. At query time, we compute the dot product similarity between the query embedding and all document embeddings.

A.3 Retrieval methods

Our system supports two retrieval methods, which can be dynamically selected by the language model during inference:

Sparse Retrieval (BM25). Given a query, we perform BM25 retrieval against the Elasticsearch index. The query is preprocessed by removing wh-words (e.g., "what", "where", "when") to improve lexical matching. We retrieve the top-k documents ranked by BM25 score, where $k = 6$ for all baselines and our method.

[3] https://dl.fbaipublicfiles.com/dpr/wikipedia_split/psgs_w100.tsv.gz

Dense Retrieval. Given a query, we encode it using the same BGE-large-en-v1.5 model used for indexing and retrieve the top-k documents based on dot product similarity with the document embeddings. We use the same $k = 6$ as sparse retrieval.

Dynamic Retrieval Selection. Unlike prior methods that rely on a single fixed retrieval approach, our agentic system treats different retrieval methods as *tools* that can be intelligently selected based on the query characteristics. At each retrieval step, the language model decides which retrieval method to invoke. This design choice is motivated by the complementary strengths of sparse and dense retrieval: BM25 excels at lexical matching and handling rare entities, while dense retrieval better captures semantic similarity.

We acknowledge that this introduces a potential fairness consideration when comparing against baselines, as most prior methods use a single retrieval method. However, we note that: (i) these baseline methods do not natively support dynamic retrieval selection in their architectures, (ii) there is limited motivation for them to use multiple retrieval methods since they lack the decision-making mechanism to choose between them, and (iii) ablation study for SPARC-RAG using only dense retrieval in Appendix C.8 indicates that the improvement does not originate from the dynamic retrieval selection.

Note on `global_max_num_paras`. For all baseline methods that accumulate retrieved passages into an ever-growing context, we set a unified `global_max_num_paras` threshold. Without such a cap, these methods may receive excessively long contexts across iterations, which is known to harm LLM generation quality and distort fair comparison. This design follows the IRCoT (Trivedi et al., 2023) implementation, where limiting the accumulated context is necessary to prevent the model from being overwhelmed by large, redundant evidence pools.

In contrast, note-based or context-compression methods, i.e., Deepnote (Wang et al., 2025b) and our SPARC-RAG, do not rely on raw accumulation of retrieved passages. Their internal representations remain compact, so `global_max_num_paras` is not applicable to them.

A.4 Implementation Details

Hardware. All training and inference workloads run on a single server with four NVIDIA RTX PRO 6000 Blackwell Server Edition GPUs (97,887 MiB each; 384 GB total), dual Intel Xeon 6730P CPUs (2 sockets × 32 cores × 2 threads = 128 logical cores), and 1.0 TiB RAM.

Software Environment. Experiments are executed under Python 3.11.13 with the following key libraries: PyTorch 2.8.0+cu128, vLLM 0.10.2, Transformers 4.56.1, PEFT 0.17.1, TRL 0.23.1, and DeepSpeed 0.18.0. Sparse retrieval is implemented using Elasticsearch 7.10.2. Dense retrieval uses BGE-large-en-v1.5 embeddings indexed with FAISS FlatIP.

Inference Configuration. Generation modules use temperature 0.5, top-$p = 1.0$, and a maximum generation length of 600 tokens. Answer Evaluator modules operate deterministically with temperature 0. Parallel branches use global random seed 42. The sequential controller is allowed up to 8 reasoning rounds per query.

Training Hyperparameters. The DPO training uses LoRA adapters with rank 16, α=32, dropout 0.05, and target modules `q_proj`, `k_proj`, `v_proj`, `o_proj`. DPO training uses learning rate 5e–5 (LoRA), 5e–7 (full fine-tuning), batch size 1 with gradient accumulation 4 (effective batch size 16 across 4 GPUs), maximum sequence length 8192, BF16 precision, and DPO β=0.1. We use the AdamW optimizer and train for 3 epochs. The complete configuration, including the asymmetric DPO loss with up-weight λ=2.0 on ANSWER EVALUATOR wrong-stop pairs, is consolidated in Table 3; the preference dataset construction pipeline is described in detail below.

Retriever Configuration. Sparse retrieval uses Elasticsearch 7.10.2 with BM25 scoring. Dense retrieval uses model-encoded embeddings served via lightweight FastAPI endpoints. The same retrieval stack is used for all baselines to ensure fair comparison.

Preference data sources and held-out splits. The DPO preference dataset is constructed exclusively from the *training splits* of three multi-hop QA benchmarks: HotpotQA, 2WikiMultiHopQA, and MuSiQue. Critically, the two remaining evaluation datasets—**Natural Questions** and **Bamboogle**—are entirely excluded from preference construction. This separation is a design choice that lets us treat NQ and Bamboogle as held-out testbeds for cross-dataset transfer; the held-out im-

provements are reported alongside in-training results in the cross-dataset analysis (Appendix C.3 and the DPO results table). Because no per-dataset balancing was applied, the resulting pair counts reflect the natural yield of the three source datasets after filtering uninformative steps (Table 2).

Preference pair generation pipeline. Each preference pair (y^+, y^-) originates from a single intermediate step of an *actual* SPARC-RAG rollout, rather than from a synthetic, isolated prompt. This grounds every pair in a realistic distribution of upstream context (history, retrieved evidence, current note) that the agent will see at inference time. The construction follows a three-stage protocol:

1. **Rollout collection.** We run the un-tuned base model (Qwen2.5-7B-Instruct) through the full SPARC-RAG pipeline on training questions. To populate diverse controller states—in particular, branches with different evidence trajectories—we vary the dispatch width across rollouts: $W \in \{2, 3, 4\}$. The retriever, prompts, and decoding for the upstream context use the same configuration as inference time.

2. **Per-step candidate sampling.** At each intermediate step where the QUERY REWRITER or ANSWER EVALUATOR is invoked, we freeze the upstream context and re-sample 9 candidate outputs from the same backbone. The 9 candidates correspond to the full 3×3 temperature×top-p grid with $T \in \{0.1, 0.5, 0.9\}$ and top-$p \in \{0.1, 0.5, 0.9\}$, chosen to span low- and high-entropy regions of the decoder distribution. Sampling from the same backbone (rather than a stronger teacher) ensures the chosen-rejected gap reflects *realizable* quality differences rather than off-policy distribution shift.

3. **Best/worst pair extraction.** For each step we form one preference pair by retaining the best (*chosen*) and worst (*rejected*) candidate under an agent-specific criterion (below). When multiple candidates tie at the extreme, one is sampled uniformly at random. Steps where all 9 candidates yield the same outcome (no informative gradient) are discarded.

Per-agent selection criteria.

- **Query Rewriter (recall-grounded).** Each candidate rewrite is issued to the same retriever and produces a top-k paragraph set (k=6). The candidate is scored by paragraph-level recall against the gold supporting paragraphs of the original question. The highest-recall candidate becomes the *chosen* and the lowest-recall the *rejected*. This pure recall-based criterion does not invoke any LLM-as-judge signal and so avoids judge-model bias; we treat retrieval recall as a downstream-aligned proxy because higher recall directly increases the chance that the answer is grounded in retrieved evidence.

- **Answer Evaluator (correctness-grounded STOP/CONTINUE).** Each candidate produces a binary STOP/CONTINUE decision together with a free-form justification. Using the gold answer as ground truth, the underlying candidate answer being evaluated is labeled *correct* or *incorrect*. (i) If the candidate answer is *correct*, the chosen response is the highest-quality `STOP` ("accept") output and the rejected is the highest-quality `CONTINUE` ("reject") output. (ii) If the candidate answer is *incorrect*, the polarity flips: chosen = best `CONTINUE`, rejected = best `STOP`. Among the resulting 502 AE pairs, the chosen decisions split nearly evenly between STOP (239 pairs) and CONTINUE (263 pairs), preventing the model from collapsing to a trivial always-stop or always-continue policy. Within each STOP / CONTINUE bucket, ties on the binary label are broken by an LLM judge (same backbone) that ranks the candidates' justification quality.

Table 2: Composition of the DPO preference dataset by component and source dataset. Pairs are sampled from training splits only. Each pair contains exactly one chosen and one rejected response. NQ and Bamboogle are held out from preference construction.

Source Dataset	QR pairs	AE pairs	Total
HotpotQA	1,001	160	1,161
MuSiQue	809	141	950
2WikiMultiHopQA	1,044	201	1,245
Natural Questions	*held out (no preferences constructed)*		
Bamboogle	*held out (no preferences constructed)*		
Source-question pool	1,701	502	—
Total pairs	**2,854**	**502**	**3,356**

Train/validation split. The 3,356 pairs are randomly split 90/10 with random seed 42, yielding approximately 3,020 training and 336 validation pairs. The QR and AE pairs share the same split and are jointly optimized on a *single* shared LoRA adapter; the agent role is implicitly conveyed via

Table 3: DPO training hyperparameters. The same configuration is used for the joint QR+AE preference dataset on the Qwen2.5-7B-Instruct backbone. Trainable parameter count is dominated by the LoRA adapter ($\sim$0.13% of the backbone).

Category	Hyperparameter	Value
Adapter (LoRA)	Rank (r)	16
	α	32
	Dropout	0.05
	Target modules	`q_proj`, `k_proj`, `v_proj`, `o_proj`
Optimization	Optimizer	AdamW
	Learning rate (LoRA)	5×10^{-5}
	Learning rate (full FT)	5×10^{-7}
	Per-GPU batch size	1
	Grad. accumulation steps	4
	Effective batch size	16 (4 GPUs)
Training schedule	Max sequence length	8,192
	Mixed precision	BF16
	Number of epochs	3
DPO loss	Reference temperature β	0.1
	AE wrong-stop up-weight λ	2.0
Data split	Train pairs (seed 42)	$\sim$3,020
	Validation pairs	$\sim$336

the prompt format (the QUERY REWRITER prompt and the ANSWER EVALUATOR prompt differ in instruction header and expected output schema, providing sufficient role conditioning for the shared adapter).

Asymmetric DPO loss for stopping bias. A naive symmetric DPO loss tends to reinforce the base model's existing tendency to stop early, because correctly-accepted-easy-questions outnumber correctly-rejected-hard-questions in the natural rollouts. To counteract this, we apply an asymmetric weighting: pairs in which the chosen decision is `CONTINUE` (i.e., the model should reject an incorrect answer) receive a multiplicative loss weight λ=2.0, while pairs whose chosen decision is `STOP` keep weight 1.0. The value λ=2.0 was selected via a sweep over $\{1, 2, 3\}$ on a 100-question dev subset; λ=1 produced an over-stopping policy and λ=3 over-corrected into excessive continuation, while λ=2 achieved the best accuracy-cost trade-off. The same λ is shared across all datasets and model scales.

A.5 Baseline Configurations

We compare against the following representative RAG methods, categorized by their scaling strategy:

Vanilla RAG is a single-step retrieval baseline: given a query, the system retrieves the top-k documents using dense retrieval and generates an answer conditioned on the concatenation of the query and retrieved passages. No iterative refinement or parallel exploration is performed.

Parallel (Naive) is a naive parallel scaling baseline that generates W independent answers at temperature 0.7, then aggregates via majority voting (self-consistency). This baseline uses the same answer generation prompt as Vanilla RAG but samples multiple responses to leverage diversity. The final answer is selected by counting exact matches among the post-processed candidates. We sweep $W \in \{2, 4, 6, 8, 10\}$ and report the best result for each dataset.

Sequential (Naive) is a naive sequential scaling baseline that implements a verify-and-retry strategy. The system generates an answer, then prompts the same LLM to self-evaluate and output a confidence score (0–1). If confidence falls below a threshold (0.7), the system retries up to D=3 attempts. The final answer is selected from the attempt with highest self-evaluated confidence.

IRCoT (Trivedi et al., 2023) is an interleaved retrieval method with Chain-of-Thought prompting. The model alternates between generating a reasoning step and retrieving relevant documents, iterating until the answer is produced. We use the official implementation[4] and follow its original design, which utilizes a BM25 sparse retriever, as the method relies heavily on keyword-level lexical overlap.

Iter-RetGen (Shao et al., 2023) is an iterative retrieval-generation method that alternates between retrieval and generation, using the previous generation as context for the next retrieval query. Since no official implementation is publicly available, we re-implemented this method following the algorithm described in the original paper.

Speculative-RAG (Wang et al., 2025d) is a parallel drafting approach that generates multiple draft answers using different retrieved document subsets and selects the best response via a verification step. Since no official implementation is publicly available, we re-implemented this method based on the original paper. We use k=6 document subsets with m=10 drafts for 2WikiMultiHopQA and MuSiQue, and k=3 subsets with m=5 drafts for the other datasets, with a maximum of 15 retrieved documents.

[4] https://github.com/StonyBrookNLP/ircot

Self-RAG (Asai et al., 2024) is a self-reflective retrieval-augmented generation method that learns to retrieve, generate, and critique its own output through special reflection tokens. Rather than using the original fine-tuned model, we adopt the LangGraph-based re-implementation[5] which reproduces the Self-RAG workflow using standard LLMs without specialized fine-tuning. This enables fair comparison using the same backbone models (Qwen2.5-7B-Instruct and Qwen3-32B) across all methods.

DeepNote (Wang et al., 2025b) is a sequential RAG method that maintains a structured note representation across iterations, compressing retrieved evidence into a running summary. We use the official implementation[6] but replace the retrieval backend and generator LLM to match our unified evaluation setup.

B Extended Definitions

B.1 Task Formulation

We formalize the retrieval-augmented reasoning task that our system aims to solve. The objective is to answer a complex query q_0 by iteratively interacting with an external corpus C and refining intermediate hypotheses over multiple rounds of reasoning. To facilitate a precise formulation, we introduce the notations used throughout this section in Table 4.

Table 4: Notation summary for the RAG task formulation.

Symbol	Description
C	Corpus available for retrieval.
q_0	Global (initial) query.
$q^{(t)}$	Query at round t.
$m^{(t)} \in \mathcal{M}$	Memory state at round t.
$a^{(t)} \in \mathcal{A}$	Candidate answer at round t.
D	Total number of reasoning rounds (depth).
$W^{(t)}$	Number of parallel reasoning branches at round t (width).

We describe the reasoning process as a hierarchical composition of three operators: an *intra-path* operator I, a *parallel* operator P, and a *sequential* operator S. The overall system can be expressed as

$$\Pi(q_0, C) \triangleq (\mathsf{S} \circ \mathsf{P} \circ \mathsf{I})(q_0, C).$$

The operator I performs one round of retrieval-augmented inference along a single reasoning path, updating its memory and optionally producing a candidate answer. Note that an *operator* in our formulation is an abstract computational role, rather than a concrete agent.

The parallel operator $\mathsf{P}^{(t)}$ applies I across $W^{(t)}$ branches at round t and aggregates their outcomes:

$$\begin{aligned}&\mathsf{P}^{(t)}(q^{(t)}, m^{(t-1)}, C)\\ &= \mathsf{Merge}\Big(\big\{\mathsf{I}(q^{(t,k)}, m^{(t-1,k)}, C)\big\}_{k=1}^{W^{(t)}}\Big).\end{aligned}$$

Finally, the sequential operator composes the round-specific parallel processes:

$$\mathsf{S}\big(\{\mathsf{P}^{(t)}\}_{t=1}^{D}\big)(q_0, C) = \big(\mathsf{P}^{(D)} \circ \cdots \circ \mathsf{P}^{(1)}\big)(q_0, C),$$

where D denotes the number of reasoning rounds until termination.

This formulation captures the hierarchical structure of multi-step RAG: sequential composition models refinement over time, while parallel composition captures the breadth of exploration within each round.

B.2 Jaccard Overlap of Query Groups.

Given a set of rewritten queries generated for the same question, we measure the degree of redundancy between them using the Jaccard overlap of their retrieved document sets. For a rewritten query q, let $\mathcal{R}(q)$ denote the set of top-k retrieved paragraphs obtained from the dense retriever. For two rewritten queries q_i and q_j, their Jaccard overlap is defined as

$$\mathrm{Jac}(q_i, q_j) = \frac{|\mathcal{R}(q_i) \cap \mathcal{R}(q_j)|}{|\mathcal{R}(q_i) \cup \mathcal{R}(q_j)|}.$$

This quantity ranges from 0 (no shared retrieved evidence) to 1 (identical retrieval results).

For a group of rewrites $\{q_1, \ldots, q_m\}$, we report the *average pairwise Jaccard overlap*, defined as

$$\mathrm{JacAvg} = \frac{2}{m(m-1)} \sum_{i<j} \mathrm{Jac}(q_i, q_j).$$

Lower overlap indicates that the rewrites retrieve more complementary evidence, whereas high overlap suggests that rewrites collapse to similar retrieval trajectories. In our analysis, we compute

[5] https://langchain-ai.github.io/langgraph/tutorials/rag/langgraph_self_rag/

[6] https://github.com/thunlp/DeepNote/tree/main

this metric separately for DPO-chosen and DPO-rejected rewrites; the former consistently exhibit lower overlap, indicating that recall-based supervision implicitly encourages more diverse and evidence-complementary sub-queries.

C Additional Results

C.1 Token cost before and after fine-tuning

Table 5: DPO fine-tuning result: accuracy and efficiency comparison between fixed-depth baseline and DPO-tuned model with adaptive depths. 7B uses D=6, W=2; 32B uses D=8, W=2.

Model	Dataset	F1 (%)		Avg Depth		Tokens (K)	
		Fixed	Tuned (Δ)	Fixed	Tuned	Fixed	Tuned
7B	HotpotQA	55.9	61.3 (+5.4)	6	1.8	20.2	6.5
	MuSiQue	24.3	32.4 (+8.1)	6	3.0	23.6	16.5
	2WikiMQA	56.2	61.2 (+5.0)	6	2.0	22.0	7.8
	NQ	34.7	51.8 (+17.1)	6	1.7	16.2	7.1
	Bamboogle	42.7	44.7 (+2.0)	6	1.6	24.3	5.3
32B	HotpotQA	68.7	65.0 (-3.7)	8	1.4	39.0	4.6
	MuSiQue	36.6	33.9 (-2.7)	8	1.7	53.2	7.2
	2WikiMQA	74.1	69.8 (-4.3)	8	1.8	88.1	7.0
	NQ	54.7	51.1 (-3.6)	8	1.1	129.6	2.9
	Bamboogle	63.0	57.9 (-5.1)	8	1.3	104.6	3.9

Table 5 presents the impact of DPO fine-tuning on answer evaluation accuracy and computational efficiency. For the 7B model, DPO training yields consistent improvements across all five datasets, with F1 gains ranging from +2.0% (Bamboogle) to +17.1% (NQ). Notably, these accuracy improvements are accompanied by substantial efficiency gains: the tuned model reduces average retrieval depth from 6 to 1.6–3.0 iterations, resulting in 56–78% reduction in token consumption. Importantly, **NQ and Bamboogle are entirely held out from preference construction** (Table 2); their +17.1 F1 and +2.0 F1 gains indicate that the DPO-learned rewriting and stopping behaviors generalize beyond the three source multi-hop datasets rather than overfitting to benchmark-specific patterns.

For the 32B model, while DPO tuning did not improve the answer performance, it achieves dramatic efficiency improvements—reducing depth from 8 to 1.1–1.8 iterations and cutting token usage by 87–98%. We attribute this to a distribution mismatch: the DPO training data was generated using the 7B model, causing the 32B model to adopt overly conservative early-stopping behavior. Despite this, the 32B tuned model still achieves competitive absolute performance (e.g., 69.8% F1 on 2WikiMQA) while consuming only 7–8% of the baseline's computational budget.

Table 6: Token cost comparison between DeepNote and our DPO-tuned models. Values represent average tokens per query. As we directly report the results from the DeepNote paper for some datasets using the Qwen2.5-7B-Instruct model, token costs on some datasets are marked missing '-'.

Dataset	7B Model			32B Model		
	DeepNote	Ours	Δ	DeepNote	Ours	Δ
HotpotQA	6,235	6,511	+4.4%	10,561	4,640	-56.1%
MuSiQue	–	16,477	–	13,206	7,166	-45.7%
2WikiMQA	–	7,785	–	11,416	6,981	-38.8%
NQ	8,359	7,081	-15.3%	18,816	2,935	-84.4%
Bamboogle	7,452	5,348	-28.2%	17,102	3,887	-77.3%
Avg	**7,349**	**6,313**	**-14.1%**	**14,220**	**5,122**	**-64.0%**

These results suggest that DPO fine-tuning effectively teaches the model to recognize answer sufficiency, enabling adaptive depth control that balances accuracy and efficiency. Future work could explore model-specific DPO training to better preserve the capabilities of larger models.

C.2 Comparison of token cost with other methods

Table 6 compares the computational cost of our DPO-tuned models against DeepNote, a state-of-the-art adaptive RAG baseline that employs iterative retrieval with early stopping. We did not report the cost comparison with other methods as their performance are significantly worse than these two methods. We conduct the comparison using both 7B and 32B model variants to ensure a fair comparison.

For the 32B model, our approach achieves substantial token savings across all five datasets, with an average reduction of 64.0% compared to DeepNote. For the 7B model, on single-hop datasets (NQ and Bamboogle), our method reduces token consumption by 15.3% and 28.2% respectively and achieves a 14.1% average reduction while maintaining competitive accuracy.

A key advantage of our approach lies in learning *when* to stop retrieval through direct preference optimization. While DeepNote uses heuristic stopping criteria, our method learns a fine-grained stopping policy from preference data, leading to more efficient scaling behaviors.

C.3 Analysis of DPO fine-tuning on paragraph recall

To isolate the effect of DPO fine-tuning on query rewriting quality, we conduct a controlled comparison on samples where the tuned model proceeds to

Table 7: Recall comparison on samples where the DPO-tuned model proceeds to the second retrieval iteration. This controlled comparison isolates the effect of DPO on query rewriting quality, excluding samples that terminate early.

Model	Dataset	Samples	Base	Tuned	Δ
7B	HotpotQA	209	70.6	77.3	+6.7
	MuSiQue	348	58.8	61.1	+2.3
	2WikiMQA	271	77.6	84.9	+7.3
	NQ	157	18.7	54.8	+36.1
32B	HotpotQA	104	67.8	71.6	+3.8
	MuSiQue	224	55.5	60.0	+4.5
	2WikiMQA	246	81.3	86.5	+5.2
	NQ	29	39.1	39.7	+0.5

the second retrieval iteration (Table 7).

The results reveal that DPO fine-tuning consistently improves recall across all datasets. For the 7B model, there is an average recall improvement of +13.1%. The substantial improvement on NQ (+36.1%) is particularly notable: the base model achieves only 18.7% recall on these samples, indicating poor query rewriting for single-hop questions, while the tuned model reaches 54.8%. For multi-hop datasets, improvements are more modest but consistent (+2.3% to +7.3%), suggesting that DPO training enhances the model's ability to generate effective follow-up queries.

The 32B model exhibits similar trends with gains of +0.5% to +5.2% across datasets. The smaller improvement magnitude compared to 7B can be attributed to two factors: (1) fewer samples reach the second iteration (e.g., only 29 NQ samples for 32B vs. 157 for 7B), limiting statistical power, and (2) the base 32B model already produces higher-quality rewrites, leaving less room for improvement.

These findings demonstrate that DPO fine-tuning improves *how* queries are rewritten when additional retrieval is necessary.

C.4 Role conditioning and step-level precision under joint DPO training

This subsection describes how a single LoRA adapter is shared across the QUERY REWRITER and ANSWER EVALUATOR via prompt-level role conditioning, and reports the resulting step-level and pipeline-level retrieval quality after DPO.

Adapter setup and role conditioning. We deliberately fine-tune one backbone with a *single* LoRA adapter (rank 16 on `q_proj`, `k_proj`, `v_proj`, `o_proj`; ∼0.13% trainable parameters) on the combined QR+AE preference set (Table 2). Role-specific behavior emerges from *explicit prompt-level conditioning*, not from parameter partitioning:

- QUERY REWRITER prompt: instruction header "generate N diverse rewrites" + input fields {original question, current note/context} → XML-structured list of (strategy, rewrite) items.
- ANSWER EVALUATOR prompt: instruction header "decide whether to accept or continue" + input fields {question, candidate answer, supporting evidence} → binary `STOP`/`CONTINUE` decision with justification.

The two prompt templates differ in instruction header, input-field schema, and output schema, giving the shared adapter sufficient conditioning signal to specialize its behavior by role. During training we additionally track per-role validation metrics so that degradation in either role would be detected.

Empirical check: no measurable cross-role interference. After joint training, *both* roles improve on their respective held-out metrics relative to the un-tuned base model:

- QUERY REWRITER: step-level paragraph recall improves by +2.3 to +36.1 points on the 7B model across HotpotQA, MuSiQue, 2WikiMQA, and NQ (Table 7).
- ANSWER EVALUATOR: incorrect-acceptance rate drops by 3–8 points on 7B and by 11–15 points on 32B across datasets (Figure 7).

Both roles improve monotonically on their respective supervision signals after joint training, suggesting that prompt-level role conditioning is sufficient for our two-role setup.

Recall- versus precision-oriented supervision. A natural concern with recall-based QR preferences is that "retrieve more gold paragraphs" might be achieved by retrieving a wider, noisier set—inflating recall at the cost of precision. To test this directly, we re-evaluate retrieval quality under matched conditions: fixed retriever, fixed top-k per call (k=6), same question set, and the same step (second retrieval round / first follow-up rewrite). Under this controlled step-level comparison, DPO improves *both* Recall@k and Precision@k across all four 7B datasets (Table 8). Because k is held fixed, a *simultaneous* increase in Recall@k and Precision@k cannot be explained by retrieving a wider set—it can only come from retrieving a *better-ranked* set within the top-k window. Pre-

Table 8: Step-level Recall@k and Precision@k at k=6 on the second retrieval round (first follow-up rewrite), evaluated only on samples where the DPO-tuned 7B model proceeds past the first round. Base = Qwen2.5-7B-Instruct; Tuned = Base + DPO. Both recall and precision improve across all four datasets, ruling out the hypothesis that recall-oriented supervision merely inflates retrieval noise.

Dataset	Samples	Recall@k			Precision@k		
		Base	Tuned	Δ_R	Base	Tuned	Δ_P
HotpotQA	209	70.6	77.3	+6.7	14.5	17.9	+3.4
MuSiQue	348	58.8	61.1	+2.3	14.9	16.8	+1.9
2WikiMQA	271	77.6	84.9	+7.3	23.3	29.6	+6.3
NQ	157	18.7	54.8	+36.1	13.7	16.5	+2.8

cision gains range from +1.9 to +6.3 points, and mirror the direction and ordering of the recall gains (larger on datasets where recall gains are larger).

Pipeline-level evidence: precision rises with reduced retrieval volume. The step-level analysis above holds the retrieval budget fixed at k=6 per call. At the *pipeline level*—where the DPO-tuned ANSWER EVALUATOR additionally reduces the number of retrieval rounds via adaptive early stopping—the precision gains amplify substantially while paragraph recall sees a modest decrease (Table 9). The recall decrease is primarily a denominator effect: the tuned pipeline retrieves 1.5–3.0× fewer paragraphs per question (e.g., HotpotQA 31.0 → 11.2, 2WikiMQA 27.1 → 10.2), so a paragraph-ID–level recall metric naturally drops even when the gold supporting evidence remains covered. Crucially, downstream answer F1 *improves* under the tuned pipeline (Table 1), confirming that evidence sufficiency is preserved despite the lower paragraph count—the discarded retrievals were predominantly non-supporting noise, as evidenced by the 2–3× jump in paragraph precision on every dataset.

Takeaways. (i) We find no measurable cross-role interference when sharing a single LoRA adapter across the QUERY REWRITER and ANSWER EVALUATOR in our two-role setup—both roles show positive movement on their respective metrics after joint training, though we do not claim this guarantees absence of interference in larger or more conflicting role sets. (ii) Recall-grounded QR preferences do not appear to trade off precision at the step level: at a fixed retrieval budget, DPO simultaneously improves Recall@k and Precision@k, which suggests the gains come from *better-ranked* retrieval rather than *wider* retrieval. (iii) At the pipeline level, the combined effect of better-ranked QR retrievals and adaptive AE stopping yields a 2–3× jump in paragraph precision while reducing retrieval volume by 1.5–3.0×; the modest paragraph-recall decrease is a volume-driven denominator effect rather than a loss of evidence sufficiency, as confirmed by improved downstream answer F1 on every dataset (Table 1).

Table 9: Pipeline-level paragraph precision, recall, and average retrieval volume (paragraphs per question) before and after DPO, on the 7B model. Pipeline-level results reflect the combined effect of better-ranked rewrites from the QUERY REWRITER (Table 8) and adaptive early stopping by the ANSWER EVALUATOR. Recall is computed against gold supporting paragraphs at the paragraph-ID level; the recall decrease is dominated by the lower retrieval volume rather than reduced evidence coverage (see paragraph above).

Dataset	Setting	Precision	Recall	Avg #Paras
HotpotQA	Base (no DPO)	0.061	0.879	31.0
	Tuned (full pipeline)	**0.202**	0.814	11.2
MuSiQue	Base (no DPO)	0.065	0.710	30.4
	Tuned (full pipeline)	**0.132**	0.642	17.4
2WikiMQA	Base (no DPO)	0.104	0.944	27.1
	Tuned (full pipeline)	**0.271**	0.852	10.2
NQ	Base (no DPO)	0.117	0.295	14.7
	Tuned (full pipeline)	**0.226**	**0.444**	10.0

Table 10: Ablation study comparing adaptive retrieval with dense-only retrieval, using finetuned model Qwen-2.5-7B-Instruct-DPO. The grey texts shows differences compared to hybrid retrieval.

Dataset	Hybrid Retrieval			Dense-Only		
	EM	F1	Acc	EM	F1	Acc
HotpotQA	49.4	61.3	53.8	48.8 (−0.6)	62.1 (+0.8)	53.0 (−0.8)
MuSiQue	20.8	32.4	25.2	20.4 (−0.4)	31.7 (−0.7)	25.4 (+0.2)
2WikiMQA	48.4	61.2	63.0	49.2 (+0.8)	61.6 (+0.4)	62.8 (−0.2)
NQ	40.0	51.8	51.4	39.2 (−0.8)	52.5 (+0.7)	52.6 (+1.2)
Bamboogle	33.6	44.7	38.4	33.6 (0.0)	44.5 (−0.2)	40.0 (+1.6)
Average	38.4	50.3	46.4	38.2 (−0.2)	50.5 (+0.2)	46.8 (+0.4)

C.5 Agent Component Ablation

To attribute SPARC-RAG's gains to specific agents and rule out "more compute" as the primary explanation, we conduct component-level ablations under a controlled setting: identical fine-tuned backbone (Qwen2.5-7B-Instruct-DPO), same retriever, and W=2 across all variants on MuSiQue and HotpotQA (500-query test sets; see Appendix A.1). We report Acc/EM/F1 together with Avg Tokens/Q as a compute proxy (Table 11).

Table 11: Agent component ablation on MuSiQue and HotpotQA (Qwen2.5-7B-Instruct-DPO, W=2). Avg Tokens/Q is used as a compute proxy. All variants share the same backbone and retriever, so differences isolate the coordination contribution of each agent.

Dataset	Setting	Acc	EM	F1	Tokens/Q	ΔAcc
MuSiQue	Full system	0.252	0.208	0.324	16,477	—
	QR-1: no context	0.162	0.126	0.228	15,869	−0.090
	CM-1: naive concat	0.188	0.144	0.245	71,313	−0.064
	AE-1: fixed D=6	0.236	0.184	0.284	28,603	−0.016
HotpotQA	Full system	0.538	0.494	0.613	6,511	—
	QR-1: no context	0.490	0.438	0.565	7,703	−0.048
	CM-1: naive concat	0.524	0.420	0.557	37,659	−0.014
	AE-1: fixed D=6	0.546	0.492	0.621	25,455	+0.008

QUERY REWRITER is essential for multi-hop coordination. Fully removing the QUERY REWRITER collapses the pipeline into a single-round baseline that is already covered by our single-round RAG baselines in the main results. To isolate its coordination contribution instead, we ablate by removing the global context feedback into the rewriter (**QR-1: no context**)—the rewriter sees only the original question without prior-round evidence. This causes the largest accuracy drop among all variants: 9.0 points on MuSiQue and 4.8 points on HotpotQA, confirming that feeding the shared global context back into query rewriting is a primary contributor to the gains beyond "just rewriting once".

CONTEXT MANAGER's merging dominates over raw compute. Replacing the CONTEXT MANAGER with naive concatenation of all retrieved paragraphs (**CM-1: naive concat**) consumes substantially more compute—MuSiQue Tokens/Q jumps from 16,477 to 71,313 (4.3×) and HotpotQA from 6,511 to 37,659 (5.8×)—yet accuracy *decreases* (6.4 points on MuSiQue, 7.4 EM points on HotpotQA). This directly rebuts the "more compute yields better quality" hypothesis: the gains depend on the CONTEXT MANAGER's ability to consolidate complementary evidence and control context growth, not on raw context size.

ANSWER EVALUATOR provides adaptive efficiency. Disabling adaptive stopping and forcing a fixed depth D=6 (**AE-1: fixed** D=6) dramatically increases compute (MuSiQue +74%; HotpotQA +291%) with negligible quality change ($|\Delta\text{Acc}| \leq 1.6$ points), confirming that the ANSWER EVALUATOR's learned stopping behavior is primarily an efficiency mechanism that avoids wasteful iterations without sacrificing answer quality.

Summary. The three agents exhibit a clear division of labor: QUERY REWRITER drives cross-iteration coordination quality, CONTEXT MANAGER drives evidence consolidation quality, and ANSWER EVALUATOR drives compute efficiency. Notably, the CM ablation shows that naively scaling compute cannot substitute for principled coordination—a finding that directly supports the framing of the unified context management mechanism as the core architectural contribution of SPARC-RAG.

C.6 Adaptive-RAG: Classifier vs. Oracle Routing

We analyze Adaptive-RAG separately in this appendix rather than in Table 1. Its released question-complexity classifier was trained with silver labels tied to a specific LLM/retriever configuration that differs from ours, so its routing decisions transfer imperfectly under our backbone and retriever. To disentangle "bad routing" from "limited strategy set," we additionally report an *oracle* routing variant: for each question we execute all three candidate strategies (*no retrieval*, *single-step RAG*, *multi-step iterative RAG*) on our backbone and retriever, and retain the best per-question score. Oracle routing is therefore an upper bound that no realistic classifier could exceed under the same strategy menu.

Table 12 reports the macro-average across the five datasets. Two observations stand out: (i) oracle routing recovers 17–18 EM points over the classifier on both backbones, confirming that a large portion of the classifier's gap is transfer-induced rather than intrinsic to the strategy menu; (ii) SPARC-RAG matches or exceeds oracle routing on average—SPARC-RAG+DPO at 7B (38.4 EM) exceeds the 7B oracle (34.8 EM), and SPARC-RAG at 32B (45.8 EM) exceeds the 32B oracle (37.9 EM). Since the oracle already enjoys perfect strategy selection, the remaining gap cannot be closed by improving the classifier alone, and reflects the value of iterative evidence composition and context-controlled consolidation beyond single-strategy routing.

C.7 Variance and Statistical Significance

To quantify evaluation-set sampling variability in the Base vs. DPO comparison (Qwen2.5-7B-Instruct backbone), we report paired bootstrap 95% confidence intervals: for each dataset, we resample N questions with replacement 10,000 times

Table 12: Adaptive-RAG classifier vs. oracle routing, compared to SPARC-RAG, as macro-averages across NQ, HotpotQA, 2WikiMQA, MuSiQue, and Bamboogle (Acc / EM / F1). Oracle routing picks the best of the three candidate strategies per question and is an upper bound.

Backbone	Method	Acc	EM	F1
Qwen2.5-7B	Adaptive-RAG (classifier)	30.2	17.3	26.6
	Adaptive-RAG (oracle)	47.4	34.8	47.5
	SPARC-RAG	40.9	35.7	45.2
	SPARC-RAG + DPO	**46.4**	**38.4**	**50.3**
Qwen3-32B	Adaptive-RAG (classifier)	35.8	20.2	32.9
	Adaptive-RAG (oracle)	52.6	37.9	52.8
	SPARC-RAG	**56.5**	**45.8**	**59.4**
	SPARC-RAG + DPO	54.5	42.6	55.9

Table 13: Paired bootstrap 95% CIs for F1 and token cost (Base vs. DPO, Qwen2.5-7B-Instruct), 10,000 resamples per dataset. "Sig." indicates whether the paired-delta 95% CI excludes zero.

Dataset	Metric	Δ (95% CI)	Sig.
HotpotQA	F1	+5.4 (+1.7, +9.2)	Yes
	Tokens	−68% (−72%, −63%)	Yes
MuSiQue	F1	+8.4 (+4.9, +11.9)	Yes
	Tokens	−31% (−38%, −25%)	Yes
2WikiMQA	F1	+5.0 (+0.7, +9.2)	Yes
	Tokens	−66% (−70%, −62%)	Yes
NQ	F1	+5.0 (+1.8, +8.1)	Yes
	Tokens	−75% (−78%, −71%)	Yes
Bamboogle	F1	+2.3 (−5.5, +10.0)	No
	Tokens	−78% (−83%, −72%)	Yes

and recompute the dataset-level F1 and token cost for Base (fixed D=6, W=2) and DPO (adaptive D, W=2) on the same resampled questions each round. The delta's 95% CI excludes zero on $4/5$ datasets for F1 and on $5/5$ datasets for token cost (Table 13). Bamboogle has a wider F1 CI due to its smaller test set (N=125 vs. 500). These CIs capture evaluation-set variability only; we treat multi-seed fine-tuning variance as future work due to training-cost constraints.

C.8 Ablation on retrieval methods

Most baselines in our comparison only use a single retrieval method. To enable a fair comparison, we therefore evaluate our framework under a *dense-only* setting, matching the retrieval configuration used by the baselines. Although mixed retrieval is a common and robust practice in RAG systems, prior work (Thakur et al., 2021) has also noted that strong dense retrievers often perform comparably to hybrid configurations when the corpus is well-structured and the embedding model is high quality. In this ablation, the retriever is fixed to use dense retrieval for all queries.

Table 10 reports results across five datasets. Overall, the differences between the adaptive hybrid setting and dense-only retrieval are small and inconsistent across datasets.

These results suggest that dense retrieval alone is sufficiently effective for our task. This finding indicates that the primary performance improvements of our system stem from other components, such as the iterative query rewriting and answer evaluation modules, rather than the retrieval strategy selection.

C.9 Case study

To illustrate why increasing parallel width improves multi-hop retrieval, we present a representative HotpotQA example (Figure 8). The system is asked: *"Where is Anticimex's parent company headquartered?"* Under a single path (W=1), the system fails to retrieve the document about EQT—the parent company—making the answer unreachable. With two parallel branches (W=2), different sub-queries explore complementary directions: one identifies EQT as the parent company, while the other retrieves EQT's headquarters. Together they recover all necessary evidence and produce the correct final answer.

Analysis. The example in Figure 8 reveals a clear pattern:

- **W=1 fails due to insufficient retrieval.** The single branch retrieves "Anticimex" but never surfaces "EQT", preventing the system from answering the headquarters question.
- **W=2 enables specialization.** One rewritten query targets the parent-company relationship; the other explicitly seeks EQT's headquarters. This division of labor increases coverage.
- **Complementarity resolves ambiguity.** Only when the two branches are combined does the system access all gold evidence (both Anticimex and EQT documents).
- **Merging selects the supported answer.** Although branches propose different answers, the system resolves them by grounding final selection in retrieved evidence.

This case concretely demonstrates our central claim: **parallel width expands the search space**

Question HotpotQA | 5ab969ca55429970cfb8eac9

Where is Anticimex's parent company headquartered?

Gold Answer: **EQT Plaza in Pittsburgh, Pennsylvania**

Gold Supporting Documents

EQT | Anticimex

Width = 1 (Single Path) FAILED

Model Prediction **Not explicitly mentioned**

Execution Trace

Question

Where is Anticimex's parent company headquartered?

Retrieval 6 docs

Cemex | Anticimex | Esteve | ExxonMobil | Telmex | Mexichem

Answer Generation & Evaluation Rejected

Not explicitly mentioned

Query Rewriting

"What is the headquarters of Anticimex's parent company?"

Retrieval 6 docs

Cemex | Anticimex | Esteve | ExxonMobil | Mexichem | Telmex

Final Answer

Not explicitly mentioned

Width = 2 (Parallel Branches) SUCCESS

Model Prediction **Pittsburgh, Pennsylvania**

Execution Trace

Question

Where is Anticimex's parent company headquartered?

Retrieval 6 docs

Cemex | Anticimex | Esteve | ExxonMobil | Telmex | Mexichem

Answer Generation & Evaluation Rejected

Stockholm, Sweden

Query Rewriting

B0: *"EQT headquarters location"*

B1: *"relationship between Anticimex and EQT"*

Retrieval

B0: EQT | EQT Plaza | EQT Ventures | EQT (disambiguation) | EQT Partners | Headquarters and Headquarters Squadron

B1: Anticimex | Esterified estrogens/methyltestosterone | AEM (psychedelic) | Etifoxine | Dexetimide | Guaifenesin/phenylephrine

Answer Generation & Merging

B0: **Pittsburgh, Pennsylvania**

B1: **Stockholm, Sweden**

Final Answer: **Pittsburgh, Pennsylvania**

Retrieval Coverage Comparison

Gold Document	W=1	W=2	
		B0	B1
EQT	✗	✓	✗
Anticimex	✓	✗	✓
Total Gold Retrieved	**1/2**	**2/2**	

Key Insight: Branch Complementarity

- **Branch 0** directly searches for "EQT headquarters location", successfully retrieving the "EQT" document that contains the headquarters information
- **Branch 1** explores the "relationship between Anticimex and EQT", retrieving the "Anticimex" document that establishes EQT as the parent company
- The two branches are **complementary**: Branch 0 finds the headquarters location, while Branch 1 confirms the parent-subsidiary relationship
- W=1 fails because it cannot retrieve the "EQT" document needed to answer where the headquarters is located. W=2 succeeds by having Branch 0 directly target the key entity: **Pittsburgh, Pennsylvania**

Figure 8: Case study illustrating how W=2 retrieves complementary evidence that W=1 misses.

and recovers evidence that sequential refinement alone cannot reach.

C.10 Inference-Time Cost Breakdown

To decompose the per-query compute cost behind each scaling configuration—complementing the token-only column in Appendix C.12—we report a per-configuration inference-time breakdown of SPARC-RAG on HotpotQA and MuSiQue. We sweep five fixed $W \times D$ settings (1×4, 1×8, 2×4, 2×8, 4×4) and include our adaptive DPO variant; all configurations share the same backbone (Qwen2.5-7B-Instruct), retriever, and evaluation splits (per-query averages on 100-query subsets; the **DPO (2×Adp)** row uses the 500-query test subset from Table 1). For each configuration we measure the average number of LLM calls (counting QUERY REWRITER, ANSWER GENERATOR, and ANSWER EVALUATOR invocations), retrieval operations, total passages returned by the retriever, input+output tokens processed by the LLM, the post-consolidation context size (sum of CONTEXT MANAGER new-context tokens across all updates), per-query wall-clock latency, and F1 (Tables 14–15).

Table 14: Per-configuration inference-time breakdown on HotpotQA (Qwen2.5-7B-Instruct). LLM calls count QR+AG+AE invocations; Retr and Passages are retrieval operations and returned passages; Tok (K) is LLM input+output tokens; Ctx is the sum of CONTEXT MANAGER consolidated-context tokens across updates; Lat is per-query wall-clock latency (measurement protocol below).

$W \times D$	LLM calls	Retr	Passages	Tok (K)	Ctx (tok)	Lat (s)	F1
1×4	7.0	4.0	24	8.2	576	12.7	52.9
1×8	15.0	8.0	48	19.0	1,176	27.5	57.9
2×4	20.0	7.0	42	11.9	1,191	26.2	61.0
2×8	44.0	15.0	89	37.5	3,037	86.3	56.2
4×4	37.3	12.8	76	17.7	2,393	44.8	57.2
DPO (2×Adp)	7.3	2.8	17	5.9	501	13.2	**61.3**

Table 15: Per-configuration inference-time breakdown on MuSiQue (same setup as Table 14).

$W \times D$	LLM calls	Retr	Passages	Tok (K)	Ctx (tok)	Lat (s)	F1
1×4	7.0	4.0	23	10.6	734	11.1	21.5
1×8	15.0	8.0	46	24.8	1,565	24.0	20.2
2×4	20.0	7.0	40	14.4	1,417	24.9	21.6
2×8	44.0	15.0	86	44.0	3,558	81.5	27.5
4×4	37.3	12.8	73	18.6	2,591	40.1	24.6
DPO (2×Adp)	14.8	5.4	31	15.2	1,293	26.8	**32.4**

Context compression. The CONTEXT MANAGER compresses the raw retrieved evidence into a smaller consolidated note: at $W{=}2, D{=}4$, HotpotQA consolidates 42 passages (∼ 3,525 raw tokens) to 1,191 context tokens—a 3.0× compression—and MuSiQue consolidates 40 passages (∼ 5,329 raw tokens) to 1,417 context tokens—a 3.8× compression. This keeps the LLM input manageable as depth grows and supports our claim that the *aggregation quality* rather than the raw context size drives the gains (Appendix C.5).

Improvements come from better coordination, not just more compute. Four conclusions are directly visible in Tables 14–15:

- **Width helps at matched compute.** At total budget $W \times D{=}8$, 2×4 beats 1×8 on HotpotQA (61.0 vs. 57.9 F1) with 37% fewer tokens (11.9K vs. 19.0K), because width explores complementary reformulations instead of redoing the same chain deeper.
- **DPO attains the best quality at the lowest cost.** DPO (2×Adp) obtains the highest F1 on both datasets (61.3 HotpotQA, 32.4 MuSiQue) with the fewest LLM calls (7.3–14.8), retrievals (2.8–5.4), and tokens (5.9–15.2K). On HotpotQA it matches the best fixed configuration (61.3 vs. 61.0) with 50% fewer tokens and 64% fewer LLM calls; on MuSiQue it improves F1 by +4.9 over the best fixed configuration while still using far less compute than 2×8.
- **Over-scaling *either* axis hurts.** Extending depth (HotpotQA 2×4→2×8: 61.0→56.2 F1 at 3.2× tokens) or widening beyond $W{=}2$ (HotpotQA 2×4→4×4: 61.0→57.2 F1 at 1.9× LLM calls) both degrade the Pareto tradeoff. Effective scaling requires a balanced operating point, not more compute along any single axis.
- **Comparable cost to retrieval-augmented baselines.** DPO (2×Adp) uses a similar per-query token budget to DeepNote (5.9K vs. 6.2K on HotpotQA; see Appendix C.2) while improving F1 by +2.1.

Latency measurement protocol. Wall-clock latencies are measured end-to-end from the arrival of a query at SPARC-RAG to the emission of its final answer, and *include* all retrieval operations, CONTEXT MANAGER, ANSWER GENERATOR, ANSWER EVALUATOR, and QUERY REWRITER calls. Measurements are taken on the shared inference server described in Appendix A.4 (four NVIDIA RTX PRO 6000 Blackwell GPUs, 384 GiB total

VRAM; vLLM v0.10.2 with continuous batching and tensor parallelism across the four GPUs; sparse retrieval via Elasticsearch 7.10.2 on the same host). Queries are processed *sequentially* (one query at a time; no cross-query batching), so reported latencies reflect the full serialized cost of each configuration. Each number is the mean over the 100-query (or 500-query for the DPO row) subset; in deployment, cross-query batching and prefix/KV caching would reduce absolute latencies but preserve the relative ordering between configurations.

C.11 Failure Analysis

While SPARC-RAG+DPO substantially improves both accuracy and efficiency, two residual failure modes persist.

Residual branch overlap. At $W{=}2$, the average pairwise Jaccard overlap between the two branches' retrieved document sets remains non-trivial even after DPO—0.19–0.25 across datasets (down from 0.26–0.48 before DPO; see Appendix B.2 for the definition). Recall-oriented preferences clearly reduce redundancy but do not eliminate it, leaving some fraction of parallel capacity under-utilized on questions whose rewrites naturally overlap.

Premature stopping on hard multi-hop questions. DPO reduces the ANSWER EVALUATOR's incorrect-acceptance rate by 3–8 points on 7B and 11–15 points on 32B (Figure 7), but the rate remains non-zero on the harder multi-hop questions—roughly 9% on HotpotQA, 12% on MuSiQue, and 14% on 2WikiMQA for the 7B model. These are questions where partial evidence appears superficially sufficient, causing the model to stop before compositional reasoning completes.

Accuracy drops monotonically with hop count. The second failure mode is most visible when breaking MuSiQue down by required hop count (Table 16). Accuracy falls from 44.5% on 2-hop questions to 13.0% on 4-hop questions—a 3.4× absolute gap. Each additional hop raises the probability that some intermediate retrieval misses evidence or that the ANSWER EVALUATOR stops before the full chain is assembled, and errors compound across hops.

Depth and width help in different regimes. Looking across the grid-search results (Appendix C.12), the relative value of depth vs. width depends on the question type. On long-chain MuSiQue, additional depth pays off more than additional width: 2×6 ($F1{=}25.7$) beats 4×2 ($F1{=}22.6$) at comparable compute, consistent with the need for sequential evidence chaining. On 2-hop HotpotQA, moderate width at moderate depth is more efficient: 2×4 ($F1{=}52.3$) beats 1×8 ($F1{=}49.6$) with roughly half the tokens, because the two hops can often be explored in parallel. This regime-dependence suggests that fixed $W{=}2$ is a reasonable default but may leave additional performance on the table at the extreme ends (very short or very long chains).

Table 16: SPARC-RAG+DPO performance on MuSiQue by required hop count (Qwen2.5-7B-Instruct). Accuracy falls sharply as hop count grows, indicating that long-chain compositional reasoning remains the dominant residual failure mode.

Hops	Count	Acc	EM	F1
2	254	44.5	27.6	42.4
3	154	27.3	18.2	26.1
4	92	13.0	5.4	14.2

C.12 Width × Depth Grid Search Results

Table 17 presents the full grid search results across different width (W) and depth (D) configurations using the Qwen2.5-7B-Instruct model on multi-hop QA datasets. For the efficiency of the grid search, we report the result on a 100-sample subset of the dev set. For each configuration, we report accuracy, F1 score, exact match (EM), average score, and token usage across four multi-hop QA datasets. These results inform our selection of $W{=}2$ as the default width setting, which provides the best trade-off between performance and computational cost.

Table 17: Width × Depth grid search results for Qwen2.5-7B-Instruct on multi-hop QA datasets.

W×D	HotpotQA					2WikiMQA					MuSiQue					Bamboogle				
	acc.	f1	em	avg.	tok.	acc.	f1	em	avg.	tok.	acc.	f1	em	avg.	tok.	acc.	f1	em	avg.	tok.
1×2	37.0	45.5	35.0	39.2	3.2k	57.0	52.7	44.0	51.2	3.1k	11.0	20.7	10.0	13.9	4.2k	30.4	37.1	29.6	32.4	3.5k
1×4	44.0	52.9	38.0	45.0	8.2k	56.0	52.2	42.0	50.1	7.1k	15.0	21.5	13.0	16.5	10.6k	36.0	41.8	34.4	37.4	8.0k
1×6	41.0	51.4	38.0	43.5	13.2k	52.0	51.4	44.0	49.1	11.3k	18.0	23.3	16.0	19.1	17.3k	31.2	38.6	30.4	33.4	12.7k
1×8	46.0	57.9	45.0	49.6	19.0k	50.0	50.2	42.0	47.4	15.8k	15.0	20.2	13.0	16.1	24.8k	39.2	46.3	36.8	40.8	17.7k
2×2	43.0	52.1	42.0	45.7	3.8k	53.0	49.8	42.0	48.3	5.5k	15.0	20.7	12.0	15.9	4.1k	33.6	38.4	32.0	34.7	5.8k
2×4	50.0	61.0	46.0	52.3	9.4k	56.0	53.3	43.0	50.8	13.6k	12.0	21.6	11.0	14.8	11.1k	35.2	42.4	35.2	37.6	14.6k
2×6	50.0	56.2	42.0	49.4	15.8k	61.0	56.1	44.0	53.7	22.2k	26.0	30.0	21.0	25.7	18.8k	36.8	46.2	34.4	39.1	24.3k
2×8	46.0	56.2	41.0	47.7	30.7k	62.0	55.9	42.0	53.3	31.6k	21.0	27.5	18.0	22.2	37.7k	36.8	41.3	33.6	37.2	35.3k
4×2	50.0	55.9	43.0	49.6	5.9k	52.0	50.5	44.0	48.8	8.6k	22.0	27.9	18.0	22.6	6.7k	36.0	41.1	32.8	36.6	9.3k
4×4	48.0	57.2	44.0	49.8	17.4k	64.0	61.3	49.0	58.1	24.0k	17.0	24.6	17.0	19.5	20.4k	37.6	43.9	36.8	39.4	25.9k
4×6	50.0	58.7	43.0	50.6	31.7k	59.0	56.0	47.0	54.0	40.7k	21.0	30.9	18.0	23.3	37.2k	35.5	44.2	31.4	37.0	45.3k
4×8	48.0	55.1	37.0	46.7	62.4k	56.0	53.9	44.0	51.3	53.4k	20.0	26.6	18.0	21.5	72.9k	34.4	39.0	31.2	34.9	66.3k

D Prompts for the LLM Agents

Multi-Path Dispatch Rewrite Prompt (QUERY REWRITER)

```
You are an intelligent assistant in a
Retrieval-Augmented Generation (RAG) system. Your
goal is to (a) diagnose retrieval needs for the
current question and (b) produce exactly {N}
rewritten queries, each paired with the most
suitable retrieval strategy for that specific
rewrite.
Information:
- Original Query: {query}
- Current Query: {current_query}
- Context: {context}
Available Retrieval Strategies:
- bm25 (sparse / lexical): Prioritizes exact token
and phrase matches.
- dense (semantic / vector similarity): Matches by
meaning despite paraphrases.
Instructions:
1. Use the context to reflect on what is missing
to answer the query. Think about both the big
picture and the small atomic facts that might need
verification.
2. Generate exactly {N} rewritten queries.
- Do not just paraphrase — each query should
explore a different angle, granularity, or fact.
- Avoid near-duplicates.
- Each query must serve a distinct retrieval
purpose.
3. For each query, select the most suitable
retrieval strategy:
- Use bm25 when exact names, phrases, or quoted
terms matter; remove "wh" words.
- Use dense when searching for meanings,
definitions, or related concepts.
```

Multi-Path Dispatch Output Format (QUERY REWRITER)

```
Output Format (strict):
1. First provide your analysis and rationale in a
<think> block, including per-item strategy
justification.
2. Then output exactly {N} query rewrites using
the following structure:
<queries>
<item rank="1">
<strategy>bm25|dense</strategy>
<query>...</query>
</item>
<item rank="2">
<strategy>bm25|dense</strategy>
<query>...</query>
</item>
...
<item rank="{N}">
<strategy>bm25|dense</strategy>
<query>...</query>
</item>
</queries>
3. End your response with the literal tag [END].
Output:
```

Context Manager Prompt (CONTEXT MANAGER)

```
Act as the context manager for a
Retrieval-Augmented Generation (RAG) system. Your
job is to maintain a single, up-to-date note that
contains all the information relevant to answering
the original query. Please ensure that the note
includes all original text information useful for
answering the question.
Steps:
- Based on the retrieved documents, supplement the
notes with content not yet included but useful for
answering the question.
- Resolve conflicts: if statements disagree, keep
the most reliable or recent version.
End your response with the literal tag [END].
Original query: {query}
Old note: {note}
New information: {new_context}
Updated note:
```

Answer Selection Prompt (CONTEXT MANAGER)

```
Question: {question}
All Generated Answers:
{answer_blocks}
Based on all the answers and reasoning provided,
select the best answer. Consider accuracy and
relevance to the question. Give the final answer
directly. Then, provide a direct, concise, and
accurate answer inside <answer> </answer> tags.
End your response with the literal tag [END].
Final answer:
```

Context Merging Prompt (CONTEXT MANAGER)

```
You are an expert at combining contexts in a
Retrieval Augmented Generation system for
answering question {question}. Here are several
notes: {reasoning_list}.
Combine the above notes into a single note that
includes all information and is useful for final
answer generation. Please ensure that the note
includes all original text information useful for
answering the question. End your response with the
literal tag [END].
Your response:
```

Answer Generator Prompt (ANSWER GENERATOR)

```
Answer the question based on the given notes.
Output ONLY the exact answer in as few words as
possible.
Do not include the question, reasoning, or any
extra text.
End your response with the literal tag [END].
The following are given notes:
{note}
Question: {query}
Answer:
```