

SPARKLE: A Structured and Plug-and-play Agentic Retrieval Policy for Adaptive RAG Models

Anonymous ACL submission

Abstract

Adaptive retrieval-augmented generation (RAG) models offer an effective approach for integrating external knowledge. However, existing methods either rely on frozen large language models (LLMs) without explicit supervision or require costly LLM finetuning. Therefore, we propose **SPARKLE**, a structured and plug-and-play agentic retrieval policy where an additional *proxy model* is introduced to control the retrieval process. The proxy model leverages *knowledge graph*-based reasoning to make retrieval decisions in a structured manner, while operating independently of the retriever and the LLM. This plug-and-play design allows SPARKLE to generalise across different retrievers and LLMs. SPARKLE is optimised via reinforcement learning (RL), treating the retriever and the LLM as part of the environment. To enable more effective exploration during RL training, we further introduce a binary tree-structured rollout strategy. Experiments on three in-domain and four out-of-domain QA benchmarks show that SPARKLE outperforms state-of-the-art baselines, achieving average improvements of 9.17% and 2.85%, respectively¹.

1 Introduction

Retrieval-augmented generation (RAG) models have demonstrated remarkable performance in enhancing large language models (LLMs) for question answering (QA) tasks (Ram et al., 2023; Lin et al., 2024; Wei et al., 2025). To fulfil the varying information requirements of different questions and support multi-step reasoning, recent research has explored **adaptive or agentic RAG** models (Jiang et al., 2023; Jeong et al., 2024; Feng et al., 2025), which dynamically retrieve the necessary information during the LLM’s reasoning process.

Recent work on adaptive RAG has primarily focused on three key aspects (Su et al., 2024): (1)

¹Code is available at: <https://anonymous.4open.science/r/sparkle-6A7A>

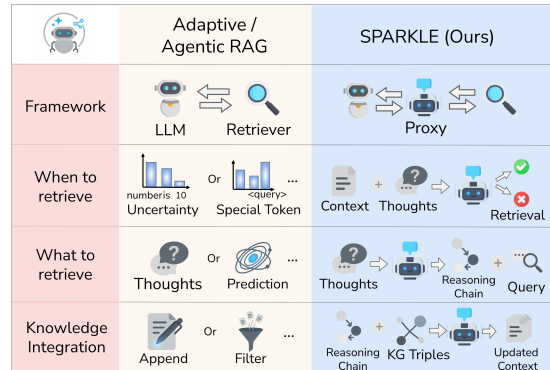


Figure 1: Comparison between existing adaptive/agentic RAG models and the proposed SPARKLE.

(2) *when to retrieve*, i.e., identifying the appropriate moment to retrieve; (3) *what to retrieve*, i.e., formulating effective queries for missing information; (4) *how to integrate retrieved information* into the LLM’s reasoning process for accurate generation. Figure 1 provides a summary of existing methods. One line of work relies on the LLM’s abilities to control the retrieval process, triggering retrieval based on uncertainty (Jiang et al., 2023; Su et al., 2024) or generating a special token that indicates the need for external information (Li et al., 2025a). However, these methods often use frozen LLMs without additional training, which can lead to sub-optimal performance due to the lack of explicit supervision (Jin et al., 2025). To this end, another line of work explores supervised finetuning or reinforcement learning to train LLMs to make more effective retrieval decisions (Asai et al., 2024; Xu et al., 2024; Jin et al., 2025; Song et al., 2025). While these methods have demonstrated improved performance, they often require substantial computational resources to train the LLM and may negatively impact the LLM’s generalisation to other tasks (Hu et al., 2025; Li et al., 2025b).

These limitations highlight the need for a more efficient and modular solution to enable adaptive retrieval without modifying the LLM itself. Ideally, such an approach should decouple retrieval con-

trol from the LLM, enabling independent learning while preserving the model’s generalisation capability. To this end, we introduce a lightweight *proxy model* to manage the retrieval process. This design offers a retrieval policy that is both lightweight to train and able to generalise across different retrievers and LLMs. Building on this idea, we propose **SPARKLE**, a **Structured and Plug-and-play** agentic retrieval policy for improved Adaptive Retrieval and **KnowLEdge** integration in RAG. SPARKLE employs the proxy model to make step-wise decisions on when and what to retrieve, and how to incorporate retrieved information. In this process, the proxy model leverages structured knowledge graphs to better formulate retrieval queries and integrate relevant information into the reasoning context. Figure 1 provides a comparison between SPARKLE and existing adaptive RAG models.

The proxy model in SPARKLE operates in an agentic manner by taking on different functional roles, each guided by targeted instructions. These roles serve as specialised agents that work collaboratively to support the adaptive RAG process. SPARKLE defines three agents: (1) a *Retrieval Decision Agent* for determining when external knowledge is needed, (2) a *Query Formulation Agent* for identifying knowledge gaps and generating a targeted search query, and (3) a *Knowledge Integration Agent* for selecting and integrating relevant information. Specifically, at each LLM’s reasoning step, the Retrieval Decision agent decides whether additional knowledge is needed based on the current context and the LLM’s intermediate thoughts. If retrieval is required, the Query Formulation and Knowledge Integration agents are invoked to obtain an updated context for the next step of reasoning.

In practice, we observe that the LLM-generated thoughts, particularly those from large reasoning models like DeepSeek (Guo et al., 2025a), may contain distracting or unnecessary content (Xia et al., 2025), making it difficult to accurately identify knowledge gaps and formulate queries. Therefore, following prior work that employed knowledge graph (KG)-based reasoning chains to identify useful information from noisy content (Fang et al., 2024b), SPARKLE extracts a KG-based reasoning chain from the LLM’s intermediate thoughts, providing a structured and informative representation of its reasoning trajectory. This structure helps the Query Formulation agent identify knowledge gaps more precisely for targeted retrieval. Moreover, retrieved documents are decomposed into knowl-

edge triples, allowing the Knowledge Integration agent to perform fine-grained selection of relevant content for incorporation into the context.

We leverage proximal policy optimisation (PPO) to train the proxy model, while keeping the retriever and the LLM frozen. To facilitate more effective exploration during training, we propose a *binary tree-structured rollout strategy* with a pruning mechanism, which allows the proxy model to explore multiple reasoning paths while discarding unpromising branches. We assess SPARKLE on three in-domain and four out-of-domain QA datasets. Compared to the strongest adaptive RAG baselines, SPARKLE achieves significant improvements, with an average gain of 9.17% on in-domain datasets and 2.85% on out-of-domain datasets.

Our contributions can be summarised as follows: (1) We propose SPARKLE, a plug-and-play agentic retrieval policy that uses KG-based reasoning chains to identify knowledge gaps and integrate relevant information; (2) We introduce a binary tree-structured rollout strategy that enables more effective exploration of retrieval policies; (3) Empirical results on both in-domain and out-of-domain datasets demonstrate that SPARKLE consistently outperforms existing adaptive RAG baselines.

2 Agentic Retrieval Policy

We focus on the adaptive RAG setting (Su et al., 2024), where the LLM dynamically retrieves external information to answer questions. Formally, given a question q and a corpus $\mathcal{C} = \{d_i\}_{i=1}^N$, the goal is to generate an answer y through a sequence of reasoning steps. To facilitate adaptive retrieval, SPARKLE employs a lightweight proxy model to make retrieval decisions and integrate information from retrieved documents. The proxy model operates in an agentic manner to support the adaptive RAG process. Figure 2 illustrates the overall framework and training strategy of SPARKLE.

2.1 Agentic Retrieval Formulation

SPARKLE defines three agents to support the adaptive RAG process: a *Retrieval Decision Agent*, a *Query Formulation Agent* and a *Knowledge Integration Agent*. Following Chen et al. (2025a), we model their collective behaviour as a *multi-agent Markov Decision Process* (MDP), where each agent operates over its own state and action space, while collaboratively contributing to the overall process.

Formally, we define the multi-agent MDP pro-

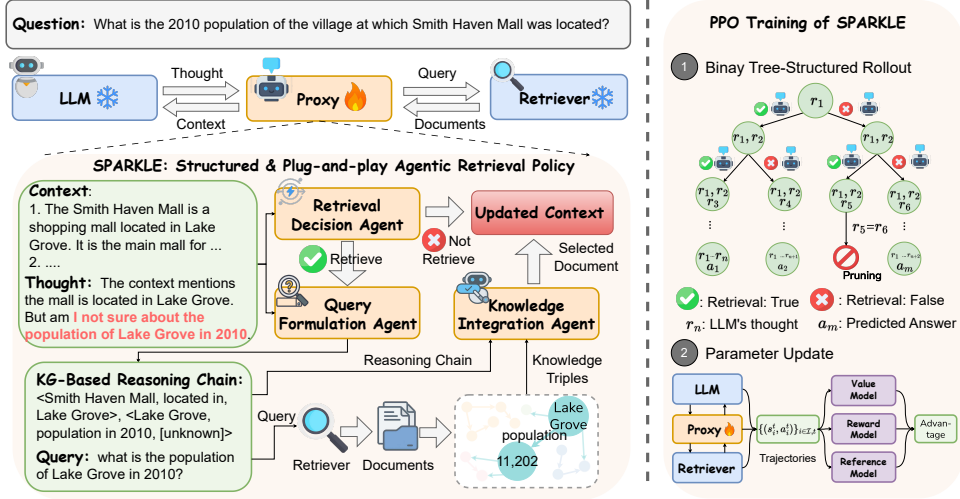


Figure 2: (left) Overview of SPARKLE. At each reasoning step, the *Retrieval Decision Agent* determines whether retrieval is needed. If so, the *Query Formulation Agent* and the *Knowledge Integration Agent* are invoked to generate a search query and incorporate relevant information into the context, respectively. (right) PPO training of SPARKLE, which employs a binary tree-structured rollout to explore multiple reasoning paths during reinforcement learning.

cess as a tuple $(\mathcal{I}, \{\mathcal{S}_i\}_{i \in \mathcal{I}}, \{\mathcal{A}_i\}_{i \in \mathcal{I}}, \mathcal{R})$, where $\mathcal{I} = \{1, 2, 3\}$ represents the set of agents, \mathcal{S}_i and \mathcal{A}_i represent the state space and action space of the i -th agent, respectively. Each agent follows its policy $\pi_i : \mathcal{S}_i \rightarrow \mathcal{A}_i$ to select actions based on its state. Intuitively, a state can be viewed as the agent’s input, while the action corresponds to its output. A trajectory is a sequence of states and actions generated as agents interact with the environment over time. \mathcal{R} is the system-level reward function that provides feedback to guide cooperative behaviour across agents.

2.2 Agent Roles and Collaborative Strategy

In this section, we detail how each agent operates and collaborates within this framework.

Retrieval Decision Agent. This agent determines whether external knowledge is required at each reasoning step. At the t -th step, the agent observes a state $s_1^t = (q, c_t, r_{\leq t})$, where c_t denotes the current context consisting of documents retrieved in previous steps, and $r_{\leq t}$ is the intermediate thoughts generated by the LLM up to step t . Based on this state, the agent selects an action $a_1^t \in \{\text{“Yes”}, \text{“No”}\}$, where “Yes” indicates that a retrieval is required for subsequent reasoning, while “No” indicates external knowledge is not needed. The prompt for this agent can be found in Appendix A.1.

Query Formulation Agent. If retrieval is required at step t , the Query Formulation agent is activated to obtain a query q_t based on the state $s_2^t = (q, r_{\leq t})$. While one could directly generate the query from the LLM’s thoughts, such strategy often leads to

suboptimal performance (see **RQ4** in §4.2) due to the presence of distracting or speculative content in LLM’s thoughts, such as:

Question: Who is the mother of the Texas businessman that co-founded Ignite! Learning?
Thoughts: Ignite! Learning was co-founded by someone. . . I think Neil Bush? Yes, Neil Bush. He’s the son of George H. W. Bush. Wait, is that right? Maybe it was another Bush. . . no, it’s Neil. Then his mother — probably Barbara Bush. Though I’m not entirely sure.

This example illustrates how the LLM’s thoughts can include uncertainty and speculative content, making it difficult to directly extract an accurate search query. To address this, our Query Formulation agent first constructs a *KG-based reasoning chain* g_t (Fang et al., 2024b, 2025), which abstracts the LLM’s thoughts into a structured form that captures the essential reasoning steps, such as:

$\langle \text{Ignite!Learning, co-founded by, Neil Bush} \rangle, \langle \text{Neil Bush, child of, [unknown]} \rangle$

This structured representation filters out distracting, speculative or irrelevant information from the raw thoughts, preserving only the core reasoning steps necessary for the task. By doing so, it makes the underlying knowledge gaps more explicit, allowing the agent to clearly identify what additional information is needed to advance the reasoning process and to formulate a targeted search query q_t accordingly. The prompt for the Query Formulation agent can be found in Appendix A.2.

Knowledge Integration Agent. Given the query q_t , the retriever returns a list of documents $\mathcal{D}_t \subset \mathcal{C}$,

which often contain irrelevant content (Yoran et al., 2024). To facilitate more fine-grained identification of useful information, following Fang et al. (2025), SPARKLE decomposes each document into a set of knowledge triples². This structured representation enables more precise reasoning over the retrieved content by reducing textual noise. The Knowledge Integration agent observes a state $s_3^t = (q, g_t, \mathcal{K}_t)$, where \mathcal{K}_t is the set of knowledge triples extracted from \mathcal{D}_t . It then selects the triple that best fills the most important gap in the current reasoning chain. The prompt for this agent can be found in Appendix A.3. Given the selected triple, SPARKLE adds the source document³ from which this triple is extracted to the context c_t , resulting in the updated context c_{t+1} for subsequent reasoning.

Collaborative Strategy. Finally, we introduce how these agents collaborate with the LLM to support adaptive retrieval and reasoning. Given a question q and an empty initial context c_0 , the LLM generates the first thought⁴. The Retrieval Decision agent then determines whether external knowledge is required. If so, the Query Formulation and the Knowledge Integration agents are activated to obtain an updated context c_1 for subsequent reasoning. Otherwise, the context remains unchanged. This process is performed iteratively until the LLM generates a final answer. The prompt for the LLM and the pseudo code for the whole process can be found in Appendices A.4 and B, respectively.

3 Agentic Policy Optimisation

Since the effectiveness of the agentic retrieval policy can only be assessed after the LLM completes its full reasoning and produces a final answer, it is natural to frame the training process as a RL problem. We adopt PPO (Schulman et al., 2017) to train the proxy model to learn a retrieval policy that improves the answer generation performance.

3.1 Binary Tree-Structured Rollout

In vanilla on-policy PPO training, a rollout stage executes the current policy to collect trajectories of states, actions and rewards, which are used for optimisation. However, in practice, we found that standard rollouts are constrained by the capabilities

²Since triple extraction is query-agnostic, it can be performed offline without adding inference-time cost.

³We add the source document rather than the triple to preserve its full context and avoid missing information.

⁴We explicitly prompt the LLM to produce an intermediate reasoning step enclosed in “<think> ... </think>” tags.

of the proxy model, resulting in limited exploration of more effective retrieval behaviours. Therefore, inspired by previous works (Feng et al., 2025; Chen et al., 2025a), we propose a *binary tree-structured rollout strategy* that enables broader exploration by expanding multiple retrieval paths during RL:

$$\mathbf{a}^t = \begin{cases} a_1^t = \text{“Yes”}, a_i^t \sim \pi_i(s_i^t), \forall i \in \{2, 3\} \\ a_1^t = \text{“No”} \end{cases} \quad (1)$$

where $\mathbf{a}^t = \{a_i^t\}_{i \in \mathcal{I}}$ denotes the joint action at step t . Specifically, at each step, instead of relying on the Retrieval Decision agent to make a single retrieval decision, we explicitly explore *both* possible actions, i.e., retrieval and non-retrieval. For the retrieval branch, we sample the subsequent actions directly from the current policies of the Query Formulation and the Knowledge Integration agents to simulate the retrieval process. By iteratively applying this process over successive steps, we obtain a binary decision tree containing multiple trajectories: $\tau = \{(s_i^t, a_i^t)\}_{i \in \mathcal{I}, t}$ for optimisation. An illustration of this process can be found in Figure 2.

To improve rollout efficiency, we introduce a pruning mechanism to reduce redundant branches. We observe that when the retrieval branch selects a document already present in the current context, the resulting updated context remains unchanged. Then the LLM would produce an identical next reasoning step for both the retrieval and non-retrieval branches. Therefore, we prune the retrieval branch to avoid unnecessary computation. Moreover, to mitigate exponential growth with reasoning depth, we constrain binary expansion to the first L reasoning steps. After that, each trajectory proceeds as a standard rollout using the current policy.

3.2 Monte Carlo Reward Assignment

In our multi-agent MDP formulation, each agent operates over its own state and action space, yet only a single system-level reward \mathcal{R} is observed at the end of each trajectory. To assign credit to individual agent decisions, we follow previous work (Chen et al., 2025a) and employ a Monte Carlo-based reward assignment scheme:

$$r(s_i^t, a_i^t) = \mathbb{R}_{\tau \sim \pi}[\mathcal{R}(\tau) | s_i^t, a_i^t] \approx \frac{\sum_{j \in \mathcal{N}(s_i^t, a_i^t)} \mathcal{R}_j}{|\mathcal{N}(s_i^t, a_i^t)|}, \quad (2)$$

where $\mathcal{N}(s_i^t, a_i^t)$ denotes leaf nodes reachable from (s_i^t, a_i^t) and \mathcal{R}_j is the reward of the leaf node j .

Since SPARKLE focuses on learning an effective retrieval policy, we define the system-level reward

Category	Model	In-Domain Datasets				Out-of-Domain Datasets						
		HotPotQA	2Wiki	NQ	Avg.	Δ	MuSiQue	Bamboogle	TQA	ASQA	Avg.	Δ
w/o retrieval	LLM [*]	27.85	29.25	16.20	24.43	↑ 31.61	11.94	40.49	46.50	29.26	32.05	↑ 14.29
Single-Step RAG	RAG [*]	49.69	41.60	33.30	41.53	↑ 14.51	21.04	37.72	58.20	30.36	36.83	↑ 9.51
	TRACE [*]	54.32	43.28	37.60	45.07	↑ 10.97	22.61	47.36	64.20	31.84	41.50	↑ 4.84
Iterative RAG	IRCoT [*]	59.05	47.13	36.60	47.59	↑ 8.45	27.63	43.08	61.90	21.47	38.52	↑ 7.82
	ITER-RETGEN [*]	61.03	<u>57.09</u>	38.60	<u>52.24</u>	↑ 3.80	30.14	48.92	64.80	24.64	42.13	↑ 4.21
	KiRAG [*]	<u>61.39</u>	55.85	36.00	51.80	↑ 4.96	31.07	47.36	63.20	31.84	43.37	↑ 2.97
Adaptive RAG	Self-RAG [*]	26.69	20.90	36.20	27.93	↑ 28.11	12.46	15.89	43.50	32.81	26.17	↑ 20.17
	DRAGIN [*]	48.23	48.46	26.10	40.93	↑ 15.11	27.64	42.47	57.80	29.25	39.29	↑ 7.05
	Adaptive-RAG [*]	56.62	46.06	33.30	45.33	↑ 10.71	26.43	38.87	57.80	29.42	38.13	↑ 8.21
	SlimPLM [*]	50.95	36.29	37.90	41.71	↑ 14.33	20.14	36.75	63.40	<u>36.30</u>	39.15	↑ 7.19
	Auto-RAG [*]	49.30	52.40	<u>38.90</u>	46.87	↑ 9.17	21.21	44.89	62.00	10.98	34.77	↑ 11.57
C-3PO [*]	56.67	48.87	33.60	46.38	↑ 9.66	<u>31.87</u>	<u>48.96</u>	62.50	30.61	<u>43.49</u>	↑ 2.85	
Ours	SPARKLE [*]	63.14 [†]	64.78 [†]	40.20 [†]	56.04	-	32.85 [†]	50.19 [†]	64.80	37.50 [†]	46.34	-

Table 1: Performance (%) of SPARKLE and baselines across both in-domain and out-of-domain datasets. The best and the second-best results are marked in bold and underlined, respectively, and [†] indicates p-value<0.05 compared with the strongest baseline. The average improvement of SPARKLE to each baseline is highlighted in the Δ column. ^{*} and ^{*} indicate whether an approach requires additional training or is training-free, respectively.

\mathcal{R} as a combination of retrieval quality and final answer accuracy: $\mathcal{R} = \alpha \cdot \text{Recall}(c_T) + (1 - \alpha) \cdot \text{F1}(y)$, where c_T is the final context and y is the predicted answer, while α is the trade-off parameters⁵. More details about training strategy are in Appendix C.

4 Experiments

4.1 Experimental Setup

Datasets. We conduct experiments on three types of QA datasets: *multi-hop QA* datasets such as **HotPotQA** (Yang et al., 2018), **2WikiMultiHopQA** (2Wiki) (Ho et al., 2020), **MuSiQue** (Trivedi et al., 2022), and **Bamboogle** (Press et al., 2023); *single-hop QA* datasets such as **Natural Questions** (NQ) (Kwiatkowski et al., 2019) and **TriviaQA** (TQA) (Joshi et al., 2017); and *long-form QA* dataset such as **ASQA** (Stelmakh et al., 2022). Following previous work (Chen et al., 2025a), we randomly sample 1,000⁶ examples from the original test/dev set as our evaluation set. More detailed information about the datasets is in Appendix D.1.

Baselines. Since SPARKLE focuses on retrieval policy within adaptive RAG, we compare it against RAG models with different retrieval strategies: (1) **LLM**, which generates answers without retrieval; (2) **single-step RAG**, which employs a single-step retrieval strategy, such as the standard RAG and TRACE (Fang et al., 2024b); (3) **iterative RAG**, which adopts a multi-step retrieval strategy with a fixed number of iterations, such as IRCoT (Trivedi et al., 2023), ITER-RETGEN (Shao et al., 2023), and KiRAG (Fang et al., 2025); (4) **adaptive RAG**,

such as Self-RAG (Asai et al., 2024), DRAGIN (Su et al., 2024), Auto-RAG (Yu et al., 2024), Adaptive-RAG (Jeong et al., 2024), SlimPLM (Tan et al., 2024) and C-3PO (Chen et al., 2025a). More details about baselines can be found in Appendix D.2.

Evaluation. We adopt standard evaluation metrics for each dataset to assess QA performance. Specifically, we report **Exact Match (EM)** for NQ and TQA, **F1** for HotPotQA, 2Wiki, MuSiQue, and Bamboogle, and **ROUGE** for ASQA.

Implementation Details. We leverage E5 (Wang et al., 2022) as the retriever, Qwen2.5-7B-Instruct (Yang et al., 2024) as the LLM, and Qwen2.5-3B-Instruct as the proxy model. Following Fang et al. (2025), we leverage Llama3 (Dubey et al., 2024) to extract knowledge triples from documents, which are precomputed and cached to avoid additional overhead. A single proxy model is shared across all agents, and the proxy is trained on a subset of samples randomly drawn from the training sets of HotPotQA, 2Wiki and NQ. More implementation and training details, as well as hyperparameter settings can be found in Appendix D.3.

4.2 Results and Analysis

We provide our main experimental results in this section. Additional results are in Appendix E.

(RQ1): How does SPARKLE perform compared with baselines? We evaluate the performance of SPARKLE on both *in-domain* datasets, which are used for proxy training, and *out-of-domain* datasets that are not seen during training. Table 1 presents the results of SPARKLE and the baselines, yielding the following findings: (1) SPARKLE consistently achieves the best performance on the three

⁵The sensitivity analysis of α is in Appendix E.2.

⁶We retain all evaluation examples for datasets with fewer than 1,000 instances, such as Bamboogle and ASQA.

Category	Model	Retriever: E5 → BGE						LLM: Qwen2.5 → DeepSeek-R1-Distill-Qwen-14B					
		HotPotQA	2Wiki	MuSiQue	NQ	Avg.	Δ	HotPotQA	2Wiki	MuSiQue	NQ	Avg.	Δ
Single-Step RAG	RAG ❄️	50.49	41.18	21.13	31.30	36.03	↑ 13.51	52.39	43.74	24.69	31.50	38.08	↑ 16.89
	TRACE ❄️	53.86	42.99	22.37	36.94	39.04	↑ 10.50	53.99	48.57	25.56	36.70	41.21	↑ 13.76
Iterative RAG	IRCoT ❄️	56.37	43.75	28.36	36.20	41.17	↑ 8.37	60.14	63.04	33.61	38.10	48.97	↑ 6.00
	ITER-RETGEN ❄️	62.28	56.06	30.17	35.60	46.03	↑ 3.51	57.26	56.32	28.05	35.30	44.23	↑ 10.74
	KIRAG 🔥	61.92	60.56	30.37	36.45	47.33	↑ 2.21	58.01	63.23	33.25	35.00	47.37	↑ 6.92
Adaptive RAG	Self-RAG 🔥	26.78	19.90	11.78	33.10	22.89	↑ 26.65	-	-	-	-	-	-
	DRAGIN ❄️	47.84	46.25	27.44	25.80	36.83	↑ 12.71	50.40	44.01	27.78	31.03	38.31	↑ 16.18
	Adaptive-RAG 🔥	53.76	43.95	26.23	31.30	38.81	↑ 10.73	58.47	56.88	33.70	35.80	46.21	↑ 8.76
	SlimPLM 🔥	51.06	35.33	20.61	37.40	36.10	↑ 13.44	51.12	38.01	27.03	31.90	37.02	↑ 17.95
	Auto-RAG 🔥	49.23	52.19	22.38	38.00	40.45	↑ 9.09	-	-	-	-	-	-
	C-3PO 🔥	59.39	50.26	31.50	35.70	44.21	↑ 5.33	-	-	-	-	-	-
Ours	SPARKLE 🔥	63.52 [†]	63.75 [†]	32.87 [†]	38.00	49.54	-	67.13 [†]	74.22 [†]	39.62 [†]	38.90 [†]	54.97	-

Table 2: Performance (%) of SPARKLE and baselines under different retrievers and LLMs, without retraining of the proxy model. Notations follow those in Table 1. Some results using the Deepseek-distilled model are missing because these baselines either require LLM training or are incompatible with the model’s output format.

in-domain datasets, with 3.80% and 9.17% average score improvements over the strongest iterative and adaptive RAG baselines, respectively. These results highlight the effectiveness of our proxy-based retrieval policy in enhancing adaptive RAG pipelines; (2) SPARKLE also achieves the best performance across the four out-of-domain datasets, outperforming the best iterative and adaptive RAG baselines by 2.97% and 2.85% on average, respectively, demonstrating its strong generalisation capability to unseen datasets.

(RQ2): Can SPARKLE generalise to different retrievers and LLMs without retraining? We assess the plug-and-play ability of the proxy model by evaluating its performance across different retrievers and LLMs *without* retraining. We change the retriever from E5 to BGE (Xiao et al., 2024), and the LLM from Qwen2.5 to DeepSeek-R1-Distill-Qwen-14B (Guo et al., 2025a). Table 2 indicates that when only changing the retriever model, SPARKLE achieves the best average performance across four datasets, surpassing the strongest baseline by 2.21%. Moreover, when changing the LLM, the improvement becomes more substantial, with an average gain of 6.00% over the best-performing baseline. These results demonstrate that SPARKLE generalises effectively across different retrievers and LLMs, highlighting its adaptability as a plug-and-play retrieval policy without retraining.

(RQ3): How does SPARKLE perform in terms of training overhead, effectiveness, and generalisation compared to RL-finetuned LLMs? Table 3 evaluates the effectiveness and generalisation of SPARKLE, we compare it against two RL-finetuned baselines: R1-Searcher (Song et al., 2025) and Search-R1 (Jin et al., 2025). We conduct experiments using the same LLM (Qwen2.5-7B), and further evaluate SPARKLE’s effectiveness by

scaling it to stronger LLMs, such as QwQ-32B and GPT-4o, without any retraining. Additionally, to evaluate the generalisation ability, we report results on ASQA, a long-form QA benchmark that differs significantly from the phrase-level answers RL models are typically trained on.

Table 3 yields the following findings: (1) Compared to RL-finetuned approaches that require updating the full LLM and retraining for each model variant, SPARKLE trains only a lightweight 3B proxy model on just 7.5K examples (vs. up to 169K for the baselines), while keeping the LLM frozen. This substantially reduces the training overhead and enables generalisation across LLMs without retraining; (2) In terms of effectiveness, SPARKLE achieves competitive performance when using the same LLM as RL-finetuned baselines and further surpasses them substantially when combined with stronger LLMs, an advantage that RL approaches lack due to their reliance on model-specific finetuning; (3) In terms of generalisation, RL-finetuned methods show a clear drop on ASQA⁷, likely due to their training objective, which emphasises short, phrase-level answers, limiting their ability to generalise to open-ended generation. In contrast, by not modifying the LLM, SPARKLE preserves its ability to generate coherent and long-form answers.

(RQ4): What are the effects of each agent and the training strategy in SPARKLE? To explore the effects of the three agents within SPARKLE, we introduce the following variants: *w/o RD Agent*, *w/o QF agent*, and *w/o KI agent*, where the Retrieval Decision (RD), Query Formulation (QF), and Knowledge Integration (KI) agents are individually removed⁸. In each case, the corresponding

⁷We slightly modify the original prompts of RL-finetuned models to output long-form answers.

⁸Details about these variants are in Appendix D.2.

Model	Retriever	LLM	Proxy	Trainable Parameters	Training Size	HotPotQA	2Wiki	NQ	ASQA	Avg.
(1) R1-Searcher	E5	Qwen2.5-7B	-	7B	8.4K	68.75	68.06	41.00	25.66	50.86
(2) Search-R1	E5	Qwen2.5-7B	-	7B	169K	67.01	65.61	48.21	29.20	52.51
(3) SPARKLE	E5	Qwen2.5-7B	Qwen2.5-3B	3B	7.5K	63.14	64.78	40.20	37.50	51.41
(4) SPARKLE	E5	Qwen2.5-32B	Qwen2.5-3B	-	-	67.59	69.12	41.90	36.98	53.90
(5) SPARKLE	E5	QwQ-32B	Qwen2.5-3B	-	-	67.90	73.88	41.30	37.80	55.22
(6) SPARKLE	E5	GPT-4o	Qwen2.5-3B	-	-	70.15	74.70	42.38	38.23	56.37

Table 3: Training statistics and performance (%) comparison of SPARKLE and RL baselines, where and denote trainable and frozen components, respectively, and indicates the proxy model is reused *without any retraining*. SPARKLE supports generalisation to different LLMs in a plug-and-play manner.

Model	HotPotQA	2Wiki	NQ	ASQA	Δ
SPARKLE	63.14	64.78	40.20	37.50	-
<i>Effectiveness of different Agents</i>					
w/o RD Agent	60.55 [†]	60.79 [†]	37.89 [†]	34.22 [†]	↓ 3.04
w/o QF Agent	61.84 [†]	59.86 [†]	38.80 [†]	36.57 [†]	↓ 2.14
w/ Direct QF Agent	61.81 [†]	59.29 [†]	39.10 [†]	36.47 [†]	↓ 2.24
w/o KI Agent	61.55 [†]	61.39 [†]	38.90 [†]	36.98 [†]	↓ 1.70
w/ Direct KI Agent	51.78 [†]	53.69 [†]	37.20 [†]	35.77 [†]	↓ 6.80
<i>Effectiveness of RL Training</i>					
w/o RL Training	56.48 [†]	55.55 [†]	39.23	35.83 [†]	↓ 4.63
w/ Standard Rollout	61.14 [†]	58.54 [†]	39.89	36.52 [†]	↓ 2.38

Table 4: Ablation results (%) of SPARKLE, where [†] indicates p-value<0.05 compared with SPARKLE and Δ denotes average drop of each variant to SPARKLE.

functionality is handled by other agents or heuristics. In addition, we introduce two variants to examine the effects of structured reasoning: *w/ Direct QF*, where the QF agent generates queries directly from the LLM’s thoughts without using the KG-based reasoning chain, and *w/ Direct KI agent*, where the KI agent selects a document directly from the retrieved set without using KG triples.

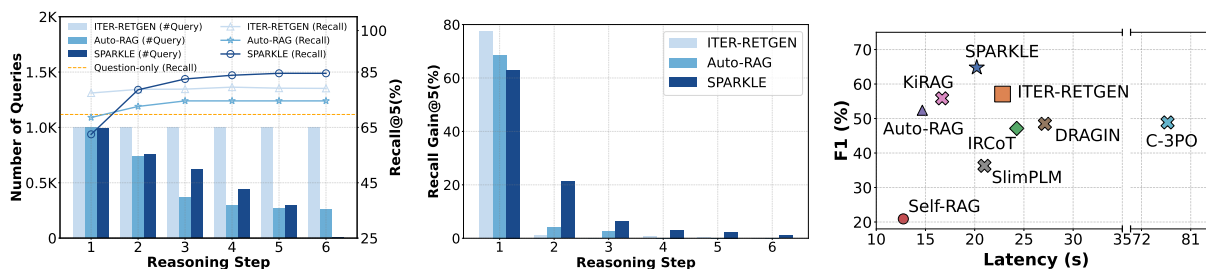
Table 4 reports SPARKLE and its variants’ performance on four datasets, revealing the following findings: (1) Compared to SPARKLE, removing the RD agent, the QF agent, and the KI agent results in average drops of 3.04%, 2.14%, and 1.70%, respectively, indicating that all three agents contribute meaningfully to overall performance, with the RD agent being the most critical for guiding retrieval decisions; (2) For the QF agent, removing the KG-based reasoning chain, i.e., *w/ Direct QF agent*, leads to an average decline of 2.24%, which is likely due to the presence of distracting content in LLM’s thoughts. This highlights the effectiveness of structuring the LLM’s reasoning through a KG-based chain, which helps distil essential information and enables more accurate identification of knowledge gaps for targeted retrieval; (3) For the KI agent, *w/ Direct KI agent* suffers from a substantial drop of 6.80% compared to SPARKLE. The significant decline is likely due to the noise within retrieved documents, which makes it difficult for the proxy model to select the most relevant one,

leading to missed critical information.

To examine the effects of our training strategy, we introduce two variants: *w/o RL Training*, where the proxy model is not trained, and *w/ Standard Rollout*, where the tree structured rollout is replaced with standard on-policy rollout. The results of SPARKLE and the variants are reported in Table 4, which shows that removing RL leads to an average drop of 4.63%, highlighting the importance of end-to-end optimisation. Using the standard rollout results in an average 2.38% drop, likely due to insufficient exploration, which limits the model’s ability to learn more effective retrieval behaviours.

(RQ5): Can SPARKLE generate high-quality queries and effectively retrieve relevant documents? To understand the retrieval behaviour of SPARKLE, we analyse the number of generated queries and the cumulative Recall@5 at each reasoning step, where cumulative Recall@5 refers to the recall computed over the union of retrieved documents up to that step. As shown in Figure 3a, SPARKLE generates fewer queries than the iterative baseline ITER-RETGEN, especially in later steps, indicating its ability to avoid redundant retrieval. While SPARKLE generates more queries than the adaptive baseline Auto-RAG, its higher cumulative recall indicates that these queries are more effective in retrieving relevant information.

To further assess query quality, we compute *step-wise recall gain*, defined as the increase in cumulative Recall@5 from step $t-1$ to step t . Intuitively, this metric reflects how much new relevant information is introduced by the query at each step, thereby indicating the quality of the query. The results in Figure 3b show that SPARKLE achieves higher recall gains at later steps, particularly at step 2, suggesting that its generated queries are more effective at retrieving new, relevant information. In contrast, baselines like ITER-RETGEN and Auto-RAG rely heavily on initial retrieval, with diminishing recall gains in later steps. These results highlight SPARKLE’s ability to generate high-quality queries that lead to more effective retrieval.



(a) Number of queries (bars) and cumulative recall (lines) at each reasoning step. (b) Step-wise recall gain, i.e., the increase in recall compared to the previous step. (c) Effectiveness–efficiency trade-off.

Figure 3: Analysis of SPARKLE’s retrieval performance and effectiveness-efficiency trade-off on the 2Wiki dataset.

(RQ6): How efficient is SPARKLE compared to baselines? As shown in Figure 3c, SPARKLE achieves a strong trade-off between effectiveness and efficiency. It obtains the highest F1 score with relatively low inference latency. While some baselines have slightly lower latency, they fall short in accuracy. Particularly, compared to recent adaptive RAG models such as C-3PO and DRAGIN, SPARKLE not only achieves substantially higher accuracy but also reduces latency by a large margin.

5 Related Work

Adaptive and Agentic RAG Models. RAG models have demonstrated strong performance in QA tasks (Ram et al., 2023; Fang et al., 2024a; Fan et al., 2024). To better support multi-step reasoning, recent studies have explored *adaptive and agentic* retrieval strategies that enable models to dynamically decide when and what to retrieve during inference (Jeong et al., 2024; Guo et al., 2025b; Feng et al., 2025; Singh et al., 2025). Early approaches rely on the LLM’s predicted probabilities to make retrieval decisions and formulate queries (Jiang et al., 2023; Su et al., 2024; Yao et al., 2024). Recent work adopts an agentic perspective (Yao et al., 2023; Li et al., 2025a; Chen et al., 2025a), defining specialised agents to retrieve and integrate information, an approach that we follow in this paper.

Another line of research directly finetunes the LLM to optimise its retrieval behaviour using SFT or RL (Asai et al., 2024; Yu et al., 2024; Jin et al., 2025; Song et al., 2025; Chen et al., 2025b). However, these methods require extensive training and may compromise the base LLM’s generalisation ability. In contrast, SPARKLE uses a lightweight proxy model to guide retrieval in an agentic manner, enabling plug-and-play integration with different retrievers and LLMs without retraining.

While both SPARKLE and C-3PO (Chen et al., 2025a) use a proxy model in RAG, they differ in

several key aspects. First, C-3PO relies on a planning stage, where a larger LLM, e.g., Qwen2-72B, is used to generate a fixed plan to guide subsequent retrieval and reasoning. In contrast, SPARKLE enables dynamic retrieval planning based on the LLM’s evolving thoughts, allowing the retrieval decisions to adapt on-the-fly to the current reasoning context. Second, C-3PO filters retrieved documents by passing all retrieved content into an agent for selection, while SPARKLE leverages a structured representation, KG triples, to enable more precise identification of useful information, which yields better performance as shown in RQ4 of §4.2.

KG-Enhanced RAG Models. Recent studies have explored integrating KGs into RAG models (Edge et al., 2024; Gutiérrez et al., 2024; Panda et al., 2024; Fang et al., 2024a; Liu et al., 2025). Some methods construct a graph index for retrieval (Edge et al., 2024; Gutiérrez et al., 2024), while others decompose documents into triples to support structured reasoning (Li and Du, 2023; Fang et al., 2024b) and fine-grained retrieval (Fang et al., 2025). In contrast to these approaches that rely on static KG indices or retrieval stages, SPARKLE dynamically constructs reasoning-time KG chains to guide adaptive retrieval and knowledge integration. We provide an empirical comparison between our approach and graph RAG models in Appendix E.3.

6 Conclusion

This paper proposes SPARKLE, a structured and plug-and-play agentic retrieval policy for adaptive RAG. SPARKLE employs a lightweight proxy model and KG-based reasoning to make step-wise decisions about when and what to retrieve, and how to integrate retrieved information. Extensive experiments show that SPARKLE consistently outperforms strong adaptive RAG baselines across in-domain and out-of-domain benchmarks, achieving average gains of 9.17% and 2.85%, respectively.

603 Limitations

604 We identify the following limitations of our work:
605 (1) SPARKLE relies on precomputed knowledge
606 triples to construct the KG-based reasoning chain.
607 In our implementation, we adopt the triple extrac-
608 tion method from Fang et al. (2025), which has
609 been empirically shown to produce high-quality
610 results. In future work, we plan to explore alterna-
611 tive KG construction strategies and evaluate how
612 the quality of extracted triples affects downstream
613 performance. (2) Our training strategy employs
614 PPO combined with a binary tree-structured roll-
615 out strategy to optimise the proxy model. While
616 this approach demonstrates strong empirical per-
617 formance, we leave the exploration of alternative
618 RL algorithms, such as Group Relative Policy Op-
619 timisation (GRPO) (Shao et al., 2024), to future
620 work. (3) A key motivation of our design is to
621 maintain a lightweight and efficient retrieval proxy
622 model. Therefore, we primarily experiment with
623 a lightweight 3B proxy model. While this choice
624 achieves competitive performance with low train-
625 ing cost, it remains an open question whether scal-
626 ing up the proxy model could yield improved per-
627 formance. We leave the exploration of larger proxy
628 architectures to future work.

629 References

630 Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and
631 Hannaneh Hajishirzi. 2024. Self-RAG: Learning to
632 retrieve, generate, and critique through self-reflection.
633 In *The Twelfth International Conference on Learning
634 Representations*.

635 Guoxin Chen, Minpeng Liao, Peiying Yu, Dingmin
636 Wang, Zile Qiao, Chao Yang, Xin Zhao, and Kai Fan.
637 2025a. C-3PO: Compact plug-and-play proxy opti-
638 mization to achieve human-like retrieval-augmented
639 generation. In *International Conference on Machine
640 Learning*.

641 Mingyang Chen, Linzhuang Sun, Tianpeng Li, Haoze
642 Sun, Yijie Zhou, Chenzheng Zhu, Haofen Wang,
643 Jeff Z Pan, Wen Zhang, Huajun Chen, and 1 oth-
644 ers. 2025b. Learning to reason with search for
645 LLMs via reinforcement learning. *arXiv preprint
646 arXiv:2503.19470*.

647 Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey,
648 Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman,
649 Akhil Mathur, Alan Schelten, Amy Yang, Angela
650 Fan, and 1 others. 2024. The Llama 3 herd of models.
651 *arXiv preprint arXiv:2407.21783*.

652 Darren Edge, Ha Trinh, Newman Cheng, Joshua
653 Bradley, Alex Chao, Apurva Mody, Steven Truitt,

and Jonathan Larson. 2024. From local to global: A
graph RAG approach to query-focused summariza-
tion. *arXiv preprint arXiv:2404.16130*.

654
655
656
657 Wenqi Fan, Yujuan Ding, Liangbo Ning, Shijie Wang,
658 Hengyun Li, Dawei Yin, Tat-Seng Chua, and Qing
659 Li. 2024. A survey on RAG meeting llms: Towards
660 retrieval-augmented large language models. In *Pro-
661 ceedings of the 30th ACM SIGKDD Conference on
662 Knowledge Discovery and Data Mining*, pages 6491–
663 6501. ACM.

664 Jinyuan Fang, Zaiqiao Meng, and Craig Macdonald.
665 2024a. REANO: Optimising retrieval-augmented
666 reader models through knowledge graph generation.
667 In *Proceedings of the 62nd Annual Meeting of the
668 Association for Computational Linguistics (Volume
669 1: Long Papers)*, pages 2094–2112.

670 Jinyuan Fang, Zaiqiao Meng, and Craig Macdon-
671 ald. 2024b. TRACE the evidence: Constructing
672 knowledge-grounded reasoning chains for retrieval-
673 augmented generation. In *Findings of the Associa-
674 tion for Computational Linguistics: EMNLP*, pages
675 8472–8494.

676 Jinyuan Fang, Zaiqiao Meng, and Craig Macdonald.
677 2025. KiRAG: Knowledge-driven iterative retriever
678 for enhancing retrieval-augmented generation. In
679 *Proceedings of the 63rd Annual Meeting of the As-
680 sociation for Computational Linguistics (Volume 1:
681 Long Papers)*, pages 18969–18985.

682 Wenfeng Feng, Chuzhan Hao, Yuwei Zhang, Jingyi
683 Song, and Hao Wang. 2025. AirRAG: Activat-
684 ing intrinsic reasoning for retrieval augmented gen-
685 eration using tree-based search. *arXiv preprint
686 arXiv:2501.10053*.

687 Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song,
688 Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong
689 Ma, Peiyi Wang, Xiao Bi, and 1 others. 2025a.
690 Deepseek-R1: Incentivizing reasoning capability in
691 LLMs via reinforcement learning. *arXiv preprint
692 arXiv:2501.12948*.

693 Hanghui Guo, Jia Zhu, Shimin Di, Weijie Shi, Zhangze
694 Chen, and Jiajie Xu. 2025b. DioR: Adaptive cogni-
695 tive detection and contextual retrieval optimization
696 for dynamic retrieval-augmented generation. In *Pro-
697 ceedings of the 63rd Annual Meeting of the Associa-
698 tion for Computational Linguistics (Volume 1: Long
699 Papers)*, pages 2953–2975.

700 Bernal Jiménez Gutiérrez, Yiheng Shu, Yu Gu, Michi-
701 hiro Yasunaga, and Yu Su. 2024. HippoRAG: Neu-
702robiologically inspired long-term memory for large
703 language models. *arXiv preprint arXiv:2405.14831*.

704 Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara,
705 and Akiko Aizawa. 2020. Constructing A multi-
706 hop QA dataset for comprehensive evaluation of
707 reasoning steps. In *Proceedings of the 28th Inter-
708 national Conference on Computational Linguistics*,
709 pages 6609–6625.

710	Chuxuan Hu, Yuxuan Zhu, Antony Kellermann, Caleb Biddulph, Suppakit Waiwitlikhit, Jason Benn, and Daniel Kang. 2025. Breaking barriers: Do reinforcement post training gains transfer to unseen domains? <i>arXiv preprint arXiv:2506.19733</i> .	767
711		768
712		769
713		770
714		771
715	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 <i>Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)</i> , pages 7036–7050.	772
716		773
717		774
718		775
719		776
720		777
721		
722		
723	Zhengbao Jiang, Frank F. Xu, Luyu Gao, Zhiqing Sun, Qian Liu, Jane Dwivedi-Yu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. Active retrieval augmented generation. In <i>Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing</i> , pages 7969–7992.	778
724		779
725		780
726		781
727		782
728		783
729		784
730		
731		
732		
733		
734	Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan Arik, Dong Wang, Hamed Zamani, and Jiawei Han. 2025. Search-R1: Training LLMs to reason and leverage search engines with reinforcement learning. <i>arXiv preprint arXiv:2503.09516</i> .	785
735		786
736		787
737		788
738		789
739		790
740		791
741		
742		
743		
744	Jiajie Jin, Yutao Zhu, Xinyu Yang, Chenghao Zhang, and Zhicheng Dou. 2024. FlashRAG: A modular toolkit for efficient retrieval-augmented generation research. <i>arXiv preprint arXiv:2405.13576</i> .	792
745		793
746		794
747		795
748		796
749		
750		
751		
752		
753		
754		
755		
756		
757	Mandar Joshi, Eunsol Choi, Daniel S. Weld, and Luke Zettlemoyer. 2017. TriviaQA: A large scale distantly supervised challenge dataset for reading comprehension. In <i>Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics</i> , pages 1601–1611.	797
758		798
759		799
760		800
761		801
762		
763		
764		
765		
766		
767		
768		
769		
770		
771		
772		
773		
774		
775		
776		
777		
778		
779		
780		
781		
782		
783		
784		
785		
786		
787		
788		
789		
790		
791		
792		
793		
794		
795		
796		
797		
798		
799		
800		
801		
802		
803		
804		
805		
806		
807		
808		
809		
810		
811		
812		
813		
814		
815		
816		
817		
818		
819		
820		
821		
822		
823		
824		
825		
826		
827		
828		
829		
830		
831		
832		
833		
834		
835		
836		
837		
838		
839		
840		
841		
842		
843		
844		
845		
846		
847		
848		
849		
850		
851		
852		
853		
854		
855		
856		
857		
858		
859		
860		
861		
862		
863		
864		
865		
866		
867		
868		
869		
870		
871		
872		
873		
874		
875		
876		
877		
878		
879		
880		
881		
882		
883		
884		
885		
886		
887		
888		
889		
890		
891		
892		
893		
894		
895		
896		
897		
898		
899		
900		
901		
902		
903		
904		
905		
906		
907		
908		
909		
910		
911		
912		
913		
914		
915		
916		
917		
918		
919		
920		
921		
922		
923		
924		
925		
926		
927		
928		
929		
930		
931		
932		
933		
934		
935		
936		
937		
938		
939		
940		
941		
942		
943		
944		
945		
946		
947		
948		
949		
950		
951		
952		
953		
954		
955		
956		
957		
958		
959		
960		
961		
962		
963		
964		
965		
966		
967		
968		
969		
970		
971		
972		
973		
974		
975		
976		
977		
978		
979		
980		
981		
982		
983		
984		
985		
986		
987		
988		
989		
990		
991		
992		
993		
994		
995		
996		
997		
998		
999		
1000		

823	Aditi Singh, Abul Ehtesham, Saket Kumar, and Tala Talaie Khoei. 2025. Agentic retrieval-augmented generation: A survey on agentic RAG. <i>arXiv preprint arXiv:2501.09136</i> .	<i>Conference on Research and Development in Information Retrieval</i> , pages 641–649.	879
824			880
825			
826			
827	Huatong Song, Jinhao Jiang, Yingqian Min, Jie Chen, Zhipeng Chen, Wayne Xin Zhao, Lei Fang, and Ji-Rong Wen. 2025. R1-Searcher: Incentivizing the search capability in LLMs via reinforcement learning. <i>arXiv preprint arXiv:2503.05592</i> .	Shicheng Xu, Liang Pang, Mo Yu, Fandong Meng, Huawei Shen, Xueqi Cheng, and Jie Zhou. 2024. Un-supervised information refinement training of large language models for retrieval-augmented generation. In <i>Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)</i> , pages 133–145.	881
828			882
829			883
830			884
831			885
832	Ivan Stelmakh, Yi Luan, Bhuwan Dhingra, and Ming-Wei Chang. 2022. ASQA: factoid questions meet long-form answers. In <i>Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing</i> , pages 8273–8288.	An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, and 1 others. 2024. Qwen2.5 technical report. <i>arXiv preprint arXiv:2412.15115</i> .	886
833			887
834			888
835			889
836			890
837	Weihang Su, Yichen Tang, Qingyao Ai, Zhijing Wu, and Yiqun Liu. 2024. DRAGIN: Dynamic retrieval augmented generation based on the real-time information needs of large language models. In <i>Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)</i> , pages 12991–13013.	Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. 2018. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In <i>Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing</i> , pages 2369–2380.	891
838			892
839			893
840			894
841			895
842			896
843			897
844	Jiejun Tan, Zhicheng Dou, Yutao Zhu, Peidong Guo, Kun Fang, and Ji-Rong Wen. 2024. Small models, big insights: Leveraging slim proxy models to decide when and what to retrieve for LLMs. In <i>Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics</i> , pages 4420–4436.	Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing reasoning and acting in language models. In <i>The Eleventh International Conference on Learning Representations</i> .	898
845			899
846			900
847			901
848			902
849			903
850	Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. 2022. MuSiQue: Multi-hop questions via single-hop question composition. <i>Trans. Assoc. Comput. Linguistics</i> , 10:539–554.	Zijun Yao, Weijian Qi, Liangming Pan, Shulin Cao, Linmei Hu, Weichuan Liu, Lei Hou, and Juanzi Li. 2024. SEAKR: Self-aware knowledge retrieval for adaptive retrieval augmented generation. <i>arXiv preprint arXiv:2406.19215</i> .	904
851			905
852			906
853			907
854	Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. 2023. Interleaving retrieval with chain-of-thought reasoning for knowledge-intensive multi-step questions. In <i>Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)</i> , pages 10014–10037.	Ori Yoran, Tomer Wolfson, Ori Ram, and Jonathan Berant. 2024. Making retrieval-augmented language models robust to irrelevant context. In <i>International Conference on Learning Representations</i> .	908
855			909
856			910
857			911
858			912
859			913
860			914
861	Liang Wang, Nan Yang, Xiaolong Huang, Binxing Jiao, Linjun Yang, Daxin Jiang, Rangan Majumder, and Furu Wei. 2022. Text embeddings by weakly-supervised contrastive pre-training. <i>arXiv preprint arXiv:2212.03533</i> .	Tian Yu, Shaolei Zhang, and Yang Feng. 2024. Auto-RAG: Autonomous retrieval-augmented generation for large language models. <i>arXiv preprint arXiv:2411.19443</i> .	915
862			916
863			917
864			918
865			919
866	Zhepei Wei, Wei-Lin Chen, and Yu Meng. 2025. InstructRAG: Instructing retrieval-augmented generation via self-synthesized rationales. In <i>The Thirteenth International Conference on Learning Representations</i> .	Nan Zhang, Prafulla Kumar Choubey, Alexander R. Fabri, Gabriel Bernadett-Shapiro, Rui Zhang, Prasenjit Mitra, Caiming Xiong, and Chien-Sheng Wu. 2025. SiReRAG: Indexing similar and related information for multihop reasoning. In <i>The International Conference on Learning Representations</i> .	920
867			921
868			922
869			923
870			924
871	Heming Xia, Chak Tou Leong, Wenjie Wang, Yongqi Li, and Wenjie Li. 2025. TokenSkip: Controllable chain-of-thought compression in llms. <i>arXiv preprint arXiv:2502.12067</i> .	A Prompts	925
872			926
873			927
874			928
875	Shitao Xiao, Zheng Liu, Peitian Zhang, Niklas Muenighoff, Defu Lian, and Jian-Yun Nie. 2024. C-Pack: Packed resources for general chinese embeddings. In <i>Proceedings of the 47th International ACM SIGIR</i>	A.1 Prompt for Retrieval Decision Agent	929
876			930
877			931
878			932

support further reasoning. This decision is made in a *context-aware* and *step-wise* manner, i.e., considering not only what has already been retrieved and reasoned over, but also whether the current thought introduces new information needs.

Prompt Used for Retrieval Decision Agent

Instruction:
 You are an intelligent assistant. Your job is to decide whether external information retrieval is required to continue solving the problem.

You will be given the following inputs:

- Question: The original user query.
- Previous Reasoning: A list of prior reasoning thoughts the model has generated.
- Current Reasoning Step: The latest reasoning step to evaluate.
- Retrieved Context: Text previously retrieved that may contain relevant information.

Your task is to answer whether the **current reasoning step** requires external retrieval...

Output Format
 ONLY output "Yes (retrieval is required)" or "No (retrieval is not required)" and do not include any other text.

Inputs:
Retrieved Context: {*context*}
Question: {*original question*}
Previous Reasoning: {*reasoning history*}
Current Reasoning Step: {*current reasoning thought*}

Figure 4: Prompt for Retrieval Decision Agent.

A.2 Prompt for Query Formulation Agent

The prompt used in the Query Formulation agent is provided in Figure 5. The agent takes as input the original question and the LLM-generated reasoning thoughts up to the current step, and follows a three-step process to formulate a high-quality retrieval query: (1) extract a KG-based reasoning chain; (2) identify the next missing piece of information needed to progress; and (3) generate a retrieval query targeting that gap. While the process is logically divided into three steps for clarity and structure, all steps are executed within a single LLM invocation.

A.3 Prompt for Knowledge Integration Agent

The prompt for the Knowledge Integration agent is shown in Figure 8. The agent receives: (1) the original question, (2) the reasoning chain generated by the Query Formulation agent, (3) the retrieval query, and (4) a set of candidate knowledge triples. These candidate triples are extracted by first retrieving documents using the query, then applying the triple extraction and filtering method from Fang et al. (2025) to decompose the retrieved documents

Prompt Used for Query Formulation Agent

Instruction:
 You are an intelligent reasoning assistant. Your task is to help bridge the information gap between a current reasoning trace (thought) and the full answer to a given question by generating a high-quality retrieval query.

You will follow these three steps:

[Step 1: Generate Reasoning Chain]
 Given the question and the corresponding thought, extract the information that is directly helpful for answering the question. Represent each piece of information as a structured knowledge triple in the format <head; relation; tail>. Output Format: <head1; relation1; tail1>, <head2; relation2; tail2>, ... <headN; relationN; tailN>

[Step 2: Identify Knowledge Gaps]
 Based on the extracted reasoning chain, identify the **next piece of information** that is required to continue reasoning toward the answer to the question. This could be a missing fact, relation, or intermediate concept that, if known, would allow the reasoning to proceed.
 Output Format: To continue answering the question, we need to know: [describe the next necessary piece of information]

[Step 3: Formulate Retrieval Query]
 Write a concise and effective natural language query that can help retrieve the specific information identified in Step 2. The query should be focused, contain the key entities and relations, and be suitable for a retrieval system.

Inputs:
Question: {*original question*}
Thoughts: {*reasoning thoughts up to current step*}

Figure 5: Prompt for Query Formulation Agent.

into structured triples. The agent aims to select a single triple that best supports the next reasoning step by extending the current chain. This design allows for fine-grained relevant knowledge selection.

A.4 Prompt for LLMs

To support step-wise reasoning in SPARKLE, the LLM is required to generate one reasoning thought at each step. For non-reasoning LLMs, such as Qwen2.5-7B-Instruct, we explicitly instruct the model to wrap each intermediate thought within “<think>” and “</think>” tags, making it easier to extract individual steps. The prompt used for this setting is shown in Figure 6. For reasoning LLMs, such as DeepSeek-R1-Distill-Qwen-14B, which generates multi-step reasoning thoughts by default, we segment the reasoning steps based on the newline character. The prompt used in this case is in Figure 7.

B Pseudo Code for SPARKLE

The overall inference procedure of SPARKLE is illustrated in Algorithm 16, which outlines how the proxy model collaborates with the LLM to it-

Algorithm 1: End-to-End Inference Procedure of SPARKLE.

Input : question q , a retriever model $Retriever(\cdot)$, an LLM model $LLM(\cdot)$, a KG corpus \mathcal{G} , the Retrieval Decision agent $RD(\cdot)$, the Query Formulation agent $QF(\cdot)$, and the Knowledge Integration agent $KI(\cdot)$.

```
1  $t = 0$ ; // current reasoning step
2  $c_0 = \emptyset$ ; // initialise context
3 while  $t$  is less than maximum reasoning steps do
4    $r_t = LLM(q, c_t)$ ;
5   if  $r_t$  contains the answer then
6     break; // stop the loop if the answer has been generated
7   retrieval =  $RD(q, c_t, r_{\leq t})$ ; // Determine whether current step requires retrieval
8   if retrieval is "Yes" then
9      $g_t, q_t = QF(q, r_{\leq t})$ ; // Obtain KG-based reasoning chain and retrieval query
10     $\mathcal{D}_t = Retriever(q_t)$ ;
11     $\mathcal{K}_t = \mathcal{G}(\mathcal{D}_t)$ ; // Obtain the knowledge triples for retrieved documents from the KG corpus
12     $c_{t+1} = KI(q, g_t, \mathcal{K}_t)$ ; // Update context using the selected triple
13  else
14     $c_{t+1} = c_t$ ; // the context remains unchanged if retrieval is not required.
15   $t = t + 1$ ;
16  $a = extract\_answer(r_{\leq t})$ ; // extract final answer from the reasoning thoughts with predefined rules
```

Output : Predicted Answer a .

Prompt Used for LLM**Instruction:**

Please answer the following question. You should think step-by-step to solve it. You should first output all your internal thinking steps. Each thought should be put inside `<think>` and `</think>` tags. After reasoning, you must put your final answer inside `<answer>` and `</answer>` without detailed instructions.

Your output should look like:

`<think>` [put your first thought here] `</think>`

`<think>` [put your second thought here] `</think>`

...

`<think>` [put your final thought here] `</think>`

`<answer>` [put your final answer (short phrase or keyword only) here] `</answer>`

Inputs:

Context: {retrieved documents}

Question: {original question}

Figure 6: Prompt for LLM.

eratively retrieve and integrate external information. This iterative process is terminated if the final answer is produced or the maximum number of reasoning steps is reached. The computational complexity of SPARKLE during inference primarily depends on the number of reasoning steps T and the cost of individual components at each step. Assuming a maximum of T reasoning steps, the total cost is $\mathcal{O}(T \cdot (C_{LLM} + C_{Retriever} + C_{Proxy}))$, where C_{LLM} denotes the cost of invoking the LLM, $C_{Retriever}$ is the cost of retrieval, and C_{Proxy} captures the combined cost of the three proxy agents. Since the proxy model is lightweight and retrieval is performed selectively, SPARKLE maintains relatively low overall latency compared to other multi-step retrieval approaches (see Figure 3c for the full effectiveness-efficiency trade-off analysis).

Prompt Used for Reasoning LLM**Instruction:**

Please answer the following question. You should think step-by-step to solve it. Provide your final answer in the format `\boxed{YOUR_ANSWER}`.

Inputs:

Context: {retrieved documents}

Question: {original question}

Figure 7: Prompt for Reasoning LLM.

C Training Strategy

In SPARKLE, only the proxy model is trainable, while both the retriever and the LLM are kept frozen throughout training. The goal is to jointly optimise the proxy model across its three functional roles: Retrieval Decision, Query Formulation, and Knowledge Integration, so that it can effectively coordinate these behaviours to support adaptive retrieval and knowledge integration.

Our overall training pipeline follows the high-level framework proposed by Chen et al. (2025a), which we briefly outline here for completeness. Please refer to the original paper for more details. In our multi-agent MDP framework, each agent receives its own state (input) and produces an action (output). A complete reasoning trajectory involves all agents cooperating to generate an answer, after which a system-level reward is computed. We adopt the Monte Carlo reward assignment method described in §3.2 to decompose the final reward and assign it backward to each agent’s decision step. This gives us a set of $\langle state, action, reward \rangle$ triplets for each agent, which we aggregate to train the proxy model using standard RL techniques.

Prompt Used for Knowledge Integration Agent

Instruction:

You are an intelligent reasoning assistant. Your task is to select the most useful knowledge triple from a list of candidates to help continue a reasoning process toward answering a given question.

You will be given:

- A question
- A current reasoning chain represented as a sequence of knowledge triples
- A query, which may reflect an information need to guide the next reasoning step
- A set of candidate triples

Your goal: Select only one triple that is most useful for continuing the reasoning toward answering the question. This selected triple should logically extend the current reasoning chain and help bridge the gap toward the final answer.

Inputs:

Candidate Triples: {triples extracted from documents}

Question: {original question}

Reasoning Chain: {KG-based reasoning chain generated by the Query Formulation agent}

Query: {query generated by the Query Formulation agent}

Figure 8: Prompt for Knowledge Integration Agent.

1024 Unlike traditional on-policy rollouts, we employ
1025 the *binary tree-structured rollout* strategy intro-
1026 duced in §3.1 to explore multiple reasoning trajec-
1027 tories. Each branch in the tree corresponds to a
1028 complete reasoning trajectory. For each trajectory,
1029 we apply the reward decomposition method and
1030 collect training data accordingly. *It is important*
1031 *to note that the binary tree-structured rollout is*
1032 *only used during training to enhance exploration.*
1033 During inference, the proxy model operates deter-
1034 ministically without branching, relying solely on its
1035 learned policy. In § E.4, we provide a case analysis
1036 illustrating the mechanisms of this rollout strategy.

D Experimental Details

D.1 Datasets

1039 In our experiments, we employ three different types
1040 of QA datasets: (1) *multi-hop QA* datasets such as
1041 HotPotQA, 2Wiki, MuSiQue and Bamboogle; (2)
1042 *single-hop QA* datasets such as NQ and TQA; (3)
1043 *long-form QA* dataset ASQA. In terms of the re-
1044 trieval corpus, for HotPotQA, we use the corpus
1045 provided by its authors for retrieval. For 2Wiki-
1046 MultihopQA and MuSiQue, we construct the re-
1047 trieval corpus following the exact same procedure
1048 outlined by Trivedi et al. (2023). For all other
1049 datasets, we leverage the Wikipedia corpus intro-
1050 duced by Karpukhin et al. (2020).

1051 For datasets with public test sets (Bamboogle,
1052 NQ and TQA, ASQA), we report results on their

Hyperparameter	Value
Base Model	Qwen/Qwen2.5-3B-Instruct
Learning Rate	5e-6
Batch Size	64
Max Length	2048
# Epochs	1

Table 5: Training hyperparameters for SFT stage.

Hyperparameter	Value
Learning Rate of Policy Model	5e-7
Learning Rate of Value Model	5e-6
KL Coefficient	0.002
Batch Size	1024
Max Length	2048
LR scheduler type	Cosine

Table 6: Training hyperparameters for RL stage.

official test sets. For those without public test sets
1053 (HotPotQA, 2Wiki and MuSiQue), we instead use
1054 their development sets as test sets and report the
1055 corresponding performance. Following Chen et al.
1056 (2025a), we randomly sample up to 1,000 exam-
1057 ples from their evaluation set for comparison. 1058

D.2 Baselines

1059 In our experiments, we compare SPARKLE against
1060 three categories of baseline models: (1) *LLM*; (2)
1061 *single-step RAG* models; (3) *iterative RAG* models;
1062 and (4) *adaptive RAG* models. 1063

1064 The *LLM* model takes the original question as
1065 input and directly outputs an answer to the ques-
1066 tion. The *Standard RAG* model employs a single-
1067 round retrieval strategy, where the top- K docu-
1068 ments are retrieved and passed to the LLM for an-
1069 swer generation. For *IRCoT*, *ITER-RETGEN* and
1070 *Self-RAG*, we use the implementations provided
1071 by FlashRAG (Jin et al., 2024). For other models,
1072 including *TRACE*, *KiRAG*, *DRAGIN*, *Auto-RAG*,
1073 *Adaptive-RAG*, *SlimPLM* and *C-3PO*, we adapt the
1074 code released by their authors to align with our
1075 experimental setup. Notably, for fair comparison,
1076 both our SPARKLE and baselines use the same re-
1077 triever for retrieving documents from the corpus
1078 and the same LLM for generating answers. 1079

1080 In **RQ4**, to investigate the effectiveness of each
1081 agent within our framework, we introduce several
1082 ablated variants of SPARKLE: *w/o RD Agent*, *w/o*
1083 *QF Agent*, *w/o KI Agent*. Here we describe how
1084 each variant is implemented and how the removed
agent’s functionality is handled. Specifically, in the

1085 *w/o RD Agent* variant, the RD agent is removed
 1086 and the LLM itself decides whether to retrieve at
 1087 each reasoning step by generating a special token
 1088 (e.g., “<search>”) to indicate the need for retrieval.
 1089 This design relies solely on the LLM’s internal rea-
 1090 soning without external supervision for retrieval
 1091 decision. In the *w/o QF Agent* variant, the QF
 1092 agent is removed and the RD agent is responsi-
 1093 ble for both deciding whether retrieval is needed
 1094 and generating the corresponding query. Note that
 1095 this variant also generates the KG-based reasoning
 1096 chain, which is handled by the RD agent. In the
 1097 *w/o KI agent*, the KI agent is removed and all the
 1098 retrieved documents are concatenated and directly
 1099 appended to the LLM prompt for further reasoning.

1100 D.3 Implementation and Training Details

1101 Throughout the experiments, we use `intfloat/e5`
 1102 `-large-v2` as the retriever model, `Qwen/Qwen2.5`
 1103 `-7B-Instruct` as the LLM and `Qwen/Qwen2.5`
 1104 `-3B-Instruct` as the proxy model, unless speci-
 1105 fied otherwise. The prompts for both the LLM and
 1106 the proxy model can be found in Appendix A. At
 1107 each iteration, when retrieval is triggered, we re-
 1108 trieve the top-10 documents. These documents are
 1109 then decomposed into a set of knowledge triples
 1110 using `meta-llama/Meta-Llama-3-8B-Instruct`
 1111 model. However, the total number of extracted
 1112 triples can be large and may contain substantial
 1113 noise. Therefore, following Fang et al. (2025), we
 1114 rank all triples using the E5 retriever and retain
 1115 only the top-20 highest-scoring triples as input to
 1116 the Knowledge Integration agent. All inference
 1117 experiments (including SPARKLE and baselines)
 1118 are conducted on a machine with a 32-core AMD
 1119 Ryzen Threadripper Processor (3.5 GHz) and an
 1120 NVIDIA A6000 GPU.

1121 Note that our framework only introduces a sin-
 1122 gle proxy model, which is shared across all agents,
 1123 with different prompts used to elicit specialised be-
 1124 haviours. To train the proxy model, we randomly
 1125 sample 3,000 examples from each of the training
 1126 sets of HotPotQA, 2Wiki and NQ. These examples
 1127 are further split into 7,500 for training and 1,500
 1128 for validation. Following the standard PPO training
 1129 recipe, we begin with a supervised finetuning (SFT)
 1130 warm-up phrase. We use `Qwen2.5-72B-Instruct` as
 1131 the proxy model to generate trajectories, retaining
 1132 only those that lead to the *correct* final answer for
 1133 SFT. The collected trajectories are then used to
 1134 train the proxy model for 1 epoch before RL op-
 1135 timisation, with detailed hyperparameter settings

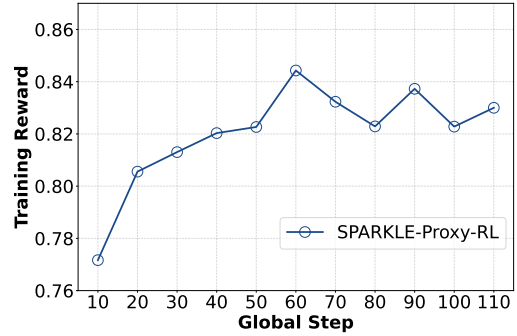


Figure 9: Training rewards across PPO training steps.

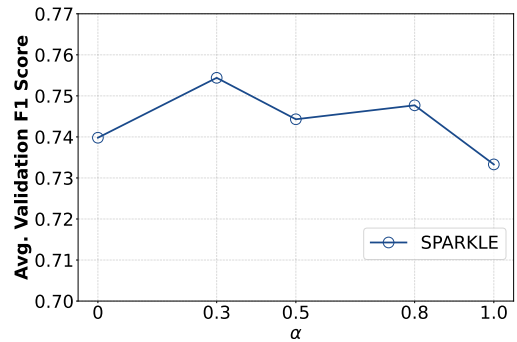


Figure 10: Sensitivity analysis of α , where the average F1 scores on the validation set under different α values are reported.

1136 provided in Table 5. In the RL phrase, we compute
 1137 the system-level reward with a trade-off hyperpa-
 1138 rameter $\alpha = 0.3$. The proxy model is trained with
 1139 hyperparameters outlined in Table 6. Moreover, in
 1140 the binary tree-structured rollout, we set the max-
 1141 imum tree expansion depth to $L = 4$ to balance
 1142 exploration and computational cost. All hyperpa-
 1143 rameters for both SFT and RL stages are selected
 1144 based on performance on the validation set. In
 1145 terms of training resources, training is conducted
 1146 on the same hardware setup as inference, but uses
 1147 two A6000 GPUs to enable parallelisation.

1148 E Additional Experimental Results and 1149 Analysis

1150 E.1 RL Training Analysis

1151 To assess the effectiveness of our RL training strat-
 1152 egy, we monitor the average system-level rewards
 1153 obtained by sampled trajectories during PPO train-
 1154 ing. Figure 9 illustrates the evolution of training
 1155 rewards over PPO steps. As the training progresses,
 1156 the reward increases steadily in the early stage and
 1157 then exhibits mild fluctuations, indicating that the
 1158 proxy model successfully learns effective retrieval

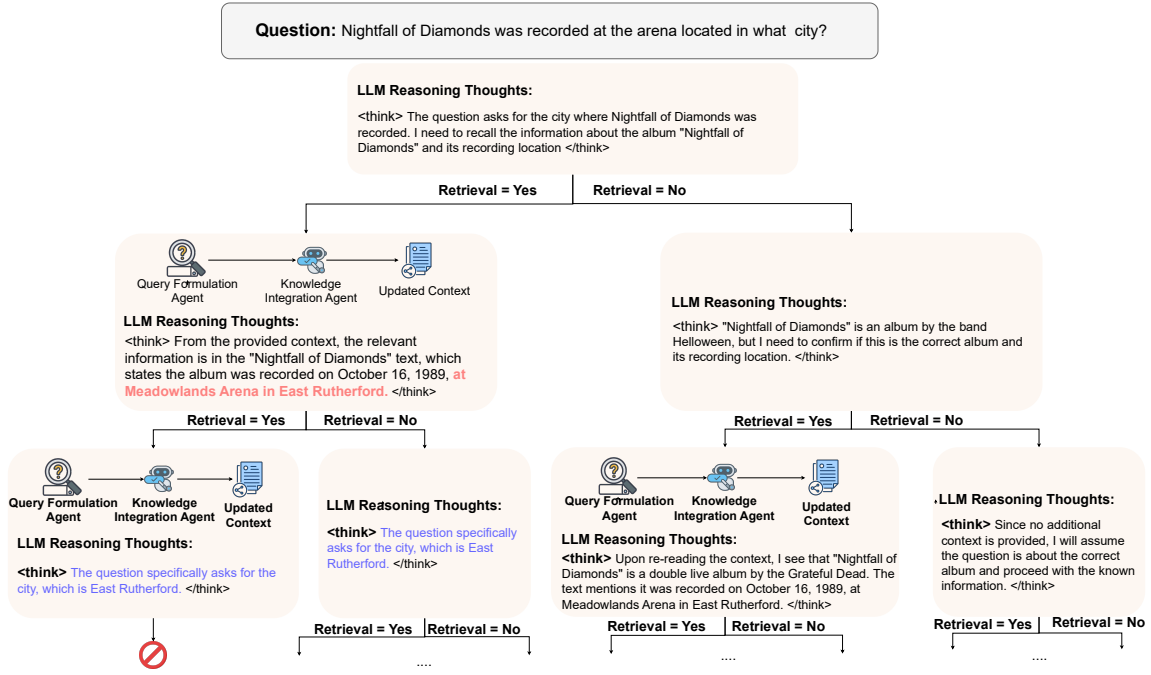


Figure 11: Illustration of the proposed binary tree-structured rollout strategy during PPO training. At each reasoning step, instead of relying on the Retrieval Decision agent to make retrieval decisions, both possible directions, i.e., retrieval and non-retrieval, are explicitly explored. For the retrieval branch, the Query Formulation agent and the Knowledge Integration agent are invoked to update context. If the selected document is already present in the context, resulting in no context change, the LLM would generate identical reasoning in both branches (highlighted in purple). In such cases, the retrieval branch is pruned to avoid redundant computation.

1159 policies. Notably, the rewards converge after about
 1160 60 PPO steps, highlighting the rapid convergence
 1161 of our training strategy and its ability to achieve
 1162 strong performance with relatively few updates.

1163 E.2 Sensitivity Analysis

1164 The system-level reward used in our PPO training is
 1165 a combination of retrieval quality and final answer
 1166 accuracy, balanced by a trade-off hyperparameter
 1167 α . To investigate the effects of α , we vary its value
 1168 from 0 to 1.0, and train the proxy model accord-
 1169 ingly. We then evaluate the model’s performance,
 1170 measured by the average F1 score on the validation
 1171 set, under different α values, as shown in Figure 10.
 1172 The results show that the curve remains relatively
 1173 flat across different α values, suggesting that our
 1174 training method is not highly sensitive to this hy-
 1175 perparameter. However, we observe performance
 1176 drops when α is set to 0 and 1.0, indicating that
 1177 relying solely on either retrieval recall or answer
 1178 accuracy leads to suboptimal training. The best
 1179 performance is achieved when $\alpha=0.3$, highlighting
 1180 that a balanced reward combining both retrieval
 1181 effectiveness and answer quality provides stronger
 1182 learning signals and leads to better performance.

Model	HotPotQA	2Wiki	MuSiQue	Avg.
HippoRAG	74.29*	74.01*	43.78*	64.03
RAPTOR	72.48*	61.20*	47.47*	60.38
SiReRAG	77.36*	68.20*	52.08*	65.88
SPARKLE	<u>75.25</u>	78.38	52.28	68.64

Table 7: Performance (F1%) of SPARKLE and graph RAG baselines, where * indicates results reported from the SiReRAG paper. The results of SPARKLE are produced under SiReRAG’s settings for fair comparison.

E.3 Comparison with Graph RAG Models

1183 We conduct experiments to compare SPARKLE
 1184 with recent graph RAG methods, including Hip-
 1185 poRAG (Gutiérrez et al., 2024), PAPTOR (Sarthi
 1186 et al., 2024) and SiReRAG (Zhang et al., 2025).
 1187 To ensure a fair comparison, we adopt the exact
 1188 same experimental setups as used in SiReRAG,
 1189 including both datasets and model configurations.
 1190 In terms of datasets, while SiReRAG also evalu-
 1191 ates on HotPotQA, 2Wiki, and MuSiQue, it fol-
 1192 lows the setting of HippoRAG, where 1,000 ques-
 1193 tions are sampled from each dataset, along with
 1194 a subset of the original corpus (see Table 1 in
 1195 HippoRAG for details). In our experiments, we
 1196

1197 use the same sampled questions and correspond- 1247
1198 ing corpus subsets for consistency. In terms of 1248
1199 model configurations, following SiReRAG, we use 1249
1200 OpenAI’s `text-embedding-3-small` as the em- 1250
1201 bedding model for retrieval, GPT-4o for answer 1251
1202 generation, and GPT-3.5-Turbo to extract knowl- 1252
1203 edge triples from documents for KG construction 1253
1204 (see Section 5.4 in the SiReRAG paper). 1254

1205 Table 7 presents the performance comparison be- 1255
1206 tween SPARKLE and graph RAG baselines across 1256
1207 three datasets. The results indicate that SPARKLE 1257
1208 achieves the highest average F1 score of 68.64%, 1258
1209 outperforming the strongest baseline SiReRAG by 1259
1210 2.76%. This highlights the effectiveness of our 1260
1211 adaptive retrieval framework and structured rea- 1261
1212 soning mechanism, which together enable more 1262
1213 accurate and efficient knowledge integration than 1263
1214 existing graph RAG methods. Notably, SPARKLE 1264
1215 achieves a substantial improvement of 4.37% over 1265
1216 SiReRAG on 2Wiki. This improvement is likely 1266
1217 due to SPARKLE’s dynamic construction of KG- 1267
1218 based reasoning chains based on the LLM’s inter- 1268
1219 mediate thoughts, which guide both retrieval and 1269
1220 knowledge integration in a goal-directed manner. 1270
1221 In contrast, graph RAG baselines rely on static 1271
1222 graph indices built over the corpus, which can suf- 1272
1223 fer from error propagation or noise introduced dur- 1273
1224 ing graph construction. 1274

1225 E.4 Case Analysis 1275

1226 **Binary Tree-Structured Rollout Strategy.** Fig- 1276
1227 ure 11 illustrates a concrete example of our binary 1277
1228 tree-structured rollout strategy during PPO train- 1278
1229 ing. At each reasoning step, instead of relying on 1279
1230 the Retrieval Decision agent to make a single re- 1280
1231 trieval decision, we explicitly explore both possible 1281
1232 branches, i.e., retrieval and non-retrieval, to con- 1282
1233 struct a binary reasoning tree. This enables the 1283
1234 proxy model to explore a broader space of trajec- 1284
1235 tories and acquire more informative reward signals 1285
1236 for learning effective retrieval policy. 1286

1237 While the Retrieval Decision agent is not acti-
1238 vated during rollout, the retrieval decision on each
1239 branch is recorded and used for policy optimisa-
1240 tion. In contrast, during inference, we do not per-
1241 form branching and directly leverage the learned
1242 Retrieval Decision agent to make a single determin-
1243 istic decision at each step.

1244 **End-to-End Inference.** To better illustrate the end-
1245 to-end inference process of SPARKLE, we present
1246 four case studies (Tables 8-11) that show how the

three agents within SPARKLE, i.e., Retrieval De-
cision, Query Formulation and Knowledge Inte-
gration, interact with the LLM and the retriever to
handle *different types* of questions.

Tables 8-10 present successful inference trajec-
tories for *multi-hop*, *single-hop*, and *commonsense*
questions, respectively. In Table 8, the two-hop
question triggers two retrievals, with each corre-
sponding to an intermediate information need. In
contrast, the single-hop question in Table 9 requires
only a single retrieval, while the commonsense
question in Table 10 is answered entirely using
the LLM’s internal knowledge without invoking
retrieval. These examples demonstrate that the Re-
trieval Decision agent effectively identifies when
external information is needed, triggering retrieval
only when the LLM’s current context is insufficient.
Conversely, when the LLM is engaged in interpret-
ing the question or reasoning over existing context,
the agent appropriately suppresses retrieval to avoid
unnecessary overhead. Moreover, the Query For-
mulation and Knowledge Integration agents can
also generate targeted queries and accurately incor-
porate relevant content into the context, enabling
the LLM to progressively refine its reasoning and
arrive at the correct answer.

Table 11 presents a failure case where SPARKLE
is unable to correctly answer the question due to
insufficient retrieved information. In this case, the
Retrieval Decision agent successfully identifies the
need for external information, and the Query For-
mulation agent generates a relevant and precise
query. However, due to the absence of necessary
knowledge in the retrieval corpus, the system fails
to obtain the correct answer. This highlight a key
limitation of relying on a single static corpus and
suggests that incorporating multiple knowledge
sources, such as web-based retrieval, could be a
promising direction for enhancing the systems’s
coverage and robustness.

Question: Who is the spouse of the director of film The Promise (1995 Film)?

Ground Truth: Volker Schlöndorff

 **LLM:** <think> To answer this question, I need to identify the director of the 1995 film "The Promise". </think>

 **Retrieval Decision Agent: Yes (retrieval is required)**

 **Query Formulation Agent:**

Step 1: Reasoning Chain: <"The Promise (1995 Film)"; directed by; [unknown director]>

Step 2: Knowledge Gaps: To continue answering the question, we need to know: Who directed the film "The Promise" (1995)? Identifying the director will allow us to find out who their spouse is.

Step 3: Retrieval Query: **Who directed the film "The Promise" (1995)?**

 **Retriever:**

Doc 1 (The Promise (1995 film)): The Promise is a 1995 German-language film directed by Margarethe von Trotta. It was an international co-production ... but did not receive a nomination.

...

 **Knowledge Integration Agent:**

<The Promise (1995 film); director; Margarethe von Trotta> → update context with Doc 1

 **LLM:** <think> From the context, it states "The Promise is a 1995 German-language film directed by Margarethe von Trotta." </think>

 **Retrieval Decision Agent: No (retrieval is not required)**

 **LLM:** <think> Now, I need to find out who Margarethe von Trotta's spouse is. </think>

 **Retrieval Decision Agent: Yes (retrieval is required)**

 **Query Formulation Agent:**

Step 1: Reasoning Chain: <"The Promise (1995 Film)"; directed by; Margarethe von Trotta>, <Margarethe von Trotta; has spouse; [unknown]>

Step 2: Knowledge Gaps: To continue answering the question, we need to know: Who is Margarethe von Trotta's spouse? Identifying her spouse will provide the answer to the question.

Step 3: Retrieval Query: **Who is Margarethe von Trotta's spouse?**

 **Retriever:**

Doc 1 (Title: Margarethe von Trotta): Margarethe von Trotta (born 21 February 1942) is a German film director who has been referred to as a "leading force" of the New German Cinema movement... She was married to and collaborated with director Volker Schlöndorff... Nevertheless, she rejects the suggestion that she makes "women's films".

...

 **Knowledge Integration Agent:**

<Margarethe von Trotta; spouse; Volker Schlöndorff> → update context with Doc 1

 **LLM:** <think> The context mentions "She was married to and collaborated with director Volker Schlöndorff." </think>

 **Retrieval Decision Agent: No (retrieval is not required)**


 **LLM:** <think> Therefore, the spouse of the director of the 1995 film "The Promise" is Volker Schlöndorff. </think>


 **Retrieval Decision Agent: No (retrieval is not required)**


 **LLM:** <answer> Volker Schlöndorff </answer>

Table 8: A case study (successfully) demonstrating how SPARKLE performs multi-step reasoning and adaptive retrieval to answer a multi-hop question, where the proxy model accurately determines when retrieval is required, formulates targeted queries, and integrates retrieved knowledge to guide the LLM toward the correct answer.


Question: who did corbin dance with on dancing with the stars
Ground Truth: Karina Smirnoff

 **LLM:** <think> The question is asking about a specific dance partner on the show "Dancing with the Stars." I need to recall which celebrity Corbin Bleu was paired with during his time on the show. </think>

 **Retrieval Decision Agent:** **Yes (retrieval is required)**

 **Query Formulation Agent:**
Step 1: Reasoning Chain: <Corbin Bleu; participated in; Dancing with the Stars>, <Corbin Bleu; danced with; [unknown]>
Step 2: Knowledge Gaps: To continue answering the question, we need to know: Who was Corbin Bleu paired with on "Dancing with the Stars"? This specific information will directly answer the question.
Step 3: Retrieval Query: **Who was Corbin Bleu paired with on "Dancing with the Stars"?**


 **Retriever:**
Doc 1 (Title: Corbin Bleu): He competed in the 17th season of "Dancing with the Stars", partnered with professional dancer Karina Smirnoff. He has also pursued a music career, and released his debut album "Another Side" on May 1, 2007, which included the single "Push It to the Limit ...

Doc 2 (Title: Corbin Bleu): Bleu also revealed it will feature a little "tap section" at the end, and because Smirnoff had never tapped before, he was thrilled to be able to teach her something for once. On October 18, 2013, appeared with his professional dancer Karina Smirnoff on "The Ellen DeGeneres Show" ...
...
 **Knowledge Integration Agent:**
<Corbin Bleu; partnered with; professional dancer Karina Smirnoff> → update context with Doc 1

 **LLM:** <think> Looking through the provided texts, I see that Corbin Bleu is mentioned in the context of "Dancing with the Stars" season 17. </think>

 **Retrieval Decision Agent:** **No (retrieval is not required)**

 **LLM:** <think> The text clearly states that Corbin Bleu was partnered with Karina Smirnoff. </think>

 **Retrieval Decision Agent:** **No (retrieval is not required)**





 **LLM:** <answer> Karina Smirnoff </answer>


Table 9: A case study (successfully) demonstrating how SPARKLE performs adaptive retrieval to answer single-hop question, where the retrieval is only triggered once to retrieve necessary information.


Question: what is the common name for gravitational force?
Ground Truth: Gravity


 **LLM:** <think> The question is asking for the common name of gravitational force. I need to recall the basic physics terminology related to gravity. </think>

 **Retrieval Decision Agent:** **No (retrieval is not required)**

 **LLM:** <think> Gravitational force is the fundamental force of attraction between any two masses in the universe. It's a well-known concept in physics. </think>

 **Retrieval Decision Agent:** **No (retrieval is not required)**

 **LLM:** <think> The common name for gravitational force in everyday language is simply "gravity." </think>

 **Retrieval Decision Agent:** **No (retrieval is not required)**



 **LLM:** <answer> gravity </answer>

Table 10: A case study (successfully) demonstrating how SPARKLE handles commonsense question, where no retrieval is involved and the LLM relies on its internal knowledge to answer the question.

Question: Where was the place of death of Beata Łaska Z Kościeleckich's father?

Ground Truth: Kraków

 **LLM:** <think> The question asks about the place of death of Beata Łaska Z Kościeleckich's father. I need to recall information about Beata Łaska Z Kościeleckich and her father's details. </think>

 **Retrieval Decision Agent:** Yes (retrieval is required)

 **Query Formulation Agent:**

Step 1: Reasoning Chain: <Beata Łaska Z Kościeleckich; has father; [unknown]>

Step 2: Knowledge Gaps: To continue answering the question, we need to know: What is the name of Beata Łaska Z Kościeleckich's father? Identifying him will allow us to find out where he died.

Step 3: Retrieval Query: **What is the name of Beata Łaska Z Kościeleckich's father?**


 **Retriever:**

Doc 1 (Beata Łaska z Kościeleckich): Beata Łaska z Kościeleckich (1515-1576), was a Polish magnate heiress. She was reputed to be the illegitimate daughter of king Sigismund I the Old. She was the daughter of Andrzej Kościelecki and Katarzyna Telniczanka and married to Illia Ostrogski (d. 1539) in 1539, and Olbracht Łaski in 1564. ...

...

 **Knowledge Integration Agent:**

<Beata Łaska z Kościeleckich; father; Andrzej Kościelecki> → update context with Doc 1

 **LLM:** <think> From the text, I know Beata Łaska Z Kościeleckich was a Polish magnate heiress. She was the daughter of Andrzej Kościelecki and Katarzyna Telniczanka. However, there is no information provided about the place of death of her father. </think>

 **Retrieval Decision Agent:** Yes (retrieval is required)

 **Query Formulation Agent:**

Step 1: Reasoning Chain: <Beata Łaska Z Kościeleckich; is daughter of; Andrzej Kościelecki>, <Andrzej Kościelecki; has place of death; [unknown]>

Step 2: Knowledge Gaps: To continue answering the question, we need to know: What was the place of death of Andrzej Kościelecki? This information will provide the answer to the question regarding Beata Łaska Z Kościeleckich's father's place of death.

Step 3: Retrieval Query: **What was the place of death of Andrzej Kościelecki?**

 **Retriever:**

Doc 1 (Title: Mieczysław Kościelniak): Mieczysław Kościelniak(29 January 1912 in Kalisz – 5 March 1993 in Słupsk) was a Polish painter, graphic designer, and draftsman, brother of Władysław and Tadeusz Kościelniak. Already a prominent artist, he was arrested in 1941 and sent to the Auschwitz concentration camp..


Doc 2 (Title: Andrzej Kościelecki): Andrzej Jędrzej Kościelecki (b. 1455–1515) was a Marshal of the household of Cardinal Fryderyk Jagiellończyk, starosta of Bydgoszcz from 1485, starosta of Świecie from 1487, Marshal of the Court from 1501, starosta of Spiš from 1507, Castellan of Wiślica and Biecz from 1508, Żupnik of Wieliczka–Bochnia from 1508, Podskarbi from 1509, starosta of Oświęcim from 1509, starosta of Inowrocław from 1510, starosta of Nowy Sącz from 1512, starosta of Zator from 1513, Castellan of Wojnicz from 1513, Gran Steward and the burgrave of Kraków. He was born in Kościelec. He is one of the characters on the famous painting by Jan Matejko "Prussian Homage".

Doc 3 (Title: Andrzej Kotkowski): Andrzej Kotkowski(17 February 1940 – 15 January 2016) was a Polish film director and screenwriter. He directed seventeen films between 1972 and 2009.

...

 **Knowledge Integration Agent:**

<Andrzej Kościelecki; birth place; Kościelec> → update context with Doc 2

 **LLM:** <think> Since the provided texts do not contain information about the place of death of Beata Łaska Z Kościeleckich's father, I cannot determine the answer from the given context. </think>

 **Retrieval Decision Agent:** No (retrieval is not required)


 **LLM:** <answer> Unknown </answer>

Table 11: A case study (failed) demonstrating how SPARKLE fails to answer a question with insufficient retrieved information.