HAMMER: An Automatic RAG Tuning System via Hierarchical Memory-Guided Monte Carlo Tree Search

YINGLI ZHOU, The Chinese University of Hong Kong, Shenzhen, China
ZIXUAN WANG, Harbin Institute of Technology, China
YIXIANG FANG*, The Chinese University of Hong Kong, Shenzhen, China

Retrieval-Augmented Generation (RAG) effectively integrates external knowledge into large language models (LLMs), enhancing accuracy, adaptability, interpretability, and trustworthiness. The modern RAG systems contain multiple modules with many tunable parameters, each of which can greatly influence the overall performance of downstream applications. To identify the optimal configuration for a given RAG pipeline, existing studies have relied on hyperparameter optimization (HPO) methods. However, these methods overlook parameter dependencies, lack experience-based learning, and incur substantial computational costs. To tackle those issues, we propose HAMMER, a hierarchical memory–guided Monte Carlo Tree Search (MCTS) system. Inspired by human learning, HAMMER employs a hierarchical graph memory to organize experimental insights and integrates it with MCTS for more reliable tuning. We further design a theoretically guaranteed query selection technique to reduce cost while preserving effectiveness. Extensive experiments on eight real-world datasets show that HAMMER improves exact match by up to 20.0% and F1-score by 15.2%, while reducing both tuning time and token consumption by up to 9×.

CCS Concepts: • **Information systems → Information systems applications**.

Additional Key Words and Phrases: RAG, MCTS, System tuning

ACM Reference Format:
Yingli Zhou, Zixuan Wang, and Yixiang Fang. 2026. HAMMER: An Automatic RAG Tuning System via Hierarchical Memory-Guided Monte Carlo Tree Search. *Proc. ACM Manag. Data* 4, 3 (SIGMOD), Article 194 (June 2026), 29 pages. https://doi.org/10.1145/3802071

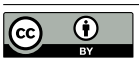
1 Introduction

The development of Large Language Models (LLMs) like GPT-5 [72], Qwen3 [96], and Llama 4 [16] has sparked a revolution in the field of artificial intelligence [23, 30, 50, 60, 71, 90, 92, 105]. Despite their remarkable comprehension and generation capabilities, LLMs may still generate incorrect outputs due to a lack of domain-specific knowledge, real-time updated information, and proprietary knowledge, which are outside LLMs' pre-training corpus, known as "hallucination" [76].

To bridge this gap, the Retrieval Augmented Generation (RAG) technique [20, 22, 27, 32, 94, 99, 104, 111] has been proposed, which supplements LLM with external knowledge to enhance its factual accuracy and trustworthiness. Given a user query Q, the key idea of naive RAG [45] is to retrieve relevant chunks from the external corpus, and then feed them along with Q as a prompt into LLM to generate answers. Consequently, RAG techniques have been widely applied in various fields, especially in domains where LLMs need to generate reliable outputs, such as

*Corresponding author.

Authors' Contact Information: Yingli Zhou, yinglizhou@link.cuhk.edu.cn, The Chinese University of Hong Kong, Shenzhen, Shenzhen, China; Zixuan Wang, 2023113027@stu.hit.edu.cn, Harbin Institute of Technology, Haerbin, China; Yixiang Fang, The Chinese University of Hong Kong, Shenzhen, Shenzhen, China, fangyixiang@cuhk.edu.cn.



ACM 2836-6573/2026/6-ART194
https://doi.org/10.1145/3802071

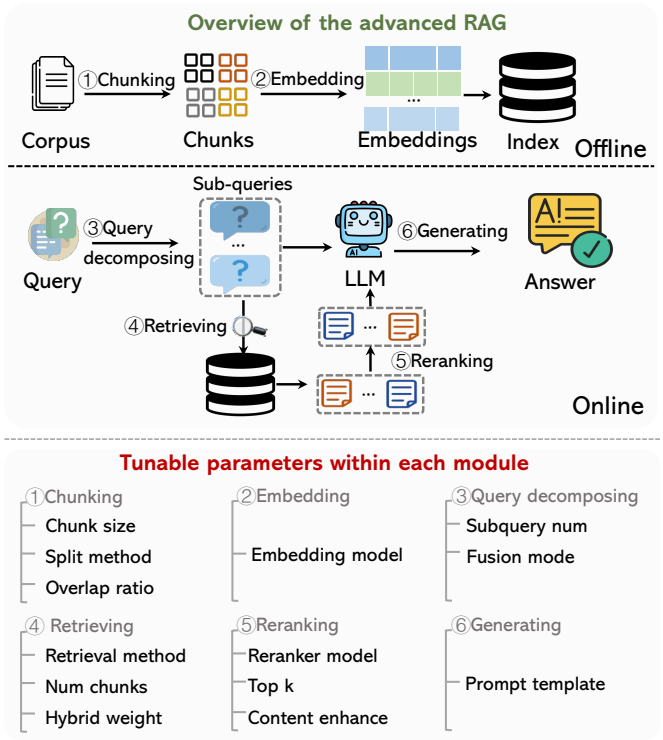
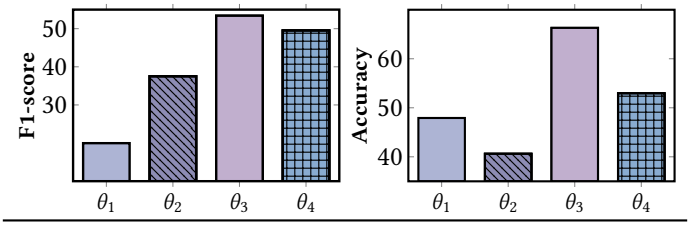


Fig. 1. Overview of the advanced RAG pipeline and its six modules with 13 tunable parameters.

healthcare [60, 90, 105], finance [50, 71], and education [23, 92]. Moreover, RAG has proven highly useful in many data management tasks, including NL2SQL [18, 47], data cleaning [19, 48, 69, 77], knob tuning [24, 42], DBMS diagnosis [81, 106, 109], and SQL rewrite [51, 85].

The modern advanced RAG pipeline, as shown in Fig. 1, typically comprises six modules [41, 63], such as chunking, embedding, and retrieving, each containing multiple tunable parameters. For example, the chunking module determines how to split the corpus into chunks (`Split method`) and the appropriate chunk size (`Chunk size`); the retrieving module should determine how many chunks to retrieve (`Num chunks`). These parameters significantly influence the overall performance of the RAG system across different task types and domains [27, 63, 111], as they directly affect how each module processes information. For instance, varying the chunk size alters the granularity of retrieved text, while the choice of embedding model determines how semantic relations are captured, where smaller chunks favor context-sensitive embeddings, whereas larger chunks benefit from semantically rich ones. Moreover, domain-specific tasks (e.g., biomedical vs. financial) often require different reranker models, and complex reasoning questions benefit from deeper query decomposition, such as using a larger number of subqueries (`Subquery num`). Fig. 2 illustrates an example of a RAG pipeline on the HotpotQA dataset [98], a well-known multi-hop reasoning benchmark, under four different configurations. We observe that varying configurations (i.e., parameters) lead to substantial performance differences, highlighting the sensitivity of RAG systems to parameter choices. To this end, in this paper, we investigate how to determine the optimal parameters of a RAG pipeline, or equivalently, the optimal configuration of a RAG pipeline, namely, the *RAG tuning problem*.



Parameter	θ_1	θ_2	θ_3	θ_4
`Chunk size`	128	1,024	256	512
`Split method`	token	sentence	recursive	token
`Embedding model`	Finance2_en	gte-large	bge-base-en	bge-m3
`Num chunks`	1	3	5	10

Fig. 2. Results of a RAG pipeline on the HotpotQA dataset with different configurations, where θ_i denotes a specific configuration of tunable parameters.

• **Prior works.** While the RAG tuning problem is highly valuable, only a few works have explored it so far. The first systematic study by IBM Research evaluated five hyperparameter optimization (HPO) methods for tuning naive RAG pipelines [73]. Recently, Bayesian optimization–based approaches have been proposed. In particular, Dataroot introduced `Syftr` [12], a Tree-Structured Parzen Estimator (TPE)–based framework that supports advanced RAG pipelines with large configuration spaces. Gaussian process–based optimization has also been explored for RAG tuning [1]. In addition, `AutoRAG` [21] adopts a multi-armed bandit strategy for hyperparameter selection in RAG pipelines. In total, eight HPO-based methods have been explored for RAG tuning.

• **Motivations.** Although the existing methods above have achieved some positive progress, they still exhibit three major limitations (**L** for short):

(L1): Ignore parameter dependencies. Existing HPO-based methods typically treat parameters as independent variables. However, in a RAG pipeline, many parameters are inherently interdependent. For example, the choice of embedding model often influences the optimal chunk size, as different embeddings capture semantic information at varying granularities. Ignoring such dependencies can easily lead to suboptimal configurations.

(L2): Lack of experience-based learning. HPO-based methods need to conduct tuning through multiple iterations, yet they completely ignore the relationships across different tuning rounds as well as the valuable experimental signals and insights embedded in historical tuning processes. Learning from past experiments is a principle deeply rooted in human problem-solving. Without such experience-based learning, these methods tend to repeat similar mistakes and repeatedly explore suboptimal parameters, leading to inefficient and ineffective tuning.

(L3): Extensive tuning cost. Existing methods evaluate each candidate configuration on the full training set during every tuning iteration, resulting in excessive token usage and long runtimes before convergence to a near-optimal configuration.

• **Our technical contributions.** In this paper, we propose `HAMMER`, a novel **H**ier**A**rchical **M**emory-guided **M**onte Carlo Tre**E** Search based **R**AG tuning system, to address the three limitations discussed above. Specifically, `HAMMER` consists of two key stages: 1) a *submodular-driven query selection stage* that selects the most representative queries from the query set with a rigorous theoretical guarantee, and 2) a *memory-guided MCTS-based configuration optimization stage* that mimics how humans learn from intermediate feedback and accumulate experience across trials,

thereby making the tuning process more reliable and effective. In the first stage, HAMMER leverages query embeddings to quantify pairwise similarities between queries and identify a subset of queries that are both representative and diverse. Specifically, we model the query selection problem as a *submodular cover* problem [34], which is well known to be NP-hard. In light of this, we design an approximation algorithm with a rigorous and provable $\frac{1}{2}$-approximation guarantee, which efficiently produces a near-optimal subset.

In the second stage, we integrate MCTS with a hierarchical memory structure, SE-Bank, which organizes tuning history and search experience into a three-layer graph. This design enables effective, reliable, and knowledge-driven RAG tuning. HAMMER is the first system to employ MCTS for RAG tuning, explicitly modeling sequential parameter dependencies (addressing (L1)). It further incorporates two memory-guided components: 1) *Memory-guided simulation*, which retrieves similar configurations and distilled insights from SE-Bank to guide parameter selection, mitigating (L2); and 2) *Memory-guided evaluation*, which samples a small subset of queries and refines their results through LLM-assisted estimation, thereby reducing tuning costs ((L3)). After each iteration, SE-Bank is incrementally updated, mirroring continual human learning.

Extensive experiments on eight real-world RAG scenarios across medicine and science domains show that HAMMER consistently outperforms state-of-the-art methods, improving exact match by up to 20.0% and F1-score by 15.2%, while reducing tuning time and token usage by 9×. We have released the source codes and datasets of our work: https://github.com/Cyan9061/HAMMER.

In summary, our main contributions are as follows.

- We design the first hierarchical memory-guided Monte Carlo Tree Search (MCTS) based RAG tuning system that mimics the process of human learning, learning from historical experiences.
- We organize the historical experiences in a hierarchical graph memory and integrate it with MCTS to enable more reliable tuning. In addition, we propose a query selection algorithm with rigorous theoretical guarantees to reduce the tuning cost.
- We conduct experiments on eight real-world datasets to demonstrate the effectiveness and efficiency of our system.

Outline. We introduce the preliminaries and analyze the limitations of existing algorithms in Section 2. We introduce the overview of HAMMER in Section 3, and present our query selection algorithm in Section 4. Section 5 introduces our memory-guided MCTS algorithm. The experimental results are reported in Section 6. We review the related works in Section 7 and conclude in Section 8.

2 Preliminaries

In this section, we review the key concepts of large language models (LLMs), the pipeline of retrieval-augmented generation (RAG), and then formalize the problem studied in this paper.

2.1 Large Language Models (LLMs)

We introduce some fundamental concepts of LLMs, including LLM prompting and retrieval augmented generation (RAG).

▸ **LLM Prompting.** After instruction tuning on the large corpus of human interaction scenarios, LLM is capable of following human instructions to complete different tasks [14, 74]. Specifically, given the task input, we construct a prompt that encapsulates a comprehensive task description. The LLM processes this prompt to fulfill the task and generate the corresponding output. Note that pre-training on trillions of bytes of data enables LLM to generalize to diverse tasks by simply adjusting the prompt [74].

▸ **Retrieval Augmented Generation.** During task completion with prompting, LLMs often generate erroneous or meaningless responses, i.e., the hallucination problem [30]. To mitigate this problem, retrieval augmented generation (RAG) is utilized as an advanced LLM prompting technique by using the knowledge within the external corpus. Typically, there are two widely used RAG pipelines in the popular LLM-based systems, such as Llama Index [62] and Langchain [41], namely *1) naive RAG* and *2) advanced RAG*.

2.2 Pipeline of RAG

In this subsection, we introduce the pipelines of naive RAG and advanced RAG [22].
▸ **Naive RAG.** The corpus $\mathcal{D}$ is first segmented into smaller chunks. Given a query q, the retriever fetches the top-k most relevant chunks. These chunks, together with the query, are concatenated into a prompt, which is then processed by the LLM to generate the final answer.
▸ **Advanced RAG.** To improve the effectiveness of RAG, advanced RAG extends the naive framework with additional modules. Beyond basic retrieval, it often incorporates reranking to refine candidate chunks, hybrid retrieval that combines dense and sparse strategies to avoid missing relevant chunks, and query decomposition that breaks complex questions into simpler sub-queries. Advanced RAG typically achieves substantially better performance than naive RAG.

By examining both naive and advanced RAG pipelines, we identify *six key modules* that are widely used in the modern RAG systems [41, 62], as shown in the Fig. 1: *1) Chunking* Splits documents into small fixed-length chunks. *2) Embedding*. Maps each chunk into a high-dimensional dense vector representation using a pre-trained embedding model. *3) Retrieving*. Selects the top-k relevant chunks from the corpus using dense, sparse, or hybrid retrieval methods. *4) Query decomposing*. Decomposes the query into smaller sub-queries to improve retrieval accuracy and support fine-grained reasoning. *5) Reranking*. Re-scores and re-orders retrieved chunks with a more accurate ranking model. *6) Generating*. Produces the final answer with an LLM, guided by parameters such as the prompt template. Note that naive RAG typically involves only modules 1)–3) and 6), whereas advanced RAG incorporates all six modules.

2.3 Problem definition

In this subsection, we formally define the research problem studied in this paper. The six modules introduced above, together with their 13 configuration parameters, define the configuration space of a RAG pipeline. Detailed descriptions are provided in the supplementary material due to space limitations [112].

Definition 2.1 (Configuration θ and Space Θ). A configuration θ is a concrete instantiation of the RAG pipeline, obtained by assigning values to all tunable parameters of its modules. The configuration space Θ is the Cartesian product of parameter domains:

$$\Theta = \Theta_1 \times \Theta_2 \times \cdots \times \Theta_m, \tag{1}$$

where m denotes the number of all parameters in the RAG pipeline, and each Θ_i is the domain of the i-th tunable parameter.

We note that the 13 parameters in the advanced RAG pipeline jointly determine its behavior: for example, the chunk size and embedding model affect the granularity and semantic quality of retrieved information, while the reranking and query decomposition strategies influence reasoning accuracy. A particular assignment of these parameters forms a *configuration* θ, representing a specific instantiation of the RAG pipeline. Collectively, all possible configurations constitute the configuration space Θ, which defines the search domain for RAG tuning.

We denote the performance metric by f, which can represent any evaluation measure to be optimized, such as F1-score or accuracy. Given a query set Q, an external corpus $\mathcal{D}$, and a specific configuration θ, we use $f_Q(\theta)$ to denote the corresponding performance of the RAG system. Given these definitions, we now formalize the RAG configuration optimization problem or *RAG tuning problem*.

Problem 1 (RAG tuning problem [12, 73]). *Given a RAG pipeline, its configuration space Θ, an external corpus $\mathcal{D}$, and a query set Q. Assuming that the objective is a maximization problem, RAG tuning aims to find a configuration $\theta^* \in \Theta$, where*

$$\theta^* = \arg\max_{\theta \in \Theta} f_Q(\theta). \tag{2}$$

In practice, RAG tuning requires splitting the query set into training and testing sets, to learn a configuration that performs well on the training queries while generalizing effectively to the unseen test queries. In addition, we notice that recent studies have explored automatic RAG pipeline construction, including MCTS-based approaches and agentic RAG frameworks, which primarily focus on composing pipeline architectures for a given dataset [80]. Our work is orthogonal to this line of research: once the pipeline structure is fixed, we focus on optimizing its configuration to achieve optimal end-to-end performance. This separation allows our approach to naturally complement existing RAG automation systems. In this work, we consider the advanced RAG pipeline, which is widely used in well-known modern LLM systems (e.g., LangChain, LangGraph, and LlamaIndex), reflecting realistic deployment settings.

2.4 Analysis of HPO-based RAG Tuning

HPO-based hyperparameter tuning methods are widely applied in machine learning model optimization [67] and database knob tuning [102]. In essence, the RAG tuning problem can also be regarded as a form of HPO. Recently, research groups such as IBM and Dataroot have proposed HPO-based approaches for RAG tuning [12, 73], employing methods such as Bayesian optimization to search for the "best" configurations for a specific RAG pipeline.

We note that some recent studies have proposed LLM-guided HPO approaches for database knob tuning [24, 42, 84]. These methods typically leverage LLMs to read DBMS manuals and identify promising knobs or reduce the search space of individual knobs. However, directly applying this idea to RAG tuning is infeasible. Unlike DBMSs, RAG systems lack abundant external knowledge sources — such as official manuals or community forums (e.g., Stack Overflow) — that provide scenario-specific parameter guidance exploitable by LLMs. In the absence of such knowledge, LLMs tend to produce unreliable reasoning, leading to overlooking optimal configurations (see Exp.4 in Section 6.2).

The three key limitations of existing HPO-based methods are: 1) These methods treat parameters as independent, overlooking their interdependencies, such as how different embedding models align with specific chunk sizes, which often leads to suboptimal configurations. 2) HPO-based methods ignore insights from past tuning rounds, failing to follow the human principle of learning from experience. As a result, they often repeat mistakes and re-explore suboptimal parameters, causing inefficient and ineffective tuning. 3) All methods need to evaluate each candidate configuration on the entire query set, which incurs extensive tokens and time to reach near-optimal configurations. In this paper, we aim to design a novel RAG tuning system to address the three key challenges.

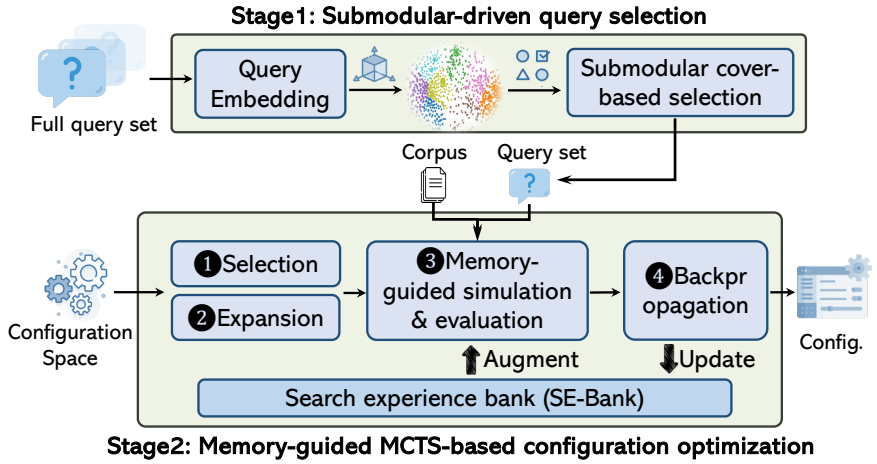


Fig. 3. The overview of HAMMER.

3 Our HAMMER Approach

In this paper, we propose HAMMER, a novel **H**ier**A**rchical **M**emory-guided **M**onte Carlo Tre**E** Search based **R**AG tuning system, to address the three limitations discussed above; the overall framework is shown in Fig. 3. Specifically, given a document, a query set, and the configuration space Θ of a specific RAG workflow, our HAMMER contains two stages: *1) submodular-guided query selection*, and *2) memory-guided MCTS-based configuration optimization*. The first stage aims to select a small subset of representative queries from the entire query set, as detailed in Section 4. We formulate the representative query selection problem as a submodular cover problem, which is a well-known NP-hard problem [34]. To address this, we design an efficient greedy algorithm with a theoretical approximation guarantee.

In the second phase, the key idea is to leverage the knowledge stored in the hierarchical graph memory, termed SE-Bank (see Section 5.2), to guide the MCTS-based RAG tuning process. Specifically, we employ MCTS for RAG tuning to capture the dependencies among different parameters. During this process, HAMMER incorporates two key components: *memory-guided simulation* and *memory-guided evaluation*, to ensure reliable and efficient tuning (see Section 5.3). In *memory-guided simulation*, HAMMER retrieves high-level insights and similar configurations from SE-Bank to guide parameter selection. These historical experiments provide informative signals for decision-making, analogous to the human learning process of accumulating and leveraging past experience. In *memory-guided evaluation*, instead of evaluating the entire query set, HAMMER samples a small subset of representative queries and refines their evaluation results by leveraging insights stored in SE-Bank to prompt the LLM for more accurate refinement. This strategy effectively reduces computational and token costs while maintaining evaluation reliability.

4 Query selection

In this section, we introduce our query selection algorithm, which aims to select a small subset of representative queries from the query set Q, thereby reducing the overall cost of the entire process of HAMMER.

4.1 The query selection problem

Our goal is to select a subset of queries that capture the key characteristics of the full query set. In other words, tuning the RAG system using the selected queries should achieve performance comparable to that obtained using the full query set.

Intuitively, the selected queries should be mutually dissimilar so that they collectively form a representative subset preserving the diversity and coverage of the original query set for effective RAG tuning. To quantify query dissimilarity, we measure the semantic similarity between queries. In practice, each query is mapped to a vector representation using a pre-trained embedding model (e.g., BERT [13], bge-m3 [68], or ChatGPT [72]). Given two queries q_i and q_j with embedding vectors $\mathbf{v}_i$ and $\mathbf{v}_j$, respectively, we define their similarity using cosine similarity: $\phi(q_i, q_j) = \frac{\mathbf{v}_i \cdot \mathbf{v}_j}{\|\mathbf{v}_i\| \, \|\mathbf{v}_j\|}$, where $\phi(q_i, q_j) \in [-1, 1]$, where larger values indicate higher similarity between the two queries.

Based on this, a pair of queries (q_i, q_j) is considered *dissimilar* if their similarity is below the threshold $\tau \in [0, 1]$, that is, (q_i, q_j) is dissimilar if $\phi(q_i, q_j) < \tau$. Here, we quantify the quality of a selected subset $S \subseteq Q$ by counting the number of mutually dissimilar query pairs, $h(S, \tau) = \left|\{\{q_i, q_j\} \subseteq S : \phi(q_i, q_j) < \tau\}\right|$. However, directly maximizing $h(S, \tau)$ is not meaningful, since selecting the entire query set Q always yields the maximum value.

To avoid such a trivial solution, we require the selected subset to be representative yet not excessively large. Thus, we normalize the objective by the subset size, $\frac{h(S,\tau)}{|S|}$, where $|S|$ denotes the number of queries in the subset. In this formulation, the subset size serves as a penalty term, discouraging overly large selections. Besides, we enforce a minimum size constraint $|S| \geq k$ to prevent the selected subset from being too small, ensuring sufficient coverage for robust evaluation on the test set.

With these considerations, the query selection problem is formally defined as:

Problem 2 (Query selection problem). *Given a query set Q, a similarity function $\phi(\cdot, \cdot)$ that measures the similarity between any pair of queries $q_i, q_j \in Q$, and a threshold τ, the goal is to select a subset $S \subseteq Q$ with $|S| \geq k$ that maximizes*

$$\max_{S \subseteq Q, \ |S| \geq k} \frac{h(S, \tau)}{|S|}. \tag{3}$$

In the following, we first show that finding the optimal solution to this problem is NP-hard, and then present an efficient polynomial-time $\frac{1}{2}$-approximation algorithm to solve it.

4.2 Our query selection algorithm

In this subsection, we present our query selection algorithm. We first note that finding the optimal solution to problem 2 is NP-hard. The proof is provided in the supplementary material [112].

Given this hardness result, we focus on designing an efficient approximation algorithm that provides a provable approximation ratio guarantee, rather than attempting to compute the exact optimal solution. Here, we propose a greedy-based approximation algorithm, as shown in Algorithm 1.

The algorithm iteratively expands the solution by greedily selecting, at each iteration, the subset of queries that maximizes the marginal gain in dissimilar query pairs (lines 2–5). This strategy incrementally promotes diversity during the construction process. After reaching the budget k, all intermediate solutions are completed to size k (line 8), and the solution with the highest diversity ratio is selected as the final output (line 10).

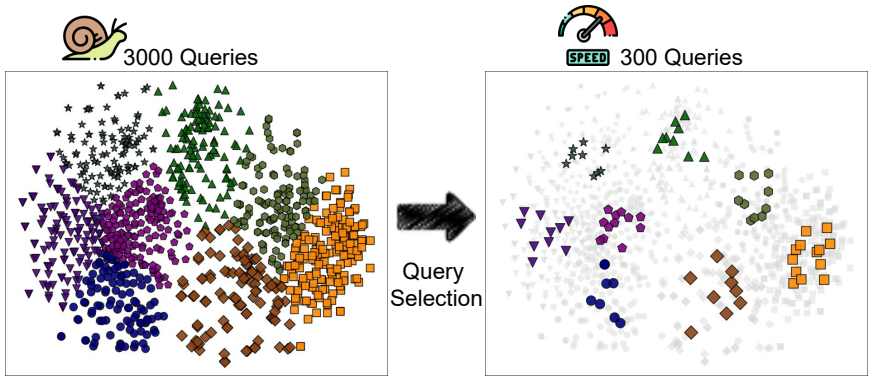


Fig. 4. Illustration of query selection.

Algorithm 1: `Greedy`

Input: A query set Q, a budget k, and a threshold τ
Output: $S \subseteq Q$ with $|S| \geq k$

$S_0 \leftarrow \emptyset$; $j \leftarrow 0$;
while $|S_j| < k$ **do**
  $j \leftarrow j + 1$;
  // choose a set of queries
  $H_j \leftarrow \arg\max_{T \subseteq Q \setminus S_{j-1}} \frac{h(S_{j-1} \cup T, \tau) - h(S_{j-1}, \tau)}{|T|}$;
  $S_j \leftarrow S_{j-1} \cup H_j$;
$\ell \leftarrow j$;
for $j \leftarrow 1$ **to** ℓ **do**
  // pad to size k if needed
  add any $\max\{k\text{-}|S_j|, \emptyset\}$ vertices to S_j to form S'_j ;
$j^* \leftarrow \arg\max_{j \in [\ell]} \frac{h(S'_j, \tau)}{|S'_j|}$;
return S'_{j^*} ;

Example 1. *As shown in Fig. 4, the left plot visualizes the embeddings of 3,000 queries, where queries with similar semantics are marked in the same color. On the right, after applying our query selection method, only 300 representative queries are retained. We can observe that queries from each color group are covered, indicating that the selected subset preserves the diversity of the entire query set.*

By using our query selection strategy for RAG tuning, the selected subset (typically 30% of queries) remains highly representative of the full query set, enabling our method HAMMER to achieve nearly the same tuning performance as using all queries, while significantly reducing computational and token costs.

Theorem 4.1. *Given a query set Q, a budget k, and a threshold τ, the algorithm* `Greedy` *returns a set S with $|S| \geq k$ and $\frac{h(S,\tau)}{|S|} \geq \frac{1}{2} \times \frac{h(S^*,\tau)}{|S^*|}$, where S^* is an optimal solution to the Problem 2.*

Discussion. We note that our query selection problem can be viewed as an instance of the *coreset selection* problem [6]. Both aim to select a representative subset S from a large dataset Q

(where $|S| \ll |Q|$) to approximate the performance on the full set. That is, the existing coreset selection algorithm can also be applied here. While coreset methods minimize the distance between each query and its nearest selected counterpart in the embedding space, our approach instead maximizes the dissimilarity among selected queries to reduce redundancy and capture broader semantic diversity. In our later experiments, we show that our query selection method has two main advantages over *coreset selection*: it is up to five times faster, and achieves up to 9.33% higher EM and 7.83% higher F1 compared to the coreset-based alternative.

5 Hierarchical MEMORY-GUIDED MONTE CARLO TREE SEARCH

In this section, we first present how Monte Carlo Tree Search (MCTS) can be applied to RAG tuning and analyze its limitations. We then propose a hierarchical memory-guided MCTS method, introducing a newly designed hierarchical graph memory structure and describing how its knowledge is integrated into MCTS to overcome the limitations of direct MCTS-based tuning.

5.1 MCTS for RAG tuning

In this subsection, we first briefly review the MCTS, and then discuss how to use MCTS for RAG tuning.

Algorithm 2: `MCTS for RAG tuning`

Input: Tuning iterations T
Output: Configuration with maximum reward

Initialize MCTS tree root v_0;
foreach $t \leftarrow 1, 2, 3, \cdots, T$ **do**
  $v_t \leftarrow \texttt{Select}(v_0)$;
  $(v'_t, a_t) \leftarrow \texttt{Expand}(v_t)$;
  `// Complete partial parameters randomly`
  $\theta_t \leftarrow \texttt{Rollout}(v'_t)$;
  $\widehat{R}(v_t, a_t) \leftarrow f_Q(\theta_t)$;
  Backpropagation via equations (6) - (8);
return configuration with maximum reward;

▸ **Introduction of MCTS.** MCTS is a widely used search algorithm for decision-making [38]. Given an objective and a set of possible actions, MCTS aims to identify the "best" sequence of actions (i.e., an execution trajectory) that maximizes the expected reward for achieving the goal. The key idea of MCTS is to iteratively simulate action sequences by traversing a search tree, where each node represents a state and each edge corresponds to an action leading to a new state. At every iteration, MCTS performs four steps: (1) *Selection*, traversing from the root to a leaf node by recursively choosing, at each layer, the child node with the highest score; (2) *Expansion*, adding a new child node to explore previously unvisited actions; (3) *Simulation*, conducting rollouts from the new state to estimate potential outcomes; and (4) *Backpropagation*, updating the statistics (e.g., visit counts and average rewards) of nodes along the traversal path. Through repeated iterations, MCTS gradually refines its estimates and guides the search towards action sequences with higher expected rewards. In the context of RAG tuning, the configuration process is naturally formulated as a sequential decision problem. As illustrated in Fig. 1, an advanced RAG pipeline consists of six ordered modules that must be configured step by step. Specifically, the system first determines the chunking strategy (e.g., splitting method and chunk size), and then selects the embedding model based on the resulting chunks, followed by subsequent retrieval and generation components. Each

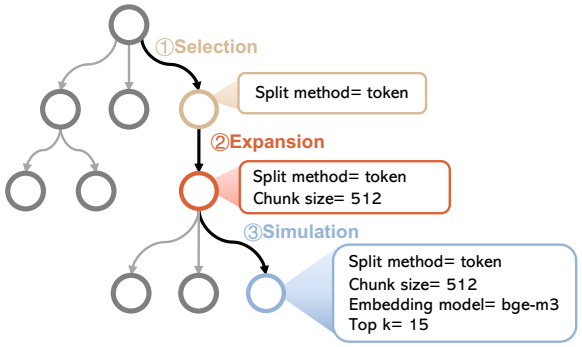


Fig. 5. Illustration of MCTS for RAG tuning.

decision directly conditions the feasible choices and effectiveness of later stages, forming a tree-style dependency chain over the configuration space. Therefore, MCTS is particularly well-suited for RAG tuning, as it explicitly models sequential decisions and parameter dependencies, effectively addressing Limitation (L1).

▸ **Using MCTS for RAG tuning.** Motivated by the discussions above, we model the RAG tuning problem as a tree-structured search process and employ MCTS to efficiently explore the configuration space. Formally, given a configuration space Θ consisting of m tunable parameters, we construct a directed search tree $\mathcal{T} = (V, E)$. Each node $v \in V$ represents a partial configuration in which the first several parameters have been assigned, while each edge $e \in E$ corresponds to selecting a specific value for the next parameter, thereby extending the current configuration. The tree has depth m, where a node at depth i encodes a partial assignment of the first i parameters. Consequently, each root-to-leaf path uniquely defines a complete configuration $\theta \in \Theta$, which can be instantiated into a concrete RAG pipeline for evaluation. Following this formulation, MCTS iteratively performs the following four steps: selection, expansion, evaluation, and backpropagation to progressively identify high-quality configurations. The detailed procedure is presented in Algorithm 2.

❶ **Selection.** At each iteration, MCTS starts from the root node v_0 and recursively traverses the search tree until reaching a leaf node v_t by selecting, at each step, the child node with the highest selection score (line 3 in Algorithm 2). We adopt the Upper Confidence Bound for Trees (UCT) [38] as the selection policy to balance exploration and exploitation.

Specifically, for a state v and an action $a \in \mathcal{A}(v)$, where $\mathcal{A}(v)$ denotes the set of feasible actions from v, the UCT score is:

$$U(v, a) = \frac{R(v, a)}{N(v, a)} + c \cdot \sqrt{\frac{\ln N(v)}{N(v, a)}}, \tag{4}$$

where $R(v, a)$ is the cumulative reward obtained by executing action a at state v, $N(v, a)$ is the number of times action a has been taken from v, and $N(v)$ is the total number of visits to v. The constant c balances exploration and exploitation.

❷ **Expansion.** At the t-th iteration, if the selected node v_t is not at the final depth m or is not fully expanded (i.e., there exists an untried action in $\mathcal{A}(v_t)$), the algorithm randomly selects one such unexplored action a_t. Executing a_t generates a new child node v'_t, which is then added to the search tree (line 4).

❸ **Simulation.** Given the newly expanded node v'_t at depth i, MCTS performs a rollout to complete the remaining $m - i$ parameters, thereby constructing a full configuration θ_t (lines 5–6).

A common strategy is random rollout, where the remaining parameters are sampled uniformly at random and combined with the partial configuration defined by the path to v'_t.

❹ **Backpropagation.** After evaluating the completed configuration θ_t and obtaining the reward $\widehat{R}(v_t, a_t) = f_Q(\theta_t)$, the reward is propagated backward along the traversal path to update all visited state–action pairs. Let Π_t denote the search path in the t-th iteration, represented as a sequence of state–action pairs:

$$\Pi_t = \{(v_t^1, a_t^1), (v_t^2, a_t^2), \ldots, (v_t^m, a_t^m)\} \tag{5}$$

where each $v_t^i \in V$ is a node, and a_t^i is the applied action at that node. For each pair $(v_t^i, a_t^i) \in \Pi_t$, we update the following three statistics as follows:

$$R(v_t^i, a_t^i) \leftarrow R(v_t^i, a_t^i) + \widehat{R}(v_t^i, a_t^i) \tag{6}$$

$$N(v_t^i) \leftarrow N(v_t^i) + 1 \tag{7}$$

$$N(v_t^i, a_t^i) \leftarrow N(v_t^i, a_t^i) + 1 \tag{8}$$

Example 2. *Fig. 5 illustrates an example of using MCTS for RAG tuning. Each iteration begins at the root node. The algorithm traverses the tree by repeatedly selecting the node with the highest UCT score until it reaches an expandable node. From there, the algorithm expands the tree by randomly choosing an untried action (in this case, selecting an embedding model) to create a new node. Next, a simulation (or rollout) is performed from the current state to obtain a reward. Finally, this reward is used to update the statistics of all nodes along the path back to the root, which is highlighted in black in this figure.*

▸ **Limitations of using MCTS for RAG tuning.** So far, limitation (L1) can be effectively addressed. While we observe that directly applying MCTS for RAG tuning still has some limitations:

1) Low exploration efficiency. MCTS only relies on the final reward for backpropagation and does not explicitly leverage intermediate trajectories or historical search experience. This delayed credit assignment makes it difficult to identify which parameter choices contribute to strong performance [70], leading to repeated exploration of suboptimal regions. 2) Unreliable random rollouts. The simulation phase randomly completes partial configurations, often generating implausible or incompatible parameter combinations. Such blind rollouts introduce high reward variance and provide noisy guidance for tree expansion. 3) Excessive evaluation cost. Each intermediate configuration in the tuning process (i.e., each node) requires executing the entire set of queries on the RAG pipeline under that configuration and evaluating the results to generate scores for subsequent search. This exhaustive evaluation incurs a high computational cost.

To overcome the limitations of vanilla MCTS, we resort to the principle of human learning from experience by distilling historical tuning trajectories into a hierarchical memory structure, termed the Search Experience Bank (`SE-Bank`), and proposing a memory-guided MCTS framework (`MG-MCTS`). By replacing random rollouts with memory-guided simulations and introducing a memory-guided evaluation module, `MG-MCTS` enables informed exploration and substantially reduces tuning cost. We next present the memory design and its integration into MCTS.

5.2 The Hierarchical Graph Memory

In this subsection, we introduce the structure of our hierarchical graph memory, termed `SE-Bank`, which organizes search experiences from the MCTS tuning process into a structured multi-layer graph.

▸ **The structure of `SE-Bank`.** We design `SE-Bank` as a three-layer hierarchical graph $\mathcal{M}$, where each layer stores search knowledge at a different level of abstraction. From bottom to top, the information becomes progressively more abstract and generalizable, forming a cognitive-style

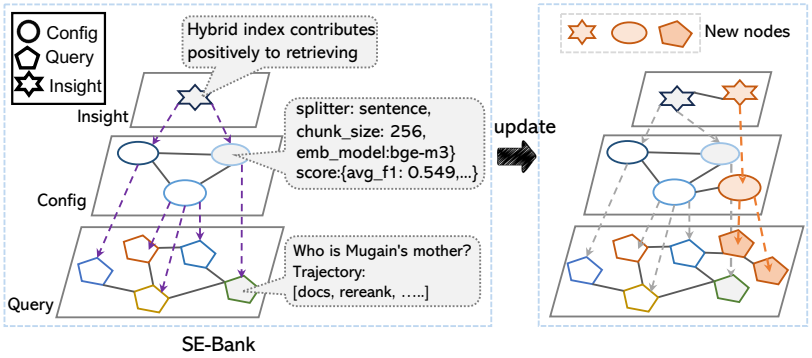


Fig. 6. The overview of SE-Bank.

memory hierarchy that transforms raw tuning experiences into reusable knowledge. This design captures useful information at different levels of granularity, enabling effective abstraction, storage, and reuse of historical knowledge.

An overview of SE-Bank is illustrated in Fig. 6. It consists of three interconnected layers:

- ✱ **Query Graph**, which represents individual queries and preserves their fine-grained execution traces.
- ✱ **Configuration Graph**, where each node corresponds to a concrete RAG configuration and stores the performance behavior of queries under that configuration.
- ✱ **Insight Graph,** which captures distilled knowledge and performance patterns from historical tuning trajectories, where each node corresponds to reusable textual insights.

We formally model SE-Bank as a three-layer hierarchical graph $\mathcal{M} = \{\mathcal{G}^{(Q)}_{\texttt{query}}, \mathcal{G}_{\texttt{Config}}, \mathcal{G}_{\texttt{Insight}}\}$, where each layer captures search experience at a distinct level of abstraction.

✱ **Query Graph.** For a query set Q, we define the *query graph* $\mathcal{G}^{(Q)}_{\texttt{query}} = (\mathcal{V}_Q, \mathcal{E}_Q)$, where $\mathcal{V}_Q = \{(q_i, \mathcal{T}_i, r_i)\}_{i=1}^{|Q|}$. Here, each vertex corresponds to a query $q_i \in Q$, together with its execution record: $\mathcal{T}_i$ denotes the RAG execution trace of q_i, including the retrieved chunks and the generated responses from the LLM; r_i represents the evaluation result of q_j (e.g., $r_i = 1$ if the response is correct and $r_i = 0$ otherwise). The edges $\mathcal{E}_Q \subseteq \mathcal{V}_Q \times \mathcal{V}_Q$ encode semantic similarity relationships between queries. The query graph preserves fine-grained execution behavior and outcome feedback, serving as the experience unit for memory construction.

✱ **Configuration Graph.** The configuration graph, which records explored configurations along with their corresponding performance on previously evaluated query sets, is defined as follows:

$$\mathcal{G}_{\texttt{Config}} = (\mathcal{V}_C, \mathcal{E}_C) = \left(\{\theta_i, \mathcal{G}^{(Q_i)}_{\texttt{query}}, f_{Q_i}(\theta_i)\}_{i=1}^{|\mathcal{V}_C|}, \mathcal{E}_C\right), \tag{9}$$

where $\mathcal{V}_C$=$\{c_i\}$ is the node set, node c_i=$\{\theta_i, \mathcal{G}^{(Q_i)}_{\texttt{query}}, f_{Q_i}(\theta_i)\}$, and the edges $\mathcal{E}_C \subseteq \mathcal{V}_C \times \mathcal{V}_C$ encode semantic relationships between configurations.

The edges between the query layer and the configuration layer record the execution history: they tell us which configuration was applied to which query, along with the observed performance, such as the F1-score or accuracy.

✱ **Insight Graph.** The highest-level insight graph is defined as:

$$\mathcal{G}_{\texttt{Insight}} = (\mathcal{V}_I, \mathcal{E}_I) = (\underbrace{\langle \Delta_i, \Omega_i \rangle}_{\lambda_i}{}_{i=1}^{|\mathcal{V}_I|}, \mathcal{E}_I), \tag{10}$$

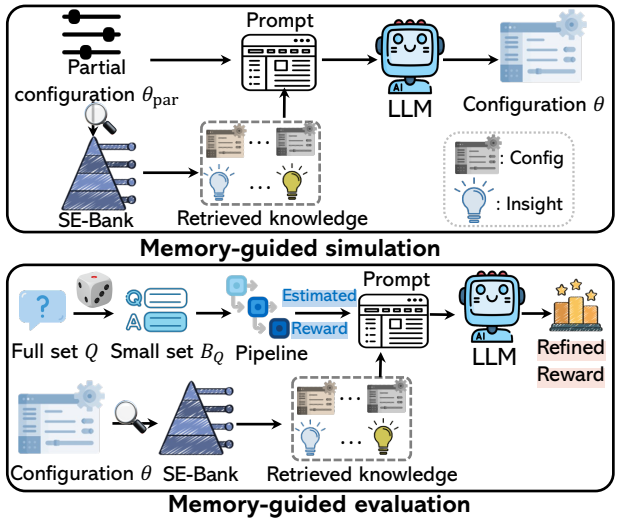


Fig. 7. Overview of memory-guided simulation and evaluation.

where the node set $\mathcal{V}_I=\{\lambda_i\}$ represents distilled insights, each node λ_i is composed of the insight content Δ_i and the set of supporting configurations $\Omega_i \subseteq \mathcal{V}_C$. The edges $\mathcal{E}_I \subseteq \mathcal{V}_I \times \mathcal{V}_I \times \mathcal{V}_C$, forming hyper-connections.

▸ **Construction of** SE-Bank. The SE-Bank is built incrementally, mirroring the continuous learning process of humans. During RAG tuning, when a new configuration θ is evaluated on a selected set of queries (Q_i), both the configuration graph and the query graph are updated accordingly. In the configuration graph, θ is connected to its similar configurations, capturing relationships among parameter settings. Finally, high-level insights are distilled from configurations and queries by prompting the LLM. The prompt is illustrated in the supplementary material [112]. As shown in Fig. 6, the orange nodes represent newly added entries, depicting the process of updating the memory. Through this hierarchical design, SE-Bank bridges fine-grained execution feedback and high-level tuning strategies. The Query Graph preserves detailed experiences, the Configuration Graph summarizes performance behavior, and the Insight Graph abstracts reusable knowledge. This multi-layer structure directly supports the memory-guided simulation and evaluation mechanisms introduced in the following subsection, allowing MG-MCTS to replace blind random rollouts with informed exploration driven by accumulated tuning experience.

5.3 The MG-MCTS Algorithm

In this subsection, we present our carefully designed hierarchical memory-guided MCTS algorithm, MG-MCTS, which aims to overcome the limitations of directly applying MCTS to RAG tuning.

▸ **The key idea of** MG-MCTS. The main idea of MG-MCTS is to leverage reusable knowledge stored in SE-Bank to guide the entire RAG tuning process. It consists of two key modules: *1) memory-guided simulation* and *2) memory-guided evaluation*, designed to improve exploration efficiency and reduce expensive configuration evaluation. Specifically, in *memory-guided simulation* module, we replace random rollouts in MCTS with experience-driven guidance by reusing knowledge distilled from historical tuning trajectories. It leverages the knowledge stored in SE-Bank to prompt the LLM for completing partial configurations. In the *memory-guided evaluation module*, to further reduce the tuning cost of each MCTS iteration, we evaluate configurations using only a small randomly

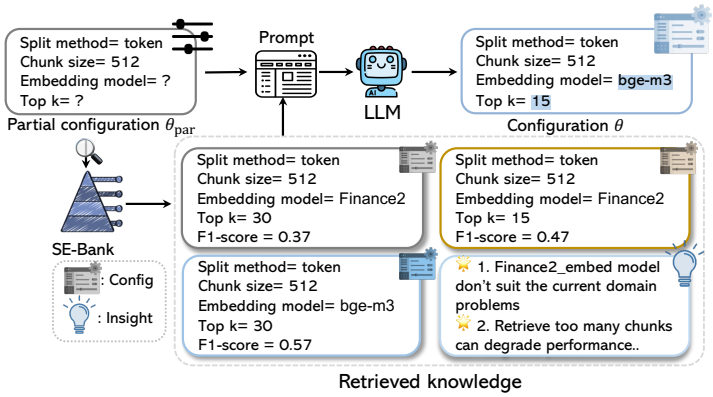


Fig. 8. Illustration of memory-guided simulation.

sampled subset of queries and then exploit historical experiences and distilled insights in SE-Bank to refine this approximate reward. In Fig. 7, we show the overview of these two components.

▸ **Memory-guided simulation.** Instead of relying on random rollouts as MCTS does, MG-MCTS uses memory-guided simulation. Given a partial configuration θ_{par}, it first retrieves top-k similar configurations from the configuration graph $\mathcal{G}_{\texttt{Config}}$ and then collects the corresponding insights from the insight graph $\mathcal{G}_{\texttt{Insight}}$ through the edges connecting the two graphs. By analyzing these similar configurations and their evaluated results, together with high-level insights, we prompt the LLM to fill in the remaining $m - |\theta_{\text{par}}|$ parameters, where the prompt is shown in the supplementary material. By analyzing retrieved historical configurations and their associated insights, LLMs infer effective and ineffective parameter choices to guide configuration completion. This experience-driven rollout replaces random rollouts with informed simulation, while the textual insights merely capture coarse-grained feedback, enabling the model to propose more plausible configurations and effectively address Limitation (L2).

Example 3. *In Fig. 8, we show the process of memory-guided simulation. When a partial configuration is provided, the system retrieves three similar past configurations along with a related insight indicating that the* `finance_2` *embedding model is unsuitable for the current domain and that overly large retrieval sizes can degrade performance. Guided by this knowledge, the LLM selects* `bge-m3` *as the embedding model and sets the retrieval top-k to 15, resulting in a more effective completed configuration.*

▸ **Memory-guided evaluation.** To further reduce the tuning cost at each MCTS iteration, we introduce a memory-guided evaluation mechanism inspired by stochastic gradient descent (SGD) [3], which approximates full objectives using small mini-batches. Specifically, given a configuration θ and a query set Q, instead of evaluating θ on all queries, we first randomly sample a subset $B_Q \subset Q$ to obtain a partial score $f_{B_Q}(\theta)$. Afterwards, to compensate for the unobserved queries, we retrieve the top-k most similar historical configurations of θ from the configuration graph $\mathcal{G}_{\texttt{Config}}$ and leverage their recorded evaluation results and corresponding insights stored in SE-Bank to refine this partial score, yielding an enhanced estimate $f'_{B_Q}(\theta)$. The key insight is that for the remaining queries $Q \setminus B_Q$, their performance under a new configuration θ can be estimated using historical evaluation results together with corresponding insights that capture effective and ineffective parameter choices stored in SE-Bank, enabling experience-based approximation without explicit re-evaluation. Thus, $f'_{B_Q}(\theta)$ provides an efficient yet accurate approximation of the full evaluation $f_Q(\theta)$.

Algorithm 3: MG-MCTS

Input: Tuning iterations T, and dataset info $D_{\texttt{info}}$
Output: Best configuration found

1 Initialize MCTS tree root v_0 and SE-Bank $\mathcal{M}$;
2 **foreach** $t \leftarrow 1, 2, 3, \cdots, T$ **do**
3 $\quad v_t \leftarrow \texttt{Select}(v_0)$;
4 $\quad (v'_t, a_t) \leftarrow \texttt{Expand}(v_t)$;
5 $\quad \theta_t \leftarrow$ run memory-guided simulation via $\mathcal{M}$;
6 $\quad \widehat{R}(v_t, a_t) \leftarrow$ run memory-guided evaluation via $\mathcal{M}$;
7 $\quad \mathcal{M} \leftarrow$ update the memory via v'_t, and $\widehat{R}(v_t, a_t)$;
8 **return** configuration with maximum reward;

By integrating these two memory-guided modules into the MCTS-based RAG tuning process, we obtain the MG-MCTS algorithm, as shown in Algorithm 3. In particular, when a new state v'_t (i.e., a partial configuration) is expanded during the MCTS process (lines 3–4), SE-Bank retrieves similar configurations from the configuration graph $\mathcal{G}_{\texttt{Config}}$, then traverses upward (configuration $\Rightarrow$ insight) to exploit high-level knowledge and downward (configuration $\Rightarrow$ query) to retrieve fine-grained execution records. These retrieved configurations and insights are then used in the memory-guided simulation and evaluation (lines 5-6) After each iteration, all three layers are updated with new insights, enriched query records, and refined configuration relationships, ensuring continuous accumulation and reuse of tuning knowledge (line 7).

Remark. We understand that in some practical settings, the gold data may not be available. To tackle this issue, there are several possible methods to obtain the reward signals, to name a few: 1) We can design a human-in-the-loop feedback strategy by incorporating expert feedback, such as acceptance, rejection, or refinement, as high-quality reward signals. 2) We can leverage powerful LLMs (e.g., ChatGPT 5.2) as proxy evaluators to assess answer faithfulness, relevance, or confidence, producing approximate scalar reward signals. Notably, our memory-guided evaluation module, following the same principle, has empirically demonstrated the feasibility of such proxy-based feedback. 3) We can also strategically combine the above two methods, since human feedback is expensive and LLMs may not be fully reliable. For example, for simple queries, we directly rely on LLM-based evaluation, while for complicated cases, we escalate the evaluation to human experts for verification. By doing so, we believe that our approach will remain effective for RAG tuning in real-world deployments.

6 Experiments

We now present the experimental results. Section 6.1 discusses the setup. We discuss the results in Sections 6.2 and 6.3.

6.1 Setup

▸ **Benchmark datasets.** We employ eight real-world datasets to evaluate the performance of graph-based RAG methods. These datasets cover diverse domains, including multi-hop reasoning, scientific inquiry, and medical question answering, ensuring both breadth and challenge in evaluation. Their statistics, including the numbers of tokens and questions as well as their corresponding domains, are summarized in Table 1. Besides, due to resource limitations, for some datasets (e.g., HotpotQA and PopQA), we randomly sample a subset of queries from the full query set for tuning and evaluation, since RAG tuning is computationally expensive and requires repeatedly executing the retrieval

Table 1. Statistics of the datasets used in our experiments.

Dataset	# of Tokens	# of Questions	Domain
MedQA [37]	262,958	1,113	Medicine
FiQA [97]	1,852,155	437	Finance
2WikiMultiHopQA (2Wiki) [26]	640,205	875	Wikipedia
HotpotQA [98]	1,239,838	875	Wikipedia
PopQA [65]	16,638	875	Wikipedia
QuaRTZ [86]	47,641	684	Science
Web Questions (WebQuest) [2]	31,075	1,777	Web pages
ELI5 [17]	716,782	1,317	Social Media

and generation pipeline under different configurations. While this sampling strategy allows us to conduct controlled experiments within a reasonable computational budget, it may not fully capture the complete query workload distribution of the datasets. Exploring the effectiveness of RAG tuning over the entire query set remains an important direction for future work. Detailed descriptions of each dataset are provided in the supplementary material [112].

▸ **Competitors.** We evaluate the following RAG tuning methods:

- `Random`: A naive hyperparameter optimization (HPO) method that uniformly samples configurations at random from the search space.
- `Greedy`: A standard greedy optimizer that tunes one hyperparameter at a time while keeping others fixed, following the execution flow of the RAG pipeline.
- `GreedyE`: An enhanced greedy strategy that prioritizes optimizing the embedding model before tuning the remaining pipeline parameters (e.g., chunk size, top-k).
- `TGreedy`: A variant of greedy search that first optimizes retrieval-related parameters and subsequently tunes the remaining parameters in the pipeline.
- `Grid`: A method that evaluates every possible combination of hyperparameters within a discretized search space.
- `Syftr`: A SOTA method that uses the Tree-structured Parzen Estimator (TPE)-based Bayesian optimization (BO).
- `AutoRAG` [21]: a multi-armed bandit–based method for hyperparameter selection in RAG pipeline;
- `LLM-TPE`: an LLM-enhanced TPE-based Bayesian optimization approach for RAG tuning, inspired by [61].
- `GPBO` [1]: a Gaussian Process–based Bayesian optimization approach for LLM-based system optimization.
- `HAMMER`: our proposed RAG tuning method in Section 5.

We identified several techniques that are partially related to our setting [78, 79]. Specifically, [78] focuses on selecting suitable RAG parameters under resource constraints by prompting LLMs. In our paper, we also evaluated a similar strategy that directly uses an LLM to select parameters for a given query and RAG pipeline (denoted as `LLM-Only`). In addition, the methods in [79] are entirely covered by the benchmark study [73], which is already included in our evaluation.

▸ **Evaluation metrics.** To accurately assess performance across different types of QA tasks, we adopt several standardized evaluation metrics [35, 46, 66]. Specifically, we use F1-score (F1), Accuracy, and Exact Match (EM) to evaluate whether the generated answers are correctly included in or exactly match the ground-truth answers. In addition, we employ Faithfulness [66] and ROUGE-L [54] to assess the factual consistency and textual similarity of the generated answers with respect to the reference answers.

Table 2. Detailed performance comparison of different tuning algorithms across eight datasets, where shaded areas indicate the best results and **orange** marks the second-best results.

Dataset	Metric	Random	Greedy	GreedyE	TGreedy	Grid	Syftr	AutoRAG	LLM-TPE	GPBO	HAMMER
MedQA	F1	50.8	49.9	51.0	50.1	50.6	**51.2**	50.9	47.8	50.2	**54.3** (△6.1%)
	EM	13.9	13.7	13.6	**14.7**	14.6	13.9	12.6	11.6	13.4	**17.2** (△17.0%)
	Accuracy	60.4	**62.4**	61.5	60.5	60.7	61.5	61.8	60.9	60.6	**66.9** (△7.2%)
	ROUGE-L	50.1	49.1	**51.0**	50.8	50.8	50.9	47.1	48.7	50.2	**51.8** (△1.6%)
FiQA	F1	24.8	24.0	22.8	**26.6**	25.2	21.1	24.7	24.1	26.2	**30.3** (△13.9%)
	Faithfulness	54.0	52.5	51.4	48.3	52.3	**56.1**	55.8	48.7	51.5	**62.7** (△11.8%)
	ROUGE-L	**12.3**	10.8	8.7	9.9	11.4	12.0	12.1	10.4	11.2	**13.6** (△10.6%)
2Wiki	F1	38.7	40.7	39.9	42.4	39.9	41.1	41.6	39.6	**42.5**	**47.1** (△10.8%)
	EM	29.6	28.6	29.7	**31.4**	30.1	30.7	31.1	28.9	30.7	**37.7** (△20.1%)
	Accuracy	41.4	44.0	43.2	**46.0**	43.3	44.6	41.9	41.2	44.6	**50.6** (△10.0%)
	ROUGE-L	40.2	38.0	39.1	41.3	41.8	**42.4**	39.8	38.9	42.3	**47.1** (△11.1%)
HotpotQA	F1	49.7	**55.0**	44.1	50.6	50.3	54.4	50.8	54.4	52.1	**63.4** (△15.2%)
	EM	39.2	40.9	38.4	39.0	37.0	39.6	40.4	40.8	**41.7**	**44.4** (△6.5%)
	Accuracy	**58.5**	55.3	55.5	58.1	51.9	56.0	53.2	52.1	50.8	**66.3** (△13.3%)
	Faithfulness	**83.7**	81.4	78.0	78.9	82.4	81.1	73.9	83.6	78.3	**86.9** (△3.8%)
	ROUGE-L	**54.7**	50.7	49.7	48.5	49.0	48.8	53.4	52.1	52.0	**63.3** (△15.7%)
PopQA	F1	52.0	59.8	56.1	62.1	63.6	**65.3**	64.9	58.9	61.2	**69.7** (△6.7%)
	EM	50.0	57.7	55.5	59.9	61.3	**63.1**	60.8	54.2	58.6	**67.4** (△6.8%)
	Accuracy	61.0	62.1	62.5	62.9	61.5	63.2	59.9	**66.8**	61.7	**67.7** (△1.3%)
	ROUGE-L	53.0	58.4	57.3	61.8	64.0	65.1	60.6	**67.3**	62.1	**69.7** (△3.6%)
QuaRTZ	F1	72.5	74.6	74.1	**75.5**	75.1	73.0	71.6	74.6	74.4	**82.1** (△8.7%)
	Accuracy	75.8	78.1	77.6	**79.2**	78.6	75.8	78.3	77.3	77.8	**86.3** (△9.0%)
WebQuest	F1	**18.3**	16.5	18.0	**18.3**	17.8	17.7	18.2	16.5	17.7	**20.2** (△10.4%)
	EM	5.9	5.1	**7.7**	7.2	6.3	6.1	7.6	7.6	7.1	**8.2** (△6.5%)
	Accuracy	28.5	27.2	27.3	**29.2**	27.9	27.6	26.4	26.9	28.6	**31.8** (△8.9%)
Eli5	F1	9.6	10.0	9.8	9.9	10.1	10.1	9.8	10.1	**10.3**	**11.3** (△9.7%)
	Accuracy	26.3	24.2	25.2	26.5	19.8	**27.0**	25.9	26.8	25.7	**29.4** (△8.9%)
	ROUGE-L	7.9	7.5	7.7	8.1	8.2	8.3	**8.5**	7.9	7.9	**9.0** (△5.9%)

▸ **Tuning setting.** For all HPO-based methods, we set the maximum number of tuning iterations to 50. For fairness, our method also performs 50 simulations and evaluations to align with them. For each dataset, we split the query set in an 8:2 ratio, using 80% as the training set for tuning the RAG system and the remaining 20% as the test set for evaluating the performance of each RAG tuning method. Unless otherwise specified, all experimental results reported in this paper are obtained on the test set. Note that our study does not focus on tuning the generation model itself; instead, we evaluate all methods across different LLMs. Specifically, we use Qwen2.5-7B [96] as the default model, as it demonstrates remarkable performance among open-source models of similar scale. In addition, we also test DeepSeek-R1-32B [25], Qwen2.5-72B [96], and GPT-4o-mini [72], covering reasoning-oriented models, different model sizes, and a representative model from the GPT family. Note that all experiments are conducted on four NVIDIA A5000 GPUs, each equipped with 24 GB of memory. For the models Qwen2.5-72B and GPT-4o-mini, we used external API services. All methods share the same configuration space, which includes 13 tunable parameters (i.e., $m = |\Theta| = 13$), covering various aspects such as chunk splitting strategies, chunk sizes, embedding models, and

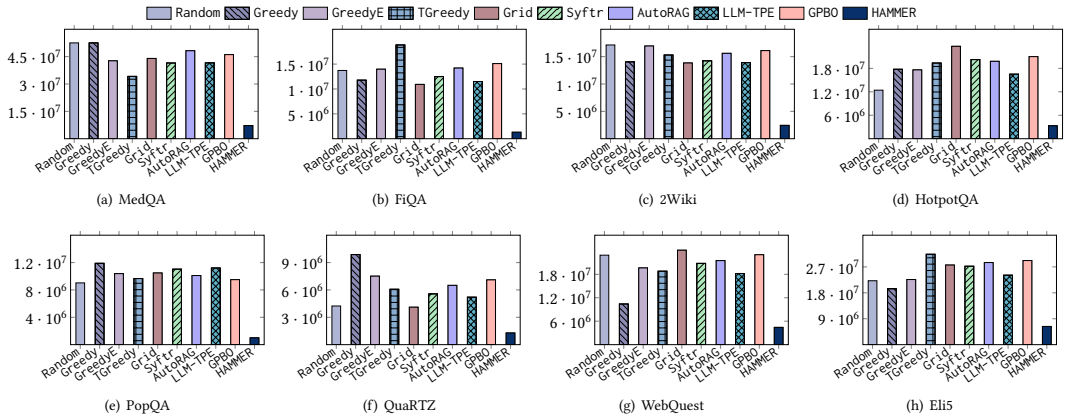


Fig. 9. Token costs of RAG tuning methods.

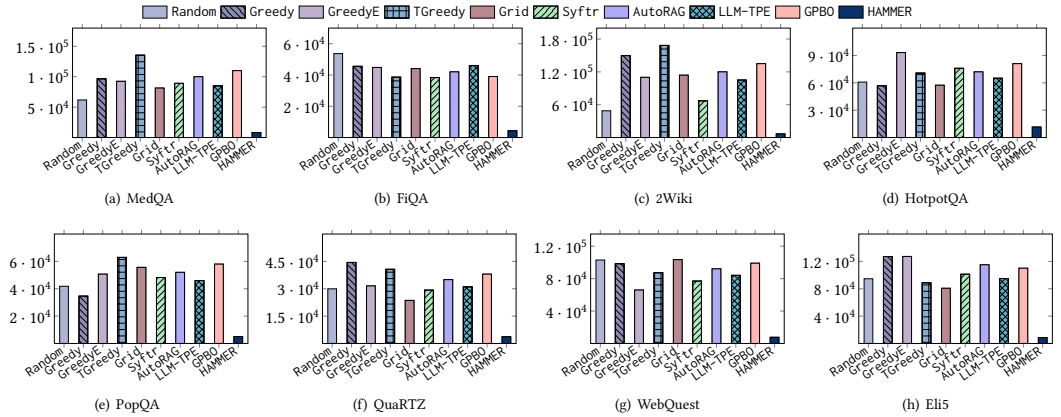


Fig. 10. Time costs of RAG tuning methods.

Table 3. Comparison with directly using LLM methods.

Model	FiQA			2Wiki			HotpotQA				QuaRTZ	
	F1	Faithfulness	ROUGE-L	F1	EM	Accuracy	F1	EM	Accuracy	Faithfulness	F1	Accuracy
MCTS-LLM	27.9	56.4	10.7	45.0	32.0	49.4	52.6	42.1	58.9	85.2	80.2	84.4
LLM-only	22.3	52.1	8.4	39.5	28.8	43.3	41.9	37.6	61.7	78.0	75.5	79.1
HAMMER	**30.3**	**62.7**	**13.6**	**47.1**	**37.7**	**50.6**	**63.4**	**44.4**	**66.3**	**86.9**	**82.1**	**86.3**

reranking models. The detailed descriptions of each parameter and its corresponding domain are provided in our supplementary material.

6.2 Comparison with the existing methods

▸ **Exp.1. Overall performance.** We report the metric values of all methods on various QA tasks in Table 2. We can make the following observations and analyses: 1) Our system HAMMER significantly outperforms all baseline methods, achieving the best performance across different task types and domain-specific corpora under all evaluation metrics. Specifically, HAMMER improves

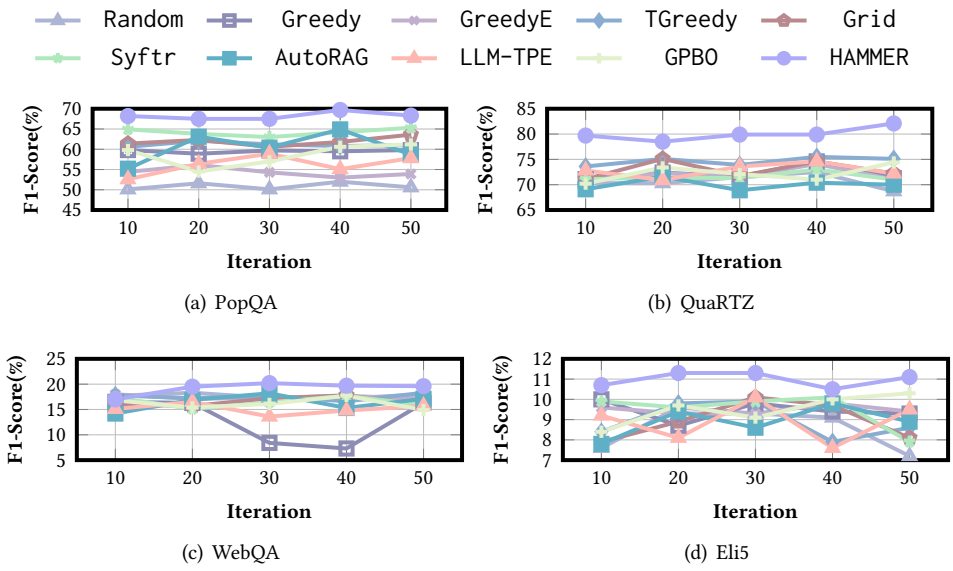


Fig. 11. F1-Score variance with iterations.

F1-score by up to 15.2%, accuracy by 13.3%, and exact match by 20.0%, respectively, demonstrating its superior effectiveness in RAG tuning. This improvement stems from HAMMER's ability to leverage the knowledge stored in the hierarchical graph memory SE-Bank, which provides informative, experience-driven guidance throughout the tuning process. While existing HPO-based methods overlook such historical experience. 2) Typically, the TGreedy and Syftr achieve the best performance among all compared methods, while the remaining methods exhibit mixed results but consistently underperform compared to HAMMER. The LLM-TPE does not always perform better than Syftr, which also aligns with our analysis in Section 2.4. This demonstrates that our query selection module effectively identifies the most representative queries, and the memory-guided evaluation provides sufficient guidance for reliable performance assessment.

▸ **Exp.2. Token costs and time costs.** In this experiment, we report the token and time costs of each RAG tuning method in Figs. 9 and 10, respectively. As shown, our system HAMMER reduces both tuning time and token consumption by up to 9× compared to existing HPO-based methods. For example, on the PopQA dataset, HAMMER only takes 996,188 tokens, while the existing methods at least take 9,028,325 tokens. This improvement stems from two key factors: 1) the query selection module significantly lowers the number of queries involved in each tuning iteration, thereby reducing computational cost, and 2) the memory-guided evaluation module further decreases the cost by evaluating only a small subset of representative queries instead of the entire query set. Notably, despite the substantially lower cost, HAMMER still achieves the best tuning performance, as discussed earlier. The results are provided in the supplementary material due to space constraints.

▸ **Exp.3. Effect of T.** In Fig. 11, we report the performance of advanced RAG pipelines on the testing set using configurations obtained by different tuning methods after T tuning iterations across four datasets, where T ranges from 10 to 50, while more results are provided in the supplementary material. This indicates that HAMMER is able to effectively accumulate useful tuning experience and progressively identify better configurations, leading to stronger generalization to unseen queries. In contrast, the other methods show noticeable fluctuations and even performance degradation as T grows, suggesting unstable search trajectories and a tendency to overfit intermediate evaluations.

Table 4. Performance comparison of HAMMER variants.

Model	FiQA	2Wiki		HotpotQA		QuaRTZ	
	F1	EM	Accuracy	EM	Accuracy	F1	Accuracy
HAMMER-Sim	21.5	**31.9**	**46.6**	39.2	**58.4**	**77.8**	**83.5**
HAMMER-Eva	**27.4**	30.5	45.2	**40.2**	57.0	75.4	81.5
HAMMER	**30.3**	**37.7**	**50.6**	**44.4**	**66.3**	**82.1**	**86.3**

Table 5. Performance comparison of variant models.

Model	FiQA	2Wiki		HotpotQA		QuaRTZ	
	F1	EM	Accuracy	EM	Accuracy	F1	Accuracy
Qwen2.5-7B	**30.3**	37.7	50.6	44.4	66.3	82.1	86.3
DeepSeek-R1-32B	29.6	37.5	54.5	48.4	65.5	88.0	**94.1**
Qwen-2.5-72B	28.9	**44.1**	**62.9**	**50.6**	**73.2**	**89.5**	**94.9**
GPT-4o-mini	**31.4**	**42.6**	**58.5**	**49.7**	**69.8**	**88.5**	**94.9**

We also note that on the training set, all methods exhibit an overall increasing trend as tuning progresses. However, from a practical perspective, the performance on the testing set is more critical, since real-world RAG systems must generalize to unseen user queries. From this view, HAMMER demonstrates clear superiority in robustness and generalization.

▶ **Exp.4.Comparison with directly using LLM methods.** In this experiment, we aim to demonstrate that directly using LLMs for RAG tuning is ineffective, primarily due to the lack of rich external knowledge to guide the LLM's decision process, unlike database knob tuning, which can leverage comprehensive DBMS manuals, as analyzed in Section 2.4. Specifically, we design two variants: MCTS-LLM, which replaces random simulations with LLM-guided rollouts and employs the LLM to complete partial configurations; and LLM-only, which relies solely on the LLM to select the optimal configuration based on corpus and query information. As shown in Table 3, both MCTS-LLM and LLM-only exhibit inferior performance, lower than our system HAMMER. This result supports our claim that directly applying LLMs to RAG tuning is unreliable and suboptimal.

6.3 Detailed analysis of HAMMER

In this subsection, we provide an in-depth analysis of why and how HAMMER achieves superior performance, from four perspectives.

▶ **Exp.1. Ablation study.** In this experiment, we conduct an ablation study for HAMMER by progressively removing its key memory-guided components. Specifically, we create two variants: HAMMER-Sim, which removes the memory-guided simulation component, and HAMMER-Eva, which removes the memory-guided evaluation component. The results, shown in Table 4, reveal that: 1) Both HAMMER-Sim and HAMMER-Eva perform worse than the full HAMMER, confirming that both components contribute effectively. The former replaces the random rollout with memory-guided simulation, enabling more informed exploration, while the latter refines the evaluation process using insights from SE-Bank, ensuring more reliable feedback. 2) Notably, HAMMER-Sim, which still evaluates configurations using the entire query set, performs worse than HAMMER. This indicates that our memory-guided evaluation not only approximates the full evaluation effectively but also refines it through LLM-based correction, achieving even higher reliability and efficiency. The results on the other metrics are shown in the supplementary material.

Table 6. Performance of HAMMER under different query selection strategies.

Method	MedQA			FiQA	2Wiki			HotpotQA			PopQA			QuaRTZ		WebQuest			Eli5	
	F1	EM	Acc	F1	F1	EM	Acc	F1	EM	Acc	F1	EM	Acc	F1	Acc	F1	EM	Acc	F1	Acc
HAMMER (Coreset)	52.6	15.8	63.5	28.1	44.5	35.3	48.0	59.5	43.3	60.1	66.2	64.1	64.5	78.8	82.5	18.8	7.5	29.6	10.5	27.2
HAMMER (Query)	**54.3**	**17.2**	**66.9**	**30.3**	**47.1**	**37.7**	**50.6**	**63.4**	**44.4**	**66.3**	**69.7**	**67.4**	**67.7**	**82.1**	**86.3**	**20.2**	**8.2**	**31.8**	**11.3**	**29.4**

Table 7. Performance comparison between HAMMER and HAMMER-Full across all datasets.

Method	MedQA			FiQA	2Wiki			HotpotQA			PopQA			QuaRTZ		WebQuest			Eli5	
	F1	EM	Acc	F1	F1	EM	Acc	F1	EM	Acc	F1	EM	Acc	F1	Acc	F1	EM	Acc	F1	Acc
HAMMER	54.3	17.2	66.9	30.3	47.1	37.7	50.6	63.4	44.4	66.3	69.7	67.4	67.7	82.1	86.3	20.2	8.2	31.8	11.3	29.4
HAMMER-Full	**56.9**	**18.7**	**67.8**	**31.1**	**49.7**	**39.3**	**51.7**	**64.4**	**45.3**	**67.2**	**70.8**	**68.3**	**68.6**	**83.2**	**87.3**	**20.7**	**8.6**	**32.4**	**11.8**	**29.9**

▶ **Exp.2. Effect of LLM backbones.** We investigate the performance of HAMMER with different LLM backbones. The results are shown in the Table 5, and we can see that generally, the more powerful model typically generates higher performance. The results for other evaluation metrics are provided in the supplementary material. We note that the generation model itself can also be treated as a tunable parameter, which is a direction explored in prior studies on model selection [29]. However, we leave this as future work, as incorporating multiple large models into the tuning configuration space would significantly increase computational cost.

▶ **Exp.3. Effect of query selection.** In this experiment, we compare HAMMER using queries selected by coreset [6] and by our strategy, denoted as HAMMER (Coreset) and HAMMER (Query). As shown in Tables 6 and 8, we identify two main advantages of our method. 1) Our proposed query selection method (Greedy) is significantly faster than the SOTA coreset selection approach [6] (Coreset). On the 2Wiki and HotpotQA datasets, selecting 300 queries from the full set requires 14.68 s and 17.91 s for Coreset, whereas Greedy only takes 3.02 s and 4.65 s, respectively (excluding similarity preprocessing time). 2) The queries selected by our method lead to better tuning performance than those chosen by the coreset. For example, HAMMER (Query) achieves up to 9.33% higher EM and 7.83% higher F1-score compared to HAMMER (Coreset). When query selection is directly applied to existing HPO baselines, their F1-scores drop noticeably (see the supplementary material), whereas our method does not suffer from such degradation. To investigate this effect, we introduce HAMMER-Full, which performs tuning using the full query set. As shown in Table 7, HAMMER-Full yields only marginal F1-score improvements over our method while incurring approximately 9× higher tuning cost in both token consumption and runtime. This advantage stems from three key designs: MCTS explicitly models parameter dependencies, knowledge in SE-Bank guides effective exploration of high-quality configurations, and memory-guided evaluation enables reliable optimization using only a small subset of queries by approximating remaining evaluations from historical results and distilled insights.

▶ **Exp.4. Case study.** To show the deployment process more clearly, we present its general workflow in Fig. 12(a). Specifically, given the data in a specific domain, the practitioner first selects a domain-appropriate RAG pipeline composed of multiple modules (e.g., chunking, retrieval, and reranking), each exposing tunable parameters (e.g., chunk size and number k of returned results). Then, the practitioner uses HAMMER to identify the optimal configuration of these parameters. Finally, we deploy the RAG pipeline with the optimal configuration.

Besides, we have provided a complete use case in Fig. 12(b), where the RAG pipeline is Advanced RAG, which has six modules. The tunable parameters across the six modules, together with their candidate values, collectively define the configuration space (e.g., chunk size in {128, 256, 512, 1024}

Table 8. Running time of Coreset and Greedy.

Dataset	Selecting time (s)		Reduction ($\frac{\texttt{Coreset}}{\texttt{Greedy}}$)
	Coreset	Greedy	
MedQA	22.05	4.92	4.48×
FiQA	9.64	2.47	3.90×
2Wiki	14.68	3.02	4.86×
HotpotQA	17.91	4.65	3.85×
PopQA	16.67	4.20	3.97×
QuaRTZ	12.72	3.55	3.58×

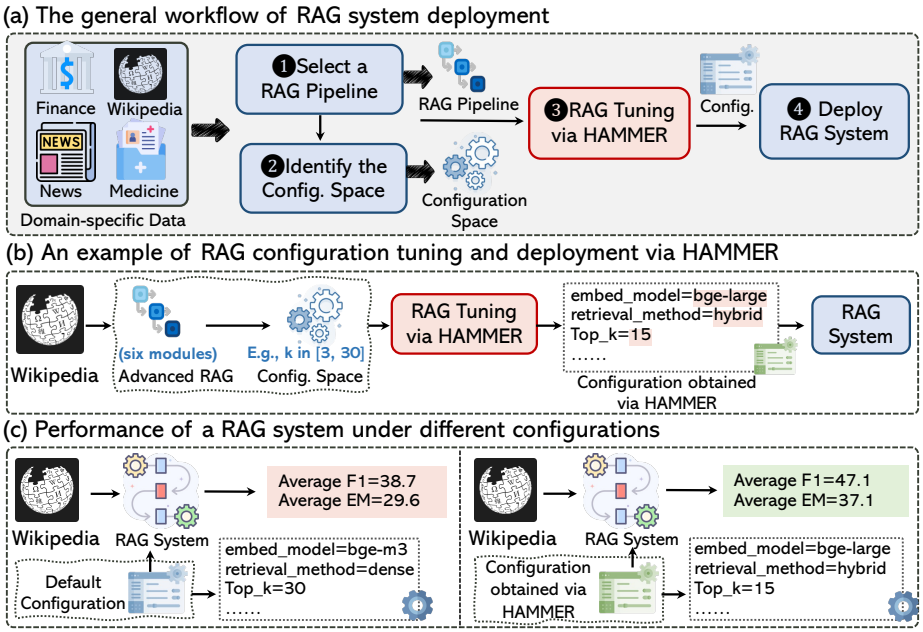


Fig. 12. Illustrating how practitioners use HAMMER to tune and deploy a domain-specific RAG system.

and $k \in [3, 30]$). Given tasks from the Wikipedia data, the advanced RAG pipeline and its corresponding configuration space, HAMMER is used to automatically identify an optimal configuration that maximizes end-to-end RAG performance. Once identified, the tuned RAG pipeline can be deployed and reused to serve future queries from the same domain. This is because queries within the same domain typically exhibit similar characteristics, which is a common phenomenon in real-world domain-specific QA settings. In Fig. 12(c), we further demonstrate that the tuned system always outperforms default configurations, highlighting its practical utility.

7 Related works

In this section, we first review related work on Retrieval-Augmented Generation (RAG) methods. We then discuss hyperparameter optimization (HPO) techniques as applied in database systems.

• **RAG methods.** RAG has been proven to be very effective in migrating the "hallucination" of LLMs [4, 8–10, 36, 82, 91, 100]. The key idea of the RAG technique relies on retrieving query-relevant information from external knowledge bases. Recently, a number of agentic RAG approaches [28,

59, 80, 101] have explored automatic RAG pipeline construction using agent-based or tree-search strategies. These methods primarily focus on dynamically composing or restructuring pipeline components for specific datasets. In contrast, our work targets configuration optimization once a pipeline structure is fixed, providing a universal solution that can be readily applied to such agentic RAG systems. While agentic RAG approaches also employ MCTS, we note that MCTS itself is a general search paradigm that has been widely adopted in many sequential optimization tasks, such as automatic rule using for SQL rewriting [108]. In contrast, the key contributions of our work lie in the integration of query selection, structured memory design, and memory-guided MCTS to enable efficient and stable RAG configuration tuning. Motivated by the rich developer knowledge in database forums, recent studies [8, 11, 11, 18, 24, 42, 52, 53, 55–58, 58, 75, 81, 83, 85, 95, 107, 109, 110] have increasingly leveraged RAG techniques to enhance database performance and optimization.

• **HPO in database.** HPO has become the backbone of modern AutoML and database knob tuning systems [39]. Existing AutoML-based methods employ intelligent search strategies and heuristics to efficiently explore configuration spaces over ad-hoc datasets [44]. Representative techniques include Bayesian Optimization (BO) [33], and directed search [89, 93]. Recently, LLM-driven AutoML approaches have been proposed to further enhance configuration exploration and transferability [7, 61, 64, 77, 87]. For example, LLAMBO [61] integrates LLMs into the BO process, which enables the LLM to iteratively propose and assess promising configurations conditioned on historical evaluations.

In knob tuning, the most representative HPO-based approaches are BO–based methods [5, 15, 40, 88, 103]. Specifically, these methods iteratively explore the configuration space by approximating the relationship between knobs and DBMS performance. Recently, LLM-assisted tuning systems have also been explored, leveraging the reasoning capabilities and domain knowledge of LLMs, such as information from DBMS manuals and community forums, to predict promising configurations, interpret workload behaviors, and generate adaptive tuning rules [24, 31, 42, 43, 49, 84]. As discussed earlier, the successful applications of LLM-based HPO methods in knob tuning cannot be directly transferred to RAG tuning.

8 Conclusions

In this paper, we investigate the problem of Retrieval-Augmented Generation (RAG) tuning. Existing solutions mainly rely on hyperparameter optimization (HPO) methods, which overlook parameter dependencies, provide unreliable tuning guidance, and incur substantial tuning costs. To address these issues, we propose HAMMER, a hierarchical memory-guided Monte Carlo Tree Search (MCTS) system for RAG tuning. By distilling knowledge from historical tuning records and leveraging it to guide future optimization, HAMMER enables more reliable and knowledge-driven tuning. Extensive experiments on eight real-world datasets show that HAMMER improves exact match by up to 20.0% and F1-score by 15.2%, while reducing both tuning time and token consumption by up to 9×. In future work, we aim to extend the core ideas of our approach to other RAG paradigms, such as graph-based RAG or agentic RAG systems and to explore tuning strategies that do not rely on gold data.

Acknowledgments

This work was supported by the 1+1+1 CUHK-CUHK(SZ)-GDSTC Joint Collaboration Fund under Grant 2025A0505000045. We also thank the anonymous reviewers for their constructive comments, which helped improve this paper.

References

[1] Matthew Barker, Andrew Bell, Evan Thomas, et al. 2025. Faster, Cheaper, Better: Multi-Objective Hyperparameter Optimization for LLM and RAG Systems. In *ICLR 2025 Workshop on Foundation Models in the Wild*.

[2] Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. 2013. Semantic Parsing on Freebase from Question-Answer Pairs. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 1533–1544. https://aclanthology.org/D13-1160/

[3] Léon Bottou. 2012. Stochastic gradient descent tricks. In *Neural networks: tricks of the trade: second edition*. Springer, 421–436.

[4] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33 (2020), 1877–1901.

[5] Stefano Cereda, Stefano Valladares, Paolo Cremonesi, and Stefano Doni. 2021. CGPTuner: a Contextual Gaussian Process Bandit Approach for the Automatic Tuning of IT Configurations Under Varying Workload Conditions. *Proc. VLDB Endow.* 14, 8 (2021), 1401–1413. doi:10.14778/3457390.3457404

[6] Chengliang Chai, Jiayi Wang, Nan Tang, Ye Yuan, Jiabin Liu, Yuhao Deng, and Guoren Wang. 2023. Efficient Coreset Selection with Cluster-based Methods. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 167–178.

[7] Guojin Chen, Keren Zhu, Seunggeun Kim, Hanqing Zhu, Yao Lai, Bei Yu, and David Z Pan. 2024. LLM-enhanced Bayesian optimization for efficient analog layout constraint generation. *arXiv preprint arXiv:2406.05250* (2024).

[8] Sibei Chen, Ju Fan, Bin Wu, Nan Tang, Chao Deng, Pengyi Wang, Ye Li, Jian Tan, Feifei Li, Jingren Zhou, et al. 2024. Automatic Database Configuration Debugging using Retrieval-Augmented Language Models. *arXiv preprint arXiv:2412.07548* (2024).

[9] Sibei Chen, Yeye He, Weiwei Cui, Ju Fan, Song Ge, Haidong Zhang, Dongmei Zhang, and Surajit Chaudhuri. 2024. Auto-Formula: Recommend Formulas in Spreadsheets using Contrastive Learning for Table Representations. *Proceedings of the ACM on Management of Data* 2, 3 (2024), 1–27.

[10] Sibei Chen, Nan Tang, Ju Fan, Xuemi Yan, Chengliang Chai, Guoliang Li, and Xiaoyong Du. 2023. Haipipe: Combining human-generated and machine-generated pipelines for data preparation. *Proceedings of the ACM on Management of Data* 1, 1 (2023), 1–26.

[11] Zui Chen, Lei Cao, Sam Madden, Tim Kraska, Zeyuan Shang, Ju Fan, Nan Tang, Zihui Gu, Chunwei Liu, and Michael Cafarella. 2023. SEED: Domain-Specific Data Curation With Large Language Models. *arXiv e-prints* (2023), arXiv–2310.

[12] Alexander Conway, Debadeepta Dey, Stefan Hackmann, Matthew Hausknecht, Michael Schmidt, Mark Steadman, and Nick Volynets. 2025. syftr: Pareto-Optimal Generative AI. *arXiv preprint arXiv:2505.20266* (2025).

[13] Jacob Devlin. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

[14] Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Jingyuan Ma, Rui Li, Heming Xia, Jingjing Xu, Zhiyong Wu, Tianyu Liu, et al. 2022. A survey on in-context learning. *arXiv preprint arXiv:2301.00234* (2022).

[15] Songyun Duan, Vamsidhar Thummala, and Shivnath Babu. 2009. Tuning database configuration parameters with ituned. *Proceedings of the VLDB Endowment* 2, 1 (2009), 1246–1257.

[16] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783* (2024).

[17] Angela Fan, Yacine Jernite, Ethan Perez, David Grangier, Jason Weston, and Michael Auli. 2019. ELI5: Long Form Question Answering. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (ACL), 2019*. https://aclanthology.org/P19-1346/ arXiv:1907.09190.

[18] Ju Fan, Zihui Gu, Songyue Zhang, Yuxin Zhang, Zui Chen, Lei Cao, Guoliang Li, Samuel Madden, Xiaoyong Du, and Nan Tang. 2024. Combining small language models and large language models for zero-shot nl2sql. *Proceedings of the VLDB Endowment* 17, 11 (2024), 2750–2763.

[19] Meihao Fan, Xiaoyue Han, Ju Fan, Chengliang Chai, Nan Tang, Guoliang Li, and Xiaoyong Du. 2024. Cost-effective in-context learning for entity resolution: A design space exploration. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*. IEEE, 3696–3709.

[20] 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*. 6491–6501.

[21] Jia Fu, Xiaoting Qin, Fangkai Yang, et al. 2024. Autorag-hp: Automatic online hyper-parameter tuning for retrieval-augmented generation. *arXiv preprint arXiv:2406.19251* (2024).

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

[23] Aashish Ghimire, James Prather, and John Edwards. 2024. Generative AI in Education: A Study of Educators' Awareness, Sentiments, and Influencing Factors. *arXiv preprint arXiv:2403.15586* (2024).

[24] Victor Giannakouris and Immanuel Trummer. 2025. λ-tune: Harnessing large language models for automated database system tuning. *Proceedings of the ACM on Management of Data* 3, 1 (2025), 1–26.

[25] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948* (2025).

[26] 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 (COLING)*. 6609–6625. doi:10.18653/v1/2020.coling-main.580

[27] Yucheng Hu and Yuxing Lu. 2024. Rag and rau: A survey on retrieval-augmented language model in natural language processing. *arXiv preprint arXiv:2404.19543* (2024).

[28] Yunhai Hu, Yilun Zhao, Chen Zhao, and Arman Cohan. 2025. Mcts-rag: Enhancing retrieval-augmented generation with monte carlo tree search. *arXiv preprint arXiv:2503.20757* (2025).

[29] Keke Huang, Yimin Shi, Dujian Ding, Yifei Li, Yang Fei, Laks Lakshmanan, and Xiaokui Xiao. 2025. Thriftllm: On cost-effective selection of large language models for classification queries. *arXiv preprint arXiv:2501.04901* (2025).

[30] Lei Huang, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Haotian Wang, Qianglong Chen, Weihua Peng, Xiaocheng Feng, Bing Qin, et al. 2023. A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. *arXiv preprint arXiv:2311.05232* (2023).

[31] Xinmei Huang, Haoyang Li, Jing Zhang, Xinxin Zhao, Zhiming Yao, Yiyan Li, Tieying Zhang, Jianjun Chen, Hong Chen, and Cuiping Li. 2024. E2etune: End-to-end knob tuning via fine-tuned generative language model. *arXiv preprint arXiv:2404.11581* (2024).

[32] Yizheng Huang and Jimmy Huang. 2024. A Survey on Retrieval-Augmented Text Generation for Large Language Models. *arXiv preprint arXiv:2404.10981* (2024).

[33] Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. 2011. Sequential model-based optimization for general algorithm configuration. In *International conference on learning and intelligent optimization*. Springer, 507–523.

[34] Rishabh K Iyer and Jeff A Bilmes. 2013. Submodular optimization with submodular cover and submodular knapsack constraints. *Advances in neural information processing systems* 26 (2013).

[35] Gautier Izacard and Edouard Grave. 2020. Leveraging passage retrieval with generative models for open domain question answering. *arXiv preprint arXiv:2007.01282* (2020).

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

[37] Di Jin, Eileen Pan, Nassim Oufattole, Wei-Hung Weng, Hanyi Fang, and Peter Szolovits. 2021. What disease does this patient have? a large-scale open domain question answering dataset from medical exams. *Applied Sciences* 11, 14 (2021), 6421.

[38] Levente Kocsis and Csaba Szepesvári. 2006. Bandit based monte-carlo planning. In *European conference on machine learning*. Springer, 282–293.

[39] Brian Kroth, Sergiy Matusevych, and Yiwen Zhu. 2025. Autotuning Systems: Techniques, Challenges, and Opportunities. In *Companion of the 2025 International Conference on Management of Data*. 821–828.

[40] Mayuresh Kunjir and Shivnath Babu. 2020. Black or white? how to develop an autotuner for memory-based analytics. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 1667–1683.

[41] Langchian. 2023. Langchian. https://python.langchain.com/docs/additional_resources/arxiv_references/.

[42] Jiale Lao, Yibo Wang, Yufei Li, Jianping Wang, Yunjia Zhang, Zhiyuan Cheng, Wanghu Chen, Mingjie Tang, and Jianguo Wang. 2024. Gptuner: A manual-reading database tuning system via gpt-guided bayesian optimization. *Proceedings of the VLDB Endowment* 17, 8 (2024), 1939–1952.

[43] Jiale Lao, Yibo Wang, Yufei Li, Jianping Wang, Yunjia Zhang, Zhiyuan Cheng, Wanghu Chen, Mingjie Tang, and Jianguo Wang. 2025. GPTuner: An LLM-Based Database Tuning System. *ACM SIGMOD Record* 54, 1 (2025), 101–110.

[44] Teddy Lazebnik, Amit Somech, and Abraham Itzhak Weinberg. 2022. Substrat: A subset-based optimization strategy for faster automl. *Proceedings of the VLDB Endowment* 16, 4 (2022), 772–780.

[45] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems* 33 (2020), 9459–9474.

[46] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems* 33 (2020), 9459–9474.

[47] Boyan Li, Yuyu Luo, Chengliang Chai, Guoliang Li, and Nan Tang. 2024. The Dawn of Natural Language to SQL: Are We Fully Ready? *arXiv preprint arXiv:2406.01265* (2024).

[48] Lan Li, Liri Fang, and Vetle I Torvik. 2024. AutoDCWorkflow: LLM-based Data Cleaning Workflow Auto-Generation and Benchmark. *arXiv preprint arXiv:2412.06724* (2024).

[49] Yiyan Li, Haoyang Li, Zhao Pu, Jing Zhang, Xinyi Zhang, Tao Ji, Luming Sun, Cuiping Li, and Hong Chen. 2024. Is large language model good at database knob tuning? A comprehensive experimental evaluation. *arXiv preprint arXiv:2408.02213* (2024).

[50] Yinheng Li, Shaofei Wang, Han Ding, and Hang Chen. 2023. Large language models in finance: A survey. In *Proceedings of the fourth ACM international conference on AI in finance*. 374–382.

[51] Zhaodonghui Li, Haitao Yuan, Huiming Wang, Gao Cong, and Lidong Bing. 2024. LLM-R2: A Large Language Model Enhanced Rule-based Rewrite System for Boosting Query Efficiency. *arXiv preprint arXiv:2404.12872* (2024).

[52] Zhaodonghui Li, Haitao Yuan, Huiming Wang, Gao Cong, and Lidong Bing. 2025. LLM-R2: A Large Language Model Enhanced Rule-based Rewrite System for Boosting Query Efficiency. *Proceedings of the VLDB Endowment* 1, 18 (2025), 53–65.

[53] Chen Liang, Donghua Yang, Zheng Liang, Zhiyu Liang, Tianle Zhang, Boyu Xiao, Yuqing Yang, Wenqi Wang, and Hongzhi Wang. 2025. Revisiting Data Analysis with Pre-trained Foundation Models. *arXiv preprint arXiv:2501.01631* (2025).

[54] Chin-Yew Lin. 2004. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*. 74–81.

[55] Yiming Lin, Mawil Hasan, Rohan Kosalge, Alvin Cheung, and Aditya G Parameswaran. 2025. TWIX: Automatically Reconstructing Structured Data from Templatized Documents. *arXiv preprint arXiv:2501.06659* (2025).

[56] Yiming Lin, Madelon Hulsebos, Ruiying Ma, Shreya Shankar, Sepanta Zeigham, Aditya G Parameswaran, and Eugene Wu. 2024. Towards Accurate and Efficient Document Analytics with Large Language Models. *arXiv preprint arXiv:2405.04674* (2024).

[57] Chunwei Liu, Matthew Russo, Michael Cafarella, Lei Cao, Peter Baille Chen, Zui Chen, Michael Franklin, Tim Kraska, Samuel Madden, and Gerardo Vitagliano. 2024. A Declarative System for Optimizing AI Workloads. *arXiv preprint arXiv:2405.14696* (2024).

[58] Chunwei Liu, Gerardo Vitagliano, Brandon Rose, Matt Prinz, David Andrew Samson, and Michael Cafarella. 2025. PalimpChat: Declarative and Interactive AI analytics. *arXiv preprint arXiv:2502.03368* (2025).

[59] Lihui Liu. 2025. Monte carlo tree search for graph reasoning in large language model agents. In *Proceedings of the 34th ACM International Conference on Information and Knowledge Management*. 4966–4970.

[60] Lei Liu, Xiaoyan Yang, Junchi Lei, Xiaoyang Liu, Yue Shen, Zhiqiang Zhang, Peng Wei, Jinjie Gu, Zhixuan Chu, Zhan Qin, et al. 2024. A Survey on Medical Large Language Models: Technology, Application, Trustworthiness, and Future Directions. *arXiv preprint arXiv:2406.03712* (2024).

[61] Tennison Liu, Nicolás Astorga, Nabeel Seedat, and Mihaela van der Schaar. 2024. Large Language Models to Enhance Bayesian Optimization. In *The Twelfth International Conference on Learning Representations*.

[62] llamaindex. 2023. llamaindex. https://www.llamaindex.ai/.

[63] llamaindex. 2024. Hyperparameter Optimization for RAG. https://developers.llamaindex.ai/python/examples/param_optimizer/param_optimizer/.

[64] Kanan Mahammadli and Seyda Ertekin. 2024. Sequential large language model-based hyper-parameter optimization. *arXiv preprint arXiv:2410.20302* (2024).

[65] Alex T. Mallen, Akari Asai, Victor Zhong, Rajarshi Das, Daniel Khashabi, and Hannaneh Hajishirzi. 2023. When Not to Trust Language Models: Investigating Effectiveness of Parametric and Non-Parametric Memories. In *Proceedings of ACL (long papers), 2023*. https://aclanthology.org/2023.acl-long.546/

[66] Joshua Maynez, Shashi Narayan, Bernd Bohnet, and Ryan McDonald. 2020. On faithfulness and factuality in abstractive summarization. *arXiv preprint arXiv:2005.00661* (2020).

[67] Alejandro Morales-Hernández, Inneke Van Nieuwenhuyse, and Sebastian Rojas Gonzalez. 2023. A survey on multi-objective hyperparameter optimization algorithms for machine learning. *Artificial Intelligence Review* 56, 8 (2023), 8043–8093.

[68] Multi-Linguality Multi-Functionality Multi-Granularity. 2024. M3-Embedding: Multi-Linguality, Multi-Functionality, Multi-Granularity Text Embeddings Through Self-Knowledge Distillation. (2024).

[69] Zan Ahmad Naeem, Mohammad Shahmeer Ahmad, Mohamed Eltabakh, Mourad Ouzzani, and Nan Tang. 2024. RetClean: Retrieval-Based Data Cleaning Using LLMs and Data Lakes. *Proceedings of the VLDB Endowment* 17, 12 (2024), 4421–4424.

[70] Andrew Y Ng, Daishi Harada, and Stuart Russell. 1999. Policy invariance under reward transformations: Theory and application to reward shaping. In *Icml*, Vol. 99. Citeseer, 278–287.

[71] Yuqi Nie, Yaxuan Kong, Xiaowen Dong, John M Mulvey, H Vincent Poor, Qingsong Wen, and Stefan Zohren. 2024. A Survey of Large Language Models for Financial Applications: Progress, Prospects and Challenges. *arXiv preprint arXiv:2406.11903* (2024).

[72] openai. 2025. Gpt-5 technical report. (2025). https://cdn.openai.com/gpt-5-system-card.pdf
[73] Matan Orbach, Ohad Eytan, Benjamin Sznajder, Ariel Gera, Odellia Boni, Yoav Kantor, Gal Bloch, Omri Levy, Hadas Abraham, Nitzan Barzilay, et al. 2025. An Analysis of Hyper-Parameter Optimization Methods for Retrieval Augmented Generation. *arXiv preprint arXiv:2505.03452* (2025).
[74] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. 2022. Training language models to follow instructions with human feedback. *Advances in neural information processing systems* 35 (2022), 27730–27744.
[75] Liana Patel, Siddharth Jha, Carlos Guestrin, and Matei Zaharia. 2024. Lotus: Enabling semantic queries with llms over tables of unstructured and structured data. *arXiv preprint arXiv:2407.11418* (2024).
[76] Boci Peng, Yun Zhu, Yongchao Liu, Xiaohe Bo, Haizhou Shi, Chuntao Hong, Yan Zhang, and Siliang Tang. 2024. Graph retrieval-augmented generation: A survey. *arXiv preprint arXiv:2408.08921* (2024).
[77] Yichen Qian, Yongyi He, Rong Zhu, Jintao Huang, Zhijian Ma, Haibin Wang, Yaohua Wang, Xiuyu Sun, Defu Lian, Bolin Ding, et al. 2024. UniDM: A Unified Framework for Data Manipulation with Large Language Models. *Proceedings of Machine Learning and Systems* 6 (2024), 465–482.
[78] Siddhant Ray, Rui Pan, Zhuohan Gu, et al. 2025. METIS: Fast Quality-Aware RAG Systems with Configuration Adaptation. In *Proceedings of the ACM SIGOPS 31st Symposium on Operating Systems Principles*. 606–622.
[79] Novem Ardan Rohmadin, Ridi Ferdiana, and Indriana Hidayah. 2025. Optimizing Retrieval-Augmented Generation Chatbot with Hyperparameter Tuning. In *2025 4th International Conference on Electronics Representation and Algorithm (ICERA)*.
[80] Aditi Singh, Abul Ehtesham, Saket Kumar, and Tala Talaei Khoei. 2025. Agentic retrieval-augmented generation: A survey on agentic rag. *arXiv preprint arXiv:2501.09136* (2025).
[81] Vikramank Singh, Kapil Eknath Vaidya, Vinayshekhar Bannihatti Kumar, Sopan Khosla, Murali Narayanaswamy, Rashmi Gangadharaiah, and Tim Kraska. 2024. Panda: Performance debugging for databases using LLM agents. (2024).
[82] Shamane Siriwardhana, Rivindu Weerasekera, Elliott Wen, Tharindu Kaluarachchi, Rajib Rana, and Suranga Nanayakkara. 2023. Improving the domain adaptation of retrieval augmented generation (RAG) models for open domain question answering. *Transactions of the Association for Computational Linguistics* 11 (2023), 1–17.
[83] Yuyang Song, Hanxu Yan, Jiale Lao, Yibo Wang, Yufei Li, Yuanchun Zhou, Jianguo Wang, and Mingjie Tang. 2025. QUITE: A Query Rewrite System Beyond Rules with LLM Agents. *arXiv preprint arXiv:2506.07675* (2025).
[84] Wenwen Sun, Zhicheng Pan, Zirui Hu, Yu Liu, Chengcheng Yang, Rong Zhang, and Xuan Zhou. 2025. Rabbit: Retrieval-Augmented Generation Enables Better Automatic Database Knob Tuning. In *2025 IEEE 41st International Conference on Data Engineering (ICDE)*. IEEE, 3807–3820.
[85] Zhaoyan Sun, Xuanhe Zhou, and Guoliang Li. 2024. R-Bot: An LLM-based Query Rewrite System. *arXiv preprint arXiv:2412.01661* (2024).
[86] Oyvind Tafjord, Matt Gardner, Kevin Lin, and Peter Clark. 2019. QuaRTz: An Open-Domain Dataset of Qualitative Relationship Questions. In *Proceedings of EMNLP-IJCNLP 2019*. https://aclanthology.org/D19-1608/ arXiv:1909.03553.
[87] Patara Trirat, Wonyong Jeong, and Sung Ju Hwang. 2024. Automl-agent: A multi-agent llm framework for full-pipeline automl. *arXiv preprint arXiv:2410.02958* (2024).
[88] Dana Van Aken, Andrew Pavlo, Geoffrey J Gordon, and Bohan Zhang. 2017. Automatic database management system tuning through large-scale machine learning. In *Proceedings of the 2017 ACM international conference on management of data*. 1009–1024.
[89] Chi Wang, Qingyun Wu, Markus Weimer, and Erkang Zhu. 2021. Flaml: A fast and lightweight automl library. *Proceedings of machine learning and systems* 3 (2021), 434–447.
[90] Jinqiang Wang, Huansheng Ning, Yi Peng, Qikai Wei, Daniel Tesfai, Wenwei Mao, Tao Zhu, and Runhe Huang. 2024. A Survey on Large Language Models from General Purpose to Medical Applications: Datasets, Methodologies, and Evaluations. *arXiv preprint arXiv:2406.10303* (2024).
[91] Shu Wang, Yixiang Fang, Yingli Zhou, Xilin Liu, and Yuchi Ma. 2025. ArchRAG: Attributed Community-based Hierarchical Retrieval-Augmented Generation. *CoRR* abs/2502.09891 (2025).
[92] Shen Wang, Tianlong Xu, Hang Li, Chaoli Zhang, Joleen Liang, Jiliang Tang, Philip S Yu, and Qingsong Wen. 2024. Large language models for education: A survey and outlook. *arXiv preprint arXiv:2403.18105* (2024).
[93] Qingyun Wu, Chi Wang, and Silu Huang. 2021. Frugal optimization for cost-related hyperparameters. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 35. 10347–10354.
[94] Shangyu Wu, Ying Xiong, Yufei Cui, Haolun Wu, Can Chen, Ye Yuan, Lianming Huang, Xue Liu, Tei-Wei Kuo, Nan Guan, et al. 2024. Retrieval-augmented generation for natural language processing: A survey. *arXiv preprint arXiv:2407.13193* (2024).
[95] Dongjie Xu, Yue Cui, Weijie Shi, Qingzhi Ma, Hanghui Guo, Jiaming Li, Yao Zhao, Ruiyuan Zhang, Shimin Di, Jia Zhu, et al. 2025. E3-Rewrite: Learning to Rewrite SQL for Executability, Equivalence, and Efficiency. *arXiv preprint*

arXiv:2508.09023 (2025).

[96] An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, et al. 2024. Qwen2. 5 Technical Report. *arXiv preprint arXiv:2412.15115* (2024).

[97] Steve Yang, Jason Rosenfeld, and Jacques Makutonin. 2018. Financial aspect-based sentiment analysis using deep representations. *arXiv preprint arXiv:1808.07931* (2018).

[98] 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. *arXiv preprint arXiv:1809.09600* (2018).

[99] Hao Yu, Aoran Gan, Kai Zhang, Shiwei Tong, Qi Liu, and Zhaofeng Liu. 2024. Evaluation of Retrieval-Augmented Generation: A Survey. *arXiv preprint arXiv:2405.07437* (2024).

[100] Fangyuan Zhang, Zhengjun Huang, Yingli Zhou, Qintian Guo, Zhixun Li, Wensheng Luo, Di Jiang, Yixiang Fang, and Xiaofang Zhou. 2025. EraRAG: Efficient and Incremental Retrieval Augmented Generation for Growing Corpora. *CoRR* abs/2506.20963 (2025).

[101] Wenchuan Zhang, Jingru Guo, Hengzhe Zhang, Penghao Zhang, Jie Chen, Shuwan Zhang, Zhang Zhang, Yuhao Yi, and Hong Bu. 2025. Patho-AgenticRAG: towards multimodal agentic retrieval-augmented generation for pathology VLMs via reinforcement learning. *arXiv preprint arXiv:2508.02258* (2025).

[102] Xinyi Zhang, Zhuo Chang, Yang Li, Hong Wu, Jian Tan, Feifei Li, and Bin Cui. 2022. Facilitating database tuning with hyper-parameter optimization: a comprehensive experimental evaluation. *Proceedings of the VLDB Endowment* 15, 9 (2022), 1808–1821.

[103] Xinyi Zhang, Hong Wu, Zhuo Chang, Shuowei Jin, Jian Tan, Feifei Li, Tieying Zhang, and Bin Cui. 2021. Restune: Resource oriented tuning boosted by meta-learning for cloud databases. In *Proceedings of the 2021 international conference on management of data*. 2102–2114.

[104] Penghao Zhao, Hailin Zhang, Qinhan Yu, Zhengren Wang, Yunteng Geng, Fangcheng Fu, Ling Yang, Wentao Zhang, and Bin Cui. 2024. Retrieval-augmented generation for ai-generated content: A survey. *arXiv preprint arXiv:2402.19473* (2024).

[105] Yanxin Zheng, Wensheng Gan, Zefeng Chen, Zhenlian Qi, Qian Liang, and Philip S Yu. 2024. Large language models for medicine: a survey. *International Journal of Machine Learning and Cybernetics* (2024), 1–26.

[106] Yihang Zheng, Bo Li, Zhenghao Lin, Yi Luo, Xuanhe Zhou, Chen Lin, Jinsong Su, Guoliang Li, and Shifu Li. 2024. Revolutionizing Database Q&A with Large Language Models: Comprehensive Benchmark and Evaluation. *arXiv preprint arXiv:2409.04475* (2024).

[107] Wei Zhou, Yuyang Gao, Xuanhe Zhou, and Guoliang Li. 2025. Cracking SQL Barriers: An LLM-based Dialect Translation System. *Proceedings of the ACM on Management of Data* 3, 3 (2025), 1–26.

[108] Xuanhe Zhou, Guoliang Li, Chengliang Chai, and Jianhua Feng. 2021. A learned query rewrite system using monte carlo tree search. *Proceedings of the VLDB Endowment* 15, 1 (2021), 46–58.

[109] Xuanhe Zhou, Guoliang Li, Zhaoyan Sun, Zhiyuan Liu, Weize Chen, Jianming Wu, Jiesi Liu, Ruohang Feng, and Guoyang Zeng. 2024. D-bot: Database diagnosis system using large language models. *Proceedings of the VLDB Endowment* 17, 10 (2024), 2514–2527.

[110] Xuanhe Zhou, Zhaoyan Sun, and Guoliang Li. 2024. Db-gpt: Large language model meets database. *Data Science and Engineering* 9, 1 (2024), 102–111.

[111] Yingli Zhou, Yaodong Su, Youran Sun, Shu Wang, Taotao Wang, Runyuan He, Yongwei Zhang, Sicong Liang, Xilin Liu, Yuchi Ma, et al. 2025. In-depth Analysis of Graph-based RAG in a Unified Framework. *arXiv preprint arXiv:2503.04338* (2025).

[112] Yingli Zhou, Zixuan Wang, and Yixiang Fang. 2025. HAMMER: An Automatic RAG Tuning System via Hierarchical Memory-Guided Monte Carlo Tree Search (technical report). http://github.com/JayLZhou/TechnicalReport/blob/main/RAG_Technical_report.pdf.

Received October 2025; revised January 2026; accepted February 2026