Cache Mechanism for Agent RAG Systems

Anonymous ACL submission

Abstract

Recent advances in Large Language Model (LLM)-based agents have been propelled by Retrieval-Augmented Generation (RAG), which grants the models access to vast external knowledge bases. Despite RAG's success in improving agent performance, agent-level cache management, particularly constructing, maintaining, and updating a compact, relevant corpus dynamically tailored to each agent's need, remains underexplored. Therefore, we introduce **ARC (Agent RAG Cache Mechanism)**, a novel, annotation-free caching framework that dynamically manages small, high-value corpora for each agent. By synthesizing historical query distribution patterns with the intrinsic geometry of cached items in the embedding space, ARC automatically maintains a high-relevance cache. With comprehensive experiments on three retrieval datasets, our experimental results demonstrate that ARC reduces storage requirements to 0.015% of the original corpus while offering up to 79.8% has-answer rate and reducing average retrieval latency by 80%. Our results demonstrate that ARC can drastically enhance efficiency and effectiveness in RAG-powered LLM agents.

1 Introduction

LLM agents have demonstrated significant potential in various domains, which exhibit remarkable capabilities in performing complex reasoning tasks (Wang et al., 2024b; Jin et al., 2024), executing knowledge-intensive operations (Jiang et al., 2024; Li et al., 2025), and implementing customized automated workflows (Li et al., 2024; Xu et al., 2025). Researchers have employed multiple techniques to enhance these agents' ability, including RAG, finetuning, and prompt engineering. Among these approaches, RAG empowers agents with the capacity to access extensive external knowledge repositories, facilitating more informed response generation and consequently becoming widely implemented across agent architectures (Gao et al., 2024; Huang et al., 2025a; Guo et al., 2025a).

The enhancement of LLM agent capabilities has led to significant growth in knowledge repositories, with extensive-scale knowledge bases now prevalent in advanced systems. With the accumulation of diverse and multifarious documents, several hundred GB or TB-level corpora are common among enterprise and government entities. These huge knowledge repositories not only enable LLM agents to address a more comprehensive range of inquiries, but also bring the following challenges: (1) high storage costs, and (2) deployment difficulties on edge and mobile devices.

Firstly, massive knowledge repositories drive up hardware and maintenance expenses—storing terabytes of embeddings and documents demands high-performance disks or distributed storage clusters—and complicate system engineering due to the need for sophisticated indexing, sharding, and cache invalidation logic (Mei et al., 2025a,b). Furthermore, deploying RAG at the edge or on mobile devices is particularly challenging: such platforms have limited local storage and CPU/GPU resources, and often operate under unreliable or high-latency network conditions. Every off-device retrieval incurs tens to hundreds of milliseconds in round-trip delay, degrading user experience and potentially causing service unavailability during network outages. These constraints motivate the introduction of an efficient, high-speed caching layer that stores only the most semantically central passages. By dramatically reducing both remote calls and on-device compute, a well-managed cache cuts latency, lowers bandwidth consumption, and simplifies the overall architecture—eliminating the need to bundle or host the full knowledge base on every device.

While the constraints outlined above motivate efficient caching solutions, recent approaches have fallen short. Numerous studies have attempted to

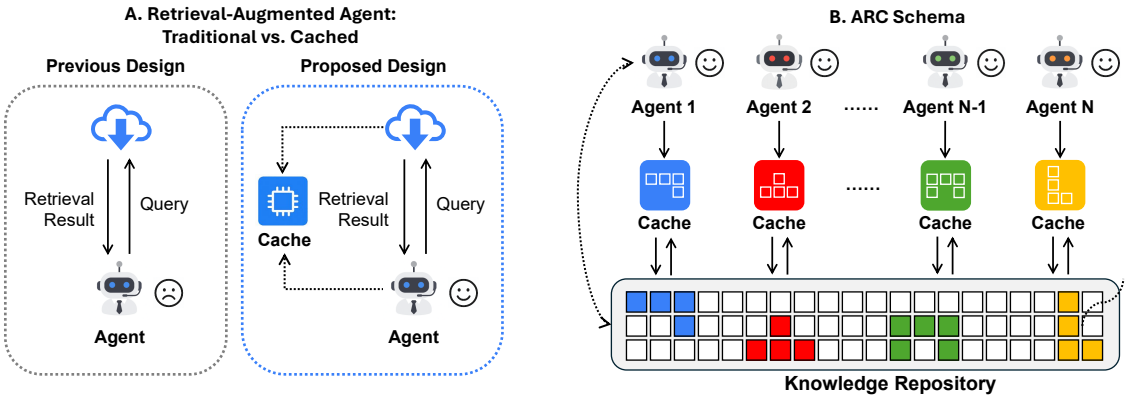


Figure 1: The motivations of our research: (a) The comparison between traditional and cached retrieval-augmented agent. (b) Our proposed ARC schema.

introduce caching mechanisms to accelerate LLM systems, yet the majority of these approaches primarily focus on key-value caches, which incur significant storage costs (Lu et al., 2025) and consequently limit their applicability in the very resource-constrained environments they aim to serve. These conventional caching strategies operate independently of semantic relationships between queries and documents, thus failing to fully leverage the semantic-aware retrieval techniques that make modern RAG systems effective in the first place.

Our insight brings guidance to cache algorithm design. We propose the **Agent RAG Cache Mechanism (ARC)**, which introduces two innovative components: (1) a rank-distance based frequency that weights query relevance by both occurrence frequency and semantic similarity to top-ranked results, and (2) a centrality quantification through hubness, where items with higher neighbor connectivity intuitively contain more general knowledge relevant to the agent's domain. We combine these two factors to create a priority score for cached items that governs cache maintenance. On a large-scale corpus comprising millions of embedding–passage pairs, ARC achieves a **79.8% cache has-answer rate** while using just **0.015%** of the original storage.

In summary, our main contributions are as follows:

- **Problem.** To the best of our knowledge, this work presents the first dedicated caching mechanism designed for retrieval-augmented LLM agents, and establishes a formulation of this problem for analyzing cache efficiency and performance.
- **Methodology.** We propose **ARC**, an agent RAG caching algorithm that leverages query-based dynamics and the structural properties of the item representation space, which can drastically reduce storage requirements while preserving retrieval effectiveness.
- **Evaluation.** We evaluate ARC on three datasets, with the 6.4-million-document Wikipedia as a realistic, noisy external corpus. Compared to prior methods, ARC attains a cache size of **0.015%** of the full index, a **79.8%** cache has-answer rate, and an **80%** average reduction in retrieval latency.

2 Related Work

RAG for agent. RAG enables agents to efficiently access external knowledge. Early studies embedded one-off retrieval calls directly into prompts for single-step decision-making or multi-hop question answering, allowing LLMs to reference external documents during generation and thereby improving the accuracy of individual responses or action instructions (Shi et al., 2024; Lee et al., 2024; Wang et al., 2025; Guo et al., 2025a). Subsequently, frameworks such as PlanRAG (Lee et al., 2024) proposed a phased pipeline, which decomposes complex tasks into subgoals (Guo et al., 2025b), retrieves pertinent information for each subtask, and synthesizes the results in a unified generation step; this reduces redundant queries and enhances the traceability and efficiency of multi-step decision processes (Lee et al., 2024; Huang et al., 2025b). In multi-hop reasoning contexts, the Generate-then-Ground (Shi et al., 2024) paradigm was introduced: the model first generates intermediate hypotheses, which are then grounded via document-level verification by the retrieval module,

improving the robustness of chain-of-thought reasoning (Shi et al., 2024). More recently, methods such as RAP (Kagaya et al., 2024) and RAT (Wang et al., 2024c) have incorporated mechanisms for contextual memory and dynamic retrieval triggering, enabling agents in multimodal or long-horizon interactions to invoke external knowledge adaptively based on internal confidence metrics or new sensory inputs, thus providing flexible, on-demand information supplementation. Cutting-edge frameworks like RAG-Gym employ fine-grained process supervision to jointly optimize retrieval and reasoning workflows (Xiong et al., 2025).

Efficient RAG. Efficient Retrieval-Augmented Generation aims to reduce resource consumption while improving end-to-end performance. Research in this field can be organized into three major areas: (1) Retrieval algorithm optimizations, including sparse indexing such as IVF/PQ (Johnson et al., 2017) and low-dimensional vector representations with approximate nearest neighbor search (Quinn et al., 2025); (2) Pipeline optimizations, such as multi-stage cascaded retrieval, which progressively refines candidate sets to focus expensive ranking on high-recall subsets (Bai et al., 2024); and (3) Hardware-aware compression techniques, such as 4-bit embedding quantization for RAG (Jeong, 2025), alongside comprehensive evaluations of compression and dimensionality-reduction trade-offs (Huerga-Pérez et al., 2025; Wang et al., 2026).

Semantic Encoding. The field of text embedding has progressively advanced from static representations such as Word2Vec (Mikolov et al., 2013) to contextualized models like ELMo (Peters et al., 2018) and BERT (Devlin et al., 2019), which generate dynamic word vectors conditioned on surrounding context. Building upon these frameworks, Siamese-network and contrastive-learning approaches such as SBERT (Reimers and Gurevych, 2019) and SimCSE (Gao et al., 2021) produce high-quality sentence and paragraph embeddings, while retrieval-augmented methods like DPR (Karpukhin et al., 2020) and ColBERT (Khattab and Zaharia, 2020) integrate explicit document retrieval with dense vector representations. More recent work including LM-Steer (Han et al., 2023) has explored mechanisms for task- or style-guided embedding control. Concurrently, intrinsic challenges of high-dimensional spaces have been widely recognized, including the curse of dimensionality (Bellman, 1957), hubness (Radovanovic et al., 2010), and anisotropy (Ethayarajh, 2019). These challenges have motivated a range of mitigation strategies, including local scaling, isotropy regularization, and manifold-alignment techniques.

Despite advances in retrieval systems, two critical gaps remain: the unexploited hierarchical nature of embeddings within topic clusters, and the neglect of embedding-space geometry in existing neural-retriever caches. We address these limitations by (1) formalizing embedding stratification to identify semantically central passages, and (2) designing a cache mechanism that leverages both spatial geometry and agent-specific semantics. Our approach combines hubness-based centrality with dynamic rank-distance metrics to enable efficient RAG retrieval in LLM agents.

3 Problem Formulation

In this section, we cast the agent's RAG pipeline as a constrained optimization problem, and further develop a detailed formulation that factorizes the pipeline into three sequential stages: (1) query generation, (2) retrieval search, and (3) cache storage.

Query generation. Unlike general-purpose conversational LLMs, agents are typically associated with a predefined domain specialization that specifies their intended application context (Russell and Norvig, 2016). This specialization induces domain-specific response preferences. For instance, a medical agent tends to provide precise answers to clinical queries (Ely et al., 2005). Once users become aware of such preferences, their interactions often exhibit implicit biases: users are more likely to ask medical agents health-related questions rather than queries from unrelated domains (Ebell, 1999), such as astrophysics. This latent query bias implies that an agent's query distribution is shaped by its operational context. Formally, this relationship can be expressed as $q \sim \mathcal{P}(q|\Theta)$, where q is the user query, Θ refers to the agent's domain specialization, and $\mathcal{P}$ denotes the probability distribution of queries conditioned on the agent's domain specialization.

Retrieval Search. In the retrieval phase, a RAG-enhanced agent leverages the user-provided query to identify highly relevant documents from an external corpus. Given a query q and an external corpus U, the set of documents retrieved by the agent can be formally defined as:

$$\mathrm{Ret}(q, U) = \{x_i : x_i \in \arg\,\mathrm{TopK}_{x_i \in U}\,(\mathrm{sim}(q, x_i))\},$$

where x_i denotes a candidate retrievable item (e.g., document passage) from the corpus U, and $\mathrm{sim}(q, x)$ is the agent's internal similarity function that scores the relevance between the query q and each item x.

Cache storage. As the agent engages in frequent interactions with users, the cache utilization of a RAG-enhanced system tends to grow continuously due to the need to retrieve new documents over time. However, retaining all retrieved content is impractical, as it would require prohibitively large cache capacity. In real-world deployments, it is therefore critical to design RAG agents that operate efficiently under limited cache budgets. Let $w(x_i)$ denote the cache space occupied by the i-th retrieved item. The constraint of operating within a fixed cache budget naturally introduces a capacity constraint into the optimization problem:

$$\sum_{x_i \in C_t} w(x_i) \;\leq\; W_{max}, \forall t \geq 0$$

where W_{max} is the max capacity of the cache. Let C_t denote the set of items stored in the cache at time t.

Cache Metrics. To quantify query-conditioned cache efficiency, we define the cumulative miss count for query q_t and the empirical has-answer rate over T queries as

$$M_t = \sum_{j=1}^{m} \mathbb{I}[x_{t,j} \notin C_{t,j-1}]$$

$$\text{Has-AnswerRate}_T = 1 - \frac{1}{mT} \sum_{t=1}^{T} M_t$$

Here, $\mathbb{I}(\cdot)$ serve as the indicator function. $x_{t,j}$ denotes the j-th item in query q_t, and $C_{t,j-1}$ is the cache state before accessing it. The has-answer rate reflects the average fraction of items served from the cache, offering a proxy objective for optimizing learned caching policies based on retrieval success.

3.1 Agent RAG Cache: From Query to Cache

Let q_t denote the t-th query sampled from a distribution $\mathcal{P}(q \mid \Theta)$. Prior to processing q_t, the cache is in state $C_{t-1} \subseteq U$, constrained by a total capacity $\sum_{x \in C_{t-1}} w(x) \leq W_{\max}$, where $w(x)$ represents the memory footprint of item x.

The retrieval module returns a ranked list of candidates:

$$R_t = \mathrm{Ret}(q_t, U) = (x_{t,1}, x_{t,2}, \ldots, x_{t,m}),$$

with each $x_{t,j}$ selected from the top-K corpus items under a similarity function:

$$x_{t,j} \in \mathrm{TopK}_{x \in U} \left(\mathrm{sim}(q_t, x)\right).$$

Upon accessing candidate $x = x_{t,j}$, the augmented candidate pool becomes $D = C_{t,j-1} \cup \{x\}$. The cache state is then updated via selection:$C_{t,j} = \arg\max_{S \subseteq D} \sum_{y \in S} p(y)$ subject to, $\sum_{y \in S} w(y) \leq W_{\max}$, where $p(y)$ denotes the utility score guiding cache prioritization. If the candidate is redundant or infeasible—i.e., $x \in C_{t,j-1}$ or $w(x) > W_{\max}$—no update is applied: $C_{t,j} = C_{t,j-1}$. After all m candidates are considered, the final cache state is set as $C_t = C_{t,m}$. S ranges over all subsets of the augmented candidate set D.

3.2 Optimization Objective

At inference step n, our objective is to optimize the cache policy—specifically, the priority function $p(\cdot)$ and update rule—so as to maximize the expected has-answer rate across a future horizon of H queries:

$$\max_p \mathbb{E}_{q_{n+1:n+H} \sim \mathcal{P}(q|\Theta)} \left[\text{Has-AnswerRate}_{n+1:n+H}\right] = \max_p \mathbb{E}\left[1 - \frac{1}{mH} \sum_{t=n+1}^{n+H} M_t\right].$$

This formulation defines our core goal: to design a cache strategy that maximizes future query performance by learning from historical retrieval patterns, subject to fixed capacity constraints.

4 Agent RAG Cache Mechanism

In this section, we introduce ARC, a cache algorithm designed for RAG-powered agents. ARC maintains the cache by deriving two complementary factors into a unified *priority* value for each cache item. Our ARC algorithm leverages both query-based dynamics and the structural properties of the item representation space: while the distance–rank frequency score (DRF) quantifies the dynamic demand for passages demonstrated through historical queries, the hubness score identifies passages within the cache that are inherently more likely to be retrieved due to their inherent spatial structure. By synthesizing these complementary signals, we establish a balanced item prioritization mechanism that adapts to agent query distribution patterns while ensuring cache priorities remain grounded in the geometric properties of the passage embedding space.

4.1 Distance–Rank Frequency Score

While a significant number of caching algorithms are designed with a heavy dependence on item access frequency, these approaches usually lack consideration of retrieval system characteristics. In these frequency-based methods, all retrievals of a passage typically contribute equally to its cache importance score. However, when a passage appears as a top-ranked result, it generally provides more value than another passage appearing in the tenth position for the identical query. In other words, higher-ranked appearances indicate stronger semantic alignment with queries and should be weighted more heavily in caching decisions. By implementing differential weighting based on retrieval rank, our caching mechanism incorporates rank position as a key signal in our priority formulation, creating a more retrieval-aware caching strategy.

Rank-based weighting improves cache prioritization by emphasizing higher-ranked retrievals, which captures the relative order of results. However, we still need to reflect actual semantic closeness between queries and passages. For example, a tenth-rank hit with high cosine distance should score lower than another tenth-rank result that is semantically closer. To address this, we also incorporate the absolute distance between query and passage embeddings into our scoring mechanism. By combining both rank for ordinal importance and distance for semantic proximity, our priority formula delivers a more nuanced evaluation of each passage's cache value.

Distance–Rank Frequency (DRF) Score. Let $\mathcal{Q}$ denote the set of cached queries and for each query $q \in \mathcal{Q}$ let $\mathrm{Ret}(q)$ be the ranked retrieval set (top-k) returned from the cache or external database. Define the DRF score of item p as

$$\mathrm{DRF}(p) = \sum_{q \in \mathcal{Q}:\, p \in \mathrm{Ret(q)}} \frac{1}{\mathrm{rank}(q,p) \;\cdot\; \mathrm{dist}(q,p)^{\alpha}}$$

where $\mathrm{rank}(q,p)$ denotes the 1-based position of p in the retrieval results for query q, $\mathrm{dist}(q,p)$ represents the embedding distance between q and p, and $\alpha > 0$ is a tunable parameter controlling distance sensitivity.

In summary, DRF factor accumulates a passage's retrieval history, weighted by both its rank position and embedding distance for each query. It therefore directly captures how frequently and meaningfully the passage responds to the current query distribution, reflecting real-time demand and relevance in the agent system.

4.2 Hubness Score

In this subsection, we introduce the hubness score, computed on the nearest neighbor search index of the cache candidate set. This metric highlights passages in the dense cores of the embedding space, specifically those that frequently appear in other embeddings' neighbor lists, without relying on any query history. As a result, it provides a truly query-agnostic approach for identifying which cache entries are most semantically central and broadly useful.

Space-Aware Approach. Hubness refers to the statistical phenomenon in high-dimensional spaces where certain points called "hubs" appear disproportionately often in others' k-nearest-neighbor lists (Tomasev et al., 2013). Prior studies have examined hubness from algorithmic and statistical perspectives but have paid little attention to its semantic implications.

Existing works (Bogolin et al., 2022; Wang et al., 2023) reveal that passages with high hubness scores consistently exhibit higher retrieval rates across diverse query sets, reflecting a space-invariant measure of semantic centrality. This insight motivates the first component of our cache mechanism. However, while hubness effectively captures the global importance of items, our formulation must also consider local retrieval dynamics. We therefore introduce both rank and absolute distance components into our prioritization formula. The rank component addresses the varying importance of different positions in retrieval results, while absolute distance proves crucial in practice; for instance, when query-document similarity is exceptionally low, such results should receive correspondingly reduced weighting regardless of rank.

Furthermore, due to sharing the same mathematical formulation in top-K selection, hubness serves as a natural bridge between retrieval behavior and embedding space characteristics, enabling us to simulate retrieval dynamics without making assumptions about the query distribution.

Hubness Score. Hubness score of x counts its occurrences in other points' k-nearest-neighbor lists. Which in math can be written as

$$h_k(x_i) = \sum_{\substack{j=1 \\ j \neq i}}^{n} \mathbb{I}(x_i \in \mathcal{N}_k(x_j)) \qquad (1)$$

$\mathcal{N}_k(x_j)$ denotes the k-nearest neighbors of x_j, $\mathbb{I}(\cdot)$ serves as the indicator function. In the ARC algorithm, we compute hubness over the cache's own candidate set.

Combined Priority. Our priority calculation integrates two key metrics: DRF and the hubness score introduced in Section 3. With an additional penalty term for memory utilization. Using $w(p)$ to denote the memory footprint of item p, we define:

$$\text{Priority}(p) = \frac{1}{\log\big(w(p)+1\big)}\Big[\beta\,\log\big(h_k(p)+1\big) + (1-\beta)\,\text{DRF}(p)\Big].$$

where $\beta \in [0,1]$ balances centrality versus query frequency. The $\log(w(p)+1)$ term in the denominator penalizes items with large memory footprints and encourages the cache to store more items. Items with lower $\text{Priority}(p)$ are deemed less valuable and evicted first to make room for new insertions.

Our comprehensive formula accounts for global importance via hubness, relative positional relationships via rank, and absolute semantic proximity via distance. By integrating these multiple dimensions of retrieval relevance, our approach offers superior performance compared to frequency-based formulations alone. This provides a more nuanced mechanism for determining cache priority that aligns with the semantic characteristics of embedding-based retrieval systems and remains robust across varying query distributions.

4.3 Cache Maintenance

Algorithm 1 shows the maintenance procedure of ARC managing a cache C with maximum capacity W_{max} and escalating the query to $\mathcal{D}$. When a new query q arrives, the algorithm first retrieves the top-k candidates $Ret(q) \leftarrow \text{topK}(q, C)$ from the cache. The relevance of these results is evaluated by computing the mean embedding distance $\text{avg}_{p \in Ret(q)}(\text{dist}(q,p))$ across all retrieved items. Only if the average distance exceeds τ, indicating low relevance, ARC escalates retrieval to the full external corpus $\mathcal{D}$ and updates the retrieved items $Ret(q) \leftarrow \text{topK}(q, \mathcal{D})$.

Subsequently, ARC updates the distance-rank metrics of each item $p \in Ret(q)$. To simplify the notation, we defined $\text{DR}(p,q) = 1/\text{rank}(q,p) \cdot \text{dist}(q,p)^{\alpha}$. Existing cache entries accumulate their DRF score via $\text{DRF}(p) \mathrel{+}= \text{DR}(p,q)$, while new items initialize $\text{DRF}(p) = \text{DR}(p,q)$ before being inserted into C. Finally, cache C iteratively evicts the lowest-priority item $\arg\min_{x \in C} \text{Priority}(x)$ until the total cache weight satisfies $\sum_{x \in C} w(x) \le W_{\max}$.

Algorithm 1 Agent RAG Cache Maintenance and Escalation

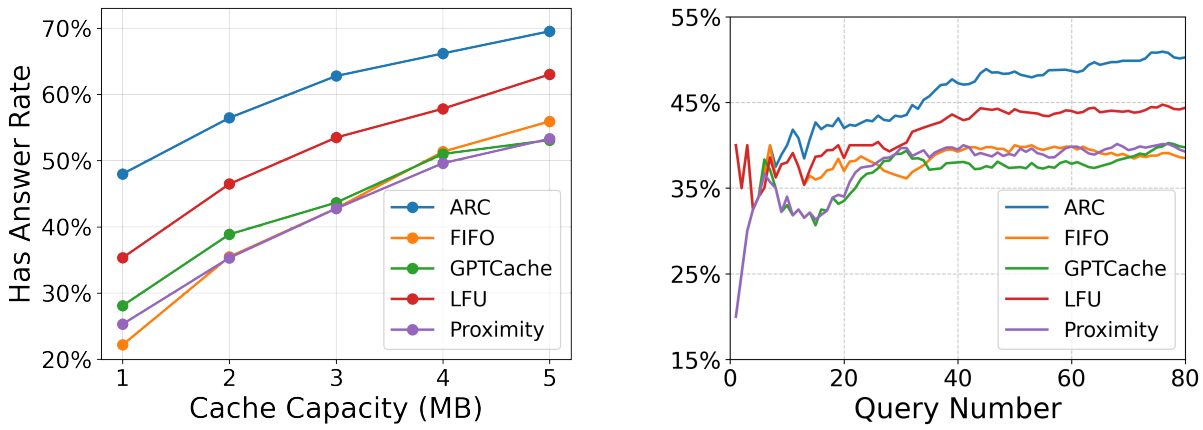
Input: Query vector q, External corpus $\mathcal{D}$, Cache C (with capacity $W_{\max}$), Top-k retrieval size k, Drift tolerance δ
Output: Top-K Items $Ret(q)$

```
Ret(q) ← Ret(q, C)  ▷ Return the topK result from the cache
if avg_{p∈Ret(q)}(dist(q, p)) > τ then
    Ret(q) ← Ret(q, U)  ▷ Return the topK result from the external corpus D
end if
for each p ∈ Ret(q) do  ▷ Update the DSF Score of inserted/existed items in cache
    if p ∈ C then
        DRF(p) += DR(p, q)
    else
        DRF(p) = DR(p, q)
        C ← C ∪ {p}
    end if
end for
while Σ_{x∈C} w(x) > W_max do  ▷ Evict if cache exceeds capacity
    C ← C \ {arg min_{x∈C} Priority(x)}
end while
return Ret(q)
```

5 Empirical Evaluation

Datasets: Large-scare Retrieval Corpus. To simulate a real-world, large-scale RAG deployment, we build our primary index from the English subset of the 2023 Wikipedia dump[1] which has over 6.4 million documents. We segment each document in passage level, where each chunk have one or more passages with no more than 2048 characters. The total chunk amount is over 14 milion. This full-scale document repository introduces substantial distractor noise—forcing the retriever to discriminate finely between relevant and irrelevant content. We also inject into this index all passages containing the ground-truth answers for our evaluation sets to evaluate performance.

[1] https://huggingface.co/datasets/wikimedia/wikipedia



(a) Has-answer Rate at Varying Cache Capacity

(b) Cache Has-answer Rate Across Query Stream

Figure 2: Cache performance analysis: (a) Effects of varying cache capacity; (b) Continuous improvement of has-answer rate with streaming queries.

Datasets: Evaluation Datasets. We assess ARC on three diverse question-answering datasets including SQuAD, MMLU and Adversarial QA.

We evaluate our method on three datasets: **SQuAD** (Rajpurkar et al., 2016), **MMLU** (Hendrycks et al., 2020), and **AdversarialQA** (Jia and Liang, 2017). SQuAD contains 536 short Wikipedia articles yielding 107,785 QA pairs, where answers are contiguous spans within each context. MMLU spans 57 academic and professional subjects, with 15,908 test and 1,540 calibration questions. AdversarialQA is built on the SQuAD v1.1 dev set (10,570 examples) and introduces perturbations via AddSent and AddOneSent distractors.

Baselines. We compare our cache method with several standard policies, including LFU, FIFO, Proximity, and GPTCache. LFU and FIFO are the most classic baselines in cache-related methods: LFU evicts the least frequently used items, while FIFO removes the oldest entries in the cache. **Proximity** (Bergman et al., 2025) maintains historical query–document pairs and returns previously retrieved passages from the most semantically similar past query when the similarity exceeds its threshold τ, evicting the oldest query–document pair. **GPTCache** (Bang, 2023) reuses results from semantically similar queries as an eviction mechanism; however, its eviction strategy relies solely on embedding similarity, neglecting valuable retrieval-specific signals such as rank-weighted frequency or embedding-space centrality that could better capture the demand intensity and semantic centrality of cached items. Optimal parameter configurations for both Proximity and GPTCache were determined as $\tau = 0.2$ through systematic grid search. In ARC, we set $\alpha = 0.4$, and $\beta = 0.7$, 0.15, and 0.2 for the SQuAD, MMLU, and AdversarialQA datasets, respectively. Unless stated otherwise, all experiments use a cache capacity of 3.0 MB and a top-K retrieval size of $K = 50$.

Models. For the embedding model, given that we need to index all wiki content, utilizing closed-source models would impose significant computational costs. Therefore, following the comprehensive RAG evaluation(Wang et al., 2024a), we adopted the open-source embedding models bge-small-en(Xiao et al., 2024) and llm-embedder(Zhang et al., 2023) for our indexing architecture, resulting in an embedding memory footprint of approximately 20 GB and 40 GB, respectively. We employed FAISS IndexFlatIP (Douze et al., 2024) for our vector similarity search.

Metrics. To evaluate the effectiveness of our cache system, we measure performance from two critical aspects: efficiency and latency reduction. We employ Has-Answer Rate, which measures the proportion of queries that can be successfully answered using cached content without accessing the full index, where a higher has-answer rate indicates better cache utilization and retrieval efficiency. Additionally, we use Average Memory Access Time (AMAT) (Hennessy and Patterson, 2011), which quantifies the average time required to retrieve information, accounting for both cache hits and misses. This metric effectively captures the overall latency reduction achieved by our caching mechanism compared to the baseline retrieval system.

Table 1: Has-Answer Rate↑ (%)

Method	bge-small-en			llm-embedder		
	MMLU	AdversarialQA	SQuAD	MMLU	AdversarialQA	SQuAD
LFU	53.26	66.11	69.46	42.09	57.57	59.37
FIFO	41.84	66.64	77.04	34.51	58.67	71.49
GPTCache	43.65	46.49	41.23	40.08	41.54	40.13
Proximity	46.41	54.25	59.79	38.41	50.73	58.84
ARC (w/o hubness)	62.37	68.14	76.09	51.92	58.48	69.55
ARC	**62.63**	**71.18**	**79.80**	**52.46**	**62.78**	**71.94**

Table 2: AMAT↓ (s)

Method	bge-small-en			llm-embedder		
	MMLU	AdversarialQA	SQuAD	MMLU	AdversarialQA	SQuAD
LFU	0.668	0.427	0.388	1.754	1.219	1.175
FIFO	0.814	0.429	0.298	1.969	1.182	0.821
GPTCache	0.788	0.704	0.773	1.810	1.680	1.720
Proximity	0.758	0.604	0.539	1.840	1.406	1.198
ARC (w/o hubness)	0.559	0.411	0.307	1.489	1.196	0.878
ARC	**0.556**	**0.377**	**0.269**	**1.468**	**1.046**	**0.818**

Baselines Comparison. To evaluate the performance of ARC, we conduct experiments comparing it against baselines including LFU, FIFO, Proximity, GPTCache, and ARC without the hubness component, across multiple retrieval benchmarks using two embedding models: bge-small-en and llm-embedder. We report the has-answer rate and AMAT in Table 1 and Table 2. *Best results are **bolded**; second-best are underlined.*

As shown in Table 2, it is observed that GPTCache and Proximity perform worst among all methods. This is because their effectiveness relies on the assumption that incoming queries are highly similar to previously seen ones, which only holds under extensive historical data and large cache. In contrast, ARC highest has-answer rates and the lowest AMAT across all benchmarks. For example, under the bge-small-en embedding, even without the hubness score, ARC achieves a has-answer rate of over 62% and an AMAT of 0.559s on MMLU, while LFU only achieves 53% and 0.668s. With the hubness score, ARC further improves by over 3% has-answer rate with hubness on Squad. Similarly, ARC outperforms all baselines in the llm-embedder setup, attaining ranging from 52.46% to 71.94%. The advantage of ARC comes from two combined score: DRF, which helps the cache identify valuable items based on their ranking position and semantic proximity to queries, and hubness score which naturally favors passages that hold central or influential positions within the embedding space. It is remarked that without caching, retrieving the entries with the highest semantic similarity on the Wikipedia index takes 1.313s per query on SQuAD, while ARC only needs 0.269s, reducing the query time by almost 80%.

Cache Capacity Ablation. Figure 2a illustrates cache has-answer rates varying cache capacity from 1MB to 5MB in Dataset MMLU with embedder bge-small-en. ARC consistently maintains the highest has-answer rates across various cache sizes, which significantly outperform baseline methods, particularly at smaller cache capacities. For example, ARC achieves 47.64% has-answer rate at 1 MB of cache capacity on MMLU, surpassing LFU by 12.8%.

Streaming Query. Figure 2b shows the streaming has-answer rates over 80 sequential queries on MMLU. ARC quickly adapts to the incoming stream and consistently maintains the highest cumulative has-answer rate. While all baselines experience an initial warm-up phase, ARC continues to improve and stabilizes above 50% after around 60 queries.

6 Conclusion

We introduce ARC, an annotation-free cache mechanism using stratified hub selection and adaptive escalation. On a 30M-pair corpus, ARC reaches a 79.8% has-answer rate while caching only 0.015% of the data, providing data-efficient retrieval in latency-constrained RAG systems.

7 Limitations

We focus our evaluation on the standard single-turn QA setting, which allows controlled measurement of caching behavior. While multi-turn dialogue is outside our current scope, ARC is compatible with session-level extensions; exploring dialogue-aware scoring and evaluation on conversational benchmarks is a natural next step. We have not yet performed experiments in transfer or cross-domain settings. Extending ARC to evaluate transfer robustness will be a direction for future work.

References

Yu Bai, Yukai Miao, Li Chen, Dawei Wang, Dan Li, Yanyu Ren, Hongtao Xie, Ce Yang, and Xuhui Cai. 2024. Pistis-rag: Enhancing retrieval-augmented generation with human feedback. *arXiv preprint arXiv:2407.00072*.

Fu Bang. 2023. Gptcache: An open-source semantic cache for llm applications enabling faster answers and cost savings. In *Proceedings of the 3rd Workshop for Natural Language Processing Open Source Software (NLP-OSS)*, pages 212–218.

Richard Bellman. 1957. *Dynamic Programming*. Princeton University Press.

Shai Aviram Bergman, Zhang Ji, Anne-Marie Kermarrec, Diana Petrescu, Rafael Pires, Mathis Randl, and Martijn de Vos. 2025. Leveraging approximate caching for faster retrieval-augmented generation. In *Proceedings of the 5th Workshop on Machine Learning and Systems*, pages 66–73.

Simion-Vlad Bogolin, Ioana Croitoru, Hailin Jin, Yang Liu, and Samuel Albanie. 2022. Cross modal retrieval with querybank normalisation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5194–5205.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, volume 1, pages 4171–4186.

Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. 2024. The faiss library. *arXiv preprint arXiv:2401.08281*.

Mark Ebell. 1999. Information at the point of care: answering clinical questions. *The Journal of the American Board of Family Practice*, 12(3):225–235.

John W Ely, Jerome A Osheroff, M Lee Chambliss, Mark H Ebell, and Marcy E Rosenbaum. 2005. Answering physicians' clinical questions: obstacles and potential solutions. *Journal of the American Medical Informatics Association*, 12(2):217–224.

Kawin Ethayarajh. 2019. How contextual are contextualized word representations? comparing the geometry of bert, elmo, and gpt-2 embeddings. In *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (EMNLP-IJCNLP)*, pages 55–65.

Chujie Gao, Siyuan Wu, Yue Huang, Dongping Chen, Qihui Zhang, Zhengyan Fu, Yao Wan, Lichao Sun, and Xiangliang Zhang. 2024. Honestllm: Toward an honest and helpful large language model. In *Advances in Neural Information Processing Systems*, volume 37, pages 7213–7255.

Tianyu Gao, Xingcheng Yao, and Danqi Chen. 2021. Simcse: Simple contrastive learning of sentence embeddings. *arXiv preprint arXiv:2104.08821*.

Minghao Guo, Qingcheng Zeng, Xujiang Zhao, Yanchi Liu, Wenchao Yu, Mengnan Du, Haifeng Chen, and Wei Cheng. 2025a. Deepsieve: Information sieving via llm-as-a-knowledge-router. *Preprint*, arXiv:2507.22050.

Minghao Guo, Xi Zhu, Haochen Xue, Chong Zhang, Shuhang Lin, Jingyuan Huang, Ziyi Ye, and Yongfeng Zhang. 2025b. Reagan: Node-as-agent-reasoning graph agentic network. *arXiv preprint arXiv:2508.00429*.

Chi Han, Jialiang Xu, Manling Li, Yi Fung, Chenkai Sun, Nan Jiang, Tarek Abdelzaher, and Heng Ji. 2023. Word embeddings are steers for language models. *arXiv preprint arXiv:2305.12798*.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2020. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*.

John L Hennessy and David A Patterson. 2011. *Computer architecture: a quantitative approach*. Elsevier.

Yue Huang, Chujie Gao, Siyuan Wu, Haoran Wang, Xiangqi Wang, Yujun Zhou, Yanbo Wang, Jiayi Ye, Jiawen Shi, Qihui Zhang, Yuan Li, Han Bao, Zhaoyi Liu, Tianrui Guan, Dongping Chen, Ruoxi Chen, Kehan Guo, Andy Zou, Bryan Hooi Kuen-Yew, and 47 others. 2025a. On the trustworthiness of generative foundation models: Guideline, assessment, and perspective. *arXiv preprint arXiv:2502.14296*.

Yue Huang, Siyuan Wu, Chujie Gao, Dongping Chen, Qihui Zhang, Yao Wan, Tianyi Zhou, Chaowei Xiao, Jianfeng Gao, Lichao Sun, and Xiangliang Zhang. 2025b. Datagen: Unified synthetic dataset generation via large language models. In *The Thirteenth International Conference on Learning Representations*.

Naamán Huerga-Pérez, Rubén Álvarez, Rubén Ferrero-Guillén, Alberto Martínez-Gutiérrez, and Javier Díez-González. 2025. Optimization of embeddings storage for rag systems using quantization and dimensionality reduction techniques. *CoRR*, abs/2505.00105.

Taehee Jeong. 2025. 4bit-quantization in vector-embedding for rag. *CoRR*, abs/2501.10534.

Robin Jia and Percy Liang. 2017. Adversarial examples for evaluating reading comprehension systems. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2021–2031.

Jinhao Jiang, Kun Zhou, Wayne Xin Zhao, Yang Song, Chen Zhu, Hengshu Zhu, and Ji-Rong Wen. 2024. Kg-agent: An efficient autonomous agent framework for complex reasoning over knowledge graph. *arXiv preprint arXiv:2402.11163*.

Mingyu Jin, Qinkai Yu, Jingyuan Huang, Qingcheng Zeng, Zhenting Wang, Wenyue Hua, Haiyan Zhao, Kai Mei, Yanda Meng, Kaize Ding, Fan Yang, Mengnan Du, and Yongfeng Zhang. 2024. How large language models acquire knowledge and concept at different layers? *arXiv preprint arXiv:2404.07066*.

Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2017. Billion-scale similarity search with gpus. *CoRR*, abs/1702.08734.

Tomoyuki Kagaya, Thong Jing Yuan, Yuxuan Lou, Jayashree Karlekar, Sugiri Pranata, Akira Kinose, Koki Oguri, Felix Wick, and Yang You. 2024. Rap: Retrieval-augmented planning with contextual memory for multimodal llm agents. *Preprint*, arXiv:2402.03610.

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

Omar Khattab and Matei Zaharia. 2020. Colbert: Efficient and effective passage search via contextualized late interaction over bert. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 39–48.

Myeonghwa Lee, Seonho An, and Min-Soo Kim. 2024. Planrag: A plan-then-retrieval augmented generation for generative large language models as decision makers. *Preprint*, arXiv:2406.12430.

Zelong Li, Shuyuan Xu, Kai Mei, Wenyue Hua, Balaji Rama, Om Raheja, Hao Wang, He Zhu, and Yongfeng Zhang. 2024. Autoflow: Automated workflow generation for large language model agents. *arXiv preprint arXiv:2407.12821*.

Zhenkun Li, Lingyao Li, Shuhang Lin, and Yongfeng Zhang. 2025. Know the ropes: A heuristic strategy for llm-based multi-agent system design. *CoRR*, abs/2505.16979.

Kuan Lu, Shuhang Lin, Sai Wu, Yichen Yao, Junhan Yang, Huan Li, Wei Chu, Xu Yinghui, Yuan Qi, and Gang Chen. 2025. Ctkvr: Kv cache retrieval for long-context llms via centroid then token indexing. *arXiv preprint arXiv:2512.15550*.

Kai Mei, Wujiang Xu, Minghao Guo, Shuhang Lin, and Yongfeng Zhang. 2025a. Omnirouter: Budget and performance controllable multi-llm routing. *SIGKDD Explor.*, 27(2):107–116.

Kai Mei, Wujiang Xu, Shuhang Lin, and Yongfeng Zhang. 2025b. Eccos: Efficient capability and cost coordinated scheduling for multi-llm serving. *Available at SSRN 5159339*.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*, volume 26, pages 3111–3119.

Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, volume 1, pages 2227–2237.

Derrick Quinn, Mohammad Nouri, Neel Patel, John Salihu, Alireza Salemi, Sukhan Lee, Hamed Zamani, and Mohammad Alian. 2025. Accelerating retrieval-augmented generation. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '25)*, pages 15–32, Rotterdam, The Netherlands.

Milos Radovanovic, Alexandros Nanopoulos, and Mirjana Ivanovic. 2010. Hubs in space: Popular nearest neighbors in high-dimensional data. *Journal of Machine Learning Research*, 11(sept):2487–2531.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. Squad: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392.

Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3982–3992.

Stuart J Russell and Peter Norvig. 2016. *Artificial intelligence: a modern approach*. pearson.

Zhengliang Shi, Shuo Zhang, Weiwei Sun, Shen Gao, Pengjie Ren, Zhumin Chen, and Zhaochun Ren. 2024. Generate-then-ground in retrieval-augmented generation for multi-hop question answering. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7339–7353, Bangkok, Thailand. Association for Computational Linguistics.

Nenad Tomasev, Milos Radovanovic, Dunja Mladenic, and Mirjana Ivanovic. 2013. The role of hubness in clustering high-dimensional data. *IEEE transactions on knowledge and data engineering*, 26(3):739–751.

Xiaohua Wang, Zhenghua Wang, Xuan Gao, Feiran Zhang, Yixin Wu, Zhibo Xu, Tianyuan Shi, Zhengyuan Wang, Shizheng Li, Qi Qian, Ruicheng Yin, Changze Lv, Xiaoqing Zheng, and Xuanjing Huang. 2024a. Searching for best practices in retrieval-augmented generation. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 17716–17736. Association for Computational Linguistics.

Yanbo Wang, Zixiang Xu, Yue Huang, Chujie Gao, Siyuan Wu, Jiayi Ye, Xiuying Chen, Pin-Yu Chen, and Xiangliang Zhang. 2025. Breaking focus: Contextual distraction curse in large language models. In *Will Synthetic Data Finally Solve the Data Access Problem?*

Yimu Wang, Xiangru Jian, and Bo Xue. 2023. Balance act: Mitigating hubness in cross-modal retrieval with query and gallery banks. *arXiv preprint arXiv:2310.11612*.

Yulong Wang, Tianhao Shen, Lifeng Liu, and Jian Xie. 2024b. Sibyl: Simple yet effective agent framework for complex real-world reasoning. *arXiv preprint arXiv:2407.10718*.

Zihao Wang, Anji Liu, Haowei Lin, Jiaqi Li, Xiaojian Ma, and Yitao Liang. 2024c. Rat: Retrieval augmented thoughts elicit context-aware reasoning in long-horizon generation. *Preprint*, arXiv:2403.05313.

Ziqi Wang, Xi Zhu, Shuhang Lin, Haochen Xue, Minghao Guo, and Yongfeng Zhang. 2026. Ragrouterbench: A dataset and benchmark for adaptive RAG routing. *CoRR*, abs/2602.00296.

Shitao Xiao, Zheng Liu, Peitian Zhang, Niklas Muennighoff, Defu Lian, and Jian-Yun Nie. 2024. C-pack: Packed resources for general chinese embeddings. In *Proceedings of the 47th international ACM SIGIR conference on research and development in information retrieval*, pages 641–649.

Guangzhi Xiong, Qiao Jin, Xiao Wang, Yin Fang, Haolin Liu, Yifan Yang, Fangyuan Chen, Zhixing Song, Dengyu Wang, Minjia Zhang, Zhiyong Lu, and Aidong Zhang. 2025. Supervising the search process produces reliable and generalizable information-seeking agents. *arXiv preprint arXiv:2502.13957*.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. 2025. A-mem: Agentic memory for llm agents. *arXiv preprint arXiv:2502.12110*.

Peitian Zhang, Shitao Xiao, Zheng Liu, Zhicheng Dou, and Jian-Yun Nie. 2023. A multi-task embedder for retrieval augmented llms. *arXiv preprint arXiv:2310.07554*.

A Theoretical Analysis of Hubness

The hubness score $h_k(x)$ defined in Section 4.2 is computed as a finite-sample average over the cache candidate set. A natural concern is whether this estimator is well-behaved. We address two specific questions: (i) whether it converges to a meaningful population quantity as more samples are accumulated, and (ii) whether the resulting top-ranked set is stable enough to support reliable cache decisions. Propositions 1 and 2 answer these two questions affirmatively.

A.1 Setup and Notation

Let documents be embedded as $x_1, \ldots, x_n \in \mathbb{R}^d$, and let $X_1, \ldots, X_m$ be i.i.d. draws from a distribution P over the candidate set $\{x_1, \ldots, x_n\}$. In ARC, P corresponds to either uniform sampling over the candidate set (used during asynchronous hubness re-estimation) or the empirical query-induced retrieval distribution (used during streaming maintenance); the analysis below holds for any such P.

For a sample X, let $\mathcal{N}_k(X) \subseteq \{x_1, \ldots, x_n\}$ denote its set of top-k nearest neighbors under the chosen distance metric. We assume that ties in nearest-neighbor distances occur with probability zero (e.g., under any absolutely continuous distribution over embeddings), so $\mathcal{N}_k(X)$ is well-defined almost surely. Throughout this section we fix k and write $h_i \triangleq h_k(x_i)$ for brevity.

Define the *true hit probability* of document x_i as

$$R_i \;\triangleq\; \Pr_{X\sim P}[\, x_i \in \mathcal{N}_k(X),\; X \neq x_i \,], \qquad (2)$$

for $i = 1, \ldots, n$. Given the m i.i.d. samples above, the *empirical hubness score* of x_i is

$$h_i \;\triangleq\; \sum_{j=1}^{m} \mathbb{I}\{\, x_i \in \mathcal{N}_k(X_j),\; X_j \neq x_i \,\}. \qquad (3)$$

Equation (3) reduces to the document-to-document hubness in Section 4.2 (Eq. (1)) when P is uniform over $\{x_1, \ldots, x_n\} \setminus \{x_i\}$ and $m = n - 1$.

A.2 Consistency of Empirical Hubness

We first show that the normalized empirical hubness score converges to the true hit probability as more samples are observed.

Proposition 1 (Consistency)**.** *For each fixed $i \in \{1, \ldots, n\}$,*

$$\frac{h_i}{m} \xrightarrow[m\to\infty]{a.s.} R_i. \qquad (4)$$

Proof. Fix i and define

$$Z_j^{(i)} \;\triangleq\; \mathbb{I}\{\, x_i \in \mathcal{N}_k(X_j),\; X_j \neq x_i \,\}, \qquad j = 1, \ldots, m. \qquad (5)$$

Since $X_1, \ldots, X_m$ are i.i.d., the random variables $Z_1^{(i)}, \ldots, Z_m^{(i)}$ are i.i.d. Bernoulli with

$$\mathbb{E}\Big[Z_j^{(i)}\Big] \;=\; \Pr[x_i \in \mathcal{N}_k(X_j),\; X_j \neq x_i] \;=\; R_i. \qquad (6)$$

By definition,

$$\frac{h_i}{m} \;=\; \frac{1}{m}\sum_{j=1}^{m} Z_j^{(i)}. \qquad (7)$$

By the Strong Law of Large Numbers,

$$\frac{1}{m}\sum_{j=1}^{m} Z_j^{(i)} \xrightarrow[m\to\infty]{\text{a.s.}} \mathbb{E}\Big[Z_1^{(i)}\Big] \;=\; R_i, \qquad (8)$$

which proves the claim. □

Proposition 1 establishes that hubness, when averaged over a sufficiently rich sample of the candidate set, provides an asymptotically unbiased estimate of the population-level retrieval frequency. The design choice of using hubness as a centrality signal in ARC is therefore grounded in a well-defined statistical quantity rather than a heuristic.

A.3 Stability of Top-s Selection

Consistency alone is insufficient for our purposes: ARC uses hubness to *rank* cached items and retain the top-s. We therefore need the empirical ranking to recover the true top-s set with high probability, not merely to estimate individual scores accurately.

Assume a cache budget of size s, where $1 \leq s < n$. Let

$$A \;\triangleq\; \{\text{indices of the } s \text{ documents with largest } R_i\} \qquad (9)$$

denote the *true optimal top-s set*. Define the empirical estimates $\widehat{R}_i \triangleq h_i/m$, and let $\widehat{A}$ denote the indices of the s documents with largest $\widehat{R}_i$.

We require a mild *separation condition*: there exists $\delta > 0$ such that

$$\min_{i\in A} R_i \;-\; \max_{j\notin A} R_j \;\geq\; \delta. \qquad (10)$$

This condition holds generically: in any finite corpus where hit probabilities are not exactly tied at the cache boundary, such a $\delta > 0$ exists. As the bound below shows, even modest separations are recoverable with reasonable sample volumes.

Proposition 2 (Stability)**.** *Under the separation condition* (10)*, for all* $m \geq 1$*,*

$$\Pr\left[\widehat{A} \neq A\right] \;\leq\; 2n \exp\left(-\frac{m\delta^2}{8}\right). \tag{11}$$

Proof. For each i, the variables $Z_1^{(i)}, \ldots, Z_m^{(i)}$ defined in the proof of Proposition 1 are i.i.d. Bernoulli with $\mathbb{E}[Z_j^{(i)}] = R_i$ and bounded in $[0, 1]$. By Hoeffding's inequality, for any $\varepsilon > 0$,

$$\Pr\left(\left|\widehat{R}_i - R_i\right| \geq \varepsilon\right) \;\leq\; 2\exp\left(-2m\varepsilon^2\right). \tag{12}$$

Setting $\varepsilon = \delta/4$ and applying a union bound over all n documents,

$$\Pr\left(\exists\, i : \left|\widehat{R}_i - R_i\right| \geq \delta/4\right) \;\leq\; 2n \exp\left(-\frac{m\delta^2}{8}\right). \tag{13}$$

Let E denote the complementary event, on which $|\widehat{R}_i - R_i| < \delta/4$ for all i. On E, for any $i \in A$ and $j \notin A$,

$$\begin{aligned} \widehat{R}_i - \widehat{R}_j \;&>\; (R_i - \delta/4) - (R_j + \delta/4) && (14)\\ &=\; (R_i - R_j) - \delta/2 && (15)\\ &\geq\; \delta - \delta/2 \;=\; \delta/2 \;>\; 0, && (16) \end{aligned}$$

where the last inequality uses (10). Since every element of A has strictly larger empirical score than every element of A^c, the s largest empirical scores are exactly those indexed by A, so $\widehat{A} = A$ on E. Combining with (13),

$$\Pr\left[\widehat{A} \neq A\right] \;\leq\; \Pr[E^c] \;\leq\; 2n \exp\left(-\frac{m\delta^2}{8}\right). \tag{17}$$

□

Sample complexity. Proposition 2 yields a concrete sample complexity: to guarantee $\Pr[\widehat{A} \neq A] \leq \epsilon$, it suffices to have

$$m \;\geq\; \frac{8}{\delta^2} \log\left(\frac{2n}{\epsilon}\right). \tag{18}$$

For a representative cache candidate set with $n = 10^3$ items, separation $\delta = 0.05$, and target failure probability $\epsilon = 0.01$, this yields $m \approx 4 \times 10^4$ samples, which is well within the streaming volumes observed in our experiments (Appendix C.2). The logarithmic dependence on n further suggests that the guarantee scales gracefully to larger candidate sets.

A.4 Discussion

Propositions 1 and 2 together justify the use of hubness in ARC from a statistical standpoint: empirical hubness is a strongly consistent and exponentially stable proxy for the latent retrieval frequency R_i. When the sampling distribution P coincides with the query-induced retrieval distribution, R_i has a direct operational reading as the population hit probability under deployment, so the same guarantees apply to streaming maintenance as to asynchronous re-estimation.

The i.i.d. assumption is standard in this style of analysis; extending the guarantees to non-i.i.d. but stationary or mixing query streams, which would replace the SLLN and Hoeffding tools with their ergodic counterparts, is a natural direction for future theoretical work. The empirical streaming results in Appendix C.2 and Section 5 suggest that the qualitative behavior predicted by these bounds, namely rapid convergence of the cache to a high-quality state, continues to hold in practice even when strict i.i.d. does not.

B Computational Complexity and Scalability

This appendix details the computational properties of ARC, complementing the brief complexity statement in the main text. We separate the discussion into the per-query online update cost, the asynchronous maintenance scheme that absorbs heavier operations, and the empirical overhead measured on our benchmarks.

B.1 Online Update Complexity

When a new query q arrives, ARC performs two operations beyond the standard retrieval step: (i) it scans the at most K retrieved passages to update their distance–rank statistics (Algorithm 1), and (ii) it maintains a priority queue over the cached items to support cheap top/bottom lookups during eviction. The first operation contributes an $O(K)$ term, and the second contributes an $O(\log |C|)$ term per insertion or update, where $|C|$ denotes the current number of cached items. The total per-query update cost is therefore

$$O(K + \log |C|). \tag{19}$$

In all our experiments we use $K = 50$ and $|C|$ remains in the order of a few thousand passages, so the cache update step is negligible compared to either the retrieval cost on the underlying index or the downstream LLM call.

B.2 Asynchronous Maintenance

Two ingredients of ARC, namely hubness re-estimation and full materialization of DRF scores over the cache candidate set, are heavier than the online update in Eq. (19). To prevent these operations from contributing to the foreground latency observed by the user, we run them in an asynchronous background worker that periodically refreshes the scores over the current candidate set. In our default configuration on MMLU we recompute hubness and DRF scores every 10 incoming queries. The foreground execution path during this window only records the retrieved passages and performs standard cache insertions and evictions using the most recently materialized scores, which keeps user-visible latency unaffected by the maintenance schedule.

B.3 Empirical Overhead

The empirical results in Table 2 already account for the full cost of ARC, including all online updates and the amortized cost of asynchronous maintenance. As a representative point, on SQuAD a direct query against the full Wikipedia index takes 1.313 s per query, whereas ARC achieves an Average Memory Access Time (AMAT) of 0.269 s, an approximately 80% reduction in latency. Across all datasets in Table 2, ARC consistently attains the lowest AMAT among the methods we compare, indicating that the additional bookkeeping introduced by hubness and DRF is more than compensated by the reduction in retrieval traffic against the external index. Each invocation of the maintenance step takes less than 0.5 s on our setup, and this cost is amortized across the queries that occur between two consecutive maintenance windows.

C Robustness Experiments

This appendix collects additional experiments that probe how ARC behaves under non-stationary or topic-focused query streams, complementing the main results in Section 5.

C.1 Sudden Domain Shift

To stress-test the adaptivity of ARC under abrupt domain changes, we construct a topic-shift query stream from MMLU. The stream first replays queries from one subject (*high school microeconomics*) and then switches instantaneously to a semantically disjoint subject (*international law*), while keeping the cache state unchanged. This setting creates a challenging transition: at the shift point, the cache has already been warmed by the previous topic and must adapt to the new topic under the same fixed capacity budget, without any explicit domain-shift detector or cache reset.

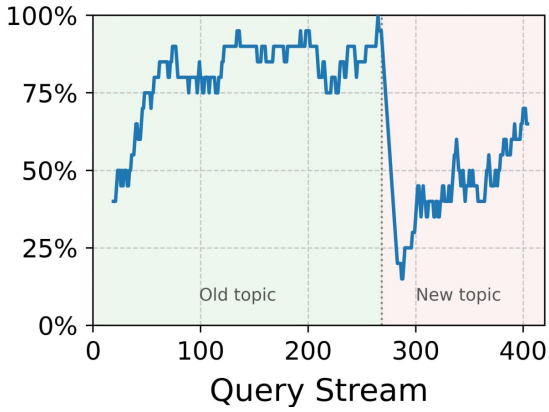


Figure 3: Hit-rate trajectory of ARC under a sudden topic shift on MMLU. The green region marks queries from the source topic (*high school microeconomics*); the red region marks queries from the target topic (*international law*) after the instantaneous switch. The curve is the rolling hit rate over a sliding window of $W = 20$ queries.

Figure 3 shows the resulting hit-rate trajectory. To smooth out per-query noise while preserving the temporal dynamics, we report a rolling hit rate computed over a sliding window of size $W = 20$. Before the shift, ARC maintains a high hit rate on the source topic, with an average pre-shift hit rate of 80.3%. Immediately after the transition, the rolling hit rate drops sharply because the cache contents are dominated by stale entries from the previous topic. ARC then quickly begins to recover as new relevant passages from the incoming topic are admitted and low-priority stale entries are evicted. The rolling hit rate rebounds from its post-shift minimum of about 15% to 55% within 100 post-shift queries, and further reaches about 65% by the end of the stream. The cumulative hit rate over the entire post-shift segment is 68.1%.

These results indicate that ARC remains robust even under a complete topic switch. Since the cache capacity is fixed, the observed recovery implies that ARC's joint DRF–hubness scoring progressively replaces outdated passages with newly useful ones, providing concrete evidence that stale entries are purged without any explicit domain-change detection.

C.2 Streaming with Topic-Focused Queries

Beyond the abrupt-shift scenario in Appendix C.1, we also evaluate ARC under a single topic-focused stream that mimics a sustained multi-turn interaction. On an MMLU subset we issue 80 consecutive queries restricted to a single Wikipedia topic and report the cumulative has-answer rate over time. As shown in Figure 2b, ARC's cumulative has-answer rate increases monotonically over the stream and stabilizes above 50% after roughly 60 queries, consistently outperforming LFU, Proximity, and all other baselines at every time step. This behavior indicates that ARC adapts efficiently to temporally correlated query streams typical of multi-turn dialogue, where consecutive queries share thematic context even if their surface forms vary.

D Hyperparameter and Design Ablations

This appendix collects additional ablations that examine ARC's sensitivity to the distance metric, priority weights, and the number of underlying query topics.

D.1 Distance Metrics

The hubness score and the DRF score both depend on a choice of distance metric on the embedding space. To verify that ARC's gains are not specific to one geometric choice, we re-run all methods under three distance metrics: cosine similarity, L_2 distance, and inner product. Table 3 reports the resulting has-answer rate and latency across MMLU, AdversarialQA, and SQuAD.

Two observations stand out. First, ARC achieves the highest has-answer rate and lowest latency across nearly all combinations of dataset and metric. The only exception is inner product on SQuAD, where the variant without hubness slightly outperforms the full ARC. Second, while the Proximity-based baselines collapse under L_2 (their fixed similarity threshold no longer separates relevant from irrelevant items), ARC degrades gracefully and retains its lead. This confirms that the gains reported in the main text are driven by the design of the cache mechanism rather than by the geometry of any particular metric.

D.2 Priority Weights (α, β)

The combined-priority formula in Section 4.2 exposes two hyperparameters: α, which controls the curvature of the distance term in DRF, and β, which balances the hubness component against the DRF component. To examine ARC's sensitivity to these choices, we sweep $\alpha \in \{0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8\}$ and $\beta \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$ on MMLU with the `bge-small-en` embedder. Table 4 reports the resulting has-answer rate for every configuration in the sweep.

Two regularities emerge from the table. First, the range of has-answer rates across the entire 7×5 grid is narrow, falling between 61.25% and 63.39%, a spread of about two percentage points. Second, every configuration in the grid outperforms the strongest non-ARC baseline (LFU at 53.47%) on this dataset by at least seven percentage points. We interpret these two observations as evidence that α and β act as soft tuning knobs rather than as critical parameters: ARC's design is robust enough that practitioners can adopt the default values used in the main text ($\alpha = 0.4$, with dataset-specific β) without further tuning, and a coarse grid search is sufficient when adapting to a new corpus.

D.3 Topic-Count Robustness

To further test that ARC adapts to diverse and previously unseen query distributions, we construct query streams that mix $k \in \{2, 3, 4, 5\}$ randomly selected topics from MMLU. For each k, the cache capacity is scaled linearly to $3k$ MB, so that all methods receive proportionally more memory as the query mixture broadens. Table 5 reports the resulting has-answer rate and latency.

ARC achieves the highest has-answer rate and the lowest latency at every value of k, with margins ranging from roughly five to fifteen percentage points over LFU and GDSF, the strongest classical baselines. The Proximity-based baselines, whose effectiveness depends on a fixed similarity threshold, fail to adapt to the randomized topic mixture and remain below 25% across the entire range of k. The contrast supports the claim that ARC's adaptivity arises from its DRF–hubness scoring rather than from any pre-computed statistics over the query distribution.

Cache	Dataset	Metric	Hit rate (%)	Latency (s)
LFU	MMLU	cosine	53.47	0.668
LFU	MMLU	L_2	53.47	0.668
LFU	MMLU	inner product	53.42	0.669
LFU	AdversarialQA	cosine	69.86	0.392
LFU	AdversarialQA	L_2	69.86	0.392
LFU	AdversarialQA	inner product	69.86	0.392
LFU	SQuAD	cosine	73.47	0.352
LFU	SQuAD	L_2	73.47	0.352
LFU	SQuAD	inner product	73.61	0.350
Proximity	MMLU	cosine	16.36	1.130
Proximity	MMLU	L_2	0.90	1.301
Proximity	MMLU	inner product	42.75	0.801
Proximity	AdversarialQA	cosine	25.25	0.970
Proximity	AdversarialQA	L_2	0.16	1.311
Proximity	AdversarialQA	inner product	55.54	0.587
Proximity	SQuAD	cosine	30.20	0.925
Proximity	SQuAD	L_2	0.43	1.309
Proximity	SQuAD	inner product	61.52	0.517
LFU + Proximity	MMLU	cosine	16.19	1.140
LFU + Proximity	MMLU	L_2	1.24	1.298
LFU + Proximity	MMLU	inner product	43.66	0.788
LFU + Proximity	AdversarialQA	cosine	22.04	1.007
LFU + Proximity	AdversarialQA	L_2	0.16	1.311
LFU + Proximity	AdversarialQA	inner product	46.55	0.704
LFU + Proximity	SQuAD	cosine	17.08	1.077
LFU + Proximity	SQuAD	L_2	0.46	1.309
LFU + Proximity	SQuAD	inner product	41.41	0.771
FIFO	MMLU	cosine / L_2 / inner product	42.77	0.806
FIFO	AdversarialQA	cosine / L_2 / inner product	68.89	0.404
FIFO	SQuAD	cosine / L_2 / inner product	78.56	0.285
ARC (w/o hubness)	MMLU	cosine	62.40	0.559
ARC (w/o hubness)	MMLU	L_2	62.35	0.560
ARC (w/o hubness)	MMLU	inner product	62.30	0.560
ARC (w/o hubness)	AdversarialQA	cosine / L_2 / inner product	70.26	0.387
ARC (w/o hubness)	SQuAD	cosine	79.00	0.279
ARC (w/o hubness)	SQuAD	L_2	79.02	0.278
ARC (w/o hubness)	SQuAD	inner product	78.99	0.279
ARC	MMLU	cosine	**63.66**	**0.543**
ARC	MMLU	L_2	**63.66**	0.546
ARC	MMLU	inner product	60.84	0.574
ARC	AdversarialQA	cosine	**71.11**	**0.377**
ARC	AdversarialQA	L_2	71.05	0.381
ARC	AdversarialQA	inner product	69.92	0.393
ARC	SQuAD	cosine	**80.80**	**0.256**
ARC	SQuAD	L_2	80.67	0.260
ARC	SQuAD	inner product	77.28	0.303

Table 3: Distance metric ablation. Has-answer rate (%) and latency (s) under cosine similarity, L_2 distance, and inner product across MMLU, AdversarialQA, and SQuAD. Rows where the three metrics yield identical numbers are collapsed for readability.

$\alpha \backslash \beta$	0.1	0.3	0.5	0.7	0.9
0.2	63.39	62.82	62.45	62.04	61.41
0.3	63.39	62.92	62.56	62.04	61.41
0.4	63.39	62.92	62.56	62.35	61.25
0.5	63.39	62.82	62.45	62.04	61.57
0.6	63.39	62.92	62.56	62.51	61.25
0.7	63.39	62.82	62.56	62.25	61.41
0.8	63.39	62.92	62.56	62.35	61.25

Table 4: Sensitivity of ARC to the priority weights α and β on MMLU (`bge-small-en`). Cells report the has-answer rate (%). All configurations fall within $[61.25, 63.39]$ %, and every configuration outperforms the strongest non-ARC baseline (LFU, 53.47%) by a wide margin.

Cache	k **/ Capacity**	**Hit (%)**	**Latency (s)**
LFU	2 / 6 MB	41.92	0.763
LFU	3 / 9 MB	73.00	0.355
LFU	4 / 12 MB	57.10	0.563
LFU	5 / 15 MB	50.16	0.655
GDSF	2 / 6 MB	41.78	0.764
GDSF	3 / 9 MB	74.70	0.332
GDSF	4 / 12 MB	57.69	0.556
GDSF	5 / 15 MB	50.64	0.648
Proximity	2 / 6 MB	14.42	1.124
Proximity	3 / 9 MB	17.27	1.086
Proximity	4 / 12 MB	14.72	1.120
Proximity	5 / 15 MB	17.44	1.084
LFU + Proximity	2 / 6 MB	17.22	1.087
LFU + Proximity	3 / 9 MB	22.47	1.018
LFU + Proximity	4 / 12 MB	17.97	1.077
LFU + Proximity	5 / 15 MB	21.36	1.033
FIFO	2 / 6 MB	30.75	0.909
FIFO	3 / 9 MB	62.18	0.497
FIFO	4 / 12 MB	46.47	0.703
FIFO	5 / 15 MB	40.70	0.779
ARC (w/o hubness)	2 / 6 MB	51.54	0.636
ARC (w/o hubness)	3 / 9 MB	79.11	0.274
ARC (w/o hubness)	4 / 12 MB	62.70	0.490
ARC (w/o hubness)	5 / 15 MB	57.78	0.555
ARC	2 / 6 MB	**51.94**	**0.633**
ARC	3 / 9 MB	**79.31**	**0.275**
ARC	4 / 12 MB	**63.19**	**0.489**
ARC	5 / 15 MB	**58.61**	**0.551**

Table 5: Performance on MMLU under query streams mixing k topics, with cache capacity scaled to $3k$ MB. ARC achieves the highest has-answer rate and lowest latency at every k, without any topic-specific tuning.