TeleRAG: Optimizing Retrieval for Retrieval-Augmented Generation

Madhav Kashyap

A thesis
submitted in partial fulfillment of the
requirements for the degree of

Master of Science

University of Washington

2025

Committee:
Gina-Anne Levow
Anant Mittal

Program Authorized to Offer Degree:
Linguistics

University of Washington

Abstract

TeleRAG: Optimizing Retrieval for Retrieval-Augmented Generation

Madhav Kashyap

Chair of the Supervisory Committee:
Gina-Anne Levow
Department of Linguistics

Retrieval-augmented generation (RAG) has become essential for grounding large language models with external datastores to enhance factual correctness and domain coverage. But deployment presents a critical challenge: large language models and vector datastores compete for limited GPU memory, often forcing datastores to the CPU and leading to slow CPU-based retrieval latency.

This thesis introduces TeleRAG, a system that resolves this bottleneck through lookahead retrieval, a technique that predicts and prefetches likely-needed vector search data concurrently with large language model inference. We discover that queries at different RAG pipeline stages exhibit semantic overlap, enabling effective predictive prefetching.

TeleRAG combines lookahead retrieval with profile-guided prefetching optimization and GPU-CPU cooperative search. Evaluation across six RAG pipelines demonstrates 1.53× average latency reduction on consumer GPUs and 1.83× throughput improvement in batched-query scenarios. Crucially, TeleRAG remains framework and algorithm agnostic, enabling immediate deployment in existing production systems. By bridging CPU and GPU retrieval, TeleRAG enables efficient RAG deployment for both latency-sensitive and high-throughput applications, advancing retrieval-augmented across diverse environments.

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGMENTS

I wish to express my sincere appreciation to my thesis advisor, Dr. Gina-Anne Levow, for her patient guidance on the spoken and unspoken rules of academia and thesis writing. Her mentorship has been invaluable in shaping my research and writing process.

I am also deeply grateful to Dr. Chien-Yu Lin for the opportunity to collaborate with him at the SAMPL Lab within the Paul G. Allen School of Computer Science Engineering, University of Washington. The research conducted here forms the core of this thesis.

I offer my sincerest thanks to Dr. Anant Mittal for agreeing to serve as a thesis reader on short notice.

Finally, thanks to my parents, brother and friends for (sometimes reluctantly) being a sounding board for many ideas that became a part of this work.

Chapter 1

INTRODUCTION

1.1 The Challenge of Modern Retrieval Augmented Generation Systems

Retrieval-augmented generation (RAG) has emerged as a transformative technique that extends the capabilities of large language models (LLMs) by integrating them with external knowledge sources [60, 31, 14]. By retrieving the relevant information from external datastores during inference time, RAG systems can help address the critical limitations of standalone LLMs, including hallucinations [54, 68, 82], and the inability to access up-to-date or domain-specific information [39, 69]. This capability has made RAG a practical solution for applications such as customer chatbots [8, 15, 21], financial analysis [66, 71], autonomous driving [19, 103], and emergency medical diagnosis [33, 3].

Modern RAG applications exhibit two characteristics that create significant system-level performance challenges. First, they are constructed as modular pipelines where a single query undergoes multiple rounds of LLM generation and retrieval operations (query transformation [29, 108], retrieval reranking [88], post-retrieval response generation [12], summarization [42], and execution flow judgment). Second, datastores are growing increasingly larger, up to billions of tokens and hundreds of gigabytes in memory; with recent evidence demonstrating that larger datastores consistently improve application accuracy [17, 54, 85]. These characteristics impose significant latency and memory requirements on RAG systems, particularly when deployed in resource-constrained environments or latency-sensitive scenarios.

The primary bottleneck arises from the competing memory demands of the LLM generation and vector search retrieval components. While GPU acceleration can dramatically reduce both LLM generation and retrieval latency, hosting both components simultaneously often exceeds available GPU memory requirement resources. For instance, deploying a repre-

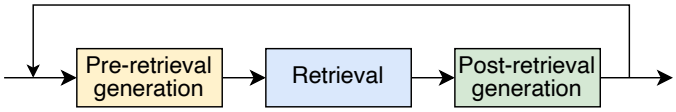
sentative question answering RAG system using a Llama-3-8B model (16 GB) [7] alongside a 61 GB FAISS vector index [27] requires approximately 77 GB of GPU memory, far exceeding the capacity of consumer GPUs like the RTX 4090 (24 GB) [73]. Consequently, in local RAG deployments, the retrieval datastore is typically offloaded to the CPU memory in order to free up the GPU memory for LLM generation. However, CPU-based retrieval is 3.87-5.96× slower than GPU-based retrieval. This results in the retrieval phase dominating 41-60% of the total pipeline latency - clearly a major performance bottleneck.

This memory-latency tradeoff extends beyond local deployments and even into large-scale data centers [1, 2, 11]. In serving scenarios where multiple concurrent requests are processed, allocating GPU memory to the retrieval datastore directly reduces memory available for the Key-Value cache of the LLM serving system, thereby limiting effective batch sizes and overall throughput [58]. Existing approaches to accelerate RAG inference - including hardware accelerators for retrieval [45, 81], KV-cache reuse [48, 64, 101], and speculative document retrieval [95] - fail to address the fundamental memory bottleneck required to accelerate retrieval during RAG.

1.2 Our Solution: TeleRAG with Lookahead Retrieval

A straightforward way to reduce RAG latency is to allow the GPU to accelerate both the retrieval and LLM generation. This thesis presents TeleRAG, an efficient RAG inference system that achieves low latency while minimizing GPU retrieval memory requirements using a novel lookahead retrieval mechanism (See Figure 1.1).

The core insight underlying our approach is that modern RAG pipelines exhibit substantial semantic overlap between queries at consecutive pipeline stages (specifically between queries before and after the pre-retrieval stage). Consequently, the data-clusters relevant to the initial input query retrieval have a high overlap with the data-clusters relevant to the transformed query retrieval. Lookahead retrieval exploits this observation by proactively prefetching likely-needed data-clusters from CPU to GPU memory during the pre-retrieval generation stage, effectively hiding data transfer latency behind concurrent LLM computa-



(a) Typical pipeline stages of a RAG application.

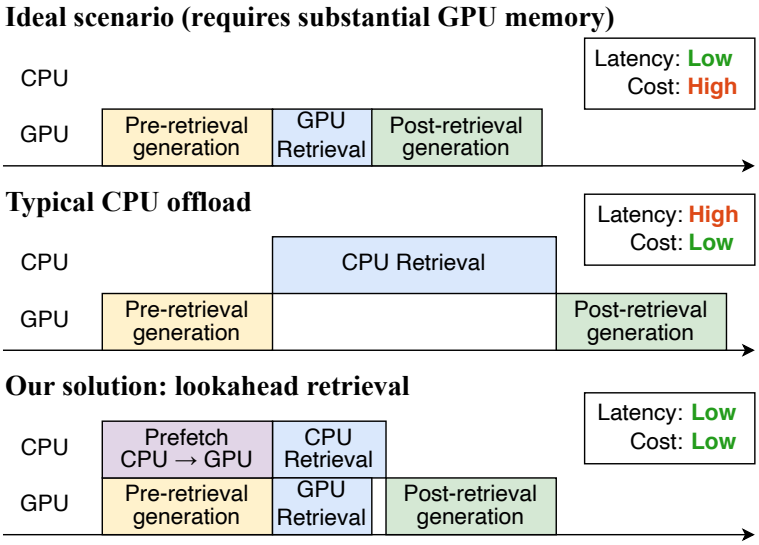


Figure 1.1: Illustration of TeleRAG's lookahead retrieval mechanism and comparison to the typical CPU offload of conventional RAG systems. We accelerate retrieval by prefetching relevant retrieval data from CPU to GPU, overlapping data transfer with the pre-retrieval stage. [61]

tion. The system then performs GPU-accelerated search on the successfully prefetched clusters while simultaneously performing CPU search on the missed dataclusters. The prefetched dataclusters are significantly smaller than the entire datastore, thereby allowing the GPU to simultaneously host the LLM generation component and the retrieval component with minimal memory overhead. The retrieval stage can now be GPU-accelerated, resulting in significant decrease in end-to-end RAG latency.

1.3 Thesis Outline

This thesis presents the following key contributions:

1. **Analysis of RAG Bottlenecks:** We provide a detailed systems analysis of representative modern RAG pipelines. We prove that CPU-GPU data transfer is the primary bottleneck for low-latency inference in low memory conditions.

2. **Insight into Query Similarity and Data-Cluster Overlap:** We identify and empirically validate the high semantic overlap between queries in different stages of a RAG pipeline, revealing a significant opportunity for predictive data prefetching.

3. **The Lookahead Retrieval Mechanism:** We propose a novel prefetching mechanism that leverages this query similarity insight to hide data transfer latency by overlapping it with LLM generation.

4. **The TeleRAG System:** We design and implement an end-to-end, high-performance RAG inference system that integrates lookahead retrieval with GPU-CPU cooperative search and an optimal prefetching strategy.

5. **Comprehensive Evaluation:** We conduct extensive empirical evaluations demonstrating that TeleRAG achieves up to a 1.53× reduction in end-to-end latency for single-query inference and up to a 1.83× improvement in throughput for batch processing, while still operating within strict GPU memory constraints (See figure 5.2).

This work was conducted in collaboration with the SAMPL lab, Paul G. Allen School of Computer Science & Engineering at the University of Washington. I am grateful to the multiple team members whose contributions were instrumental in bringing this research to fruition, especially Dr. Chien-Yu Lin. My role encompassed three primary areas: first, I was involved in predicting that semantic similarity during query transformation would be a promising opportunity for lookahead retrieval optimization; second, I constructed the

RAG pipelines by implementing designs from their respective research papers and official documentation; and third, I developed a rigorous framework for quantifying and analyzing the IVF cluster overlap rate across diverse experimental configurations, enabling objective assessment of system performance under varying conditions.

TeleRAG has been made available on arXiv [61], making the work publicly available to the research community. The paper was submitted for review at the MLSys 2026 conference, where it will be considered alongside other systems research addressing modern machine learning infrastructure challenges.

Chapter 2

BACKGROUND AND LITERATURE SURVEY

Retrieval augmented generation has evolved from a simple retrieve-then-generate pattern into a sophisticated ecosystem of modular pipelines, optimized vector stores, and memory-aware serving systems. As large-scale and complex RAG systems have started to become productionized, the systems challenges of RAG have also garnered significant attention. This literature survey collates prior work across multiple domains: RAG architectures, approximate nearest neighbor search, LLM serving with GPU memory management, and recent acceleration techniques that target RAG efficiency. The chapter emphasizes how TeleRAG's lookahead retrieval mechanism complements, rather than replaces, these advances by addressing a distinct CPU-GPU data transfer latency bottleneck not addressed by any other work.

2.1 RAG

RAG is a technique that enhances the capabilities of LLMs by integrating them with information retrieval to generate more accurate answers [14, 31, 60]. The core idea behind RAG is to augment the LLM with relevant information retrieved from a large corpus of documents, improving the LLM's ability to answer questions without hallucinations, generate contents based on private and up-to-date information, and provide verifiable evidence for answers [54, 68, 39]. The foundational RAG paradigm fused dense passage retrieval with sequence-to-sequence generation in order to ground LLM outputs in relevant external knowledge sources [60]. This basic pattern — retrieve relevant information chunks, concatenate them with the query, then generate - remains the backbone of modern RAG systems, but has been superseded by RAG architectures that interleave multiple LLM and retrieval stages.

2.2 RAG Pipelines

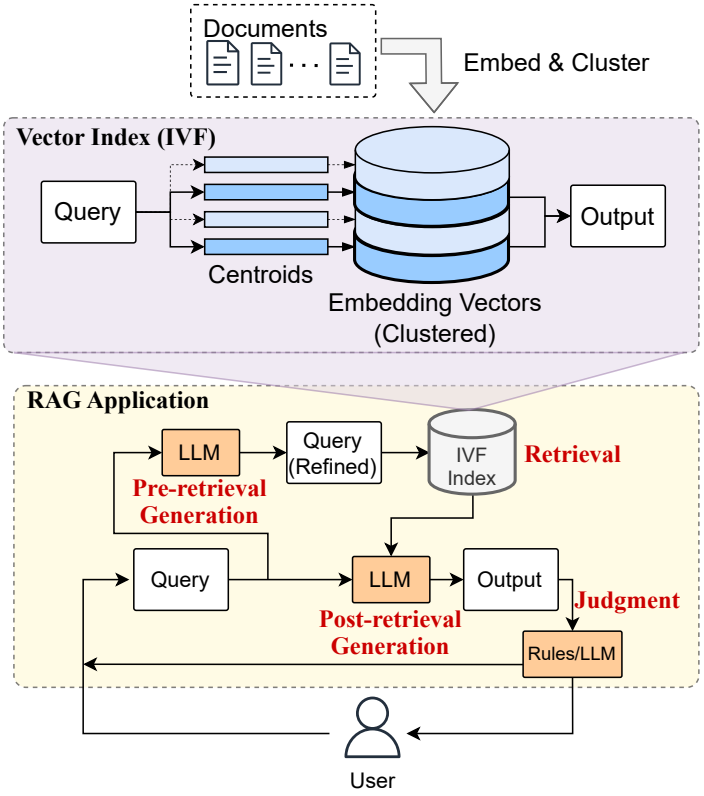


Figure 2.1: Generalized modular RAG pipeline consisting of steps: (a) pre-retrieval generation, (b) retrieval, (c) post-retrieval generation, (d) judgement. [61]

In advanced modular RAG pipelines, systems often execute multiple stages of query transformation, retrieval, filtering, and generation (See Figure 2.1). Rather than a single retrieve-then-generate step, these workflows chain together multiple sub-components, such as query rewriting, sub-question generation, summarization, and stepwise information retrieval, to better tackle reasoning-intensive queries or complex information needs [12, 47, 65, 86, 29].

In each iteration, the retriever processes a modified query to fetch relevant document chunks, and the generator contextualizes the information. The downstream judgement logic might then again iteratively refine and verify outputs. The iterative nature of these modular

pipelines increases both latency and memory pressure since multiple rounds of LLM generation and retrieval are required for a single user query. Moreover, as each query might be subject to rewriting or additional context, the set of document chunks fluctuates between stages, complicating both engineering and systems optimizations.

A generalized modular RAG pipeline has the following steps (See Figure 2.1):

- **Pre-retrieval Generation:** This step decides whether further retrieval is needed and to generates queries for future retrieval. An example of a pre-retrieval technique is query transformation [40, 62, 77, 102, 110], which reformulates the original query to make it clearer and more suitable for the retrieval task. Here is an example query transformation which decomposes the original user query into multiple easier to answer subqueries:

  "What are the impacts of climate change on the environment?"
  $\xrightarrow{\text{pre-retrieval generation}}$

  1) "What are the impacts of climate change on biodiversity?"
  2) "What are the effects of climate change on agriculture?"
  3) "What are the impacts of climate change on human health?"

- **Retrieval:** Information retrieval from the datastore to identify relevant documents.

- **Post-retrieval Generation:** Generates an LLM response based on the user query and retrieved documents. Additional processes such as summarization [55] or reranking [88] can also be performed. Here is an example of post-retrieval generation generating a response on a user query.

  "What are the impacts of climate change on the environment?"
  $\xrightarrow{\text{post-retrieval generation}}$

  "Climate change is causing widespread and accelerating impacts across Earth's environmental systems, from rising temperatures and sea levels ... "

- **Judgment:** Control the the RAG execution flow based on dynamic factors. For example heuristics can be employed to determine if more retrieval steps are needed, or if the LLM answer is good enough.

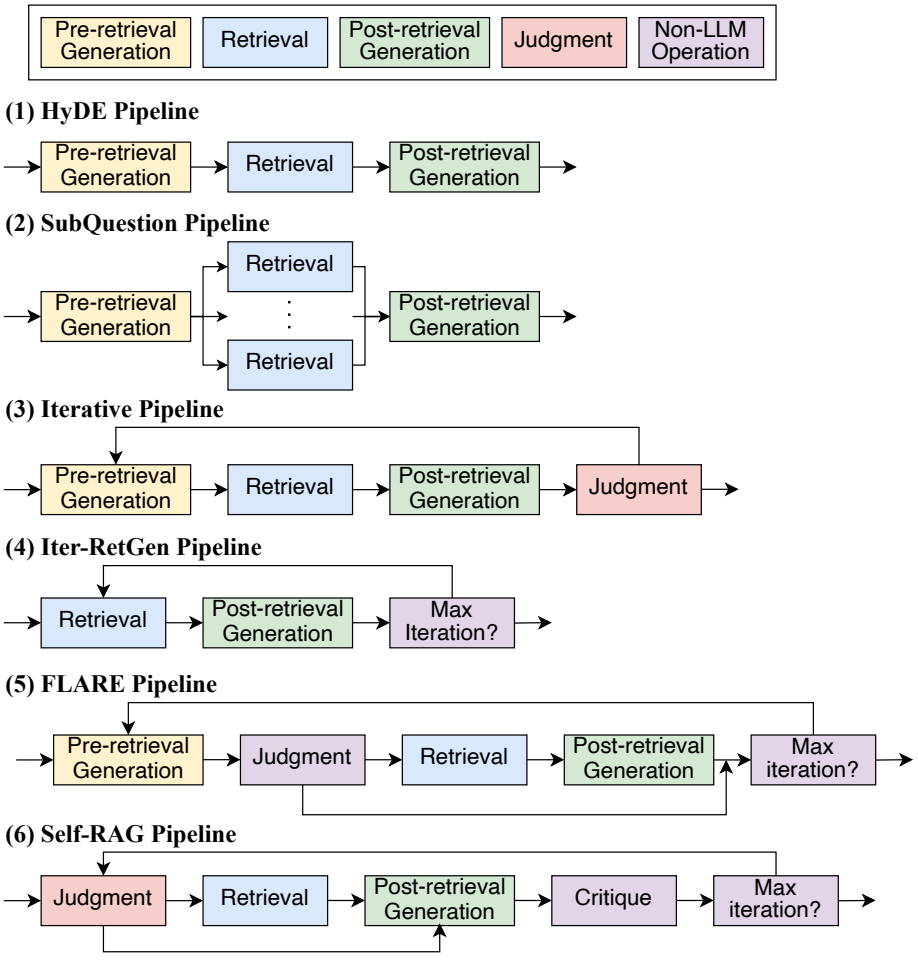


Figure 2.2: RAG pipelines we evaluate on [61]: (1) HyDE pipeline [29], (2) SubQuestion pipeline [65], Iterative pipeline [62], Iter-RetGen pipeline [86], FLARE pipeline [47], Self-RAG pipeline [12].

Existing research on RAG pipelines focuses on maximizing retrieval accuracy, often at the cost of added pipeline complexity. We evaluate TeleRAG on a set of 6 industry-standard diverse pipelines (See Figure 2.2).

- **HyDE** [29] prompts LLM to generate a hypothetical paragraph and perform retrieval based on the embedding of the generated paragraph. (See Figure 2.2.1)
- **SubQuestion (SubQ)** [65] prompts LLM to generate multiple sub-questions and performs retrievals for each generated sub-question. (See Figure 2.2.2)
- **Iterative (Iter)** [62] prompts LLM to generate narrower questions first and iteratively refine them based on previous answers. At the end of each iteration, it prompts LLM to judge if the answer is good enough. (See Figure 2.2.3)
- **Iter-RetGen (IRG)** [86] iteratively do retrieval and LLM generation for 3 iterations. (See Figure 2.2.4)
- **FLARE** [47] iteratively issues retrieval requests based on the confidence (probability score) of predicted tokens for the upcoming sentence. (See Figure 2.2.5)
- **Self-RAG (S-RAG)** [12] uses the LLM as judge for retrieval, generate responses, and self-critique on the responses. We use fine-tuned models based on Llama2-7B and Llama2-13B from their official repository [12] (See Figure 2.2.6)

2.3 Vector Index and Inverted File (IVF)

RAG systems need to support semantic similarity search on large-scale datastores, often containing millions or billions of document chunks. Most RAG deployments rely on vector search libraries such as Faiss [27] for efficient retrieval which leverage approximate nearest neighbor (ANN) search to balance recall and latency. The Faiss library provides a highly optimized GPU implementation and supports multiple indexing schemes, including HNSW

(Hierarchical Navigable Small World) index [67], PQ (Product Quantization) index [41], and the widely-used IVF (Inverted File) index [50].

IVF initially indexes the vector space into a collection of disjoint clusters, each associated with a centroid. At retrieval time, the query vector is first mapped to its nearest centroids through a coarse-grained search, which is followed by a local scan within the corresponding clusters to identify the top-k nearest neighbors. This two-stage design radically reduces memory footprint and computational overhead compared to brute-force Exact Nearest Neighbor search, enabling billion-scale semantic search on low-latency hardware [27, 41]. IVF indexes also support multiple quantization strategies, and dynamic growth/shrinking of datastores valuable for continuously updated knowledge-intensive RAG applications. Self-RAG and HyDE often use large-scale IVF indices to manage iterative retrieval rounds [12, 29]. Evaluation on large QA datasets consistently demonstrates that larger and better-clustered indexes yield higher recall and factuality in downstream RAG pipelines. However, scaling the index up creates acute memory and latency constraints for any system trying to serve LLMs and retrieval on the same GPU, resulting in retrieval being migrated to slower CPU-side execution and sharply increasing latency [27, 58].

Alternative ANN libraries like DiskANN [89], HNSW [67], and SPANN [22] aim to balance retrieval speed and memory for even larger datastores. DiskANN, for instance, leverages SSD storage and parallel posting lists to support billion-scale retrieval on commodity hardware. HNSW constructs multi-layer graphs to enable high-recall, ultra-fast search, though it consumes larger memory for index construction. These libraries vary in their support for GPU acceleration, index update dynamics, and integration with real-time RAG applications.

TeleRAG builds on the structure of IVF indices and recent ANN optimizations but specifically addresses the practical gap in efficient CPU-GPU cross-device retrieval. Prior work in efficient indexing and ANN search has largely focused on optimizing retrieval accuracy and computational efficiency, not on minimizing system-wide memory usage and hiding CPU-GPU data transfer latency. TeleRAG analyzes the semantic overlap between queries in modular RAG pipelines, demonstrating that pre-retrieval and retrieval queries hit largely overlap-

ping IVF clusters, and therefore predictively prefetches only the clusters most likely needed into the GPU during ongoing LLM generation. This retrieval-agnostic, index-compatible strategy allows TeleRAG to support any billion-scale vector datastore framework (such as Faiss). The added benefit is the latency benefits of full GPU-based retrieval. This approach does not require tuning the index beyond industry-standard practices, nor does it conflict with retriever algorithms or quantization strategies, making it immediately deployable across diverse RAG stacks.

In summary, TeleRAG not only leverages the strengths of mature vector indexing libraries such as Faiss, but it extends them by introducing cross-device concurrency and predictive transfer techniques that directly overcome real-world memory and latency barriers; a step beyond what traditional accelerated indexing and ANN libraries achieve.

2.4 Optimizations for RAG

A growing body of work has targeted RAG inference acceleration, employing diverse strategies spanning hardware accelerators, KV-cache reuse, and speculative retrieval. These systems aim to reduce the high computational and memory costs associated with modern RAG pipelines, yet each addresses a distinct subset of the overall latency problem. TeleRAG fundamentally differs from prior systems by targeting the CPU-GPU data movement bottleneck rather than computation reuse or algorithmic speculation.

KV-caching is an LLM inference optimization technique that stores the computed key and value vectors from previous tokens to speed up the generation of new tokens. RAGCache [48] is a multilevel dynamic caching system that organizes KV caches from the datastore in a knowledge tree. It introduces a replacement policy aware of LLM inference characteristics and RAG retrieval patterns, and dynamically overlaps retrieval and inference steps to minimize end-to-end latency. RAGCache reduces time-to-first-token (TTFT) and improves throughput compared to baseline RAG. Similarly, TurboRAG [64] precomputes KV caches from the data store, but only optimizes prefill latency. CacheBlend [101] introduces a selective KV-cache fusion technique for RAG to reuse pre-computed caches. While these systems

optimize KV-cache and computation reuse, they assume the retrieval index can be accessed efficiently—either on GPU or with negligible transfer cost. TeleRAG makes no such assumption and instead prefetches predicted IVF clusters during LLM generation, overlapping data transfer with computation to hide latency.

Speculative methods leverage task decomposition to accelerate retrieval and generation. Speculative RAG [95] introduces generation by smaller specialist LLMs to reduce RAG latency, but it does not target the retrieval latency. Similarly, RALM-Spec [106] speculatively retrieves documents to overlap retrieval with generation, but its focus is on algorithmic-level speculation rather than system-level data prefetching across the CPU-GPU boundary. These approaches accelerate the generation or retrieval algorithm itself but do not target the inter-CPU-GPU transfer latency. Unlike speculative retrieval methods that modify the retriever framework, TeleRAG is retrieval-agnostic and index-compatible: it works atop existing IVF-based indices (e.g. Faiss) without altering retrieval algorithms. This makes it immediately deployable on existing RAG stacks.

Hardware accelerators for retrieval have also been proposed. Chameleon [45] and Intelligent Knowledge Store [81] explore custom ASICs and FPGA-based designs to accelerate vector search itself. These approaches require specialized computing hardware and do not address the memory capacity constraints that force indices off-GPU in the first place.

2.5 Conclusion

The literature survey reveals that TeleRAG addresses a distinct and underexplored problem space. Existing optimizations focus on either improving algorithmic efficiency or reducing computational load. However, none fundamentally resolve the memory-latency tradeoff that forces RAG datastores off-GPU in resource-constrained settings. Having established the motivating problem and positioned our work within the research landscape, the next chapter focuses on a preliminary empirical analysis of real RAG pipelines to quantify the bottlenecks and identify the specific optimization opportunities.

Chapter 3

PRELIMINARY ANALYSIS

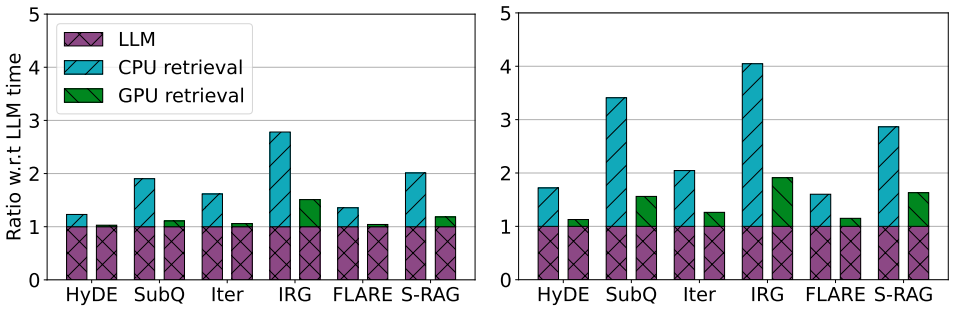
In this chapter, we analyze various state-of-the-art RAG pipelines and identify the elemental systems challenges in achieving low latency. We specifically examine three critical dimensions. First, we examine the end-to-end RAG latency breakdown to establish the magnitude of the retrieval bottleneck. Second, we examine GPU based retrieval constraints, demonstrating why straightforward GPU-accelerated solutions are infeasible. Third, we examine the semantic similarity between queries in subsequent pipeline stages, and its expression as high IVF cluster overlap. Through this empirical analysis, we quantify concrete performance gaps and identify an actionable optimization opportunity: leveraging query semantic similarity to prefetch likely-needed data clusters during LLM generation.

Comprehensive details about our experimental setup are detailed in Section 5.1. For this analysis section, the important details are the following. We use the wiki_dpr datastore [52] to construct a 61 GB Faiss IVF index [27], set the IVF clusters to 4096, and the nprobe to 256 following common benchmarking practice [12]. We use Llama-3-8B [7] as the LLM, and run our analysis on RTX 4090 (24 GB memory) [73] and H100 (80 GB memory) [74] GPU hardware.

3.1 End-to-End Latency of RAG Pipelines

We first characterize the end-to-end latency of the six representative RAG pipelines (described in Section 2.2) in two distinct scenarios:

- CPU-based retrieval: The LLM operates on the GPU while retrieval is offloaded to the CPU, thereby minimizing GPU memory consumption.



(a) Batch = 1. **(b)** Batch = 4.

Figure 3.1: Contrasting latency RAG pipelines with batch size 1 versus batch size 4. The six RAG pipelines are HyDE, SubQuestion (SubQ), Iterative (Iter), Iter-RetGen (IRG), FLARE, and Self-RAG (S-RAG). The experiment is conducted on the Natural Questions (NQ) dataset using the H100 GPU, and the nprobe is set as 256. [61]

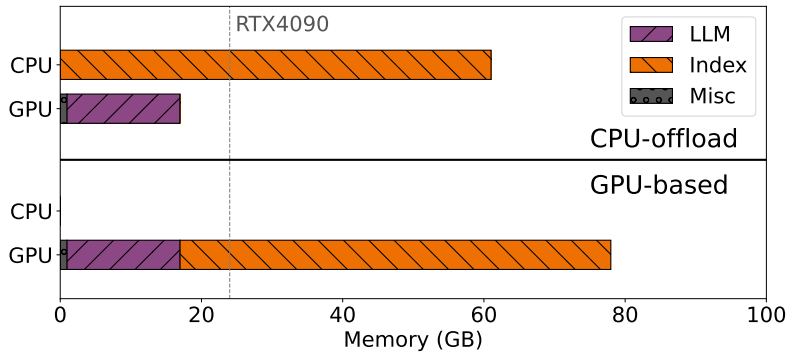


Figure 3.2: The breakdown of memory consumption at GPU and CPU for two different strategies: CPU offloading and GPU-based retrieval. The dotted line indicates the memory capacity of a RTX4090 GPU, which is commonly used for local deployment. [61]

- GPU-based retrieval: Both the LLM and the vector index reside in GPU memory, enabling GPU-based retrieval with lower latency.

The analysis reveals a striking disparity between the two scenarios. Figure 3.1 illustrates the breakdown of end-to-end latency, separating the LLM inference component from the retrieval component, computed over 1024 randomly sampled queries from the Natural

Questions dataset [57]. In the CPU-based retrieval baseline (Scenario 1), the retrieval phase emerges as the dominant latency factor, consuming 41.1% and 60.5% of total latency at batch sizes 1 and 4 respectively. Conversely, GPU-accelerated retrieval (scenario 2) substantially alleviates this bottleneck, accounting for only 10.5% and 28.3% of latency under comparable configurations. On average, GPU retrieval is 5.96× faster at batch size 1 and 3.87× faster at batch size 4 compared to CPU retrieval; which translates to end-to-end RAG latency reductions of 1.5× and 1.8× respectively. This analysis unambiguously demonstrates that GPU-accelerated retrieval is critical for improving end-to-end RAG pipeline latency.

However, GPU acceleration introduces a significant memory cost. Figure 3.2 reveals the memory requirements for both GPU and CPU storage scenarios. Hosting both LLM weights and the retrieval index on the GPU requires approximately 77 GB of memory, which exceeds the capacity of consumer-grade GPUs such as the RTX 4090, which provides only 24 GB. Consequently, GPU acceleration for retrieval is frequently infeasible when deploying on lower-end GPUs or when indexing larger datastores that can exceed hundreds of gigabytes in size.

Even on GPUs with sufficient capacity to accommodate the entire index (e.g. H100), the index's memory footprint imposes constraints on throughput in data center serving scenarios. Batched inference maximizes throughput but is inherently bottlenecked by the KV cache capacity required for LLM serving [58, 12]. In the example above, storing the index on the GPU leaves minimal memory for the KV cache (3 GB, or 20k tokens worth), restricting batch size.

These findings motivate a fundamental question: Is it possible to achieve the latency benefits of GPU-based retrieval while utilizing substantially less GPU memory? This question becomes increasingly urgent as RAG deployments expand to both resource-constrained consumer hardware and memory-intensive multi-tenant data center environments.

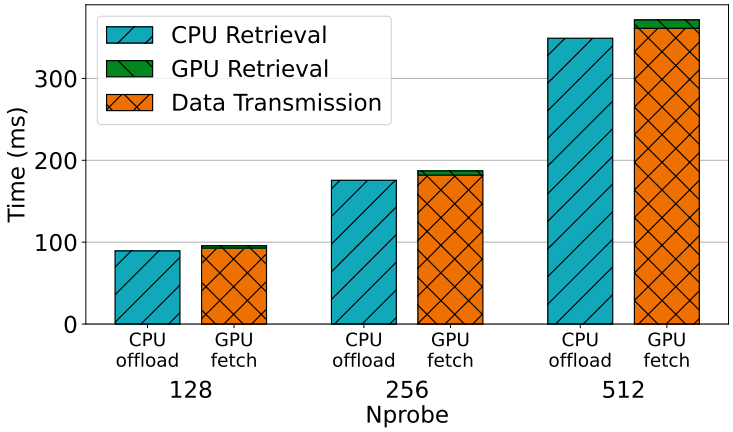


Figure 3.3: Latency breakdown for two different strategies - CPU offloading and GPU-based retrieval - over nprobe values 128, 256 and 512. We use the RTX4090 GPU and average the results over 512 random Natural Questions queries. [61]

3.2 GPU-accelerated Retrieval with Limited Memory

A direct approach to enable GPU-based retrieval under tight memory constraints is to retrieve necessary data from CPU to GPU on-demand for each query, exploiting the IVF index structure to restrict and fetch selective clusters. While this strategy facilitates faster searching on the GPU, data fetching I/O emerges as the critical latency bottleneck.

Figure 3.3 compares the latency of the on-demand GPU-fetching system against pure CPU retrieval system, evaluated across three distinct nprobe values on the RTX4090 GPU. nprobe is an IVF hyperparameter that determines how many vector clusters are exhaustively searched. The higher the nprobe, higher the recall, higher the search latency, and higher the CPU-GPU data transfer amount (and vice-versa). The results reveal that GPU fetch time dominates overall latency due to the limited PCIe bandwidth (data transfer rate) between CPU and GPU, which is constrained to 32 GB/s. Although GPU-based IVF search is substantially faster (4× - 10× faster) than its CPU counterpart [27], the transfer overhead negates this advantage, resulting in end-to-end latency that is approximately 3% slower on

average compared to CPU-based retrieval. Consequently, realizing any meaningful speedup via a GPU-accelerated approach necessitates effectively hiding data fetching latency.

To conceal data fetch costs, CPU-to-GPU transfer must be initiated prior to retrieval execution, which demands accurate prediction of data access patterns. Fortunately, modern modular RAG pipelines provide a critical hint for this prediction: the semantic similarity of the query with respect to the previous pipeline step's query.

3.3 Overlapping of IVF Clusters

Dataset	HyDE	SubQ	Iter	IRG	FLARE	S-RAG
NQ	73.1%	63.2%	91.5%	83.8%	79.1%	-
HotpotQA	75.3%	62.5%	89.6%	89.4%	80.2%	-
TriviaQA	73.1%	61.6%	86.2%	86.1%	81.7%	-

Table 3.1: IVF cluster overlap rate between the input (q_{in}) and output (q_{out}) of the pre-retrieval generation. Self-RAG is left blank as it does not incorporate query transform.

Although the exact IVF clusters required for retrieval are known only after the pre-retrieval generation phase concludes, we observe substantial semantic similarities in IVF cluster assignments between queries at different pipeline stages.

During the pre-retrieval phase of RAG pipelines, an LLM call transforms an initial user query q_{in} into a refined query q_{out}, which subsequently serves as the retrieval query. These query transformations [40, 62, 77, 102, 110] rewrite the query in alternative formats which intuitively preserve the core semantic meaning of the original query. For example the Query Decomposition [20] transformation decomposes a query into multiple simpler queries that are individually easier to retrieve (e.g. q_{in}: "Who has more siblings, Jamie or Sansa?" $\rightarrow$ q_{out}: {"How many siblings does Jamie have?", "How many siblings does Sansa have?"})

Consequently, the embedding vectors of q_{in} and q_{out} are likely to exhibit high semantic similarity and vector similarity. This vector similarity implies that the IVF clusters assigned

during retrieval to these queries will also be similar, and will substantially overlap. This renders q_{in} a valuable predictor for anticipating the cluster memory access pattern of q_{out}.

C_{in} and C_{out} are the IVF clusters corresponding to the input query q_{in} and output transformed query q_{out} respectively. The cluster overlap rate measures how well the clusters predicted using the q_{in} match the actual clusters required for q_{out}

$$\text{cluster overlap rate} = \frac{|C_{in} \cap C_{out}|}{|C_{out}|} \tag{3.1}$$

To validate this hypothesis, we measure the average cluster overlap rate (Equation 3.1) between prefetched clusters and the clusters required for actual retrieval across three prominent question-answering datasets - Natural Questions [57], HotpotQA [100], and TriviaQA [51]) (described in Section 5.1) - and the six representative RAG pipelines described in Section 2.2. Table 3.1 demonstrates the results achieved when prefetching 256 clusters. We discuss the optimal prefetch cluster amount in Section 4.2. The results reveal consistently high IVF cluster overlap rates across diverse datasets and pipelines. The overlap rates range from 61.6% for SubQuestion and up to 91.5% for Iterative. This supports our hypothesis that q_{in} can serve as a valuable predictor for predicting the IVF cluster memory access pattern of q_{out}.

3.4 Conclusion

Our preliminary analysis establishes a concrete optimization opportunity: predictive prefetching of required clusters during LLM generation can potentially hide CPU-GPU data transfer latency, enabling GPU-accelerated retrieval with minimal GPU memory overhead. This observation forms the foundation for TeleRAG's core innovation of exploiting query similarity to accelerate RAG inference latency under memory constraints. We now present TeleRAG's design, which operationalizes this insight through a lookahead retrieval mechanism alongside profile-guided prefetch optimization and GPU-CPU cooperative search.

Chapter 4

DESIGN OF TELERAG

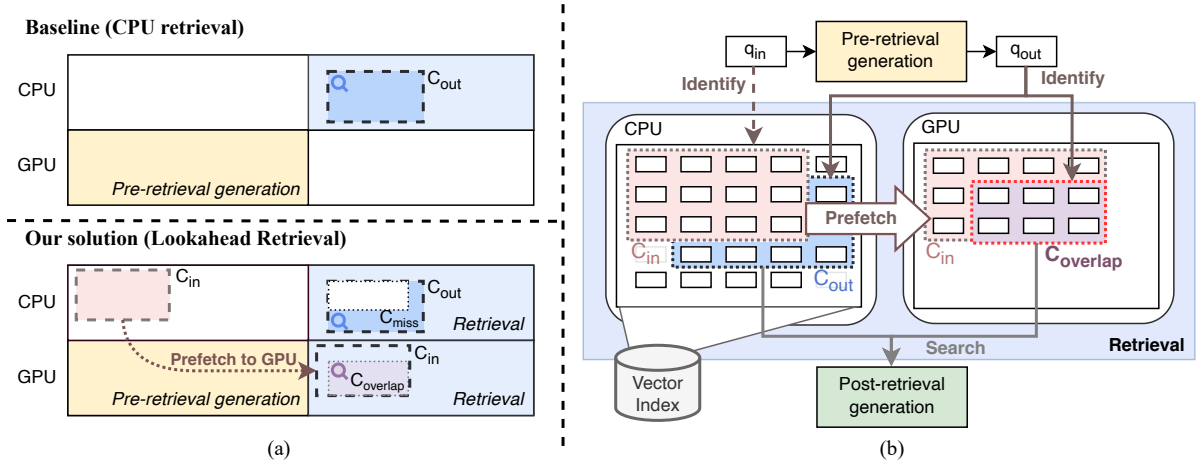


Figure 4.1: (a) The overview of lookahead retrieval compared against CPU-offloaded retrieval. (b) High level system design of TeleRAG. After identifying clusters (C_{in}) for the initial query q_{in}, C_{in} is transferred to the GPU while concurrently generating q_{out}. At retrieval time, the GPU searches the accurately prefetched clusters $C_{overlap}$, while the CPU searches the required clusters that were not prefetched C_{miss}. [61]

Building upon the high IVF cluster overlap observation detailed in Section 3.3, we propose TeleRAG, an efficient RAG inference system that employs lookahead retrieval to prefetch a set of likely IVF clusters to the GPU, thereby accelerating the retrieval process. TeleRAG's design rests on three technical innovations. First, a lookahead retrieval mechanism that predicts and prefetches likely-needed IVF clusters from CPU to GPU during pre-retrieval LLM generation. Second, a profile-guided prefetch optimization strategy that automatically determines the optimal prefetch amount for each RAG pipeline and hardware configuration. Third, a CPU-GPU cooperative similarity search that maximizes resource utilization by

searching correctly-predicted clusters on GPU while concurrently searching missed clusters on CPU. This chapter explains the theory, algorithmic details, and implementation considerations of these components.

4.1 Overview

Figure 4.1 illustrates the TeleRAG architecture with lookahead retrieval. Let q_{in} denote the initial input query provided to the pre-retrieval stage, and q_{out} denote the output transformed query provided to the subsequent retrieval stage. The IVF clusters corresponding to q_{in} are denoted as C_{in}, and those corresponding to q_{out} as C_{out}. Due to the semantic similarity between q_{in} and q_{out}, significant overlap exists between C_{in} and C_{out}, represented as $C_{overlap}$. The cluster overlap rate is defined in Equation 3.1. $C_{overlap}$ represents the clusters that were accurately prefetched from CPU to GPU. C_{miss} represents the clusters that are required for retrieval, but were not prefetched from CPU to GPU.

$$C_{overlap} = C_{in} \cap C_{out} \tag{4.1}$$

$$C_{miss} = C_{out} - C_{in} \tag{4.2}$$

The lookahead retrieval technique operates through the following coordinated steps:

- **Predict and Prefetch:** In parallel with LLM generation, identify and prefetch the IVF clusters (C_{in}) likely to be used in future retrieval to GPU memory. Prioritize clusters semantically proximal to q_{in}. This step leverages GPU DMA (Direct Memory Access) [73, 74] to enable asynchronous transfer overlapping with ongoing LLM generation, effectively hiding the CPU-GPU data transfer latency.

- **GPU Similarity Search:** After the LLM generates the transformed query q_{out}, use the query to identify the accurate C_{out}. The correctly predicted clusters ($C_{overlap}$) should already reside on the GPU due to the previous prefetching operation. The

GPU then performs an efficient similarity search on the predicted $C_{overlap}$ clusters that already reside in the GPU memory.

- **CPU Similarity Search:** C_{miss} are the remaining clusters that were not prefetched to the GPU. The CPU performs similarity search on the C_{miss} concurrently with the GPU similarity search.

- **Merge:** The similarity search results from GPU and CPU are merged on the GPU. The retrieval documents are fed to the LLM for post-retrieval generation on GPU.

Lookahead retrieval accelerates the retrieval phase by overlapping data prefetching to the GPU with LLM generation and executing concurrent similarity searches on both the GPU (for correctly predicted clusters) and CPU (for missed clusters). This design significantly reduces the CPU's computational workload while achieving near-GPU retrieval latency with minimal GPU memory overhead.

4.2 Optimal Prefetch Amount

While prefetching additional clusters increases cluster overlap rate and decreases subsequent retrieval time; extending prefetching beyond the pre-retrieval LLM generation time window (t_{LLM}) eliminates overlap advantages and introduces latency delays. We determine that prefetching up to the duration of t_{LLM} achieves an optimal balance between latency reduction and transfer overhead.

TeleRAG's prefetching mechanism targets a specific number of bytes (b_p) rather than a fixed cluster count. Since IVF cluster sizes show high variance [70], targeting a fixed cluster count can yield unstable data loading times. In contrast, a byte-based limit provides a clear upper bound on transfer duration based on available bandwidth.

The optimal prefetch amount depends on knowing the t_{LLM} for each query, which cannot be determined in advance. To address this challenge, we employ a profiling-guided approach that leverages the observation that, despite variations in query content, the output length

— and thus generation time — remains relatively consistent across most queries for a given RAG pipeline. This is supported by research that demonstrates that LLM inference time exhibits predictable patterns within datasets, with the key determinant being output token length, rather than query-specific characteristics [99].

For each RAG pipeline, we measure t_{LLM} on a calibration set containing n queries and compute the mean generation time:

$$\bar{t}_{LLM} = \text{mean}\{t_{LLM,1}, t_{LLM,2}, \ldots, t_{LLM,n}\}$$

B represents the available CPU-GPU PCIe bandwidth. We estimate the optimal prefetch amount ($\hat{b}_p^*$) as:

$$\hat{b}_p^* = B \cdot \bar{t}_{LLM}$$

This estimated $\hat{b}_p^*$ is subsequently applied to incoming queries, ensuring a near-optimal prefetch amount which maximizes cluster overlap rate while minimizing GPU data transfer latency.

4.3 Implementation Details

TeleRAG is implemented in Python [80] and leverages PyTorch's [9] ecosystem for efficient matrix computation. The vector datastore index is initially constructed using the Faiss library [27], and its data structures - including IVF centroids and cluster data - are converted to PyTorch tensors.

To enable concurrent CPU-GPU data transfer and LLM generation, we utilize PyTorch's copy_(non_blocking=True). This employs separate CUDA streams for prefetching and LLM inference operations, and ensures that data copy I/O operations (prefetch) do not block GPU computation (LLM generation).

At runtime, cluster data is loaded into a contiguous pinned memory region on the CPU, facilitating non-blocking memory transfers to the GPU. A fixed-size contiguous buffer on the

GPU is allocated based on available GPU memory capacity at initialization time.

To implement index search with GPU-CPU cooperation, we employ the following hybrid workload distribution strategy:

- **GPU component:** We perform a single matrix multiplication to compute distances for all prefetched vectors. GPU's specialized cores are optimized for highly parallelizable algorithms like matrix multiplication, which allows up to 10× faster GPU-accelerated IVF index searches compared to its CPU counterpart [35] (See Section 3.2).

- **CPU component:** We utilize Python multithreading [80] to parallelize similarity searches across clusters [27].

- **Merging component:** The merging of the CPU and GPU similarity search results is performed on the GPU. TeleRAG accelerates the final $k - argmin$ sorting step of the IVF search by leveraging GPU-accelerated parallel sorting algorithms, which provide a speedup of upto 6× compared to its CPU counterparts. To enable this we transfer the C_{miss} scalar distances computed on the CPU to the GPU. Unlike full vector data, this scalar data transfer incurs negligible overhead. The GPU then performs a global sort over the combined distances from $C_{overlap}$ and C_{miss}, delivering significant end-to-end speedup.

4.4 Conclusion

TeleRAG's design demonstrates that predictive prefetching can effectively hide CPU-GPU data transfer latency. The profile-guided approach to prefetch optimization eliminates manual tuning burden, while the CPU-GPU cooperative search strategy ensures high performance even when prefetch predictions are imperfect. Most importantly, TeleRAG remains agnostic to retrieval algorithms and LLM generation systems, enabling easy deployment on existing

RAG stacks. Having established the design principles and implementation strategies, the next chapter validates TeleRAG's effectiveness through comprehensive empirical evaluation.

Chapter 5
EVALUATION

To validate TeleRAG's effectiveness and practical impact, this chapter presents a comprehensive empirical evaluation across diverse hardware platforms, language models, RAG pipelines, IVF hyperparameters, and datasets. The evaluation is structured to answer four crucial questions. First, how much latency reduction does TeleRAG achieve for single-query inference on resource-constrained consumer GPUs? Second, how does TeleRAG's performance scale in batched inference scenarios typical of data center deployments? Third, how sensitive is TeleRAG's performance to variations in IVF retrieval hyperparameters? Fourth, how effective is our profile-guided prefetch budget in optimizing the cluster overlap rate?

5.1 Experimental Setup

Datastore: We construct a datastore based on the WikiDPR dataset [6], a widely-used collection containing 2.1 billion tokens from Wikipedia. Following established practice in prior work [12, 52, 69], we partition passages into chunks of 100 tokens each and generate embeddings for each chunk using Contriever [38], with a hidden dimension of 768.

Vector Index: For baseline retrieval, we construct an IVF vector index using Faiss [27] on the datastore. As described in Section 4.3, we convert the Faiss index to a customized PyTorch-based [9] index for TeleRAG. Table 5.1 provides detailed configurations of the vector index and datastore.

Language Models: We evaluate TeleRAG across the Llama model family [7] in three sizes (Llama-3.2-3B, Llama-3-8B, Llama-2-13B) to represent diverse use cases and model scales.

RAG Pipelines: We evaluate TeleRAG with the six popular RAG pipelines (HyDE [29],

Specification	Value
Dataset	`wiki_dpr` [52]
Dataset size	2.1 billion tokens
# of chunks	21 million
# of IVF cluster	4096
Embed model	Contriever [38]
Embed dimension	768
Index type	IVF FLAT
Distance metric	`Inner Product`
Index size	61 GB

Table 5.1: Detailed configurations of our retrieval index.

Setup	Desktop	Server
CPU	Threadripper 5975	EPYC 9554
CPU memory size	512 GB	1.5 TB
GPU	RTX4090 [73]	H100 [74]
GPU memory size	24 GB	80 GB
CPU-GPU Bus	PCIe 4	PCIe 5
Bandwidth	32 (24) GB/s	64 (51) GB/s

Table 5.2: Hardware specifications for our setups. In bandwidth, the number in the parentheses is the actual bandwidth we measured from our system.

SubQuestion [65], Iterative [62], Iter-RetGen [86], FLARE [47], and Self-RAG [12]) described in Section 2.2 . Although some pipelines lack explicit pre-retrieval generation, post-retrieval generation serves a similar functionality for subsequent iteration retrievals.

Evaluation Datasets: We use three widely-adopted question-answering datasets: Natural Questions (NQ) [57], HotpotQA [100], and TriviaQA [51]. The Natural Questions dataset contains real user queries from Google Search paired with Wikipedia passages, making it a great choice for evaluating RAG systems on natural information-seeking questions. For each dataset, we randomly sample 1024 queries and report averages unless otherwise specified. The HotpotQA dataset requires multi-hop reasoning across multiple documents to answer complex questions, which allows us to evaluate TeleRAG's performance on multi-round retrieval-intensive pipelines. The TriviaQA dataset contains trivia questions with diverse retrieval contexts from both web documents and Wikipedia, enabling evaluation of TeleRAG's robustness across varied question types and document sources.

Hardware Configurations: We evaluate TeleRAG across two representative hardware environments - Desktop and Server - reflecting consumer and data center deployment scenarios. The Desktop is equipped with an RTX 4090 (24 GB memory) [73] running 3B and 8B models. The Server has an H100 (80 GB memory) [74] running 8B and 13B models. We do not evaluate the 13B model on Desktop as its size (26 GB) exceeds RTX 4090 capacity. Table 5.2 summarizes the hardware specifications [58].

Hyperparameter Settings: For an IVF index, nprobe controls how many clusters are searched to find the nearest neighbors for a query vector. By setting a value for nprobe, you balance the trade-off between search speed and accuracy: a higher nprobe value improves recall by searching more clusters, but it increases query latency because more data is processed. A lower nprobe value speeds up the query but may result in lower recall. A common heursitic is to set nprobe to $4\sqrt{N_c}$ [113]. Given our index size of $N_c = 4096$, we use $nprobe = 256(= 4\sqrt{N_c})$ by default. For retrieval, top-k denotes the number of most relevant documents to return. We use top-k = 3, which is a standard choice in RAG [113].

RAG Pipeline Implementation: We implement the RAG pipelines using the FlashRAG framework [49]. FlashRAG is a Python toolkit for the reproduction and development of Retrieval Augmented Generation (RAG) research.

Benchmarking Methodology: We adopt the benchmarking methodology from SGLang

[109] and use GPT-3.5-Turbo [75] to trace each pipeline, recording input and output text at each step. During latency evaluation, we configure LLM output length based on recorded traces, ensuring fair latency comparisons across different models.

Baseline Systems: We construct a clean Python execution flow containing LLM generation, datastore retrieval, and necessary pipeline logic as the baseline. We use SGLang as the state-of-the-art LLM inference engine and Faiss as the CPU-offloaded retrieval baseline.

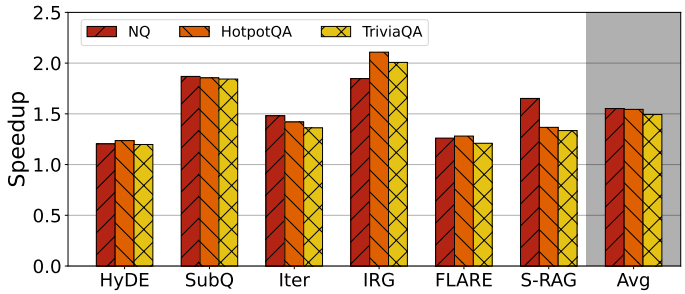
Prefetching Budget: Based on the methodology in Section 4.2, we profile each RAG pipeline using 64 random samples from the NQ dataset to derive the optimal prefetching budget. For the Server setup, we allocate 16 GB for GPU prefetching. For Desktop, we allocate 10 GB for the 3B model and 3.75 GB for the 8B model. This demonstrates efficient operation using only a small fraction (up to 40% for RTX 4090 and 20% for H100) of total GPU memory.

5.2 Single Query Latency on RTX 4090

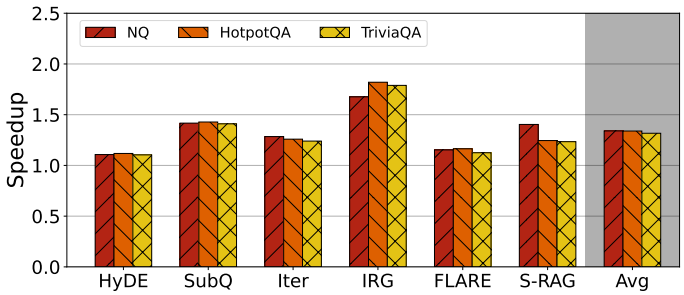
This section focuses on answering how much latency reduction does TeleRAG achieve for single-query inference on resource-constrained consumer GPUs. We evaluate the end-to-end RAG latency for single queries on a Desktop equipped with the RTX 4090 GPU (24 GB memory), representing a typical local deployment scenario.

Figure 5.1 presents the latency reduction achieved by TeleRAG across three datasets and two language models (Llama-3.2-3B and Llama-3-8B). TeleRAG consistently outperforms the Faiss CPU-offloaded retrieval baseline across all tested configurations. With Llama-3.2-3B, TeleRAG achieves average speedups of 1.55×, 1.54×, and 1.49× on Natural Questions, HotpotQA, and TriviaQA, respectively [57, 100, 51]. The peak speedup of 2.11× is achieved on the Iter-RetGen pipeline with HotpotQA, which can be attributed to this pipeline's frequent retrieval operations and relatively short LLM outputs, which amplify the relative impact of retrieval acceleration.

Notably, the SubQuestion pipeline achieves approximately 1.85× speedup across all datasets. This pipeline generates LLM-based sub-questions and performs batched retrieval



(a) End-to-end latency speedup with Llama-3.2-3B.



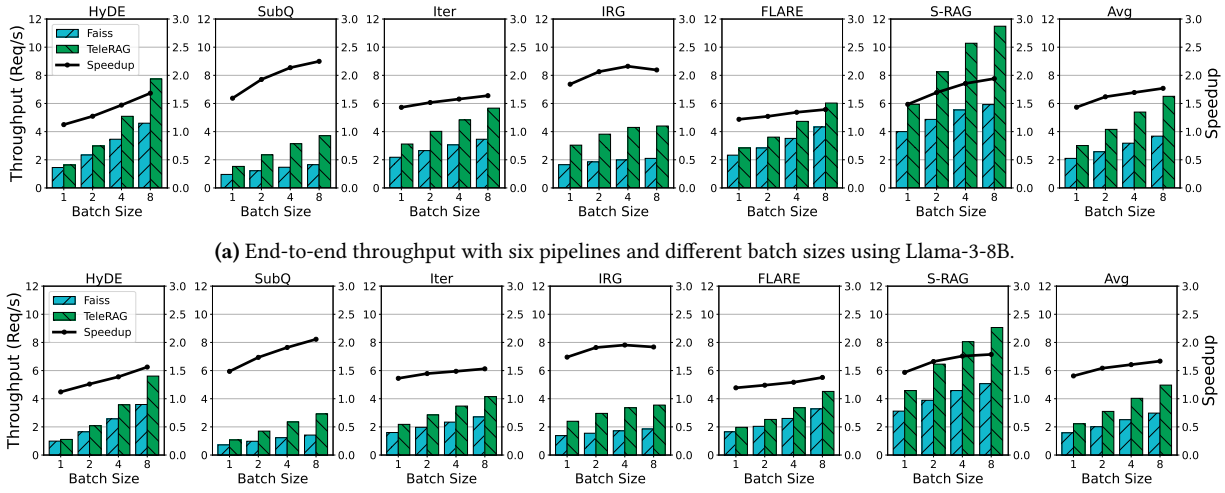
(b) End-to-end latency speedup with Llama-3-8B.

Figure 5.1: End-to-end single-query latency speedup of TeleRAG compared to the CPU-offloaded retrieval baseline. We use the RTX4090 GPU to experiment on our representative RAG pipelines and datasets. [61]

of 3-4 queries simultaneously. The batched retrieval suffers high latency in the CPU-based retrieval scenario, but gains significant performance gains in our TeleRAG retrieval scenario leveraging GPU's batch parallelism advantage.

When deploying Llama-3-8B, speedups slightly decrease compared to Llama-3.2-3B, primarily due to increased LLM latency and reduced available memory for prefetching. Nevertheless, TeleRAG achieves approximately 1.3× speedup across datasets, with peak improvement of 1.82× for Iter-RetGen on HotpotQA. These results are achieved with only 3.75 GB of remaining GPU memory after allocating space for Llama-3-8B (16 GB), the embedding model (1 GB), and miscellaneous tensors, demonstrating TeleRAG's robust capability under tight GPU memory constraints.

5.3 Multi-Query Throughput on H100



(a) End-to-end throughput with six pipelines and different batch sizes using Llama-3-8B.

(b) End-to-end throughput with six pipelines and different batch sizes using Llama-2-13B.

Figure 5.2: End-to-end multi-query throughput improvement of TeleRAG compared to the CPU-offloaded retrieval baseline. We use the H100 GPU to test Llama-3-8B and Llama-2-13B models at different batch sizes, across six RAG pipelines and the Natural Questions dataset. [61]

This section focuses on answering how TeleRAG's throughput scales in batched inference scenarios typical of data center deployments. We evaluate TeleRAG's performance in batched inference on Server (H100 GPU) across batch sizes 1, 2, 4, and 8.

Figure 5.2 shows results for six RAG pipelines with Llama-3-8B and Llama-2-13B [7, 90]. TeleRAG consistently outperforms the Faiss baseline across all pipelines and batch sizes for both language models. At batch size 1 with Llama-3-8B (equivalent to single-query setting), TeleRAG delivers throughput improvements of 1.1–2.2×, averaging 1.46×. Performance gains amplify with increasing batch size. The Faiss baseline demonstrates near-linear scalability up to batch size 4 but exhibits a noticeable plateau at batch size 8, indicating CPU retrieval's limited capacity for handling high query volumes. In contrast, TeleRAG maintains nearly linear scaling through all batch sizes, achieving 1.4–2.2× higher throughput with an average

improvement of 1.83× at batch size 8. The largest throughput gain of 2.2× is achieved for SubQuestion at batch size 8, which is reflective of the pipeline's higher retrieval demands and greater optimization potential.

5.4 Latency Breakdown across Batch Sizes

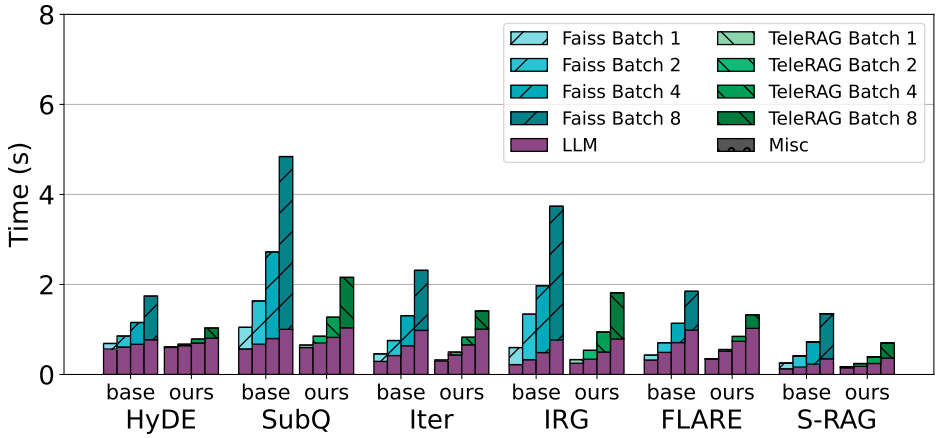
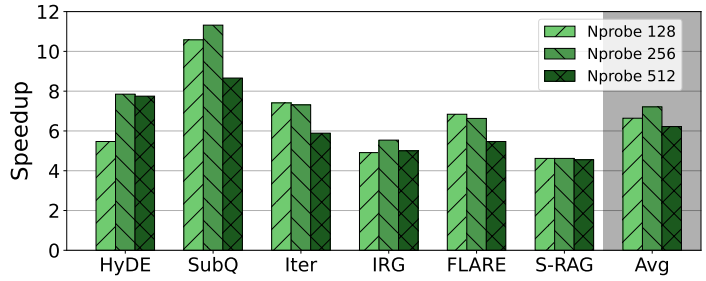


Figure 5.3: Latency breakdown for Llama-3-8B on Natural Questions with a H100 GPU with different batch sizes. [61]

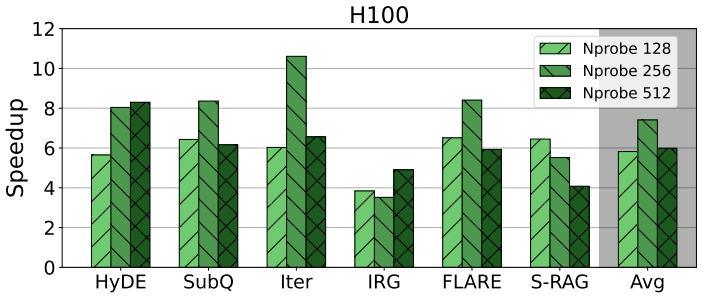
This section focuses on discussing the latency breakdown of TeleRAG in batched inference scenarios typical of data center deployments.

Figure 5.3 presents latency breakdown for RAG pipelines with Llama-3-8B and Llama-2-13B on a single H100 GPU at varying batch sizes [7, 90]. LLM latency grows sub-linearly with larger batch sizes. However, CPU-based Faiss retrieval latency increases linearly with batch size, dominating overall latency at large batches. These findings corroborate the throughput results in Figure 5.2, highlighting CPU retrieval's limited scalability in serving multi-query scenarios. In contrast, TeleRAG achieves consistent acceleration across all batch sizes, with speedup increasing from 1.4× to 1.8× as batch size grows from 1 to 8.

5.5 *Retrieval Speedup Across nprobe Values*



(a) Llama-3.2-3B with a RTX4090 GPU.



(b) Llama-3-8B with a H100 GPU.

Figure 5.4: Single-query retrieval speedup with different nprobe values. The experiment is performed on both the RTX4090 and H100 GPU, across various RAG pipelines and on the Natural Questions dataset. [61]

The IVF search hyperparameter nprobe controls the tradeoff between search recall and latency as described in Section 5.1. This section investigates how TeleRAG's retrieval latency behaves as the nprobe varies.

Figure 5.4 shows retrieval latency reduction on Natural Questions [57] across varying nprobe values . Speedups are observed for all nprobe values, with greatest improvements at nprobe 256, achieving average speedups of 7.21× on RTX 4090 and 7.41× on H100 [27].As nprobe increases, TeleRAG's performance becomes constrained by missed clusters requiring CPU processing given the fixed prefetch budget. However, higher nprobe values also ex-

tend retrieval latency, and TeleRAG maintains significant improvement over CPU-offloaded baseline retrieval [27].

5.6 Prefetch Budgets and Cluster Overlap Rates

Pipeline	H100 (Llama-3-8B)		RTX4090 (Llama-3.2-3B)	
	Budget	Cluster Overlap Rate	Budget	Cluster Overlap Rate
HyDE	9 GB	91.95%	7 GB	87.42%
SubQ	8 GB	79.04%	7 GB	76.38%
Iter	5 GB	95.51%	3 GB	84.59%
IRG	4 GB	59.52%	2.5 GB	50.34%
FLARE	6 GB	79.35%	3 GB	56.67%
S-RAG	3 GB	71.29%	1.25 GB	29.96%

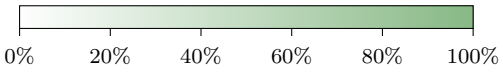


Table 5.3: The prefetch budget and corresponding averaged cluster overlap rate (See Equation 3.1) for each pipeline and hardware setup on the Natural Questions dataset.

This section examines the prefetch budgets determined by TeleRAG's profile-guided approach and the resulting cluster hit rates achieved across different RAG pipelines. By analyzing the relationship between prefetch budget size, cluster hit rate, and end-to-end speedup, it validates whether our profile-guided approach reliably estimates optimal prefetch amounts.

Table 5.3 presents prefetch budgets derived using the profile-guided approach on RTX 4090 and H100 GPUs for Natural Questions, along with average cluster hit rates. Generally, high cluster hit rates ($> 50\%$) are achieved with large prefetching budgets. For budgets below 2 GB, hit rates are relatively lower ($< 50\%$), limiting the benefits of reducing the CPU's workloads. As shown in Figure 5.1, TeleRAG achieves 1.2–1.6× end-to-end speedups for these pipelines using the conservative default prefetch budgets detailed in Section 5.1.

5.7 Conclusion

Our comprehensive evaluation demonstrates that TeleRAG successfully achieves its design objectives across diverse deployment scenarios. On consumer GPUs, TeleRAG delivers consistent 1.53× average latency reduction for single-query inference while operating within minimal GPU memory constraints. In data center settings, TeleRAG exhibits superior scaling properties, achieving 1.83× average throughput improvement at batch size 8 where CPU-based retrieval plateaus. It is also validated that our profile-guided prefetch optimization reliably identifies near-optimal prefetch amounts, and that even imperfect cluster predictions translate to meaningful end-to-end speedups. These results confirm TeleRAG's effectiveness as a practical solution that bridges the memory-latency gap in RAG deployment. Having demonstrated TeleRAG's empirical performance, we now conclude the thesis with a discussion on our contributions and its implications for the broader RAG ecosystem.

Chapter 6

CONCLUSION

This thesis has presented TeleRAG, an efficient inference system designed to address critical system-level latency challenges in retrieval-augmented generation pipelines. Modern RAG applications demand both rapid response times and access to large-scale vector datastores, creating a fundamental tension between GPU memory capacity and retrieval latency. TeleRAG resolves this tension through a novel CPU-GPU cooperative lookahead retrieval mechanism.

The key contributions of this thesis are threefold:

- **Lookahead Retrieval Mechanism:** By exploiting the high semantic overlap (61% to 91%) between queries at subsequent stages of the RAG pipeline, TeleRAG proactively prefetches likely-needed IVF clusters from CPU to GPU memory during LLM generation, effectively hiding data transfer latency behind ongoing LLM computation. This retrieval-agnostic approach requires no modifications to existing retriever algorithms or vector index structures, ensuring compatibility with standard RAG deployments.

- **Profile-Guided Prefetch Optimization:** TeleRAG employs an intelligent profiling methodology to determine optimal prefetch amounts depending on RAG pipeline configuration and hardware characteristics.

- **GPU-CPU Cooperative Retrieval:** TeleRAG implements hybrid similarity search where predicted clusters are searched on GPU while missed clusters are processed on CPU in parallel, maximizing resource utilization.

Experimental evaluation across diverse set of representative RAG pipelines, query work-

loads, and vector index configurations demonstrates TeleRAG's substantive latency gains. On consumer GPUs (RTX 4090), TeleRAG achieves 1.53× average latency reduction for single queries, with peak speedups of 2.11× on retrieval-intensive pipelines. In batched data-center scenarios (H100), TeleRAG delivers near-linear scaling delivering 1.46× average throughput improvement at batch size 1 up to 1.83× improvement at batch size 8.

These performance gains are achieved with minimal GPU memory footprint; only 3.75 GB for retrieval on consumer GPUs and 16 GB on data-center GPUs. For the first time, this unlocks simultaneous GPU hosting of massive datastore retrieval indices alongside large language models.

The main limitation of our work is that Lookahead retrieval is dependent on consistently high IVF cluster overlap rates between queries in subsequent RAG pipeline stages. This holds true for most current query transformation techniques. This presumption will fail for any query transformation technique that significantly transforms the semantic characteristics of the input query, leading to lower prefetch accuracy, lower relative GPU acceleration, and higher latencies.

TeleRAG addresses a critical gap in the RAG ecosystem by focusing on system-level optimization at the CPU-GPU boundary rather than algorithmic improvements to the retrieval or generation components. This complementary approach makes it broadly compatible with existing RAG frameworks, LLM serving engines (vLLM [58], SGLang [109]), and vector search libraries (Faiss [27], DiskANN [89]). By enabling efficient RAG deployment on both resource-constrained consumer hardware and high-throughput data center environments, TeleRAG democratizes access to advanced RAG capabilities and accelerates the practical deployment of retrieval-augmented systems across diverse applications.

BIBLIOGRAPHY

[1] Genspark. https://www.genspark.ai/.

[2] Perplexity. https://www.perplexity.ai/.

[3] Assessing Retrieval-Augmented Large Language Model Performance in Emergency Department ICD-10-CM Coding Compared to Human Coders. *medRxiv*, 2024.

[4] Reyna Abhyankar, Zijian He, Vikranth Srivatsa, Hao Zhang, and Yiying Zhang. APIServe: Efficient API Support for Large-Language Model Inferencing. *arXiv preprint arXiv:2402.01869*, 2024.

[5] Amey Agrawal, Nitin Kedia, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav S Gulavani, Alexey Tumanov, and Ramachandran Ramjee. Taming Throughput-Latency Tradeoff in LLM Inference with Sarathi-Serve. *arXiv preprint arXiv:2403.02310*, 2024.

[6] AI@Meta. Dataset Card for "wiki_dpr". https://huggingface.co/datasets/facebook/wiki_dpr, 2020.

[7] AI@Meta. Llama 3 Model Card. https://github.com/meta-llama/llama3/blob/main/MODEL_CARD.md, 2024.

[8] Rama Akkiraju, Anbang Xu, Deepak Bora, Tan Yu, Lu An, Vishal Seth, Aaditya Shukla, Pritam Gundecha, Hridhay Mehta, Ashwin Jha, et al. FACTS About Building Retrieval Augmented Generation-based Chatbots. *arXiv preprint arXiv:2407.07858*, 2024.

[9] Jason Ansel, Edward Yang, Horace He, Natalia Gimelshein, Animesh Jain, Michael Voznesensky, Bin Bao, Peter Bell, David Berard, Evgeni Burovski, et al. PyTorch 2: Faster Machine Learning Through Dynamic Python Bytecode Transformation and Graph Compilation. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, pages 929–947, New York, NY, USA, 2024. ACM.

[10] Dmitri I Arkhipov, Di Wu, Keqin Li, and Amelia C Regan. Sorting with GPUs: A Survey. *arXiv preprint arXiv:1709.02520*, 2017.

[11] Akari Asai, Jacqueline He, Rulin Shao, Weijia Shi, Amanpreet Singh, Joseph Chee Chang, Kyle Lo, Luca Soldaini, Sergey Feldman, Mike D'arcy, et al. OpenScholar: Synthesizing Scientific Literature with Retrieval-augmented LMs. *arXiv preprint arXiv:2411.14199*, 2024.

[12] Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. Self-RAG: Learning to Retrieve, Generate, and Critique through Self-Reflection. In *The Twelfth International Conference on Learning Representations*, 2023.

[13] Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. Original implementation of SELF-RAG: Learning to Retrieve, Generate and Critique through self-reflection. https://github.com/AkariAsai/self-rag, 2024.

[14] Akari Asai, Zexuan Zhong, Danqi Chen, Pang Wei Koh, Luke Zettlemoyer, Hannaneh Hajishirzi, and Wen-tau Yih. Reliable, Adaptable, and Attributable Language Models with Retrieval. *arXiv preprint arXiv:2403.03187*, 2024.

[15] AWS. Guidance for Conversational Chatbots Using Retrieval Augmented Generation on AWS. https://aws.amazon.com/solutions/guidance/conversational-chatbots-using-retrieval-augmented-generation-on-aws/.

[16] Emery Berger and Ben Zorn. AI Software Should be More Like Plain Old Software. https://www.sigarch.org/ai-software-should-be-more-like-plain-old-software/, 2024.

[17] Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George Bm Van Den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, et al. Improving Language Models by Retrieving from Trillions of Tokens. In *International Conference on Machine Learning*, pages 2206–2240. PMLR, 2022.

[18] James Briggs, Gibbs Cullen, and Greg Kogan. Vector Search in the Wild. https://www.pinecone.io/learn/series/wild/.

[19] Tianhui Cai, Yifan Liu, Zewei Zhou, Haoxuan Ma, Seth Z Zhao, Zhiwen Wu, and Jiaqi Ma. Driving with Regulation: Interpretable Decision-Making for Autonomous Vehicles with Retrieval-Augmented Reasoning via LLM. *arXiv preprint arXiv:2410.04759*, 2024.

[20] Chi-Min Chan, Chunpu Xu, Ruibin Yuan, Hongyin Luo, Wei Xue, Yike Guo, and Jie Fu. Rq-rag: Learning to refine queries for retrieval augmented generation. *arXiv preprint arXiv:2404.00610*, 2024.

[21] Binoy Chemmagate. Reducing RAG Pipeline Latency for Real-Time Voice Conversations. https://developer.vonage.com/en/blog/reducing-rag-pipeline-latency-for-real-time-voice-conversations, 2024.

[22] Qi Chen, Bing Zhao, Haidong Wang, Mingqin Li, Chuanjie Liu, Zengzhong Li, Mao Yang, and Jingdong Wang. SPANN: Highly-efficient Billion-scale Approximate Nearest Neighbor Search. *arXiv preprint arXiv:2111.08566*, 2021.

[23] Felix Chern, Blake Hechtman, Andy Davis, Ruiqi Guo, David Majnemer, and Sanjiv Kumar. TPU-KNN: K Nearest Neighbor Search at Peak FLOP/s. In *Advances in Neural Information Processing Systems*, volume 35, pages 15489–15501, 2022.

[24] Neo Christopher Chung, George Dyer, and Lennart Brocki. Challenges of Large Language Models for Mental Health Counseling. *arXiv preprint arXiv:2311.13857*, 2023.

[25] Databricks. RAG (Retrieval Augmented Generation) on Databricks. https://docs.databricks.com/en/generative-ai/retrieval-augmented-generation.html, 2024.

[26] Divyanshu Dixit. Advanced RAG Series: Generation and Evaluation. https://div.beehiiv.com/p/advanced-rag-series-generation-evaluation, 2024.

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

[28] Wenqi Fan, Yujuan Ding, Liang bo Ning, Shijie Wang, Hengyun Li, Dawei Yin, Tat-Seng Chua, and Qing Li. A Survey on RAG Meeting LLMs: Towards Retrieval-Augmented Large Language Models. In *Knowledge Discovery and Data Mining*, 2024.

[29] Luyu Gao, Xueguang Ma, Jimmy Lin, and Jamie Callan. Precise Zero-Shot Dense Retrieval without Relevance Labels. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1762–1777, 2023.

[30] Yunfan Gao. Modular RAG and RAG Flow: Part II. https://medium.com/@yufan1602/modular-rag-and-rag-flow-part-ii-77b62bf8a5d3, 2024.

[31] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, and Haofen Wang. Retrieval-Augmented Generation for Large Language Models: A Survey. *arXiv preprint arXiv:2312.10997*, 2023.

[32] Samira Ghodratnama and Mehrdad Zakershahrak. Adapting LLMs for Efficient, Personalized Information Retrieval: Methods and Implications. In *International Conference on Service-Oriented Computing*, pages 17–26. Springer, 2023.

[33] Abdussamad GM and Gopala Dhar. How Apollo 24—7 Leverages MedLM with RAG to Revolutionize Healthcare. https://cloud.google.com/blog/products/ai-machine-learning/how-apollo-247-leverages-medlm-with-rag-to-revolutionize-healthcare, 2024.

[34] Ruiqi Guo, Philip Sun, Erik Lindgren, Quan Geng, David Simcha, Felix Chern, and Sanjiv Kumar. Accelerating Large-Scale Inference with Anisotropic Vector Quantization. In *International Conference on Machine Learning*, 2020.

[35] Dipesh Gyawali. Comparative analysis of cpu and gpu profiling for deep learning models. *arXiv preprint arXiv:2309.02521*, 2023.

[36] Moritz Hardt and Yu Sun. Test-time Training on Nearest Neighbors for Large Language Models. *arXiv preprint arXiv:2305.18466*, 2023.

[37] Ivan Ilin. Advanced RAG Techniques: an Illustrated Overview. https://pub.towardsai.net/advanced-rag-techniques-an-illustrated-overview-04d193d8fec6, 2023.

[38] Gautier Izacard, Mathilde Caron, Lucas Hosseini, Sebastian Riedel, Piotr Bojanowski, Armand Joulin, and Edouard Grave. Unsupervised Dense Information Retrieval with Contrastive Learning. *arXiv preprint arXiv:2112.09118*, 2021.

[39] Gautier Izacard, Patrick Lewis, Maria Lomeli, Lucas Hosseini, Fabio Petroni, Timo Schick, Jane Dwivedi-Yu, Armand Joulin, Sebastian Riedel, and Edouard Grave. Atlas: Few-shot Learning with Retrieval Augmented Language Models. *Journal of Machine Learning Research*, 24(251):1–43, 2023.

[40] Rolf Jagerman, Honglei Zhuang, Zhen Qin, Xuanhui Wang, and Michael Bendersky. Query Expansion by Prompting Large Language Models. *arXiv preprint arXiv:2305.03653*, 2023.

[41] Herve Jegou, Matthijs Douze, and Cordelia Schmid. Product quantization for nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence*, 33(1):117–128, 2010.

[42] Huiqiang Jiang, Qianhui Wu, Xufang Luo, Dongsheng Li, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. LongLLMLingua: Accelerating and Enhancing LLMs in Long Context Scenarios via Prompt Compression. *arXiv preprint arXiv:2310.06839*, 2023.

[43] Wenqi Jiang, Shigang Li, Yu Zhu, Johannes de Fine Licht, Zhenhao He, Runbin Shi, Cedric Renggli, Shuai Zhang, Theodoros Rekatsinas, Torsten Hoefler, et al. Co-design

Hardware and Algorithm for Vector Search. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–15, 2023.

[44] Wenqi Jiang, Suvinay Subramanian, Cat Graves, Gustavo Alonso, Amir Yazdanbakhsh, and Vidushi Dadu. RAGO: Systematic Performance Optimization for Retrieval-Augmented Generation Serving. *arXiv preprint arXiv:2503.14649*, 2025.

[45] Wenqi Jiang, Marco Zeller, Roger Waleffe, Torsten Hoefler, and Gustavo Alonso. Chameleon: a Heterogeneous and Disaggregated Accelerator System for Retrieval-augmented Language Models. *arXiv preprint arXiv:2310.09949*, 2023.

[46] Wenqi Jiang, Shuai Zhang, Boran Han, Jie Wang, Bernie Wang, and Tim Kraska. PipeRAG: Fast Retrieval-Augmented Generation via Algorithm-System Co-design. *arXiv preprint arXiv:2403.05676*, 2024.

[47] Zhengbao Jiang, Frank F Xu, Luyu Gao, Zhiqing Sun, Qian Liu, Jane Dwivedi-Yu, Yiming Yang, Jamie Callan, and Graham Neubig. Active Retrieval Augmented Generation. *arXiv preprint arXiv:2305.06983*, 2023.

[48] Chao Jin, Zili Zhang, Xuanlin Jiang, Fangyue Liu, Xin Liu, Xuanzhe Liu, and Xin Jin. RAGCache: Efficient Knowledge Caching for Retrieval-Augmented Generation. *arXiv preprint arXiv:2404.12457*, 2024.

[49] Jiajie Jin, Yutao Zhu, Xinyu Yang, Chenghao Zhang, and Zhicheng Dou. FlashRAG: A Modular Toolkit for Efficient Retrieval-Augmented Generation Research. *arXiv preprint arXiv:2405.13576*, 2024.

[50] Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale Similarity Search with GPUs. *IEEE Transactions on Big Data*, 7(3):535–547, 2019.

[51] Mandar Joshi, Eunsol Choi, Daniel S Weld, and Luke Zettlemoyer. TriviaQA: A Large Scale Distantly Supervised Challenge Dataset for Reading Comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1601–1611, 2017.

[52] Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. Dense Passage Retrieval for Open-Domain Question Answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2020.

[53] Saim Khan, Somesh Singh, Harsha Vardhan Simhadri, Jyothi Vedurada, et al. BANG: Billion-Scale Approximate Nearest Neighbor Search using a Single GPU. *arXiv preprint arXiv:2401.11324*, 2024.

[54] Urvashi Khandelwal, Omer Levy, Dan Jurafsky, Luke Zettlemoyer, and Mike Lewis. Generalization through Memorization: Nearest Neighbor Language Models. 2019.

[55] Jaehyung Kim, Jaehyun Nam, Sangwoo Mo, Jongjin Park, Sang-Woo Lee, Minjoon Seo, Jung-Woo Ha, and Jinwoo Shin. SuRe: Improving Open-domain Question Answering of LLMs via Summarized Retrieval. In *The Twelfth International Conference on Learning Representations*, 2023.

[56] Sehoon Kim, Suhong Moon, Ryan Tabrizi, Nicholas Lee, Michael W Mahoney, Kurt Keutzer, and Amir Gholami. An LLM Compiler for Parallel Function Calling. *arXiv preprint arXiv:2312.04511*, 2023.

[57] Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, et al. Natural Questions: a Benchmark for Question Answering Research. *Transactions of the Association for Computational Linguistics*, 7:453–466, 2019.

[58] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E Gonzalez, Hao Zhang, and Ion Stoica. Efficient Memory Management for Large Language Model Serving with PagedAttention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023.

[59] Maximilian Lam, Jeff Johnson, Wenjie Xiong, Kiwan Maeng, Udit Gupta, Yang Li, Liangzhen Lai, Ilias Leontiadis, Minsoo Rhu, Hsien-Hsin S Lee, et al. GPU-based Private Information Retrieval for On-Device Machine Learning Inference. *arXiv preprint arXiv:2301.10904*, 2023.

[60] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474, 2020.

[61] Chien-Yu Lin, Keisuke Kamahori, Yiyu Liu, Xiaoxiang Shi, Madhav Kashyap, Yile Gu, Rulin Shao, Zihao Ye, Kan Zhu, Rohan Kadekodi, Stephanie Wang, Arvind Krishnamurthy, Luis Ceze, and Baris Kasikci. TeleRAG: Efficient Retrieval-Augmented Generation Inference with Lookahead Retrieval, 2025.

[62] Jerry Liu. LlamaIndex. https://doi.org/10.5281, 2022.

[63] Zihan Liu, Wentao Ni, Jingwen Leng, Yu Feng, Cong Guo, Quan Chen, Chao Li, Minyi Guo, and Yuhao Zhu. JUNO: Optimizing High-Dimensional Approximate Nearest Neighbour Search with Sparsity-Aware Algorithm and Ray-Tracing Core Mapping. *arXiv preprint arXiv:2312.01712*, 2023.

[64] Songshuo Lu, Hua Wang, Yutian Rong, Zhi Chen, and Yaohua Tang. TurboRAG: Accelerating Retrieval-Augmented Generation with Precomputed KV Caches for Chunked Text. *arXiv preprint arXiv:2410.07590*, 2024.

[65] Xinbei Ma, Yeyun Gong, Pengcheng He, Nan Duan, et al. Query Rewriting in Retrieval-Augmented Large Language Models. In *The 2023 Conference on Empirical Methods in Natural Language Processing*, 2023.

[66] Melissa Malec. RAG in Financial Services: Use-Cases, Impact, & Solutions. https://hatchworks.com/blog/gen-ai/rag-for-financial-services/, 2024.

[67] Yu. A. Malkov and D. A. Yashunin. Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs, 2018.

[68] Alex Troy Mallen, Akari Asai, Victor Zhong, Rajarshi Das, Daniel Khashabi, and Hannaneh Hajishirzi. When Not to Trust Language Models: Investigating Effectiveness of Parametric and Non-Parametric Memories. In *The 61st Annual Meeting Of The Association For Computational Linguistics*, 2023.

[69] Sewon Min, Suchin Gururangan, Eric Wallace, Weijia Shi, Hannaneh Hajishirzi, Noah A Smith, and Luke Zettlemoyer. SILO Language Models: Isolating Legal Risk In a Nonparametric Datastore. In *The Twelfth International Conference on Learning Representations*, 2023.

[70] Jason Mohoney, Anil Pacaci, Shihabur Rahman Chowdhury, Umar Farooq Minhas, Jeffery Pound, Cedric Renggli, Nima Reyhani, Ihab F. Ilyas, Theodoros Rekatsinas, and Shivaram Venkataraman. Incremental IVF Index Maintenance for Streaming Vector Search, 2024.

[71] MyScale. 4 Key Benefits of RAG Algorithmic Trading in Financial Markets. https://myscale.com/blog/benefits-rag-algorithmic-trading-financial-markets/, 2024.

[72] Mohammad Norouzi and David J Fleet. Cartesian k-means. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3017–3024, 2013.

[73] NVIDIA. GeForce RTX 4090. https://www.nvidia.com/en-us/geforce/graphics-cards/40-series/rtx-4090/, 2024.

[74] NVIDIA. NVIDIA H100 Tensor Core GPU. https://resources.nvidia.com/en-us-tensor-core/nvidia-tensor-core-gpu-datasheet, 2024.

[75] OpenAI. GPT-3.5 Turbo. https://platform.openai.com/docs/models/gpt-3-5-turbo.

[76] Pathway. Adaptive RAG: How we cut LLM costs without sacrificing accuracy. https://pathway.com/developers/showcases/adaptive-rag, 2024.

[77] Wenjun Peng, Guiyang Li, Yue Jiang, Zilong Wang, Dan Ou, Xiaoyi Zeng, Tongxu, and Enhong Chen. Large Language Model based Long-tail Query Rewriting in Taobao Search. *Companion Proceedings of the ACM on Web Conference 2024*, 2023.

[78] Wenjun Peng, Guiyang Li, Yue Jiang, Zilong Wang, Dan Ou, Xiaoyi Zeng, Derong Xu, Tong Xu, and Enhong Chen. Large Language Model based Long-tail Query Rewriting in Taobao Search. In *Companion Proceedings of the ACM on Web Conference 2024*, 2024.

[79] Ofir Press, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah A Smith, and Mike Lewis. Measuring and Narrowing the Compositionality Gap in Language Models. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 5687–5711, 2023.

[80] Python Software Foundation. Python Language Reference, version 3.x, 2016.

[81] Derrick Quinn, Mohammad Nouri, Neel Patel, John Salihu, Alireza Salemi, Sukhan Lee, Hamed Zamani, and Mohammad Alian. Accelerating Retrieval-Augmented Generation. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*, pages 15–32, Rotterdam, Netherlands, 2025. ACM.

[82] Ori Ram, Yoav Levine, Itay Dalmedigos, Dor Muhlgay, Amnon Shashua, Kevin Leyton-Brown, and Yoav Shoham. In-Context Retrieval-Augmented Language Models. *Transactions of the Association for Computational Linguistics*, 11:1316–1331, 2023.

[83] Rapidsai. Rapidsai/raft: RAFT contains fundamental widely-used algorithms and primitives for data science, Graph and machine learning. https://github.com/rapidsai/raft, 2022.

[84] Korakit Seemakhupt, Sihang Liu, and Samira Khan. EdgeRAG: Online-Indexed RAG for Edge Devices. *arXiv preprint arXiv:2412.21023*, 2024.

[85] Rulin Shao, Jacqueline He, Akari Asai, Weijia Shi, Tim Dettmers, Sewon Min, Luke Zettlemoyer, and Pang Wei Koh. Scaling Retrieval-Based Language Models with a Trillion-Token Datastore. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

[86] Zhihong Shao, Yeyun Gong, Yelong Shen, Minlie Huang, Nan Duan, and Weizhu Chen. Enhancing Retrieval-Augmented Large Language Models with Iterative Retrieval-Generation Synergy. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 9248–9274, 2023.

[87] Josef Sivic and Andrew Zisserman. Video Google: A Text Retrieval Approach to Object Matching in Videos. In *ICCV*, pages 1470–1477. IEEE Computer Society, 2003.

[88] David Stewart and Jamie Linsdell. Say Hello to Precision: How Rerankers and Embeddings Boost Search. https://cohere.com/blog/say-hello-to-precision-how-rerankers-and-embeddings-boost-search, 2024.

[89] Suhas Jayaram Subramanya, Fnu Devvrit, Harsha Vardhan Simhadri, Ravishankar Krishnawamy, and Rohan Kadekodi. DiskANN: Fast Accurate Billion-point Nearest Neighbor Search on a Single Node. *Advances in Neural Information Processing Systems*, 32, 2019.

[90] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open Foundation and Fine-tuned Chat Models. *arXiv preprint arXiv:2307.09288*, 2023.

[91] Kuan Tung. Enhancing User Experience by Overcoming Latency in the ION IQ Chatbot. https://www.ontinue.com/resource/enhancing-user-experience-by-overcoming-latency-in-the-ion-iq-chatbot/, 2024.

[92] Niithiyn Vijeaswaran, AJ Dhimine, Armando Diaz, Sebastian Bustillo, Farooq Sabir, and Marco Punio. Advanced RAG patterns on Amazon SageMaker. https://aws.amazon.com/blogs/machine-learning/advanced-rag-patterns-on-amazon-sagemaker/, 2024.

[93] Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, et al. A Survey on Large Language Model based Autonomous Agents. *Frontiers of Computer Science*, 18(6):186345, 2024.

[94] Zijie J Wang and Duen Horng Chau. MeMemo: On-device Retrieval Augmentation for Private and Personalized Text Generation. In *Proceedings of the 47th International*

ACM SIGIR Conference on Research and Development in Information Retrieval, pages 2765–2770, Washington DC, USA, 2024. ACM.

[95] Zilong Wang, Zifeng Wang, Long Le, Huaixiu Steven Zheng, Swaroop Mishra, Vincent Perot, Yuwei Zhang, Anush Mattapalli, Ankur Taly, Jingbo Shang, et al. Speculative RAG: Enhancing Retrieval Augmented Generation through Drafting. *arXiv preprint arXiv:2407.08223*, 2024.

[96] Khye Wei. Advanced RAG with Azure AI Search and LlamaIndex. https://techcommunity.microsoft.com/t5/ai-azure-ai-services-blog/advanced-rag-with-azure-ai-search-and-llamaindex/ba-p/4115007, 2024.

[97] Lukas Wutschitz, Boris Köpf, Andrew Paverd, Saravan Rajmohan, Ahmed Salem, Shruti Tople, Santiago Zanella-Béguelin, Menglin Xia, and Victor Rühle. Rethinking Privacy in Machine Learning Pipelines from an Information Flow Control Perspective. *arXiv preprint arXiv:2311.15792*, 2023.

[98] Zhiqiang Xie, Hao Kang, Ying Sheng, Tushar Krishna, Kayvon Fatahalian, and Christos Kozyrakis. AI Metropolis: Scaling Large Language Model-based Multi-Agent Simulation with Out-of-order Execution. *arXiv preprint arXiv:2411.03519*, 2024.

[99] Yuqing Yang, Yuedong Xu, and Lei Jiao. A Queueing Theoretic Perspective on Low-Latency LLM Inference with Variable Token Length, 2024.

[100] Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W Cohen, Ruslan Salakhutdinov, and Christopher D Manning. HotpotQA: A Dataset for Diverse, Explainable Multi-hop Question Answering. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2018.

[101] Jiayi Yao, Hanchen Li, Yuhan Liu, Siddhant Ray, Yihua Cheng, Qizheng Zhang, Kuntai Du, Shan Lu, and Junchen Jiang. CacheBlend: Fast Large Language Model Serving for RAG with Cached Knowledge Fusion. *arXiv preprint arXiv:2405.16444*, 2024.

[102] Fanghua Ye, Meng Fang, Shenghui Li, and Emine Yilmaz. Enhancing Conversational Search: Large Language Model-Aided Informative Query Rewriting. In *The 2023 Conference on Empirical Methods in Natural Language Processing*, 2023.

[103] Jianhao Yuan, Shuyang Sun, Daniel Omeiza, Bo Zhao, Paul Newman, Lars Kunze, and Matthew Gadd. RAG-driver: Generalisable Driving Explanations with Retrieval-Augmented In-Context Learning in Multi-Modal Large Language Model. *arXiv preprint arXiv:2402.10828*, 2024.

[104] Matei Zaharia, Omar Khattab, Lingjiao Chen, Jared Quincy Davis, Heather Miller, Chris Potts, James Zou, Michael Carbin, Jonathan Frankle, Naveen Rao, and Ali Ghodsi. The Shift from Models to Compound AI Systems. https://bair.berkeley.edu/blog/2024/02/18/compound-ai-systems/, 2024.

[105] Chaoliang Zeng, Layong Luo, Qingsong Ning, Yaodong Han, Yuhang Jiang, Ding Tang, Zilong Wang, Kai Chen, and Chuanxiong Guo. FAERY: An FPGA-accelerated Embedding-based Retrieval System. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pages 841–856, 2022.

[106] Zhihao Zhang, Alan Zhu, Lijie Yang, Yihua Xu, Lanting Li, Lijie Yang Phitchaya Mangpo Phothilimthana, and Zhihao Jia. Accelerating Retrieval-Augmented Language Model Serving with Speculation. *arXiv preprint arXiv:2401.14021*, 2024.

[107] Zili Zhang, Fangyue Liu, Gang Huang, Xuanzhe Liu, and Xin Jin. Fast Vector Query Processing for Large Datasets Beyond GPU Memory with Reordered Pipelining. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, pages 23–40, 2024.

[108] Huaixiu Steven Zheng, Swaroop Mishra, Xinyun Chen, Heng-Tze Cheng, Ed H Chi, Quoc V Le, and Denny Zhou. Take a Step Back: Evoking Reasoning via Abstraction in Large Language Models. *arXiv preprint arXiv:2310.06117*, 2023.

[109] Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E Gonzalez, Clark Barrett, and Ying Sheng. SGLang: Efficient Execution of Structured Language Model Programs. *arXiv preprint arXiv:2312.07104*, 2024.

[110] Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc V Le, et al. Least-to-Most Prompting Enables Complex Reasoning in Large Language Models. In *The Eleventh International Conference on Learning Representations*, 2022.

[111] Yuhao Zhu. RTNN: Accelerating Neighbor Search Using Hardware Ray Tracing. In *Proceedings of the 27th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP '22)*, pages 76–89, 2022.

[112] Shengyao Zhuang, Bing Liu, Bevan Koopman, and Guido Zuccon. Open-source Large Language Models are Strong Zero-shot Query Likelihood Models for Document Ranking. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 8807–8817, 2023.

[113] Zilliz. How to Select Index Parameters for IVF Index. https://zilliz.com/blog/select-index-parameters-ivf-index, 2020.