M-RAG: Reinforcing Large Language Model Performance through Retrieval-Augmented Generation with Multiple Partitions

Anonymous ACL submission

Abstract

Retrieval-Augmented Generation (RAG) enhances Large Language Models (LLMs) by retrieving relevant memories from an external database. However, existing RAG methods typically organize all memories in a whole database, potentially limiting focus on crucial memories and introducing noise. In this paper, we introduce a multiple partition paradigm for RAG (called `M-RAG`), where each database partition serves as a basic unit for RAG execution. Based on this paradigm, we propose a novel framework that leverages LLMs with Multi-Agent Reinforcement Learning to optimize different language generation tasks explicitly. Through comprehensive experiments conducted on seven datasets, spanning three language generation tasks and involving three distinct language model architectures, we confirm that `M-RAG` consistently outperforms various baseline methods, achieving improvements of 11%, 8%, and 12% for text summarization, machine translation, and dialogue generation, respectively.

1 Introduction

Introduced by (Lewis et al., 2020), Retrieval-Augmented Generation (RAG) represents a paradigm within the domain of Large Language Models (LLMs) to augment generative tasks. More specifically, RAG incorporates an initial retrieval step where LLMs query an external database to acquire relevant information before progressing to answer questions or generate text. This process not only guides the subsequent generation step but also guarantees that the responses are firmly anchored in the retrieved information (referred to as memories). Consequently, it enhances LLM performance, and has attracted growing research interests (Gao et al., 2023) in recent years.

While the majority of existing studies (Asai et al., 2023; Cheng et al., 2023b; Ma et al., 2023) adopt a retrieval approach that considers *a database as a whole*, which tends to yield a coarse-grained retrieval. The collective organization of all memories may hinder the focus on crucial memories and introduce noise, particularly due to the inherent challenges of Approximate k-Nearest Neighbor (AKNN) search when applied to large datasets. In this context, we investigate a retrieval approach that aims to search within a partition of the database, corresponding retrieval at a fine-grained level, which is designed to enhance the generation process by targeting specific memories. Moreover, in quite a few vector database systems, database partitions are regarded as fundamental units for analysis. This facilitates the construction and maintenance of index structures (Pan et al., 2023), ensures the protection of user privacy data (stored in specific partitions with access rights) (Xue et al., 2017), and supports distributed architectures (Guo et al., 2022). Therefore, in this work, we propose to take *a partition as a basic entity* in the execution of RAG, which is less explored in current methods.

We discuss our proposal with a motivating experiment illustrated in Figure 2. We investigate various strategies for partitioning a database (elaborated in Section 3.1), and perform RAG with varying the number of partitions for three generation tasks: summarization, translation, and dialogue generation, where we explore all partitions for the retrieval, and the best result (assessed based on a development set) across different partitions is reported. We observe that the optimal performance is typically not achieved through retrieval based on the entire database (#Partitions $= 1$). This observation inspires us to investigate a novel RAG setting with multiple partitions. To achieve this, the task should address three significant challenges, summarized below. (1) Determining a strategy for partitioning a database and the number of partitions. (2) Developing a method for selecting a suitable partition for a given input query to discover effective memories. (3) Enhancing memory quality, including inherent issues such as hallucination, or irrelevant context, which can impact the grounding of LLM generation.

Building upon the aforementioned discussion, we introduce a new solution called M-RAG, designed to facilitate RAG across multiple partitions of a database. M-RAG addresses all of the three challenges. For (1), we draw insights from the literature on vector database management (Pan et al., 2023; Han et al., 2023) and assess various strategies, namely Randomization (Indyk and Motwani, 1998), Clustering (Jegou et al., 2010), Indexing (Malkov et al., 2014; Malkov and Yashunin, 2018), and Category (Gollapudi et al., 2023), through empirical studies. The effectiveness of these strategies, along with the corresponding number of partitions, is evaluated across different generative tasks on a development set in our experiments. For (2), with multiple partitions at play, we formulate partition selection as a multi-armed bandit problem (Slivkins et al., 2019). In this context, an agent, denoted as Agent-S, iteratively selects one among several partitions. The characteristics of each partition are only partially known at the time of selection, and Agent-S gains a better understanding over time by maximizing cumulative rewards in the environment. To optimize the decision policy, we leverage reinforcement learning with a carefully designed Markov Decision Process (MDP). For (3), after selecting a partition and obtaining memories for generation, we introduce another agent, denoted as Agent-R. This agent generates a pool of candidate memories iteratively through the use of LLMs. Once a candidate is selected, Agent-R evaluates its quality by demonstrating it to generate a hypothesis. The identification of a high-quality hypothesis determined by a specific performance metric, triggers a boosting process, where it signals the exploration and replacement of the previous memory with a superior one, and continues the process. Further, we integrate the efforts of Agent-S and Agent-R through multi-agent reinforcement learning. With a shared objective of enhancing text generation for a given input query, they are jointly optimized through end-to-end training.

Our contributions can be summarized as follows: (1) we propose a multiple partition paradigm for RAG, aiming to facilitate fine-grained retrieval and concentrate on pivotal memories to enhance overall performance. In addition, the utilization of multiple partitions benefits other aspects of RAG, including facilitating the construction and maintenance of indices, protecting user privacy data within specific partitions, and supporting distributed parallel processing across different partitions. (2) We introduce M-RAG, a new solution based on multi-agent reinforcement learning that tackles the three challenges in executing RAG across multiple partitions. We show that the training objective of M-RAG is well aligned with that of text generation tasks. (3) We conduct extensive experiments on *seven* datasets for *three* generation tasks on *three* distinct language model architectures, including a recent Mixture of Experts (MoE) architecture (Jiang et al., 2024). The results demonstrate the effectiveness of M-RAG across diverse RAG baselines. In comparison to the best baseline approach, M-RAG exhibits improvements of 11%, 8%, and 12% for text summarization, machine translation, and dialogue generation tasks, respectively.

2 Related Work

2.1 Retrieval-Augmented Generation

We review the literature of Retrieval-Augmented Generation (RAG) in terms of (1) Naive RAG, (2) Advanced RAG, and (3) Modular RAG. For (1), Naive RAG follows a standard process including indexing, retrieval, and generation (Ma et al., 2023). However, its quality faces significant challenges such as low precision, hallucination, and redundancy during the process. For (2), Advanced RAG is further developed to overcome the shortcomings of Naive RAG. Specifically, during the indexing stage, the objective is to enhance the quality of the indexed content by optimizing data embedding (Li et al., 2023). During the retrieval stage, the focus is on identifying the appropriate context by calculating the similarity between the query and chunks, where the techniques involve fine-tuning embedding models (Xiao et al., 2023), or learning dynamic embeddings for different context (Karpukhin et al., 2020). During the generation stage, it merges the retrieved context with the query as an input into large language models (LLMs), where it addresses challenges posed by context window limits with re-ranking the most relevant content (Jiang et al., 2023; Zhuang et al., 2023), or compressing prompts (Litman et al., 2020; Xu et al., 2023). In addition, Self-RAG (Asai et al., 2023) is proposed to identify whether retrieval is necessary, or the retrieved context is relevant, which helps language models to produce meaningful generation (Asai

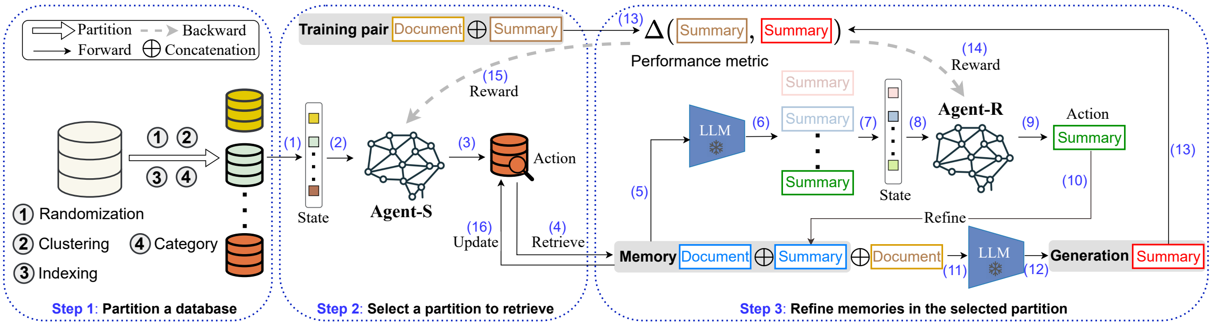


Figure 1: Illustration of M-RAG training in a summarization task: The M-RAG initiates training with multiple partitions (Section 3.1), it then selects a partition to perform retrieval via Agent-S (Section 3.2), and refines the memories within the selected partition via Agent-R (Section 3.3). Both agents are collaboratively trained to enhance generation capabilities through multi-agent reinforcement learning (Section 3.4).

et al., 2023). For (3), Modular RAG diverges from the traditional Naive RAG structure by incorporating external modules to further enhance the performance, including search module (Wang et al., 2023), memory module (Wang et al., 2022; Cheng et al., 2023b), tuning module (Lin et al., 2023), and task adapter (Cheng et al., 2023a; Dai et al., 2023). Specifically, Selfmem (Cheng et al., 2023b) incorporates a retrieval-enhanced generator to iteratively create a memory pool, it then trains a selector to choose one of the memories from the pool to generate responses. The work (Gao et al., 2023) provides a comprehensive survey of RAG for LLMs. Our work differs from existing RAG studies in two aspects. First, we introduce a multiple partition setting, where each partition serves as a fundamental entity for retrieval, rather than retrieving from the entire database. Second, we introduce an M-RAG framework built upon multi-agent reinforcement learning, which tackles three distinct challenges posed by this novel setting.

2.2 Reinforcement Learning for LLMs

Recently, reinforcement learning has seen broad applications across a variety of language-related tasks for Large Language Models (LLMs). This includes tasks such as text summarization (Wu et al., 2021), machine translation (Kreutzer et al., 2018), dialogue systems (Jaques et al., 2019; Yi et al., 2019), semantic parsing (Lawrence and Riezler, 2018), and review generation (Cho et al., 2018). For example, WebGPT (Nakano et al., 2021) incorporates a reinforcement learning framework to autonomously train the GPT-3 model using a search engine during the text generation process. Further, InstructGPT (Ouyang et al., 2022) collects a dataset containing desired model outputs provided by human labelers. Subsequently, it employs Reinforcement Learning from Human Feedback (RLHF) to fine-tune GPT-3 (Brown et al., 2020). In addition, R3 (Ma et al., 2023) introduces a Rewrite-Retrieve-Read process, where the LLM performance serves as a reinforcement learning incentive for a rewriting module. This approach empowers the rewriter to enhance retrieval queries, consequently improving the reader's performance in downstream tasks. In this work, we propose a novel multi-agent reinforcement learning framework utilizing two agents to collaboratively optimize text generation tasks. To our best knowledge, this is the first of its kind.

3 Methodology

A task involving M-RAG can be formulated below. Given a database $\mathbb{D} = \{(x_i, y_i)\}_{i=1}^{|\mathbb{D}|}$ for a language generation task (e.g., summarization), where each pair (x, y) represents a document and its corresponding summary stored in $\mathbb{D}$. The M-RAG initiates the process by partitioning $\mathbb{D}$ into multiple partitions. This can be achieved through methods like clustering or by leveraging inherent category labels in the data. The resulting partitions are denoted as $\mathbb{D} = \{D_m\}_{m=1}^{|M|}$, where each D_m $(1 \leq m \leq M)$ supports an independent RAG process (Section 3.1). The M-RAG framework comprises both training and inference processes, as outlined in Algorithm 1. For training, Agent-S learns to select a specific D_m for an input text pair (Section 3.2). Subsequently, Agent-R refines the retrieved memories, represented as $(\tilde{x}, \tilde{y}) \in D_m$, within the selected partition D_m (Section 3.3). Finally, the two agents are collaboratively trained with multi-agent reinforcement learning (see Section 3.4). Figure 1 illustrates the training process of M-RAG. For inference, the refined $\mathbb{D}$ is utilized to support an LLM in generating hypotheses, where a D_m is selected by the trained Agent-S.

3.1 Discussion on Partitioning a Database

As `M-RAG` relies on multiple partitions for RAG operations, we investigate various strategies to partition an external database (typically the training corpus). The results of these strategies are then validated through empirical studies. We review the literature, including recent vector database surveys (Pan et al., 2023; Han et al., 2023), and identify the following strategies: namely (1) Randomization (Indyk and Motwani, 1998), (2) Clustering (Jegou et al., 2010), (3) Indexing (Malkov et al., 2014; Malkov and Yashunin, 2018) and (4) Category (Gollapudi et al., 2023). Specifically, for (1), it targets the utilization of probability amplification techniques, such as locality-sensitive hashing (LSH), to hash similar items (data vectors) into the same bucket with a high probability. For (2), it involves clustering data vectors using K-means, where this clustering concept is widely applied in Inverted File Index (IVF) for tasks like Approximate k-Nearest Neighbor (AKNN) search. For (3), navigable graph indexes, such as HNSW (Malkov and Yashunin, 2018) or NSW (Malkov et al., 2014), are designed to facilitate easy traversal of different regions within a vector database. To achieve effective partitions, we employ graph partitioning with spectral clustering on a navigable graph. For (4), it involves assigning data vectors to partitions based on their respective categories. For example, in the DailyDialog dataset (Li et al., 2017), which includes 7 emotion categories (e.g., joy, anger) and 10 topic categories (e.g., work, health), vectors are partitioned according to their category labels. We note that a single vector may be assigned to multiple partitions, due to the characteristics of the dataset, where a dialogue spans multiple categories.

In Figure 2, we perform experiments on a development set, manipulating the number of partitions wrt the 4 strategies across three language generation tasks (summarization, translation, and dialogue generation). The results demonstrate the effectiveness of the strategies, and we conclude the selected strategies with the number of partitions as follows. We choose Indexing (4 partitions), Randomization (3 partitions), and Category (10 partitions) for the summarization, translation, and dialogue generation tasks, respectively. In addition, as shown in Figure 2 (a) and (b), we observe that both Top-1 and Top-3 retrieval methods exhibit comparable performance. For enhanced efficiency, we default to Top-1 retrieval in the rest of the paper.

3.2 Agent-S: Selecting a Database Partition

During the training process of an Agent-S to select a partition from $\mathbb{D}$, the environment is naturally modeled as a bandit setting. In this context, when a random partition is selected, the language model generates a response for the query with feedback (typically based on a specific performance metric), and concludes the episode. The selection process can be formulated as a Markov Decision Process (MDP), involving states, actions, and rewards.

States. Given a training pair (x, y) and a set of database partitions $\mathbb{D} = \{D_m\}_{m=1}^{|M|}$, the state $s^{(S)}$ is defined by assessing the semantic relevance, typically quantified by measures such as cosine similarity $\mathrm{sim}(\cdot,\cdot)$, between the input (x, y) and the stored memories $(\tilde{x}, \tilde{y})$ within each D_m.

$$s^{(S)} = \{\max_{(\tilde{x},\tilde{y})\in D_m} \mathrm{sim}(\sigma(\tilde{x} \oplus \tilde{y}), \sigma(x \oplus y))\}_{m=1}^{|M|}, \tag{1}$$

where $\oplus$ denotes the concatenation operation, and $\sigma(\cdot)$ denotes an embedded model utilized to obtain text representations, such as the CPT-Text (Neelakantan et al., 2022). We consider the Top-1 retrieved memories to construct the state.

Actions. Let $a^{(S)}$ represent an action undertaken by Agent-S. The design of actions corresponds to that of the state $s^{(S)}$. Specifically, the actions are defined as follows:

$$a^{(S)} = m \ (1 \leq m \leq M), \tag{2}$$

where action $a^{(S)} = m$ means to select the D_m for subsequent the generation task.

Rewards. The reward is denoted by $r^{(S)}$. When the action $a^{(S)}$ involves exploring a partition, the reward cannot be immediately observed, as no response has been received for the query x. However, when the action involves selecting a partition for Agent-R to refine the memories within the partition, the stored response $\tilde{y}$ is updated, and some reward signal can be obtained (for example, by measuring the difference between the results on the original memory and that on the refined memory). Therefore, we make Agent-S and Agent-R are trained with multi-agent reinforcement learning, since they cooperate towards the same objective of learning a policy that produces a response (hypothesis) as similar as possible to the reference y for the x.

3.3 Agent-R: Refining Memories in the Selected Partition

Next, we formulate the task of refining the retrieved memories carried out by Agent-R within a selected partition. To accomplish this, Agent-R explores potential responses denoted by $\hat{y}$ through LLMs for the retrieved $\tilde{x}$, and generates a candidate pool $\mathbb{C} = \{\hat{y}_k \leftarrow \text{LLM}(\tilde{x})\}_{k=1}^{|K|}$ for selection, where K denotes the number of candidates. Upon selecting a candidate, Agent-R evaluates its quality by demonstrating the new memory $(\tilde{x}, \hat{y}_k)$ to generate a hypothesis $h \leftarrow \text{LLM}(x \oplus (\tilde{x}, \hat{y}_k))$. In summary, a high-quality hypothesis h benefits from superior memory, which can be then refined through the produced hypothesis for subsequent selections. Consequently, Agent-R iterates in a boosting process optimized via reinforcement learning, where the states, actions, and rewards are detailed below.

States. The state $s^{(R)}$ is defined to assess the semantic relevance between the produced hypothesis h and the selected $\hat{y}_k$ from the pool $\mathbb{C}$. The rationale is to identify a memory that closely resembles the hypothesis, which aligns with the human intuition that a superior demonstration sample often leads to better generation results, that is

$$s^{(R)} = \{\text{sim}(\sigma(h), \sigma(\hat{y}_k))\}_{k=1}^{|K|}, \tag{3}$$

where $\sigma(\cdot)$ denotes an embedded model, and K governs the constructed state space.

Actions. Let $a^{(R)}$ represent an action taken by Agent-R. The design is consistent with the state $s^{(R)}$, which involves selecting a candidate memory from the pool, that is

$$a^{(R)} = k \; (1 \leq k \leq K). \tag{4}$$

Rewards. We denote the reward of Agent-R as $r_t^{(R)}$, which corresponds to the transition from the current state $\mathbf{s}_t^{(R)}$ to the next state $\mathbf{s}_{t+1}^{(R)}$ after taking action $a_t^{(R)}$. Specifically, when a memory $(\tilde{x}, \hat{y}_k)$ is updated, the hypothesis changes from h to h' accordingly. We remark that the best hypothesis (denoted as h') identified at state $s^{(R)}$ is maintained according to a specific metric $\Delta(\cdot, \cdot)$ (e.g., ROUGE for text summarization, BLEU for machine translation, BLEU and Distinct for dialogue generation), and the reward is defined as:

$$r^{(R)} = \Delta(h', y) - \Delta(h, y), \tag{5}$$

Algorithm 1: The M-RAG Framework

Require : a database $\mathbb{D}$; a frozen LLM$(\cdot)$

obtain $\mathbb{D} = \{D_m\}_{m=1}^{|M|}$ via a partitioning strategy
initialize Ag-S $\pi_\theta(a^{(S)}|s^{(S)})$, Ag-R $\pi_\phi(a^{(R)}|s^{(R)})$
while *not converged on a validation set* **do**
  sample a text pair (x, y) from the training set
  construct $s_1^{(S)}$ with (x, y) on $\mathbb{D}$ by Eq 1
  for $i = 1, 2, ...$ **do**
    sample $m = a_i^{(S)} \sim \pi_\theta(a|s_i^{(S)})$
    $r_i^{(S)} \leftarrow 0$
    $h \leftarrow \text{LLM}(x \oplus (\tilde{x}, \tilde{y}) \in D_m)$
    construct $s_1^{(R)}$ with h on $\mathbb{C} = \{\hat{y}_k \leftarrow \text{LLM}(\tilde{x})\}_{k=1}^{|K|}$ by Eq 3
    for $j = 1, 2, ...$ **do**
      sample $k = a_j^{(R)} \sim \pi_\phi(a|s_j^{(R)})$
      $h' \leftarrow \text{LLM}(x \oplus (\tilde{x}, \hat{y}_k))$
      if $\Delta(h', y) > \Delta(h, y)$ **then**
        $r_j^{(R)} \leftarrow \Delta(h', y) - \Delta(h, y)$
        $D_m.\tilde{y} \leftarrow \hat{y}_k, h \leftarrow h'$
      else
        $r_j^{(R)} \leftarrow 0$
      construct $s_{j+1}^{(R)}$ with h on a new $\mathbb{C}$
      $r_i^{(S)} \leftarrow r_i^{(S)} + r_j^{(R)}$
    construct $s_{i+1}^{(S)}$ by updating $(\tilde{x}, \tilde{y})$ and (x, y)
    optimize π_θ and π_ϕ via DQN
generate final hypotheses via LLM$(\cdot)$ on $\mathbb{D}$ (where the trained Ag-S selects a partition)

where y denotes the reference result. In this reward definition, we observe that the objective of the Markov Decision Process (MDP), which aims to maximize cumulative rewards, aligns with Agent-R's goal of discovering the best hypothesis among the memories. To illustrate, we consider the process through a sequence of states: $s_1^{(R)}, s_2^{(R)}, ..., s_N^{(R)}$, concluding at $s_N^{(R)}$. The rewards received at these states, except for the termination state, can be denoted as $r_1^{(R)}, r_2^{(R)}, ..., r_{N-1}^{(R)}$. When future rewards are not discounted, we have:

$$\begin{aligned}\sum_{t=2}^{N} r_{t-1}^{(R)} &= \sum_{t=2}^{N} (\Delta(h_t, y) - \Delta(h_{t-1}, y)) \\ &= \Delta(h_N, y) - \Delta(h_1, y),\end{aligned} \tag{6}$$

where $\Delta(h_N, y)$ corresponds to the highest hypothesis value found throughout the entire iteration, and $\Delta(h_1, y)$ represents an initial value that remains constant. Therefore, maximizing cumulative rewards is equivalent to maximizing the discovered hypothesis value. Finally, the cumulative reward is shared with Agent-S to align with the training objective, that is

$$r^{(S)} = \Delta(h_N, y) - \Delta(h_1, y). \tag{7}$$

3.4 The M-RAG Framework

Policy Learning via DQN. In a MDP, the primary challenge lies in determining an optimal policy that guides an agent to select actions at states, with the aim of maximizing cumulative rewards. Given that the states within our MDPs are continuous, we employ Deep Q-Networks (DQN) with replay memory (Mnih et al., 2013) to learn the policy, denoted as $\pi_\theta(a^{(S)}|s^{(S)})$ for Agent-S (resp. $\pi_\phi(a^{(R)}|s^{(R)})$ for Agent-R). The policy samples an action $a^{(S)}$ (resp. $a^{(R)}$) at a given state $s^{(S)}$ (resp. $s^{(R)}$) via DQN, with parameters denoted by θ (resp. ϕ).

Combining Agent-S and Agent-R. We present the M-RAG framework in Algorithm 1, which combines the functionalities of Agent-S and Agent-R on multiple partitions (line 1). The algorithm comprises two main phases: training and inference. During the training phase (lines 2-22), we randomly sample text pairs from the training set (line 4). For each pair, we generate episodes to iteratively train Agent-S and Agent-R, with the MDPs outlined in (lines 6-21) and (lines 11-20), respectively. Experiences of $(s_t^{(S)}, a_t^{(S)}, r_t^{(S)}, s_{t+1}^{(S)})$ and $(s_t^{(R)}, a_t^{(R)}, r_t^{(R)}, s_{t+1}^{(R)})$ are stored during the iteration, and a minibatch is sampled to optimize the two agents via DQN (line 22). During the inference phase (line 23), final hypotheses are generated via LLM based on the refined $\mathbb{D}$, where a partition is selected by the trained Agent-S.

4 Experiments

4.1 Experimental Setup

Datasets. By following (Cheng et al., 2023b), we conduct experiments on seven datasets for three generation tasks: (1) text summarization (XSum (Narayan et al., 2018) and BigPatent (Sharma et al., 2019)), (2) machine translation (JRC-Acquis (Steinberger et al., 2006) with Es$\rightarrow$En, En$\rightarrow$Es, De$\rightarrow$En, and En$\rightarrow$De), and (3) dialogue generation (DailyDialog (Li et al., 2017)). Specifically, XSum comprises single-document summaries for highly abstractive articles sourced from BBC news. BigPatent comprises 1.3 million records of U.S. patent documents accompanied by human-written abstractive summaries. JRC-Acquis serves as a collection of parallel legislative texts of European Union Law, commonly employed as a benchmark in machine translation tasks. DailyDialog comprises multi-turn dialogues centered around daily life topics. The detailed statistics for these datasets are available in (Cheng et al., 2023b).

Baselines. We carefully review the literature including a recent survey paper (Gao et al., 2023), and identify the following RAGs, namely Naive RAG (Ma et al., 2023), Self-RAG (Asai et al., 2023), and Selfmem (Cheng et al., 2023b), which correspond to three kinds of RAG techniques as described in Section 2. In addition, we incorporate the RAGs into three typical language model architectures, namely Mixtral 8$\times$7B (Jiang et al., 2024), Llama 2 13B (Touvron et al., 2023), and Phi-2 2.7B (Abdin et al., 2023) for the evaluation.

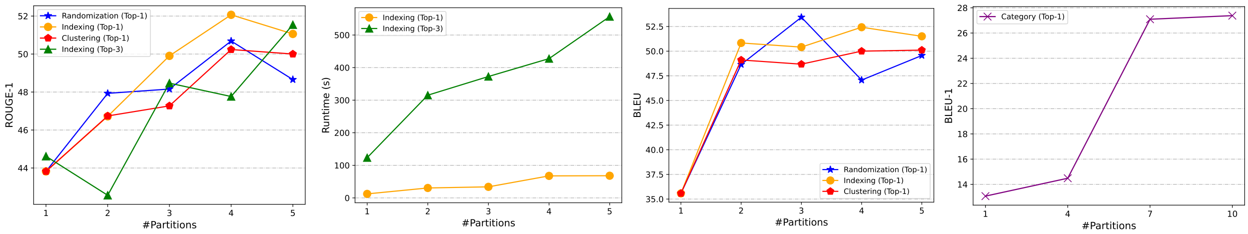
Evaluation Metrics. We evaluate the effectiveness of M-RAG in terms of the three generation tasks by following (Cheng et al., 2023b). (1) For summarization, ROUGE (R-1/2/L) (Lin, 2004) is used. (2) For machine translation, BLEU (Post, 2018) is used. (3) For dialogue generation, BLEU (B-1/2) and Distinct (D-1/2) (Li et al., 2016, 2021) are used. Overall, a higher evaluation metric (i.e., ROUGE, BLEU, Distinct) indicates a better result.

Implementation Details. We implement M-RAG and other baselines in Python 3.7 and LlamaIndex. The experiments are conducted on a server with 32 cores of Intel(R) Xeon(R) Gold 6151 CPU @ 3.00GHz 512.0GB RAM and 8 Nvidia RTX3090 GPU (24GB memory). The Agent-S (resp. Agent-R) is instantiated through a two-layered feedforward neural network. The first layer consists of 25 neurons using the tanh activation function, and the second layer comprises M (resp. K) neurons corresponding to the action space with a linear activation function. The hyperparameters M and K are empirically set to 4 and 3, respectively. During training, we randomly sample 10% of text pairs from the training set, while the remaining data is utilized for constructing the database with multiple partitions. The MDP iterations are determined by performance evaluation on a validation set. Evaluation metrics, such as ROUGE, BLEU, and Distinct, are obtained from (Cheng et al., 2023b). The language models with 4-bit quantization, including Mixtral 8$\times$7B, Llama 2 13B, and Phi-2 2.7B, are available for download via the link [1].

4.2 Experimental Results

(1) Effectiveness evaluation (partitioning strategies). We conduct experiments to evaluate various partitioning strategies across text summarization (XSum), machine translation (Es$\rightarrow$En), and dia-

[1] https://huggingface.co/TheBloke



(a) Summ. (ROUGE-1) (b) Summ. (Runtime) (c) Machine translation (d) Dialogue generation

Figure 2: Comparison with database partitioning strategies for language generation tasks.

Table 1: Text summarization.

LLM	RAG	XSum			BigPatent		
		R-1	R-2	R-L	R-1	R-2	R-L
Mixtral 8 × 7B	None	25.40	6.39	18.30	47.41	16.63	25.14
Mixtral 8 × 7B	Naive	43.82	22.07	37.44	60.11	38.33	43.44
Mixtral 8 × 7B	Selfmem	44.67	22.38	37.86	64.12	39.21	46.21
Mixtral 8 × 7B	Self-RAG	44.01	22.26	37.51	63.59	38.65	45.25
Mixtral 8 × 7B	`M-RAG`	**48.13**	**24.66**	**39.43**	**71.34**	**42.24**	**47.22**
Llama 2 13B	`M-RAG`	37.18	18.02	26.44	60.31	37.33	33.47
Phi-2 2.7B	`M-RAG`	30.70	11.57	26.20	31.25	14.72	18.98

logue generation (DailyDialog) tasks with Mixtral 8 × 7B. The best results, based on a development set across different partitions, are reported. As shown in Figure 2, we observe that retrieval based on the entire database generally fails to achieve optimal performance. Moreover, the performance slightly decreases as the number of partitions increases. This is attributed to the AKNN search, where a smaller partition size recalls more similar memories, which may not align well with the LLM preferences and impede the focus on crucial memories. Additionally, we observe that the RAG with Top-1 retrieval exhibits faster runtime compared to the Top-3 due to a shorter input length for the LLM, while maintaining comparable performance.

(2) Effectiveness evaluation (text summarization). We compare the performance of the `M-RAG` against alternative RAG methods on three distinct language models: Mixtral 8×7B, Llama 2 13B, and Phi-2 2.7B. The corresponding results are outlined in Table 1. We observe consistent improvement in language models when utilizing the RAG framework (e.g., Naive) compared to models without RAG (e.g., None). In addition, the recent MoE architecture Mistral 8 × 7B generally outperforms the typical Llama 2 13B in the summarization task. Specifically, when considering Mistral 8 × 7B as a base model, the performance of `M-RAG` outperforms that of other baseline models on both datasets. For example, it achieves better results than the best baseline model Selfmem, by 8% and 11% in terms of R-1 on XSum and BigPatent, respectively.

(3) Effectiveness evaluation (machine translation). We further conduct experiments to evaluate the performance of `M-RAG` for machine translation, and the results are reported in Table 2. We observe that a consistent improvement in the performance of translation tasks with `M-RAG` across four datasets and three architectures. Notably, it surpasses the Selfmem by 8% in the Es→En translation task.

(4) Effectiveness evaluation (dialogue generation). As shown in Table 3, `M-RAG` further enhances the language model performance for dialogue generation tasks. It outperforms the Selfmem by 12% in terms of B-1. Notably, we can also use the Distinct score as the performance metric for optimizing the two agents, denoted by `M-RAG(D)`, and it results in a more diverse dialogue.

(5) Ablation study. To evaluate the effectiveness of the two agents in `M-RAG`, we conduct an ablation study on XSum. We remove Agent-S and utilize the entire database for RAG; we replace Agent-R with a greedy rule to select a candidate memory from the pool according to Equation 3; and we remove both agents, which degrades to the Naive RAG. The results are presented in Table 4, demonstrating that both agents contribute to performance improvement. Specifically, removing Agent-S results in a significant decline in R-1 from 48.13 to 44.20. This underscores the role of the multiple partition setting in enhancing overall performance. Moreover, removing Agent-R leads to a reduction in R-1 from 48.13 to 45.75. This decline is attributed to the effectiveness of Agent-R in learning

Table 2: Machine translation.

LLM	RAG	Es→En		En→Es		De→En		En→De	
		Dev	Test	Dev	Test	Dev	Test	Dev	Test
Mixtral 8 × 7B	None	34.34	34.81	32.60	28.32	43.75	44.09	43.78	42.24
Mixtral 8 × 7B	Naive	36.64	36.22	33.18	30.70	47.84	46.77	45.83	44.23
Mixtral 8 × 7B	Selfmem	37.65	37.11	34.12	31.86	48.08	47.31	51.38	49.81
Mixtral 8 × 7B	Self-RAG	37.17	36.82	33.80	31.61	47.99	47.27	50.10	48.75
Mixtral 8 × 7B	`M-RAG`	**39.11**	**39.98**	**35.18**	**32.70**	**49.16**	**48.15**	**53.76**	**50.75**
Llama 2 13B	`M-RAG`	30.41	30.03	26.40	22.03	41.10	42.22	45.98	42.58
Phi-2 2.7B	`M-RAG`	22.83	24.22	17.64	16.60	34.21	34.71	40.01	37.08

Table 3: Dialogue generation.

LLM	RAG	DailyDialog			
		B-1	B-2	D-1	D-2
Mix. 8 × 7B	None	15.52	7.05	61.49	89.51
Mix. 8 × 7B	Naive	37.44	29.16	89.42	92.55
Mix. 8 × 7B	Selfmem	38.16	29.92	89.23	95.23
Mix. 8 × 7B	Self-RAG	37.76	29.79	88.24	95.34
Mix. 8 × 7B	`M-RAG`	**42.61**	**32.97**	88.82	95.74
Llama 2 13B	`M-RAG`	31.29	17.63	63.19	88.20
Phi-2 2.7B	`M-RAG`	7.71	3.93	44.21	82.86
Mix. 8 × 7B	`M-RAG(D)`	39.14	30.98	**93.14**	**98.34**

Table 4: Ablation study.

Components	R-1	R-2	R-L
`M-RAG`	**48.13**	**24.66**	**39.43**
w/o Agent-S (single DB)	44.20	22.72	37.40
w/o Agent-R (greedy)	45.75	23.21	38.28
w/o Agent-S and Agent-R	43.82	22.07	37.44

memory selection dynamically, as opposed to relying on a fixed rule for decision-making.

(6) Parameter study (Agent-S state space M). We study the effect of parameter M, which controls the state space of Agent-S and corresponds to the number of partitions. In Table 5, we observe that setting $M = 4$ yields the best effectiveness while maintaining reasonable runtime in terms of index construction, retrieval, and generation. This is consistent with empirical studies illustrated in Figure 2 (a). When $M = 1$, it reduces to a single database for RAG. As M increases, index construction accelerates on smaller partitions, while retrieval time sightly increases due to the additional time required for constructing states by querying each partition. As expected, the retrieval time is much smaller than the language generation time.

(7) Parameter study (Agent-R state space K). We study the effect of parameter K in Agent-R, representing the state space of Agent-R, to choose one memory from a candidate pool with a size of K. In Table 6, we observe a performance improvement as K increases from 1 to 3, and then remains

Table 5: Impacts of the number of M in Agent-S.

M	1	2	3	4	5
R-1	44.20	44.53	46.27	48.13	47.21
Index constr. (s)	299	278	257	246	227
Retrieval (s)	0.61	1.09	1.54	2.19	2.59
Generation (s)	83.59	84.88	82.81	82.89	86.64

Table 6: Impacts of the number of K in Agent-R.

K	1	2	3	4	5
R-1	45.81	46.54	48.13	48.18	48.25
Pool gen. (s)	76	191	267	290	359

stable. Particularly, when $K = 1$, `M-RAG` exhibits the worst performance, possibly due to the limited exploration of potential memories for generating improved hypotheses. We choose the setting of $K = 3$, as it demonstrates effective performance, and runs reasonably fast for generating the pool.

5 Conclusion and Limitations

In this paper, we propose a multiple partition paradigm for RAG, which aims to refine retrieval processes and emphasize pivotal memories to improve overall performance. Additionally, we introduce `M-RAG`, a novel framework grounded in multi-agent reinforcement learning, which addresses key challenges inherent in executing RAG across multiple partitions. The training objective of `M-RAG` is well aligned with that of text generation tasks, showcasing its potential to enhance system performance explicitly. Through extensive experiments conducted on seven datasets for three language generation tasks, we validate the effectiveness of `M-RAG`. For limitations, we conduct experiments with quantized versions of language models due to computational constraints. However, the observed effectiveness gains are expected to remain consistent across different model sizes and should not significantly impact the overall trends of various RAG methods. In future work, we intend to explore the incorporation of larger language models to further enhance effectiveness.

References

Marah Abdin, Jyoti Aneja, ebastien Bubeck, and Caio Cesar Teodoro Mendes et al. 2023. Phi-2: The surprising power of small language models. https://www.microsoft.com/en-us/research/blog/phi-2-the-surprising-power-of-small-language-models.

Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. 2023. Self-rag: Learning to retrieve, generate, and critique through self-reflection. *CoRR*, abs/2310.11511.

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. *NeurIPS*, 33:1877–1901.

Daixuan Cheng, Shaohan Huang, Junyu Bi, Yuefeng Zhan, Jianfeng Liu, Yujing Wang, Hao Sun, Furu Wei, Weiwei Deng, and Qi Zhang. 2023a. UPRISE: universal prompt retrieval for improving zero-shot evaluation. In *EMNLP*, pages 12318–12337.

Xin Cheng, Di Luo, Xiuying Chen, Lemao Liu, Dongyan Zhao, and Rui Yan. 2023b. Lift yourself up: Retrieval-augmented text generation with self memory. *ACL*.

Woon Sang Cho, Pengchuan Zhang, Yizhe Zhang, Xiujun Li, Michel Galley, Chris Brockett, Mengdi Wang, and Jianfeng Gao. 2018. Towards coherent and cohesive long-form text generation. *CoRR*.

Zhuyun Dai, Vincent Y. Zhao, Ji Ma, Yi Luan, Jianmo Ni, Jing Lu, Anton Bakalov, Kelvin Guu, Keith B. Hall, and Ming-Wei Chang. 2023. Promptagator: Few-shot dense retrieval from 8 examples. In *ICLR*.

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

Siddharth Gollapudi, Neel Karia, Varun Sivashankar, Ravishankar Krishnaswamy, Nikit Begwani, Swapnil Raz, Yiyong Lin, Yin Zhang, Neelam Mahapatro, Premkumar Srinivasan, et al. 2023. Filtered-diskann: Graph algorithms for approximate nearest neighbor search with filters. In *WWW*, pages 3406–3416.

Rentong Guo, Xiaofan Luan, Long Xiang, Xiao Yan, Xiaomeng Yi, Jigao Luo, Qianya Cheng, Weizhi Xu, Jiarui Luo, Frank Liu, et al. 2022. Manu: a cloud native vector database management system. *PVLDB*, 15(12):3548–3561.

Yikun Han, Chunjiang Liu, and Pengfei Wang. 2023. A comprehensive survey on vector database: Storage and retrieval technique, challenge. *CoRR*.

Piotr Indyk and Rajeev Motwani. 1998. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 604–613.

Natasha Jaques, Asma Ghandeharioun, Judy Hanwen Shen, Craig Ferguson, Agata Lapedriza, Noah Jones, Shixiang Gu, and Rosalind Picard. 2019. Way off-policy batch deep reinforcement learning of implicit human preferences in dialog. *CoRR*.

Herve Jegou, Matthijs Douze, and Cordelia Schmid. 2010. Product quantization for nearest neighbor search. *TPAMI*, 33(1):117–128.

Albert Q. Jiang, Alexandre Sablayrolles, Antoine Roux, and Arthur Mensch et al. 2024. Mixtral of experts. *CoRR*, abs/2401.04088.

Huiqiang Jiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. 2023. Llmlingua: Compressing prompts for accelerated inference of large language models. In *EMNLP*, pages 13358–13376.

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

Julia Kreutzer, Shahram Khadivi, Evgeny Matusov, and Stefan Riezler. 2018. Can neural machine translation be improved with user feedback? *CoRR*.

Carolin Lawrence and Stefan Riezler. 2018. Improving a neural semantic parser by counterfactual learning from human bandit feedback. *CoRR*.

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. *NeurIPS*, 33:9459–9474.

Jinpeng Li, Yingce Xia, Rui Yan, Hongda Sun, Dongyan Zhao, and Tie-Yan Liu. 2021. Stylized dialogue generation with multi-pass dual learning. In *NeurIPS*, pages 28470–28481.

Jiwei Li, Michel Galley, Chris Brockett, Jianfeng Gao, and Bill Dolan. 2016. A diversity-promoting objective function for neural conversation models. In *HLT-NAACL*, pages 110–119.

Xinze Li, Zhenghao Liu, Chenyan Xiong, Shi Yu, Yu Gu, Zhiyuan Liu, and Ge Yu. 2023. Structure-aware language model pretraining improves dense retrieval on structured data. In *ACL (Findings)*, pages 11560–11574.

Yanran Li, Hui Su, Xiaoyu Shen, Wenjie Li, Ziqiang Cao, and Shuzi Niu. 2017. Dailydialog: A manually labelled multi-turn dialogue dataset. In *IJCNLP(1)*, pages 986–995. Asian Federation of Natural Language Processing.

Chin-Yew Lin. 2004. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pages 74–81.

Xi Victoria Lin, Xilun Chen, Mingda Chen, Weijia Shi, Maria Lomeli, Rich James, Pedro Rodriguez, Jacob Kahn, Gergely Szilvasy, Mike Lewis, Luke Zettlemoyer, and Scott Yih. 2023. RA-DIT: retrieval-augmented dual instruction tuning. *CoRR*, abs/2310.01352.

Ron Litman, Oron Anschel, Shahar Tsiper, Roee Litman, Shai Mazor, and R. Manmatha. 2020. SCATTER: selective context attentional scene text recognizer. In *CVPR*, pages 11959–11969.

Xinbei Ma, Yeyun Gong, Pengcheng He, Hai Zhao, and Nan Duan. 2023. Query rewriting for retrieval-augmented large language models. *EMNLP*, pages 5303–5315.

Yu A Malkov and Dmitry A Yashunin. 2018. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *TPAMI*, 42(4):824–836.

Yury Malkov, Alexander Ponomarenko, Andrey Logvinov, and Vladimir Krylov. 2014. Approximate nearest neighbor algorithm based on navigable small world graphs. *Information Systems*, 45:61–68.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing atari with deep reinforcement learning. *CoRR*.

Reiichiro Nakano, Jacob Hilton, Suchir Balaji, and Jeff Wu et al. 2021. Webgpt: Browser-assisted question-answering with human feedback. *CoRR*, abs/2112.09332.

Shashi Narayan, Shay B. Cohen, and Mirella Lapata. 2018. Don't give me the details, just the summary! topic-aware convolutional neural networks for extreme summarization. In *EMNLP*, pages 1797–1807.

Arvind Neelakantan, Tao Xu, Raul Puri, Alec Radford, Jesse Michael Han, Jerry Tworek, Qiming Yuan, Nikolas Tezak, Jong Wook Kim, Chris Hallacy, et al. 2022. Text and code embeddings by contrastive pre-training. *CoRR*.

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. *NeurIPS*, 35:27730–27744.

James Jie Pan, Jianguo Wang, and Guoliang Li. 2023. Survey of vector database management systems. *CoRR*.

Matt Post. 2018. A call for clarity in reporting BLEU scores. In *WMT*, pages 186–191.

Eva Sharma, Chen Li, and Lu Wang. 2019. BIGPATENT: A large-scale dataset for abstractive and coherent summarization. In *ACL (1)*, pages 2204–2213.

Aleksandrs Slivkins et al. 2019. Introduction to multi-armed bandits. *Foundations and Trends® in Machine Learning*, 12(1-2):1–286.

Ralf Steinberger, Bruno Pouliquen, Anna Widiger, Camelia Ignat, Tomaz Erjavec, Dan Tufis, and Dániel Varga. 2006. The jrc-acquis: A multilingual aligned parallel corpus with 20+ languages. In *LREC*, pages 2142–2147. European Language Resources Association (ELRA).

Hugo Touvron, Louis Martin, Kevin Stone, and Peter Albert et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *CoRR*, abs/2307.09288.

Shuohang Wang, Yichong Xu, Yuwei Fang, Yang Liu, Siqi Sun, Ruochen Xu, Chenguang Zhu, and Michael Zeng. 2022. Training data is more valuable than you think: A simple and effective method by retrieving from training data. In *ACL (1)*, pages 3170–3179.

Xintao Wang, Qianwen Yang, Yongting Qiu, Jiaqing Liang, Qianyu He, Zhouhong Gu, Yanghua Xiao, and Wei Wang. 2023. Knowledgpt: Enhancing large language models with retrieval and storage access on knowledge bases. *CoRR*, abs/2308.11761.

Jeff Wu, Long Ouyang, Daniel M Ziegler, Nisan Stiennon, Ryan Lowe, Jan Leike, and Paul Christiano. 2021. Recursively summarizing books with human feedback. *CoRR*.

Shitao Xiao, Zheng Liu, Peitian Zhang, and Xingrun Xing. 2023. Lm-cocktail: Resilient tuning of language models via model merging. *CoRR*, abs/2311.13534.

Fangyuan Xu, Weijia Shi, and Eunsol Choi. 2023. RECOMP: improving retrieval-augmented lms with compression and selective augmentation. *CoRR*, abs/2310.04408.

Wenzhuo Xue, Hui Li, Yanguo Peng, Jiangtao Cui, and Yu Shi. 2017. Secure k nearest neighbors query for high-dimensional vectors in outsourced environments. *IEEE Transactions on Big Data*, 4(4):586–599.

Sanghyun Yi, Rahul Goel, Chandra Khatri, Alessandra Cervone, Tagyoung Chung, Behnam Hedayatnia, Anu Venkatesh, Raefer Gabriel, and Dilek Hakkani-Tur. 2019. Towards coherent and engaging spoken dialog response generation using automatic conversation evaluators. *CoRR*.

Shengyao Zhuang, Bing Liu, Bevan Koopman, and Guido Zuccon. 2023. Open-source large language models are strong zero-shot query likelihood models for document ranking. In *EMNLP (Findings)*, pages 8807–8817.