

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

Jinyuan Fang
University of Glasgow
j.fang.2@research.gla.ac.uk

Zaiqiao Meng
University of Glasgow
zaiqiao.meng@gmail.com

Craig Macdonald
University of Glasgow
craig.macdonald@glasgow.ac.uk

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%, respectively.¹

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://github.com/jyfang6/sparkle>

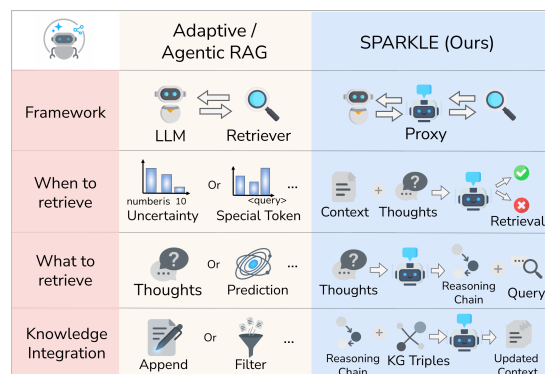


Figure 1: Comparison between existing adaptive/agentic RAG models and the proposed SPARKLE.

when to retrieve, i.e., identifying the appropriate moment to retrieve; (2) *what to retrieve*, i.e., formulating effective queries for missing information; (3) *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 ability 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 that enables adaptive

retrieval without modifying the LLM itself. Ideally, such an approach should decouple retrieval control 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 **S**tructured and **P**lug-and-play agentic retrieval policy for improved Adaptive Retrieval and **K**nowledge 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, inspired by 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 knowledge 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

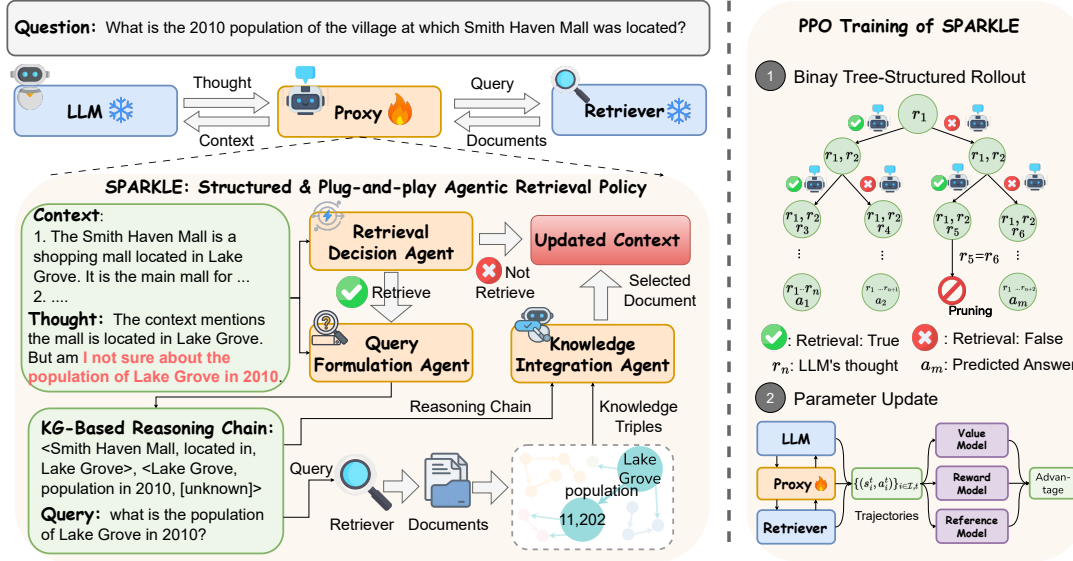


Figure 2: (left) Overview of SPARKLE. At each reasoning step, the *Retrieval Decision Agent* determines whether retrieval is needed. If so, the *Query Formulation Agent* and the *Knowledge Integration Agent* are invoked to generate a search query and incorporate relevant information into the context,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

collaboratively contributing to the overall process.

Formally, we define the multi-agent MDP process 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 a strategy often leads to suboptimal performance (see **RQ4** in §4.2) due to the presence of distracting or speculative content in the 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 distract-

ing, 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 triples.² 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 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

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.

its full reasoning and produces a final answer, it is natural to frame the training process as an RL problem. We adopt PPO (Schulman et al., 2017) to train the proxy model to learn a retrieval policy that improves 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 proxy model’s capabilities, limiting 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

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	$\uparrow 31.61$	11.94	40.49	46.50	29.26	32.05	$\uparrow 14.29$
Single-Step RAG	RAG*	49.69	41.60	33.30	41.53	$\uparrow 14.51$	21.04	37.72	58.20	30.36	36.83	$\uparrow 9.51$
	TRACE*	54.32	43.28	37.60	45.07	$\uparrow 10.97$	22.61	47.36	<u>64.20</u>	31.84	41.50	$\uparrow 4.84$
Iterative RAG	IRCoT*	59.05	47.13	36.60	47.59	$\uparrow 8.45$	27.63	43.08	61.90	21.47	38.52	$\uparrow 7.82$
	ITER-RETGEN*	61.03	<u>57.09</u>	38.60	<u>52.24</u>	$\uparrow 3.80$	30.14	48.92	64.80	24.64	42.13	$\uparrow 4.21$
	KiRAG*	<u>61.39</u>	55.85	36.00	51.80	$\uparrow 4.96$	31.07	47.36	63.20	31.84	43.37	$\uparrow 2.97$
Adaptive RAG	Self-RAG*	26.69	20.90	36.20	27.93	$\uparrow 28.11$	12.46	15.89	43.50	32.81	26.17	$\uparrow 20.17$
	DRAGIN*	48.23	48.46	26.10	40.93	$\uparrow 15.11$	27.64	42.47	57.80	29.25	39.29	$\uparrow 7.05$
	Adaptive-RAG*	56.62	46.06	33.30	45.33	$\uparrow 10.71$	26.43	38.87	57.80	29.42	38.13	$\uparrow 8.21$
	SlimPLM*	50.95	36.29	37.90	41.71	$\uparrow 14.33$	20.14	36.75	63.40	<u>36.30</u>	39.15	$\uparrow 7.19$
	Auto-RAG*	49.30	52.40	<u>38.90</u>	46.87	$\uparrow 9.17$	21.21	44.89	62.00	10.98	34.77	$\uparrow 11.57$
	C-3PO*	56.67	48.87	33.60	46.38	$\uparrow 9.66$	<u>31.87</u>	<u>48.96</u>	62.50	30.61	<u>43.49</u>	$\uparrow 2.85$
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.

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:

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

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 \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 parameters.⁵ 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 origi-

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.

nal 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 and Training Details. Throughout the experiments, we use 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, unless otherwise stated. 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 during inference. While there

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.

are three agents in the proposed SPARKLE framework, they all share a *single* proxy model, using different prompts to elicit specialised behaviours. To train the proxy model, we randomly sample a subset of examples from the training sets of HotPotQA, 2Wiki, and NQ. These examples are further split into a training set and a validation set for proxy training, resulting in 7.5K training examples and 1.5K validation examples. In terms of the reward function \mathcal{R} , we use a combination of recall and answer accuracy, with α set to 0.3. Moreover, we set the maximum tree expansion depth L to 4 in the binary tree-structured rollout during training. It is worth noting that the tree-structured rollout is only used during PPO training to encourage exploration of effective retrieval policies. During inference, the proxy model follows the learned policy and performs sequential decision-making without branching. 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 (i.e., HotPotQA, 2Wiki, and NQ), 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 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; (3) We further analyse performance across different QA types. Compared to single-hop datasets, SPARKLE yields larger gains on more challenging multi-hop and long-form QA datasets. Specifically, it outperforms the strongest baselines by an average of 2.91% on multi-hop QA datasets (i.e., HotPotQA, 2Wiki, MuSiQue, and Bamboogle) and 1.20% on the long-form QA dataset (i.e., ASQA), while maintaining competitive performance on single-hop QA datasets (i.e., NQ and TQA). This superior performance is attributed to SPARKLE’s ability to make context-aware retrieval decisions and selectively incorporate relevant information throughout multi-step reasoning, demonstrating its strength in handling complex questions that require integrating information from multiple sources.

(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. Specifically, 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), without any additional training or adaptation of the proxy model. The empirical results in Table 2 indicate that when only changing the retriever model,

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 to SPARKLE and Δ denotes the average drop of each variant wrt. SPARKLE.

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 that requires no retraining.

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

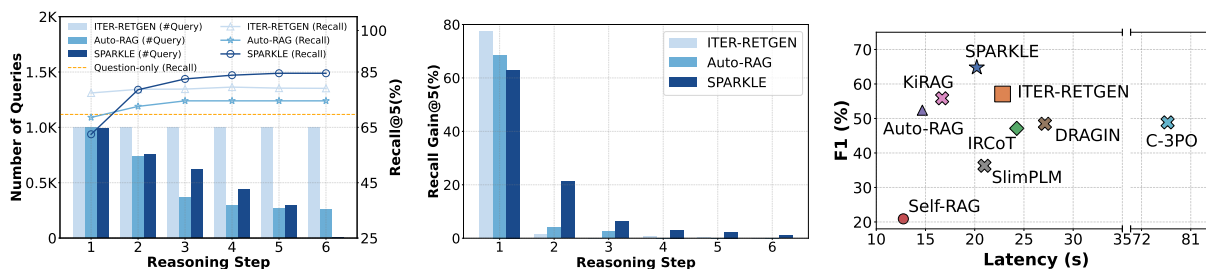
The experimental results are presented in Ta-

ble 3, which 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 ASQA,⁷ 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 removed.⁸ In each case, the corresponding 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.

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

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



(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.

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. Figure 3a presents the number of generated queries and cumulative Recall@5 per step in SPARKLE. The results indicate that 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.

(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 while maintaining relatively low inference latency. Despite some baselines exhibiting 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 and reduce unnecessary retrieval, 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. The proxy model is trained with PPO to learn effective retrieval policies. Moreover, to encourage exploration during PPO, we propose a binary tree-structured rollout strategy, which enables the proxy model to explore multiple branches during training. 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.

Limitations

We identify the following limitations of our work: (1) SPARKLE relies on precomputed knowledge triples to construct the KG-based reasoning chain. In our implementation, we adopt the triple extraction method from Fang et al. (2025), which has been empirically shown to produce high-quality results. In future work, we plan to explore alternative KG construction strategies and evaluate how the quality of extracted triples affects downstream performance. (2) Our training strategy employs PPO combined with a binary tree-structured rollout strategy to optimise the proxy model. While this approach demonstrates strong empirical performance, we leave the exploration of alternative RL algorithms, such as Group Relative Policy Optimisation (GRPO) (Shao et al., 2024), to future work. (3) A key motivation of our design is to maintain a lightweight and efficient retrieval proxy model. Therefore, we primarily experiment with a lightweight 3B proxy model. While this choice achieves competitive performance with low training cost, it remains an open question whether scaling up the proxy model could yield improved performance. We leave the exploration of larger proxy architectures to future work.

References

- Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. 2024. Self-RAG: Learning to retrieve, generate, and critique through self-reflection. In *The Twelfth International Conference on Learning Representations*.
- Guoxin Chen, Minpeng Liao, Peiying Yu, Dingmin Wang, Zile Qiao, Chao Yang, Xin Zhao, and Kai Fan. 2025a. C-3PO: Compact plug-and-play proxy optimization to achieve human-like retrieval-augmented generation. In *International Conference on Machine Learning*.
- Mingyang Chen, Linzhuang Sun, Tianpeng Li, Haoze Sun, Yijie Zhou, Chenzheng Zhu, Haofen Wang, Jeff Z Pan, Wen Zhang, Huajun Chen, and 1 others. 2025b. Learning to reason with search for LLMs via reinforcement learning. *arXiv preprint arXiv:2503.19470*.
- Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, and 1 others. 2024. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.
- Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, and Jonathan Larson. 2024. From local to global: A graph RAG approach to query-focused summarization. *arXiv preprint arXiv:2404.16130*.
- Wenqi Fan, Yujuan Ding, Liangbo Ning, Shijie Wang, Hengyun Li, Dawei Yin, Tat-Seng Chua, and Qing Li. 2024. A survey on RAG meeting llms: Towards retrieval-augmented large language models. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 6491–6501. ACM.
- Jinyuan Fang, Zaiqiao Meng, and Craig Macdonald. 2024a. REANO: Optimising retrieval-augmented reader models through knowledge graph generation. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2094–2112.
- Jinyuan Fang, Zaiqiao Meng, and Craig Macdonald. 2024b. TRACE the evidence: Constructing knowledge-grounded reasoning chains for retrieval-augmented generation. In *Findings of the Association for Computational Linguistics: EMNLP*, pages 8472–8494.
- Jinyuan Fang, Zaiqiao Meng, and Craig Macdonald. 2025. KiRAG: Knowledge-driven iterative retriever for enhancing retrieval-augmented generation. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 18969–18985.
- Wenfeng Feng, Chuzhan Hao, Yuewei Zhang, Jingyi Song, and Hao Wang. 2025. AirRAG: Activating intrinsic reasoning for retrieval augmented generation using tree-based search. *arXiv preprint arXiv:2501.10053*.
- Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, and 1 others. 2025a. Deepseek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning. *arXiv preprint arXiv:2501.12948*.
- Hanghui Guo, Jia Zhu, Shimin Di, Weijie Shi, Zhangze Chen, and Jiajie Xu. 2025b. DioR: Adaptive cognitive detection and contextual retrieval optimization for dynamic retrieval-augmented generation. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2953–2975.
- Bernal Jiménez Gutiérrez, Yiheng Shu, Yu Gu, Michihiro Yasunaga, and Yu Su. 2024. HippoRAG: Neurobiologically inspired long-term memory for large language models. *arXiv preprint arXiv:2405.14831*.
- 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.
- 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? *arXiv preprint arXiv:2506.19733*.
- 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.
- 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 *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 7969–7992.
- 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. *arXiv preprint arXiv:2503.09516*.
- Jiajie Jin, Yutao Zhu, Xinyu Yang, Chenghao Zhang, and Zhicheng Dou. 2024. FlashRAG: A modular toolkit for efficient retrieval-augmented generation research. *arXiv preprint arXiv:2405.13576*.
- Mandar Joshi, Eunsol Choi, Daniel S. Weld, and Luke Zettlemoyer. 2017. TriviaQA: A large scale distantly

- supervised challenge dataset for reading comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*, pages 1601–1611.
- Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick S. H. 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*, pages 6769–6781.
- Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, and 1 others. 2019. Natural questions: a benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:453–466.
- Ruosen Li and Xinya Du. 2023. Leveraging structured information for explainable multi-hop question answering and reasoning. In *Findings of the Association for Computational Linguistics: EMNLP*, pages 6779–6789.
- Xiaoxi Li, Guanting Dong, Jiajie Jin, Yuyao Zhang, Yujia Zhou, Yutao Zhu, Peitian Zhang, and Zhicheng Dou. 2025a. Search-O1: Agentic search-enhanced large reasoning models. *arXiv preprint arXiv:2501.05366*.
- Yu Li, Zhuoshi Pan, Honglin Lin, Mengyuan Sun, Conghui He, and Lijun Wu. 2025b. Can one domain help others? a data-centric study on multi-domain reasoning via reinforcement learning. *arXiv preprint arXiv:2507.17512*.
- Xi Victoria Lin, Xilun Chen, Mingda Chen, Weijia Shi, Maria Lomeli, Rich James, Pedro Rodriguez, Jacob Kahn, Gergely Szilvassy, Mike Lewis, and 1 others. 2024. RA-DIT: Retrieval-augmented dual instruction tuning. In *International Conference on Learning Representations*.
- Hao Liu, Zhengren Wang, Xi Chen, Zhiyu Li, Feiyu Xiong, Qinhan Yu, and Wentao Zhang. 2025. HopRAG: Multi-hop reasoning for logic-aware retrieval-augmented generation. In *Findings of the Association for Computational Linguistics*, pages 1897–1913. Association for Computational Linguistics.
- Pranoy Panda, Ankush Agarwal, Chaitanya Devaguptapu, Manohar Kaul, and Prathosh A P. 2024. HOLMES: hyper-relational knowledge graphs for multi-hop question answering using LLMs. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 13263–13282.
- Ofir Press, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah A. Smith, and Mike Lewis. 2023. Measuring and narrowing the compositionality gap in language models. In *Findings of the Association for Computational Linguistics: EMNLP*, pages 5687–5711.
- Ori Ram, Yoav Levine, Itay Dalmedigos, Dor Muhlgay, Amnon Shashua, Kevin Leyton-Brown, and Yoav Shoham. 2023. In-context retrieval-augmented language models. *Transactions of the Association for Computational Linguistics*, 11:1316–1331.
- Parth Sarthi, Salman Abdullah, Aditi Tuli, Shubh Khanna, Anna Goldie, and Christopher D. Manning. 2024. RAPTOR: Recursive abstractive processing for tree-organized retrieval. In *The International Conference on Learning Representations*.
- John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.
- 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*, pages 9248–9274.
- Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, and 1 others. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*.
- Aditi Singh, Abul Ehtesham, Saket Kumar, and Tala Talei Khoei. 2025. Agentic retrieval-augmented generation: A survey on agentic RAG. *arXiv preprint arXiv:2501.09136*.
- 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. *arXiv preprint arXiv:2503.05592*.
- Ivan Stelmakh, Yi Luan, Bhuwan Dhingra, and Ming-Wei Chang. 2022. ASQA: factoid questions meet long-form answers. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 8273–8288.
- 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 *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12991–13013.
- 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 *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, pages 4420–4436.
- Harsh Trivedi, Niranjana Balasubramanian, Tushar Khot, and Ashish Sabharwal. 2022. MuSiQue: Multi-hop questions via single-hop question composition. *Trans. Assoc. Comput. 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.

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. *arXiv preprint arXiv:2212.03533*.

Zhepei Wei, Wei-Lin Chen, and Yu Meng. 2025. InstructRAG: Instructing retrieval-augmented generation via self-synthesized rationales. In *The Thirteenth International Conference on Learning Representations*.

Heming Xia, Chak Tou Leong, Wenjie Wang, Yongqi Li, and Wenjie Li. 2025. TokenSkip: Controllable chain-of-thought compression in llms. *arXiv preprint arXiv:2502.12067*.

Shitao Xiao, Zheng Liu, Peitian Zhang, Niklas Muenighoff, Defu Lian, and Jian-Yun Nie. 2024. C-Pack: Packed resources for general chinese embeddings. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 641–649.

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 *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 133–145.

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. *arXiv preprint arXiv:2412.15115*.

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 *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2369–2380.

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 *The Eleventh International Conference on Learning Representations*.

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. *arXiv preprint arXiv:2406.19215*.

Ori Yoran, Tomer Wolfson, Ori Ram, and Jonathan Berant. 2024. Making retrieval-augmented language models robust to irrelevant context. In *International Conference on Learning Representations*.

Tian Yu, Shaolei Zhang, and Yang Feng. 2024. Auto-RAG: Autonomous retrieval-augmented generation for large language models. *arXiv preprint arXiv:2411.19443*.

Nan Zhang, Prafulla Kumar Choubey, Alexander R. Fabbri, Gabriel Bernadett-Shapiro, Rui Zhang, Prasenjit Mitra, Caiming Xiong, and Chien-Sheng Wu. 2025. SiReRAG: Indexing similar and related information for multihop reasoning. In *The International Conference on Learning Representations*.

A Prompts

A.1 Prompt for Retrieval Decision Agent

The prompt used for the Retrieval Decision agent is illustrated in Figure 4. Its input includes the current retrieved context (*retrieved documents*), the original question, the reasoning history up to the current step, and the LLM’s most recent reasoning thought. Based on this information, the agent determines whether additional retrieval is necessary to 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.

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.

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

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}

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.

et al. (2025) to decompose the retrieved documents 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 iteratively retrieve and integrate external information. This iterative process is terminated if the

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 c0 = ∅; // initialise context
3 while t is less than maximum reasoning steps do
4   rt = LLM(q, ct);
5   if rt contains the answer then
6     break; // stop the loop if the answer has been generated
7   retrieval = RD(q, ct, r≤t); // Determine whether current step requires retrieval
8   if retrieval is "Yes" then
9     gt, qt = QF(q, r≤t); // Obtain KG-based reasoning chain and retrieval query
10    Dt = Retriever(qt);
11    Kt = G(Dt); // Obtain the knowledge triples for retrieved documents from the KG corpus
12    ct+1 = KI(q, gt, Kt); // Update context using the selected triple
13  else
14    ct+1 = ct; // the context remains unchanged if retrieval is not required.
15  t = t + 1;
16 a = extract_answer(r≤t); // extract final answer from the reasoning thoughts with predefined rules
```

Output : Predicted Answer a .

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.

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 rela-

tively low overall latency compared to other multi-step retrieval approaches (see Figure 3c for the full effectiveness-efficiency trade-off analysis).

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.

Unlike traditional on-policy rollouts, we employ the *binary tree-structured rollout* strategy introduced in §3.1 to explore multiple reasoning trajectories. Each branch in the tree corresponds to a complete reasoning trajectory. For each trajectory,

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.

we apply the reward decomposition method and collect training data accordingly. *It is important to note that the binary tree-structured rollout is only used during training to enhance exploration.* During inference, the proxy model operates deterministically without branching, relying solely on its learned policy. In § E.4, we provide a case analysis illustrating the mechanisms of this rollout strategy.

D Experimental Details

D.1 Datasets

In our experiments, we employ three different types of QA datasets: (1) *multi-hop QA* datasets such as HotPotQA, 2Wiki, MuSiQue and Bamboogle; (2) *single-hop QA* datasets such as NQ and TQA; (3) *long-form QA* dataset ASQA. In terms of the retrieval corpus, for HotPotQA, we use the corpus provided by its authors for retrieval. For 2Wiki-MultihopQA and MuSiQue, we construct the retrieval corpus following the exact same procedure outlined by Trivedi et al. (2023). For all other datasets, we leverage the Wikipedia corpus introduced by Karpukhin et al. (2020).

For datasets with public test sets (Bamboogle, NQ and TQA, ASQA), we report results on their official test sets. For those without public test sets (HotPotQA, 2Wiki and MuSiQue), we instead use their development sets as test sets and report the corresponding performance. Following Chen et al. (2025a), we randomly sample up to 1,000 examples from their evaluation set for comparison.

D.2 Baselines

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

The *LLM* model takes the original question as input and directly outputs an answer to the question. The *Standard RAG* model employs a single-round retrieval strategy, where the top- K documents are retrieved and passed to the LLM for answer generation. For *IRCoT*, *ITER-RETGEN* and *Self-RAG*, we use the implementations provided by FlashRAG (Jin et al., 2024). For other models, including *TRACE*, *KiRAG*, *DRAGIN*, *Auto-RAG*, *Adaptive-RAG*, *SlimPLM* and *C-3PO*, we adapt the code released by their authors to align with our experimental setup. Notably, for fair comparison, *both our SPARKLE and baselines* use the same retriever for retrieving documents from the corpus and the same LLM for generating answers.

In **RQ4**, to investigate the effectiveness of each agent within our framework, we introduce several ablated variants of SPARKLE: *w/o RD Agent*, *w/o QF Agent*, *w/o KI Agent*. Here we describe how each variant is implemented and how the removed agent’s functionality is handled. Specifically, in the *w/o RD Agent* variant, the RD agent is removed and the LLM itself decides whether to retrieve at each reasoning step by generating a special token (e.g., “<search>”) to indicate the need for retrieval. This design relies solely on the LLM’s internal reasoning without external supervision for retrieval decision. In the *w/o QF Agent* variant, the QF agent is removed and the RD agent is responsible for both deciding whether retrieval is needed and generating the corresponding query. Note that this variant also generates the KG-based reasoning chain, which is handled by the RD agent. In the *w/o KI agent*, the KI agent is removed and all the retrieved documents are concatenated and directly appended to the LLM prompt for further reasoning.

D.3 Implementation and Training Details

Throughout the experiments, we use `intfloat/e5-large-v2` as the retriever model, Qwen/Qwen2.5-7B-Instruct as the LLM and Qwen/Qwen2.5-3B-Instruct as the proxy model, unless specified otherwise. The prompts for both the LLM and the proxy model can be found in Appendix A. At each iteration, when retrieval is triggered, we retrieve the top-10 documents. These documents are

then decomposed into a set of knowledge triples using meta-llama/Meta-Llama-3-8B-Instruct model. However, the total number of extracted triples can be large and may contain substantial noise. Therefore, following Fang et al. (2025), we rank all triples using the E5 retriever and retain only the top-20 highest-scoring triples as input to the Knowledge Integration agent. All inference experiments (including SPARKLE and baselines) are conducted on a machine with a 32-core AMD Ryzen Threadripper Processor (3.5 GHz) and an NVIDIA A6000 GPU.

Note that our framework only introduces a single proxy model, which is shared across all agents, with different prompts used to elicit specialised behaviours. To train the proxy model, we randomly sample 3,000 examples from each of the training sets of HotPotQA, 2Wiki and NQ. These examples are further split into 7,500 for training and 1,500 for validation. Following the standard PPO training recipe, we begin with a supervised finetuning (SFT) warm-up phrase. We use Qwen2.5-72B-Instruct as the proxy model to generate trajectories, retaining only those that lead to the *correct* final answer for SFT. The collected trajectories are then used to train the proxy model for 1 epoch before RL optimisation, with detailed hyperparameter settings provided in Table 5. In the RL phrase, we compute the system-level reward with a trade-off hyperparameter $\alpha = 0.3$. The proxy model is trained with hyperparameters outlined in Table 6. Moreover, in the binary tree-structured rollout, we set the maximum tree expansion depth to $L = 4$ to balance exploration and computational cost. All hyperparameters for both SFT and RL stages are selected based on performance on the validation set. In terms of training resources, training is conducted on the same hardware setup as inference, but uses two A6000 GPUs to enable parallelisation.

E Additional Experimental Results and Analysis

E.1 RL Training Analysis

To assess the effectiveness of our RL training strategy, we monitor the average system-level rewards obtained by sampled trajectories during PPO training. Figure 9 illustrates the evolution of training rewards over PPO steps. As the training progresses, the reward increases steadily in the early stage and then exhibits mild fluctuations, indicating that the proxy model successfully learns effective retrieval

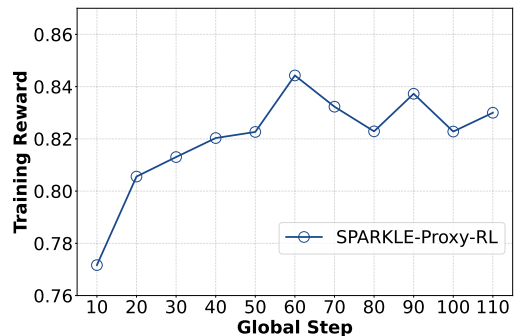


Figure 9: Training rewards across PPO training steps.

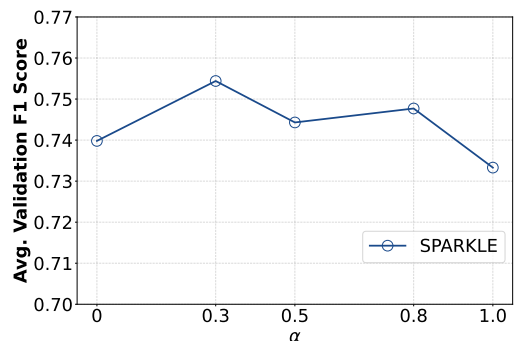


Figure 10: Sensitivity analysis of α , where the average F1 scores on the validation set under different α values are reported.

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

E.2 Sensitivity Analysis

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

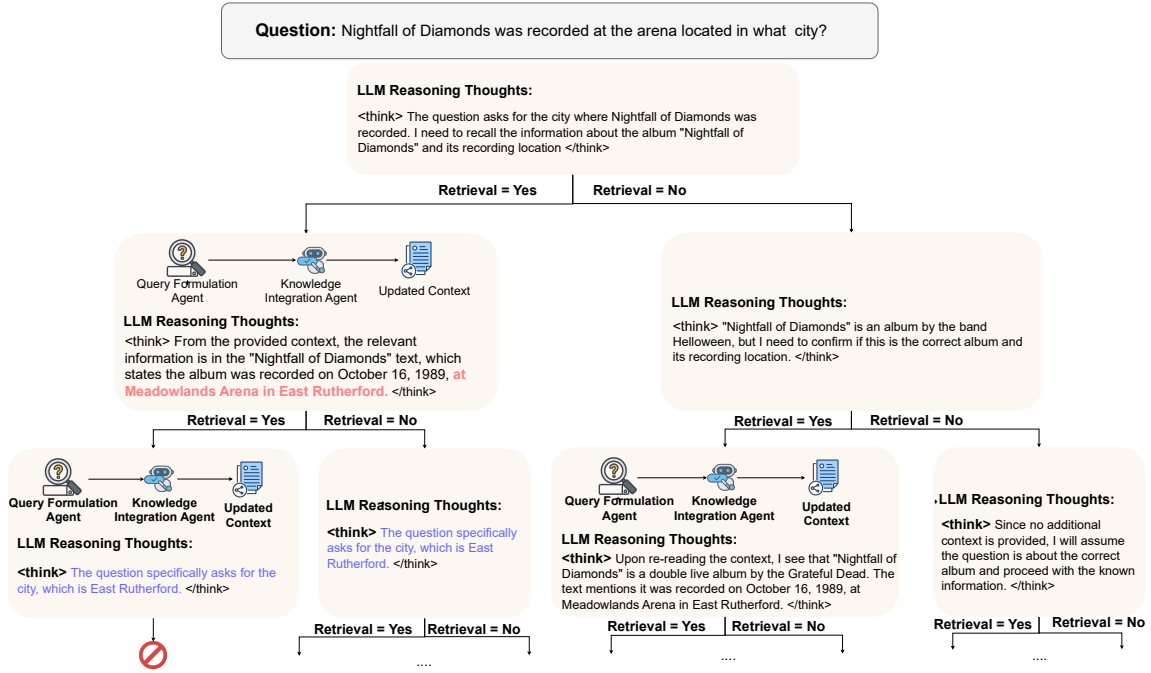


Figure 11: Illustration of the proposed binary tree-structured rollout strategy during PPO training. At each reasoning step, instead of relying on the Retrieval Decision agent to make retrieval decisions, both possible directions, i.e., retrieval and non-retrieval, are explicitly explored. For the retrieval 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.

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*	<u>52.08*</u>	<u>65.88</u>
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

We conduct experiments to compare SPARKLE with recent graph RAG methods, including HippoRAG (Gutiérrez et al., 2024), PAPTOR (Sarthi et al., 2024) and SiReRAG (Zhang et al., 2025). To ensure a fair comparison, we adopt the exact same experimental setups as used in SiReRAG, including both datasets and model configurations. In terms of datasets, while SiReRAG also evaluates on HotPotQA, 2Wiki, and MuSiQue, it follows the setting of HippoRAG, where 1,000 questions are sampled from each dataset, along with a subset of the original corpus (see Table 1 in HippoRAG for details). In our experiments, we

use the same sampled questions and corresponding corpus subsets for consistency. In terms of model configurations, following SiReRAG, we use OpenAI’s text-embedding-3-small as the embedding model for retrieval, GPT-4o for answer generation, and GPT-3.5-Turbo to extract knowledge triples from documents for KG construction (see Section 5.4 in the SiReRAG paper).

Table 7 presents the performance comparison between SPARKLE and graph RAG baselines across three datasets. The results indicate that SPARKLE achieves the highest average F1 score of 68.64%, outperforming the strongest baseline SiReRAG by 2.76%. This highlights the effectiveness of our adaptive retrieval framework and structured reasoning mechanism, which together enable more accurate and efficient knowledge integration than existing graph RAG methods. Notably, SPARKLE achieves a substantial improvement of 4.37% over SiReRAG on 2Wiki. This improvement is likely due to SPARKLE’s dynamic construction of KG-based reasoning chains based on the LLM’s intermediate thoughts, which guide both retrieval and knowledge integration in a goal-directed manner. In contrast, graph RAG baselines rely on static

graph indices built over the corpus, which can suffer from error propagation or noise introduced during graph construction.

E.4 Case Analysis

Binary Tree-Structured Rollout Strategy. Figure 11 illustrates a concrete example of our binary tree-structured rollout strategy during PPO training. At each reasoning step, instead of relying on the Retrieval Decision agent to make a single retrieval decision, we explicitly explore both possible branches, i.e., retrieval and non-retrieval, to construct a binary reasoning tree. This enables the proxy model to explore a broader space of trajectories and acquire more informative reward signals for learning effective retrieval policy.

While the Retrieval Decision agent is not activated during rollout, the retrieval decision on each branch is recorded and used for policy optimisation. In contrast, during inference, we do not perform branching and directly leverage the learned Retrieval Decision agent to make a single deterministic decision at each step.

End-to-End Inference. To better illustrate the end-to-end inference process of SPARKLE, we present four case studies (Tables 8-11) that show how the three agents within SPARKLE, i.e., Retrieval Decision, Query Formulation and Knowledge Integration, interact with the LLM and the retriever to handle *different types* of questions.


Tables 8-10 present successful inference trajectories for *multi-hop*, *single-hop*, and *commonsense* questions, respectively. In Table 8, the two-hop question triggers two retrievals, with each corresponding 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 Retrieval 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 interpreting the question or reasoning over existing context, the agent appropriately suppresses retrieval to avoid unnecessary overhead. Moreover, the Query Formulation and Knowledge Integration agents can also generate targeted queries and accurately incorporate 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 Formulation 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 highlights 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 Ostrogi (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.