Exploring Optimal Prefetching Sizes in RaLMSpec to Enhance Retrieval-Augmented Generation Efficiency

Shuxin Liu *

College of Software Engineering, Southeast University, Nanjing, China

* Corresponding Author Email: 213220639@seu.edu.cn

Abstract. Retrieval-augmented generation (RAG) frameworks like RaLMSpec enhance language model performance by integrating external knowledge. A key method to accelerate RaLMSpec efficiency is the prefetching, which determines the number of documents to retrieve in advance to balance retrieval speed and cache utilization. This study introduces and tests both static and dynamic prefetching strategies to optimize performance in RaLMSpec. Static prefetching uses fixed sizes, while dynamic prefetching adjusts based on real-time factors including task complexity, cache hit rates, and retrieval latency. Experiments across multiple datasets, retrievers, and language models demonstrate that dynamic prefetching significantly reduces latency by 18% on average, outperforming static strategies. Dynamic prefetching adapts to varying task demands, providing better balance between retrieval and caching efficiency. Among static strategies, a prefetch size of 64 offers the best trade-off between latency reduction and cache utilization. The results highlight that dynamic prefetching is optimal for environments with fluctuating task complexity, while static prefetching with a size of 64 is effective for predictable tasks. This study provides valuable insights for improving RAG system efficiency and suggests future directions, including machine learning-based adaptations and hardware optimizations.

Keywords: Retrieval-Augmented Generation, RaLMSpec, Prefetching Optimization, Cache Utilization, Model Efficiency.

1. Introduction

In recent years, advancements in large language models such as LLaMA-2 and PaLM have shown impressive outcomes in various Natural Language Processing (NLP) tasks [1, 2]. When focusing on issues that embed vast amounts of new knowledge and various downstream tasks, substantial training resources and excessive deployment computation are required. To handle these demands, Retrieval-Augmented Language Model (RaLM) has emerged as an important approach in enhancing language models for knowledge-intensive tasks [3]. By incorporating retrieval mechanisms, RaLM can draw from external knowledge sources to supplement generated content to achieve higher accuracy and relevance in responses [4, 5]. Existing RaLM methods allow models to continuously interact with the knowledge base throughout the generation process, retrieving additional information for higher quality responses. However, repeated retrieval results in considerable overhead, posing inefficiency challenges for real-time applications.

The efficiency of RaLM depends on balancing retrieval speed and accuracy, especially for large models handling extensive and comprehensive data in real-time. RaLMSpec, a Retrieval-augmented generation (RAG) acceleration framework, has been proposed by introducing speculative retrieval with batched verification to minimize serving overhead while preserving the model output [6]. One key mechanism of RaLMSpec is its prefetching strategy, which retrieves potentially relevant documents in advance to minimize delays caused by on-demand retrieval. However, the prefetching size — the number of documents retrieved in anticipation — is a critical factor in determining the balance between cache utilization and retrieval efficiency. Properly tuned, prefetching can enhance system responsiveness and reduce redundant queries. On the contrast, unsuitable prefetching sizes can lead to inefficient memory usage and prolonged processing times.

Research on retrieval optimization has explored multiple techniques, yet studies specifically focused on prefetching size in RaLM systems are still limited. For example, Izacard & Grave integrated caching mechanisms with generative transformers in open-domain question answering,

illustrating how efficient retrieval can support low-latency generation tasks [7]. Karpukhin et al. employed dense passage retrieval to boost retrieval speed, yet the study left open the potential for prefetching optimization [8]. Khattab & Zaharia explored late interaction in ColBERT, demonstrating that optimized retrieval reduces latency but largely ignored prefetch-specific configurations [9]. Furthermore, Borgeaud et al. investigated large-scale retrieval in massive datasets and highlighted caching benefits, underscoring the lack of exploration on how prefetching size affects retrieval speed and cache efficiency [10].

To fill up the lack of research on how prefetching size affects the performance of RaLM systems, this study mainly focuses on optimizing prefetching size within RaLMSpec to improve the overall efficiency. By comparing static and dynamic prefetching strategies, this research aims to find optimal configurations that maximize cache hits and minimize latency. This study provides insights for enhancing the retrieval mechanisms of RaLMSpec, ultimately contributing to the broader field of RAG optimization.

2. Method

2.1. RaLMSpec

The RaLMSpec framework is an optimized RAG system aiming to overcome latency and efficiency drawbacks during real-time document retrievals [11]. RaLMSpec reduces the service overhead of iterative RaLMs with speculative retrieval and batched verification while ensuring the consistency of model outputs [12]. Additionally, RaLMSpec optimizes its performance with prefetching, optimal speculation stride schedulers, and asynchronous verification [6].

Speculative Retrieval: RaLMSpec anticipates and retrieves potentially useful documents before they are immediately needed. This allows the system to fetch documents based on a prediction of likely informational needs, which reduces latency and accelerates the query process of the language model.

Batched Verification: Since speculative retrieval may return data that aren't directly relevant, RaLMSpec implements batched verification to validate multiple retrieved documents in a single step. By combining verification tasks, RaLMSpec reduces overhead and total processing time, ensuring that only the most relevant content is passed to the language model.

In addition to speculative retrieval and batched verification, RaLMSpec integrates three optimization strategies that further enhance retrieval and response efficiency:

- Prefetching (P): Prefetching retrieves and caches data in advance based on anticipated needs, thus reducing latency as well as the need for real-time retrieval.
- Optimal Speculation Stride Scheduler (S): To adapt to varying informational demands, RaLMSpec dynamically adjusts the number of speculative steps based on task complexity and retrieval frequency.
- Asynchronous Verification (A): This method allows retrieval and verification to occur concurrently with model generation, further minimizing latency and enhancing the throughput of the system.

2.2. Prefetching Strategies in RaLMSpec

This study mainly focuses on specific methods for optimizing prefetch strategies, including both static and dynamic approaches. In the case of RaLMSpec, prefetching is implemented by top-k cache update method, which updates the cache with the top-k most relevant entries at each verification step, further balancing latency, accuracy, and cache utilization and eventually enhancing speculative retrieval success.

2.2.1. Static Prefetching

Static prefetching defines a fixed prefetch size. At the start of each generation task, RaLMSpec prefetches a set number of documents according to the defined prefetch size, which remains constant

throughout the task. As highlighted in the yellow area of Fig. 1, the top-k cache update fetches relevant entries to the local cache in advance per verification step, contributing to higher speculation success rate.

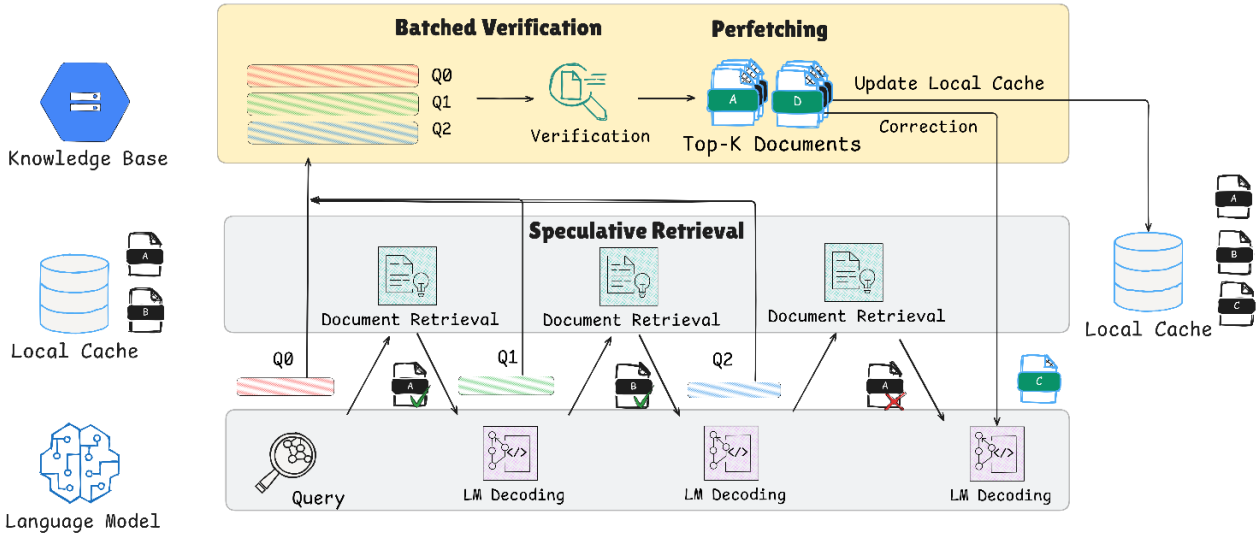

Figure 1. RaLMSpec with prefetching strategy

2.2.2. Dynamic Prefetching

Dynamic prefetching adjusts the prefetch size in real time based on task complexity, cache hit rates, and retrieval latency, allowing the system to respond adaptively to various informational needs. This approach enables RaLMSpec to optimize memory use and minimize retrieval delays, especially when handling fluctuating task demands.

Dynamic prefetching relies on three primary factors to set prefetch size adaptively. According to previous work, these factors play the most important roles in affecting the efficiency of RaLMSpec.

- Task Complexity: For complex tasks that require extensive background information, prefetch size should be increased incrementally to accommodate higher retrieval demands.
- Cache Hit Rate: A high cache hit rate indicates efficient retrieval from the existing cache, allowing the system to decrease prefetch size to conserve memory [7]. On the other hand, a low hit rate prompts an increase in prefetch size to broaden retrieval coverage.
- Retrieval Latency: When retrieval latency is low, minimizing prefetch size helps rely more on on-demand retrieval. Conversely, with higher latency, prefetch size is increased to preload more documents to reduce future delays [8].

During generation, the dynamic prefetching algorithm continuously monitors the factors above. Based on these real-time metrics, it dynamically adjusts the prefetch size to increase or decrease the number of prefetched documents as needed. This flexibility optimizes cache efficiency, enhancing the overall performance of RaLMSpec.

2.3. Experimental Setups

This section describes the experimental setups, including new prefetching details of this study, as well as datasets, language models, and retrievers of the baseline RaLMSpec structure.

2.3.1. Prefetch Implementation Details

Static Prefetching. By setting static prefetch sizes to 20, 64, 128 and 256, the experiments assess how different prefetch sizes impact latency, accuracy, and cache utilization across different tasks. Studies in baseline RaLMSpec [6] was conducted with prefetch size of 20 and 256, suggesting that the rational range of top-k cache size lies between this section. On top of that, this research tests more possibilities with moderate sizes.

Dynamic Prefetching. Prefetch sizes were adjusted dynamically in response to the three factors using the following rules. The system dynamically adjusts the prefetch size with an initial prefetch

size of 64, a maximum size of 256, and a minimum size of 10. The adjustment rules are defined as follows

1) Task Complexity

If task complexity score > 0.7, prefetch size increases by 20%. If task complexity score ≤ 0.3, it decreases by 10%. The task complexity score is calculated as: Task Complexity Score = Length of Query Tokens / Maximum Query Length. According to this, longer queries indicate higher complexity.

2) Cache Hit Rate

If cache hit rate > 80%, prefetch size reduces by 10%, prioritizing memory efficiency. If cache hit rate < 50%, it increases by 15%, improving retrieval coverage. Cache hit rate is calculated as: Cache Hit Rate = Number of Cache Hits / Number of Retrievals.

3) Retrieval Latency

If retrieval latency < 10ms, prefetch size reduces by 15%, leveraging fast on-demand retrieval. If retrieval latency > 30ms, it increases by 20%, preloading more documents to avoid delays. Retrieval latency is measured in milliseconds.

These adjustment proportions are based on experimental conclusions from similar studies. Adjustments of 20%, 15%, and 10% can significantly impact performance without causing over-adaptation or resource waste due to excessive or insufficient changes.

2.3.2. Evaluation Metrics

The experimental evaluation focuses on assessing latency, accuracy, and cache utilization. These metrics provide a comprehensive understanding of the impact of prefetching strategies on system performance.

Latency. Total latency measures the average response time for each generation task, indicating the effect of prefetching on real-time retrieval efficiency. Lower latency implies faster system response.

Cache Utilization. Cache utilization measures the proportion of retrieved documents that were actually used during speculative generation. Higher cache utilization reflects a more effective prefetching strategy.

For each dataset and model, latency, accuracy, and cache utilization for both static and dynamic prefetch strategies are recorded. Results are compared across different tasks to determine the optimal strategy under various task demands.

2.3.3. RaLMSpec Structure

Implementation Details. The maximum input prompt length is set to 512 tokens and the maximum generation length to 128 tokens. When OS3 is off, RaLMSpec uses a fixed speculation stride of s = 3. And when it’s on, it starts at s = 1 and the scheduler adapts.

Language Models. GPT2-medium, OPT-1.3B, and LLaMA-2-7B were selected to demonstrate RaLMSpec, which are not only commonly utilized as foundational language models in RaLM but also represent different model scales.

Datasets. Focus for downstream tasks is mainly on knowledge-intensive, open-domain question-answering tasks. Consequently, the experiments incorporate three QA datasets: Wiki-QA, Natural Question, and Trivia-QA.

Retrievers. For dense retrievers, the experiments use Dense Passage Retriever (DPR) as our exact dense retriever (EDR) and DPR-HNSW as our approximate methods (ADR). The BM25 retriever is adopted for sparse retrieval (SR).

3. Results

This section demonstrates a comparative analysis of static and dynamic prefetching strategies using diverse language models, retrievers, and datasets. For each dataset, 100 questions are randomly selected to test the latency and cache utilization. results are plotted over the average output of five independent runs for each setup.

3.1. Latency Analysis

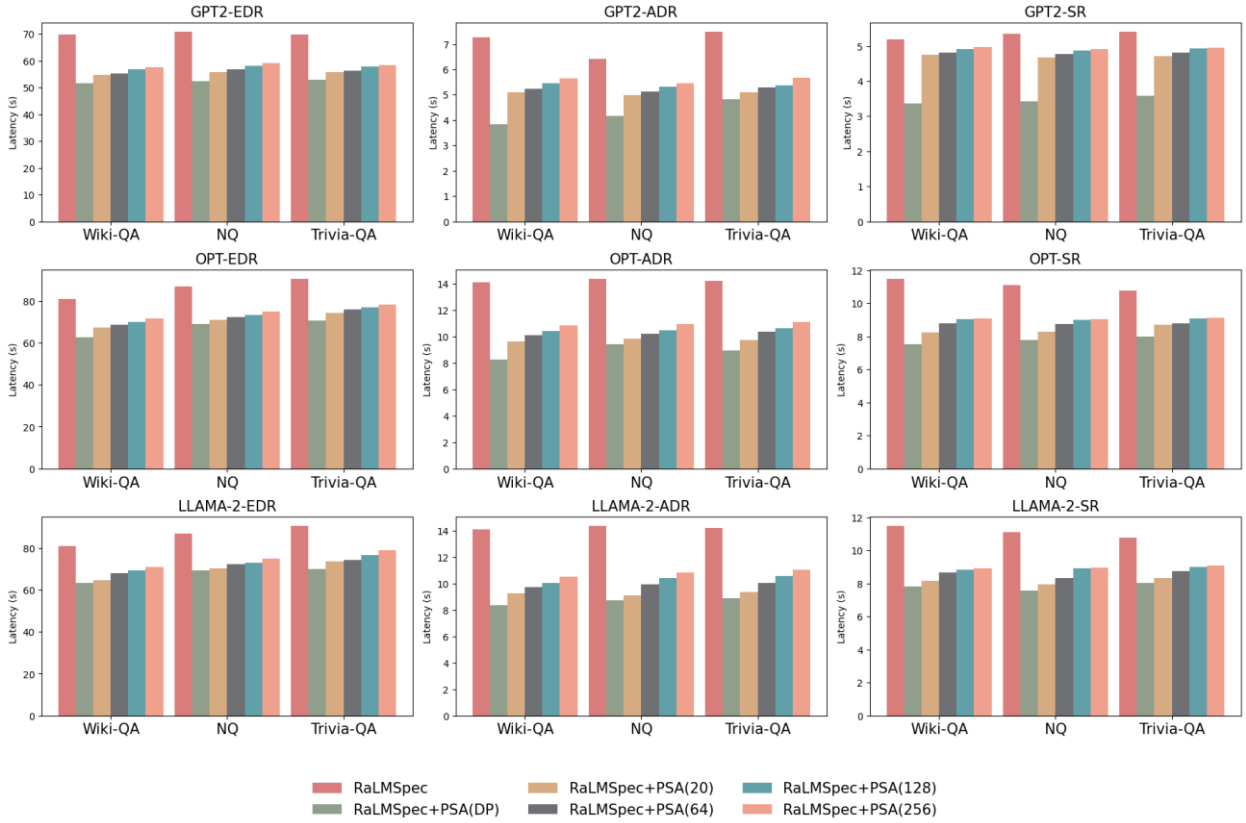

Figure 2. Latency comparison results

One of the main results is shown in Fig. 2. Latencies of RaLMSpec, and RaLMSpec+PSA with various prefetch approach on diverse language models, retrievers, and datasets are compared using bar figures. EDR, ADR, SR stand for exact dense retriever, approximate dense retriever, and sparse retriever respectively. DP stands for dynamic prefetching, and the numbers in brackets represent the prefetch sizes of static strategies.

As shown in Fig. 2, RaLMSpec+PSA(DP) achieves the best performance across all scenarios, significantly reducing latency and accelerating retrieval process. Among all datasets and retrievers, dynamic prefetching reduced latency by an average of 18% compared to static strategies with similar accuracy levels. The adaptability of the dynamic prefetching strategy is evident, with proper adjustments allowing it to achieve the lowest latency by balancing retrieval and caching.

Static prefetching strategies with different prefetch sizes all show improvements in latency reduction compared to the basic RaLMSpec. When the prefetching size increases from 20 to 256, latency increases gradually in most cases. As prefetch size grows, the large amount of prefetched data requires more loading time, occupying more memory and computational resources, leading to high latency. For instance, with a prefetch size of 256, retrieval latency was approximately 45% higher than with a prefetch size of 64.

Table 1. Speed-up ratio compared with RaLMSpec and averaged over the four datasets

Retriever	Method	GPT2	OPT	LLaMA-2
EDR	RaLMSpec+PSA (DP)	1.35x	1.29x	1.28x
	RaLMSpec+PSA (20)	1.28x	1.21x	1.25x
	RaLMSpec+PSA (64)	1.26x	1.18x	1.19x
	RaLMSpec+PSA (128)	1.23x	1.16x	1.17x
	RaLMSpec+PSA (256)	1.21x	1.13x	1.14x
ADR	RaLMSpec+PSA (DP)	1.90x	1.70x	1.68x
	RaLMSpec+PSA (20)	1.43x	1.46x	1.53x
	RaLMSpec+PSA (64)	1.39x	1.39x	1.45x
	RaLMSpec+PSA (128)	1.33x	1.35x	1.40x
	RaLMSpec+PSA (256)	1.29x	1.30x	1.34x
SR	RaLMSpec+PSA (DP)	1.54x	1.52x	1.46x
	RaLMSpec+PSA (20)	1.09x	1.39x	1.41x
	RaLMSpec+PSA (64)	1.08x	1.31x	1.33x
	RaLMSpec+PSA (128)	1.05x	1.27x	1.30x
	RaLMSpec+PSA (256)	1.04x	1.26x	1.29x

According to Table 1, improvement is most pronounced with the dynamic prefetching strategy, regardless of the retriever or model used. Smaller prefetch sizes generally yield higher improvements than larger sizes. For all retrievers, GPT2 benefits the most from RaLMSpec+PSA, reducing the most latency by 1.90× with DP.

3.2. Cache Utilization Analysis

According to the cache hit rate results in Fig. 3, RaLMSpec+PSA(DP) demonstrates the best performance among all methods as well, followed by static prefetch sizes of 64, 128, and 20, while the static prefetch size of 256 performs the worst.

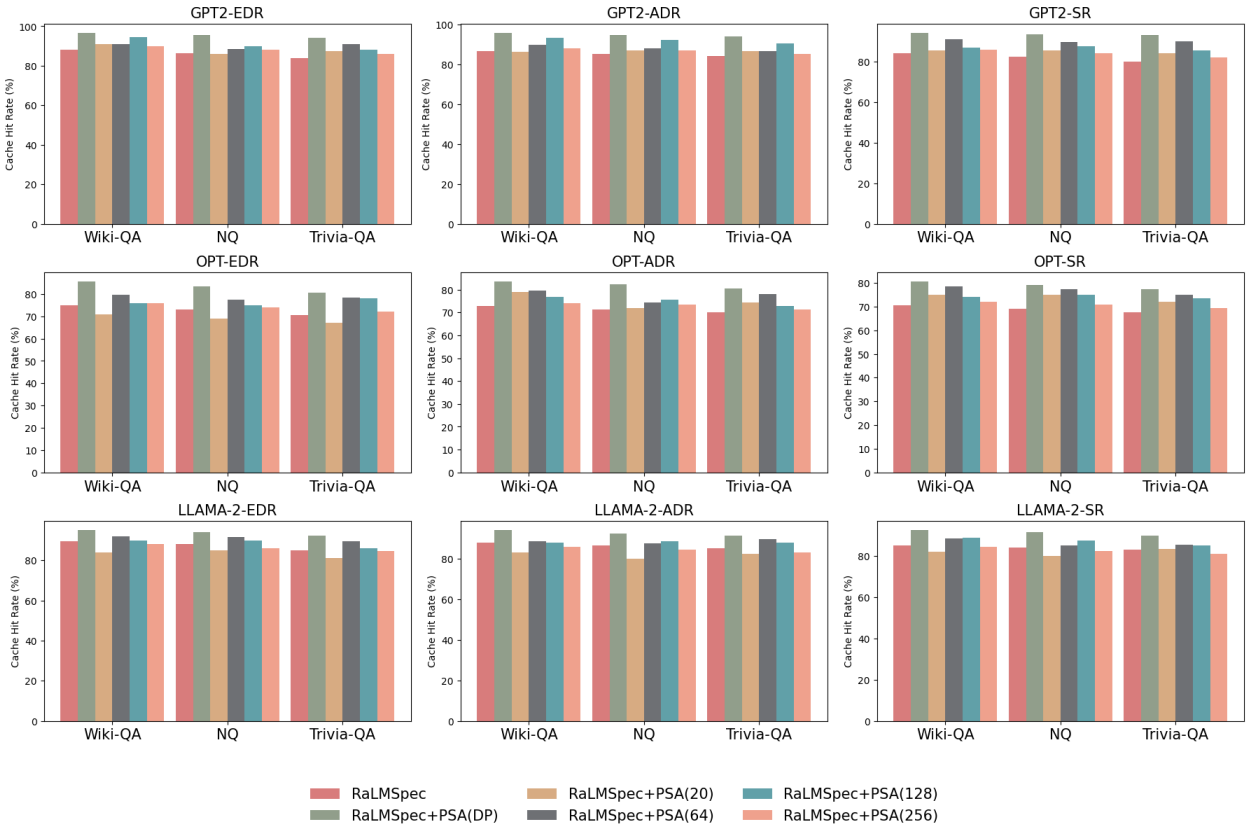

Figure 3. Cache hit rate comparison results

Dynamic prefetching minimizes cache misses by dynamically adjusting the prefetch size according to task complexity and query frequency, effectively balancing resource usage and task demands, making it the most task-optimized strategy.

The static prefetch size of 64 achieves the highest cache hit rate likely because it reaches a balance between prefetch coverage and cache utilization, providing a high cache hit while avoiding resource waste. A prefetch size too large (128 or 256) can cause the cache to fill up quickly and trigger frequent cache replacements, while too small increases the chances of cache misses, both resulting in lower cache hit rate.

4. Discussion

4.1. Optimal Prefetching Strategy

The experimental results suggest that dynamic prefetching outperforms static approaches in most scenarios. For cases requiring static prefetching, a size of 64 strikes the best balance between latency reduction and cache utilization across most retrievers and datasets.

For exact dense retrievers, dynamic prefetching reduces latency significantly while maintaining high cache hit rates, making it the preferred choice. For approximate dense and sparse retrievers, prefetch sizes of 64 or 128 are recommended for systems, as they illustrate the best trade-offs between resource usage and task performance.

4.2. Practical Recommendations for RAG Systems

To meet the needs of applications, achieving a balance between cache utilization and latency is critical. Dynamic prefetching is ideal for environments with variable task complexities, while static prefetching is more suitable for systems with predictable workloads.

Low-Latency Scenarios: Use dynamic prefetching for high-speed applications, such as conversational AI, where latency is a critical factor.

Resource-Constrained Scenarios: For applications with limited memory, prefetching sizes of 64 are optimal to ensure a high cache hit rate without excessive resource consumption.

High-Throughput Scenarios: In systems requiring processing of large batches, prefetch sizes of 128 can provide better performance by reducing cache misses.

4.3. Challenges and Future Research

Implementing dynamic prefetching is computationally intensive, requiring real-time monitoring of queries and task complexity. Additionally, tuning static prefetch sizes for diverse workloads remains a challenge, as the optimal size varies depending on model architecture.

Future work can explore advanced machine learning to predict optimal prefetch sizes, integrate hardware-level optimizations like GPU caching, and extend experiments to multilingual datasets and larger tasks for broader applicability [13].

5. Conclusion

This study introduces static and dynamic prefetching strategies in RaLMSpec framework. By analyzing their performance in reducing latency and improving cache utilization, this study confirms that dynamic prefetching is the optimal strategy across different setups. Smaller and dynamically managed prefetch sizes align best with various demands, minimizing resource waste and latency. In addition, for static strategies, a prefetch size of 64 would be the most effective, balancing performance and resource efficiency.

These findings not only provide practical insights for optimizing prefetching strategies in real-world applications, but also emphasize the broader significance for future research into intelligent prefetching mechanisms. Further research should aim to explore machine learning-driven adaptive

prefetching, hardware optimizations, and larger-scale multilingual tasks to further improve RAG systems' efficiency and scalability.

References

[1] Touvron H, Martin L, Stone K, Albert P, Almahairi A, Babaei Y, Bashlykov N, Batra S, Bhargava P, Bhosale S, ... LLaMA 2: Open foundation and fine-tuned chat models. arXiv preprint arXiv: 2307.09288, 2023.

[2] Chowdhery A, Narang S, Devlin J, Bosma M, Mishra G, Roberts A, Barham P, Chung H W, Sutton C, Gehrmann S, ... PaLM: Scaling language modeling with pathways. arXiv preprint arXiv: 2204.02311, 2022.

[3] Lewis P, Perez E, Piktus A, Petroni F, Karpukhin V, Goyal N, Küttler H, Lewis M, Yih W-T, Rocktäschel T, Riedel S, Kiela D. Retrieval-augmented generation for knowledge-intensive NLP tasks. arXiv preprint arXiv: 2005.11401v4, 2021.

[4] Khattab O, Santhanam K, Li X L, Hall D, Liang P, Potts C, Zaharia M. Demonstrate-search-predict: Composing retrieval and language models for knowledge-intensive NLP. arXiv preprint arXiv: 2212.14024, 2022.

[5] Ram O, Levine Y, Dalmedigos I, Muhlgay D, Shashua A, Leyton-Brown K, Shoham Y. In-context retrieval-augmented language models. arXiv preprint arXiv: 2302.00083, 2023.

[6] Zhang Z, Zhu A, Yang L, Xu Y, Li L, Phothilimthana P M, Jia Z. Accelerating retrieval-augmented language model serving with speculation. arXiv preprint arXiv: 2401.14021, 2023.

[7] Izacard G, Grave E. Leveraging passage retrieval with generative models for open-domain question answering. Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume, 2021: 874 - 880.

[8] Karpukhin V, Oğuz B, Min S, Lewis P, Wu L, Edunov S, Chen D, Yih W-T. Dense passage retrieval for open-domain question answering. Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP), 2020. arXiv preprint arXiv:2004.04906.

[9] Khattab O, Zaharia M. ColBERT: Efficient and effective passage search via contextualized late interaction over BERT. Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval, 2020: 39 - 48. ACM.

[10] Borgeaud S, Mensch A, Hoffmann J, Cai T, Rutherford E, Millican K, Van Den Driessche G B, Lespiau J-B, Damoc B, Clark A, et al. Improving language models by retrieving from trillions of tokens. Proceedings of the 39th International Conference on Machine Learning, 2022: 2206 - 2240. PMLR.

[11] Zhao P, Zhang H, Yu Q, Wang Z, Geng Y, Fu F, Yang L, Zhang W, Jiang J, Cui B. Retrieval-augmented generation for AI-generated content: A survey. arXiv preprint arXiv: 2402.19473v6, 2024.

[12] Guu K, Lee K, Tung Z, Pasupat P, Chang M. Retrieval-augmented language model pre-training. Proceedings of the 37th International Conference on Machine Learning, 2020: 3929 - 3938. PMLR.

[13] Asai A, Min S, Zhong Z, Chen D. ACL 2023 tutorial: Retrieval-based language models and applications. Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Tutorials), 2023. Association for Computation Linguistics.