AquaPipe: A Quality-Aware Pipeline for Knowledge Retrieval and Large Language Models

RUNJIE YU, Huazhong University of Science and Technology, China
WEIZHOU HUANG, Huazhong University of Science and Technology, China
SHUHAN BAI, Huazhong University of Science and Technology, China
JIAN ZHOU*, Huazhong University of Science and Technology, China
FEI WU*, Huazhong University of Science and Technology, China

The knowledge retrieval methods such as Approximate Nearest Neighbor Search (ANNS) significantly enhance the generation quality of Large Language Models (LLMs) by introducing external knowledge, and this method is called Retrieval-augmented generation (RAG). However, due to the rapid growth of data size, ANNS tends to store large-scale data on disk, which greatly increases the response time of RAG systems. This paper presents AquaPipe, which pipelines the execution of disk-based ANNS and the LLM *prefill* phase in an RAG system, effectively overlapping the latency of knowledge retrieval and model inference to enhance the overall performance, while guaranteeing data quality. First, ANNS's recall-aware prefetching strategy enables the early return of partial text with acceptable accuracy so the prefill phase can launch before getting the full results. Then, we adaptively choose the remove-after-prefill or re-prefill strategies based on the LLM cost model to effectively correct disturbed pipelines caused by wrong early returns. Finally, the pipelined prefill dynamically changes the granularity of chunk size to balance the overlap efficiency and GPU efficiency, adjusting to ANNS tasks that converge at different speeds. Our experiments have demonstrated the effectiveness of AquaPipe. It successfully masks the latency of disk-based ANNS by 56% to 99%, resulting in a 1.3× to 2.6× reduction of the response time of the RAG, while the extra recall loss caused by prefetching is limited to approximately 1%.

CCS Concepts: • **Information systems → Nearest-neighbor search**; **Search engine indexing**.

Additional Key Words and Phrases: Approximate Nearest Neighbor Search, Retrieval-augmented Generation, Large Language Models

ACM Reference Format:
Runjie Yu, Weizhou Huang, Shuhan Bai, Jian Zhou, and Fei Wu. 2025. AquaPipe: A Quality-Aware Pipeline for Knowledge Retrieval and Large Language Models. *Proc. ACM Manag. Data* 3, 1 (SIGMOD), Article 11 (January 2025), 26 pages. https://doi.org/10.1145/3709661

1 Introduction

Large Language Models (LLMs) have demonstrated impressive capabilities in understanding and generating human languages. However, due to the limitation of their training datasets, they face challenges such as hallucinations [70], outdated knowledge, and untraceable reasoning processes. To

*Jian Zhou and Fei Wu are co-corresponding authors.

Authors' Contact Information: Runjie Yu, d202381500@hust.edu.cn, Huazhong University of Science and Technology, Wuhan, Hubei, China; Weizhou Huang, huangweizhou@hust.edu.cn, Huazhong University of Science and Technology, Wuhan, Hubei, China; Shuhan Bai, shuhanbai0329@hust.edu.cn, Huazhong University of Science and Technology, Wuhan, Hubei, China; Jian Zhou, jianzhou@hust.edu.cn, Huazhong University of Science and Technology, Wuhan, Hubei, China; Fei Wu, wufei@mail.hust.edu.cn, Huazhong University of Science and Technology, Wuhan, Hubei, China.

ACM 2836-6573/2025/1-ART11
https://doi.org/10.1145/3709661

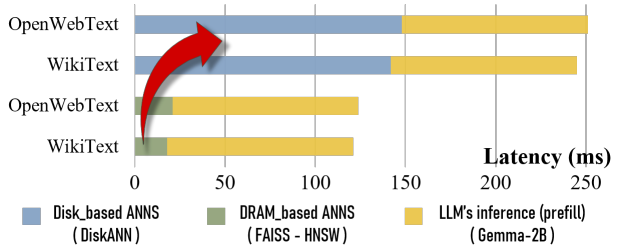


Fig. 1. Execution time breakdown for RAG systems indicates disk-based ANNS becomes the bottleneck.

address these issues, Retrieval-augmented Generation (RAG) has emerged as a promising solution by retrieving relevant knowledge from external databases to enrich users' initial prompts, which finally enhanced the generation quality of LLMs [21, 40]. In order to retrieve the most relevant knowledge (or documents) from a vast database, Approximate Nearest Neighbor Search (ANNS) provides an effective method to balance the accuracy and speed of retrieval [17, 60], which has attracted much attention. However, as the scale of the external databases expands to tens of billions [33], storing the whole index of ANNS in DRAM is becoming increasingly impractical. Therefore, many recent works have proposed disk-based ANNS algorithms [12, 47, 53, 67, 68] to store the complete dataset in disks and load the dataset to DRAM on demand. Among them, DiskANN [53] achieves an excellent trade-off between performance and accuracy by employing graph-based index, thus becomes one of the most widely used paradigm [47].

This paper finds that the disk-based ANNS becomes the bottleneck when integrated into RAG systems. Figure 1 compares the RAG systems constructed by dram-based ANNS - FAISS [17] and disk-based ANNS - DiskANN [53] under two different datasets OpenWebText [20] and Wikipedia [38]. The FAISS stores all the data in DRAM, while DiskANN caches 1% of data to simulate DRAM-constrained situations. We use SQuAD [49] as the query set and retrieve top-10 most relevant texts from the datasets to enhance the queries fed to LLM. Results indicate that the retrieval latency of DiskANN dominates the response time of the RAG system to users, that is, the entire time from user input to retrieving relevant text by ANNS to invoking LLM to generate the first token (i.e., Time-to-First-Token, TTFT). Although DiskANN optimizes the retrieval performance of graph-based ANNS by utilizing the characteristics of SSDs better, several aspects still exacerbate the growth of retrieval latency. First, as the knowledge database grows, graph-based ANNS requires an elevated number of iterations to find the relevant texts due to longer search paths on the graph (§3.1.1). Second, on the other hand, LLM tends to accept more external knowledge to improve the quality of generation, which brings a heavier workload to ANNS (§3.1.2). Third, the number of random accesses to disks increases as the expanded search scope on graph indices, which will decline cache efficiency and result in much higher latency per iteration (§3.1.3). Considering that the objective factors and trends mentioned above are difficult to change, in order to mitigate the negative impact of disk-based ANNS, it is necessary to jointly optimize ANNS and LLM from a system perspective.

The most straightforward method is developing a pipelined RAG execution paradigm to overlap part of the ANNS and LLM execution time, as shown in Figure 2. This is feasible because, while returning the complete top-k results consumes a long execution time for ANNS, part of the retrieval results can be prefetched and fed to the LLMs in the middle of the ANNS process without significantly losing accuracy. LLMs are also able to support pipeline mechanism through the *chunked-prefills* [3] method, which means that input context can be divided into several chunks and individually feed to LLM for prefill (§2.2.2). For comparison, in the naive RAG composed of normal ANNS and

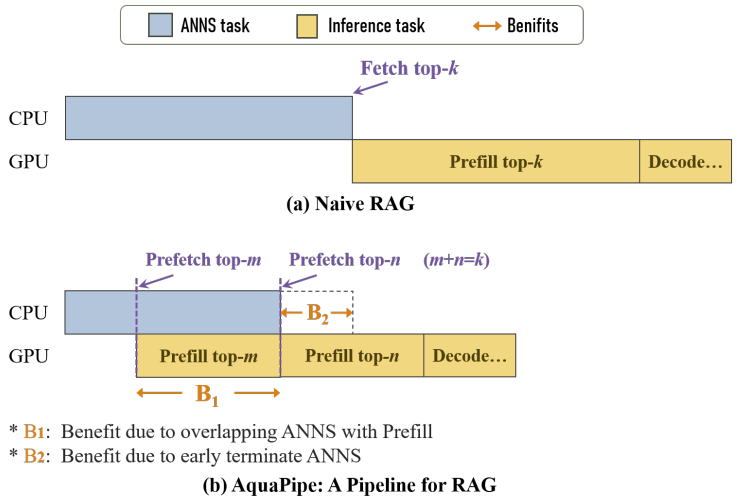


Fig. 2. Pipelined RAG Execution. The ANNS and inference are divided into two phases. Within each phase, we assume one result is fetched by ANNS and fed to LLM for inference.

LLM shown in Figure 2a, only after the ANNS fetches top-k results, the LLM's prefill task can be sequentially executed. In contrast, as shown in Figure 2b, assuming $m + n = k$, the top-m results will be prefetched and fed to LLM midway, thus overlapping the latency of ANNS with prefill task. Moreover, prefetching top-n result is load sensitive, which is able to terminate the ANNS in advance to reduce the unnecessary retrieval costs under fixed parameter search.

However, achieving efficient pipelines for ANNS and LLM confronts several pivotal challenges:

- **Recall Loss** (§3.2.1): The static prefetching method of ANNS is insensitive to recall rate, which may result in incorrect prefetching results and inevitably lead to a decrease in ANNS retrieval accuracy.
- **Disturbed Pipelines** (§3.2.2): Incorrect prefetching results can disrupt the operation of the pipeline and force it to restart new prefill tasks, which wastes a significant amount of GPU resources and increases the overall latency.
- **Splitting overheads** (§3.2.3): Splitting the entire prefill task into several chunks incurs extra overheads due to lower GPU utilization [3], making it difficult to design a fine-grained pipeline which aims to fully overlap the latency of ANNS.

To that end, we present **AquaPipe**, a quality-aware pipeline that efficiently overlap the process of disk-based ANNS and LLM to reduce the response time of RAG under various workloads, while preventing unpredictable recall loss. First, ANNS's recall-aware prefetching strategy allows for the early return of partial text without excessive accuracy loss, enabling the prefill phase to commence prior to obtaining complete results. Then, we adaptively select either the remove-after-prefill or re-prefill strategies based on the LLM cost model to effectively rectify disturbed pipelines caused by wrong early returns. Finally, the pipelined prefill dynamically adjusts the chunk size granularity to balance the pipeline efficiency and GPU utilization, accommodating ANNS tasks that converge at varying rates. Evaluation under various real production workloads shows that, due to an efficient quality-aware pipeline, AquaPipe successfully masks the latency of disk-based ANNS by 56% to 99%, thus resulting in a 1.3× to 2.6× reduction of the response time of the RAG (i.e., TTFT), while the extra recall loss caused by prefetching is limited to approximately 1%.

In summary, we make the following contributions:

- We propose AquaPipe, a quality-aware pipeline for RAG systems that overlaps the execution of ANNS and LLM prefill to effectively mask the latency of disk-based knowledge retrieval, allowing for the expansion of storage capacity without affecting RAG's response time. To the best of our knowledge, AquaPipe is the **first** work that offers equal weight to data quality, rather than merely focusing on performance as traditional pipelines do, improving pipeline efficiency without incurring high recall loss.
- We meticulously optimize multiple dimensions, including recall-aware prefetching, adaptive error correction, and dynamic granularity adjustment, to enhance pipeline efficiency while maintaining high accuracy.
- We have extended the implementation of AquaPipe to various ANNS methods, demonstrating that the core concept of AquaPipe can be flexibly and cost-effectively ported to other SOTA ANNS based RAG systems while maintaining significant optimization effects.
- We thoroughly evaluate AquaPipe with real datasets to demonstrate its efficacy and efficiency, achieving significant performance benefits of up to 2.6× while limiting extra recall loss to less than 1%.

The rest of this paper is organized as follows: §2 introduces the concepts of ANNS and RAG; The motivation and challenges of pipelining ANNS and LLM are discussed in §3; AquaPipe designs are then introduced in §4 to address these challenges and they are evaluated in §5. Finally, the related work and conclusions are presented in §6 and §7, respectively.

2 Background

2.1 Approximate Nearest Neighbor Search

2.1.1 ***Concept and Categories of ANNS.*** The approximate nearest neighbor search (ANNS) focuses on developing an efficient index structure and affiliated algorithms to swiftly pinpoint the most relevant results to any given query within a specific dataset. The mainstream ANNS algorithms include the following categories: tree-structure based [6, 9, 74], hashing-based [4, 30, 72], quantization-based [5, 35, 48], and graph-based indices [18, 26, 45, 53, 63], among which graph-based algorithms have become the SOTA paradigm due to it's better trade-off between retrieving performance and accuracy [44, 45, 47, 60].

However, to provide up-to-date and richer information, RAG continuously enriches the external corpus [11, 21, 40]. With the scale of vectors expanding to billions or even trillions, storing the whole index in DRAM is becoming increasingly impractical due to the physical limitation and high cost [18, 24, 53]. To address this, many recent works propose disk-based ANNS, such as DiskANN [23, 52, 53], AiSAQ [54], LM-DiskANN [47], SPANN [12], GRIP [68], and Zoom [67]. Notably, DiskANN, which employs graph-based indexing, becomes one of the most widely used implementations. It has been selected as the baseline for the Big-ANN competition track of NeurIPS [2] and is utilized in Vector Database(VDB) services like Weaviate [1] and Zilliz [59].

2.1.2 ***Structure and Search Mechanism of DiskANN.*** DiskANN introduces a graph-based ANNS customized for Solid State Drives (SSDs) to maximize the overall throughput. It initializes two lists when taking a user query vector: the **Candidate List** (denoted as C) and the **Result Set** (denoted as R). C is a priority queue with a fixed length of L, which stores the PQ-Distance (i.e., Product-Quantization-based approximate distances [28, 36]) of the visited nodes and their neighboring nodes and maintains the ordered state of the queue according to this lossy compression distance; R grows from an empty queue, gradually recording the actual-distance of all visited nodes (through precise vectors), while temporarily arranging these nodes in order of access.

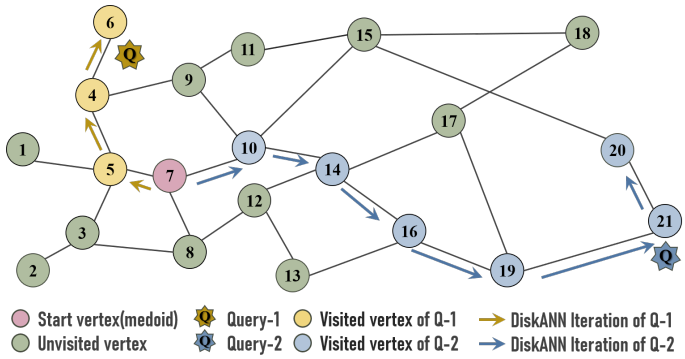


Fig. 3. DiskANN search mechanism. We visually demonstrate how DiskANN searched for top-k (assuming k=1) nearest neighbor to different queries. We further demonstrated the dynamic changes of the Candidate List and Result Set during the search process in Table 1.

Iteration	Query-1		Query-2	
	C	***R***	***C***	***R***
1	7	7	7	7
2	5,7	7,5	10,7	7,10
3	4,5	7,5,4	14,15	7,10,14
4	6,4	7,5,4,6	16,17	7,10,14,16
5			19,16	7,10,14,16,19
6			21,19	7,10,14,16,19,21
7			21,20	7,10,14,16,19,21,20

Table 1. Status of Candidate List and Result Set (denoted as C and R) upon each Iteration as shown in the example of Figure 3.

Figure 3 depicts how the candidate list C and result set R coordinates upon each iteration when processing queries with different distributions. DiskANN first selects a medoid (the red node with NO.7) of its graph index as the starting point and initializes a candidate list C. During each iteration, the algorithm will first fetch several neighbor nodes next to the most current nodes residing in the candidate list Then, it compares the quantized distance to the user query for all fetched nodes and updates C by discarding nodes whose distances are more significant. For example, for Query-1, nodes No.7 and No.5 are discarded in the second and third iterations because they are far away from the query point. Meanwhile, newly accessed nodes are appended to the result set R, which records full precision distances (more accurate than quantized distance but consumes more memory). Once all iterations are completed, DiskANN reranks the visited nodes in result set R using their distances to the query point and finally outputs the top-k nearest results, depicted as No.6 and No.4. Note that C is ordered by the quantized distance to reduce memory footprint, resulting in a difference in order between C and R after rerank.

2.1.3 ***Measuring Approximation Accuracy of ANNS.*** As depicted above, DiskANN only processes part of the nodes by traversing the graph-based index rather than computing the whole vertex set. The returned results are an approximation instead of the accurate nearest neighbors. The accurate answers must be retrieved via brute force distance comparison with all nodes, which is impossible for large datasets. Hence, graph-based approximation achieves an excellent balance of accuracy and performance. Typically, recall rate, denoted as *recall@k* [23, 52, 53], is employed

to measure the accuracy of approximating the top-k nearest neighbors. For a query vector q over a given dataset P, we would return the k nearest neighbors of q from P. Suppose that G ($G \subseteq P$) is the ground truth set of q's k nearest neighbors in P, and R ($R \subseteq P$) is the result set returned by ANNS. Then, the accuracy of the ANNS for query q is measured by $recall@k = \frac{|G \cap R|}{k}$. Moreover, $recall@k$ for a set of queries refers to the average $recall@k$ over all queries. Note that the recall rate, whose calculation relies on accurate results, is an offline measurement metric and can not be determined on the fly while responding to a user query. As the recall rate represents the proportion of correct approximations compared to the ground truth search results, recall loss denotes the proportion of incorrect retrieval results, which is another frequently used concept in this paper and equals $1 - recall@k$.

2.2 Retrieval-Augmented Generation

To address the issue of hallucinations, outdated knowledge, and opaque, untraceable reasoning processes of Large Language Models (LLMs), Retrieval-Augmented Generation (RAG) systems adopt knowledge retrieval to assist LLMs in generating more comprehensive answers [11, 21, 40, 70, 71].

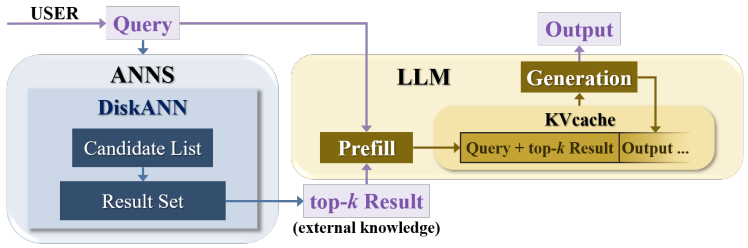


Fig. 4. An ANNS-based RAG system, exemplified by DiskANN.

2.2.1 ***Architecture of RAG System.*** As shown in Figure 4, we demonstrate a basic RAG system architecture employing DiskANN. The two most important modules of an RAG system are knowledge retrieval and a large language model (LLM), among which the knowledge retrieval module typically uses ANNS to balance the retrieval performance and accuracy. When the RAG system receives a user query, it calls the ANNS to return the top-k relevant texts and combine the user's query with the top-k external knowledge to form an enhanced context. Subsequently, the RAG system submits the enhanced query context into LLM for inference and generates an output sequence.

2.2.2 ***LLM Inference Workflow.*** The RAG system combines the user's query and the text retrieved from ANNS as the full input sequence before sending them to the LLM for inference. An inference job is divided into two phases: ***prefill*** and ***generation*** [58, 65]. During the prefill phase, LLM embeds the inputs through *embedding* matrices and leverages several layers of attention neural networks to calculate KV tensors of the input sequence, which is also known as $KVcache$. Then, the *genaration* phase uses the $KVcache$ as the input to launch multiple decoding tasks and generate one token per decoding task. The output generation stops until it encounters an ending token. Although changing numbers of decoding tasks lead to unpredictable execution times for the generation phase, the speed at which LLM can "speak" to humans is relatively trivial. On the contrary, many literatures consider the Time-to-First-Token (**TTFT**) a more important performance evaluation metric affecting the user experiences of LLM inference jobs [19, 29, 43]. Specifically, the TTFT of RAG covers the aggregated execution time of knowledge retrieval and LLM prefill phase. Considering the slower knowledge retrieval speed of disk-based ANNS and the growing latency of

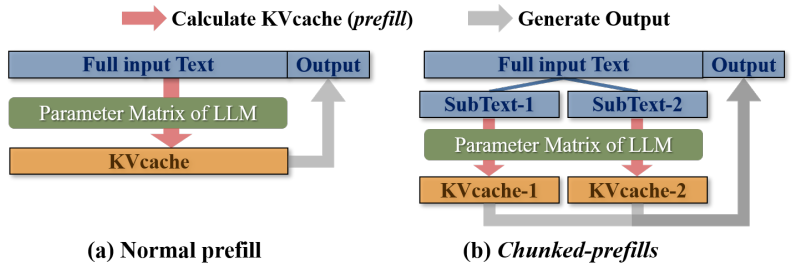


Fig. 5. *chunked-prefills*: Splitting the inputs into multiple chunks to construct multiple prefill subtasks.

prefill due to longer context, the TTFT of RAG has significantly increased. That is, we mainly focus on reducing the TTFT of RAG systems in our research.

Chunked-prefills: The prefill phase leads to high GPU utilization and long execution time when processing the long input sequence. Considering that multiple subsequent decode tasks have a relatively low compute utilization, SARATHI [3] proposes mixing decode requests with prefill requests in each batch to increase the overall compute utilization. To support this idea, SARATHI further proposes *chunked-prefills*, which splits an input sequence into multiple chunks, thereby dividing the entire prefill task into multiple subtasks accordingly, as shown in Figure 5. For our research, *chunked-prefills* provides a foundation for the pipelining ANNS and LLM, due to its ability to segment prefill phase.

3 Motivation and Challenges

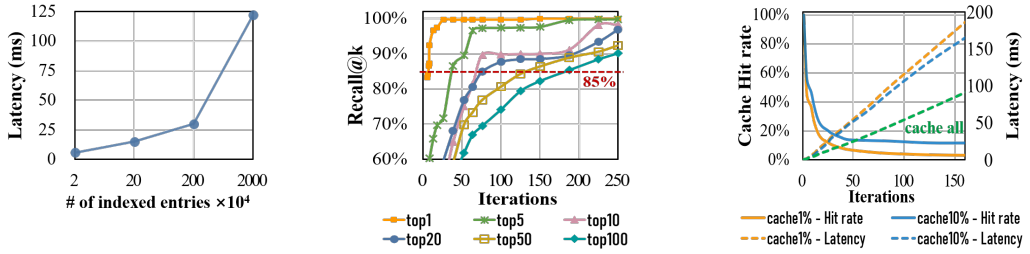
This section motivates and challenges our work. First, we analyze why disk-based ANNS becomes the bottleneck in §3.1. Then, we discuss the challenges of preventing recall loss and reducing the extra LLM chunked-prefills overhead in §3.2.

3.1 Why disk-based ANNS becomes New Bottleneck in RAG systems

Although disk-based ANNS (e.g., DiskANN) addresses the constraints on the storage capacity, it inevitably increases the retrieval latency of ANNS in RAG systems. Apart from the fact that SSD has poorer access performance than DRAM, the following three reasons fundamentally restrict the performance of ANNS.

3.1.1 ***The growth of ANNS index scale.*** The expansion of the RAG corpus has enlarged the ANNS index, resulting in more scattered ground truth of ANNS, which further expands the search range and precipitously elevates retrieval latency. To better demonstrate this issue, we constructed DiskANN index using varying sizes of corpora and conducted tests on their retrieval performance, as depicted in Figure 6a. Note that these corpora of different sizes are randomly sampled from Wikipedia [38], ensuring their distribution consistency. As data scales grow, our tests indicate an increasing retrieval latency. Specifically, the retrieval latency caused by an index containing 20M vectors is 20× higher than 20K.

3.1.2 ***Richer context requires higher top-k.*** The performance of LLMs is increasingly contingent on the quality and richness of the contextual information provided [7, 15, 38, 42], which increases the number of retrieval targets, aka top-k. However, as the k increases (e.g., 10 to 100), ANNS needs to extend the search iterations to obtain more correct results, thereby significantly increasing retrieval latency. Figure 6b shows how the number of iterations impacts the recall rate under different retrieve sizes. To ensure that the recall@k of ANNS is greater than 85%, the top-100 retrieval task requires 30× iteration steps than the top-1 retrieval task.



(a) Impact of Index Scale. (b) Recall at Different Iterations. (c) Changes of Cache Efficiency.

Fig. 6. Objective factors causing higher latency of disk-based ANNS: (a) Growing index scale results in higher ANNS latency. (b) Higher top-k drives more iterations. (c) Graph-based index leads to low cache efficiency. Note that for all the three tests, DiskANN constructs index on the Wikipedia [38] dataset and tested by SQuAD [49] dataset.

3.1.3 ***Low Cache Efficiency due to Graph-based Index.*** Given that DiskANN's caching mechanism primarily focuses on caching nodes near the center of the graph index, once the search radius expands beyond this localized area, the effectiveness of the cache will be limited. To illustrate the inefficiency of caching mechanism, Figure 6c evaluates cache efficiency across different cache ratios (1% and 10%). As iteration steps increase, Cache Hit Rate rapidly declines, ultimately approaching the cache ratio, thereby validating the inefficiency of caching mechanism when the search range is expanded. Thus, caching mechanism minimally reduces ANNS latency, with limited gains even at 100% cache, emphasizing its limited impact on overall performance enhancement.

Summary: The above trends and facts remarkably elevate the latency associated with disk-based ANNS in RAG scenarios. However, they are irreversible, and we can not alter them. Hence, this paper seeks to optimize the holistic performance of both ANNS and LLM from a system-level perspective to mitigate the adverse effects of disk-based ANNS.

3.2 Challenges of Pipelined Execution

We aim to build a system-level pipeline to overlap the execution of ANNS and LLM, thus hiding the latency. To achieve this, both ANNS and LLM's inference (mainly prefill phase) must be able to be divided into multiple subtasks. The division of prefill is enabled by *chunked-prefills* (§2.2.2), while the division of ANNS can also be performed in units of iteration steps. However, splitting ANNS and LLM to support pipeline strategy leads to several challenges.

3.2.1 ***Elevated Recall Loss due to Early Returns.*** We denote the splitting of ANNS results through iteration steps and early returning partial retrieval objects as **ANNS Prefetching**. However, the result prefetching may result in incorrect early results, which causes extra recall loss (proportion of incorrect approximation). While it would be conceivable to establish a static optimal solution for prefetching to minimize recall loss if the retrieval performance were consistent across various queries, this becomes impractical due to the significant distribution differences between various queries and knowledge bases, as depicted in Figure 3 and Table 1, where data characteristics seriously affect the performance of ANNS. Given this, the recall rate grows at varying speeds under different queries, as shown in Figure 7, it is difficult for prefetching plans to strike a balance between search performance and accuracy.

We present two typical prefetching plans in the Figure. Both plans prefetch twice and get five results each time. Plan A performs prefetching at iteration-8 and iteration-16, initiating prefetching

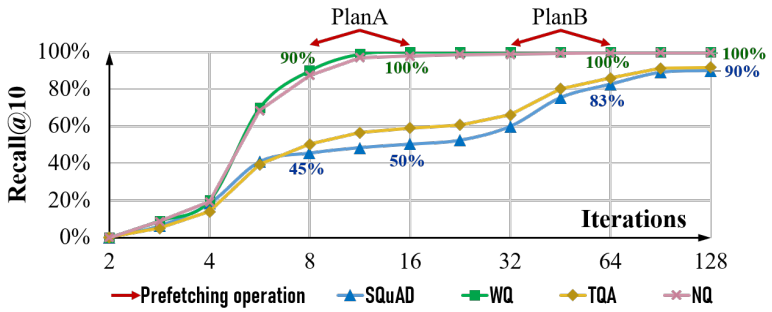


Fig. 7. Recall@10 of DiskANN under different queries(NQ [39], SQuAD [49], TQA [34] and WQ [10]) on Wikipedia [38] and schematic of different prefetching plans.

earlier with smaller intervals. In contrast, Plan B performs prefetching at iteration-64 and iteration-128, initiating prefetching later with larger intervals. For queries represented by WQ, both Plans A and B achieve negligible recall loss. However, since Plan A starts prefetching earlier, it offers better performance. For queries represented by SQuAQ, allowing iterations to terminate automatically results in a 10% recall loss. If Plan A is adopted, the recall loss will approach 50%, but the pipeline can start earlier at iteration-8, enhancing performance at the cost of accuracy. Conversely, Plan B limits the recall loss to about 10%, but the pipeline starts at iteration-64, thereby guaranteeing accuracy with suboptimal performance. In summary, given the highly variable recall rate growth speed across different queries, static prefetching solutions fail to guarantee both high performance and accuracy. The importance of this problem is evident considering the consequences of mindlessly prefetching intermediate results and feeding them to LLM.

3.2.2 ***Disturbed Pipelines due to Error Correction.*** The reduction in the accuracy of knowledge retrieval significantly impacts the generation quality of LLMs, which has been supported by numerous papers [21, 25, 31, 40, 57]. For example, in IRCoT[57], the average retrieval recall of the baseline method is 19.55% lower than that of IRCoT, resulting in an average reduction of 35.25% in the improvement effect of retrieval on answer F1 metric compared to IRCoT. Unfortunately, the ANNS Prefetching strategy(§3.2.1) will inevitably return incorrect retrieval results too early, leading to reduced retrieval recall rates and compromised generation quality of LLMs when such results are utilized.

Therefore, to ensure the high accuracy of ANNS, incorrect prefetching results need to be corrected, which will cause disturbed pipelines. Specifically, once an incorrect pipeline is discovered, one straightforward solution is to cancel it and restart a new prefill task based on the correct retrieval results. However, if the error is detected too late, canceling the current prefill task will waste substantial GPU resources already consumed and significantly increase the overall prefill latency. We further illustrate the difference in the time to discover error pipelines in Figure 8 and demonstrate the high cost of error correction in particular instances, such as "TQA top20" and "SQuAD top20". Note that the time of "Discovering Errors" in Figure 8 represents the time from starting a prefill task to discovering incorrect retrieval results contained in the prefill task.

3.2.3 ***Extra overheads due to Increased Number of Chunks.*** To support the pipeline strategy, *chunked-prefills*(§2.2.2) provide a method to split the whole prefill task into multiple subtasks. The more splits we have, the more overlapped execution time between knowledge retrieval and LLM prefill we would expect. However, as prefill is a GPU computation-intensive task, splitting a large task into several smaller ones results in lower GPU utilization, thereby increasing the overall

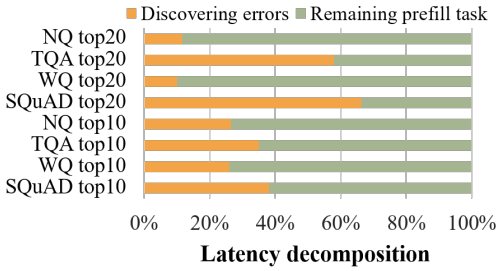


Fig. 8. Cases where incorrect pipelines occur at different stages of LLM prefill. The horizontal axis indicates the execution time of unoptimized pipelines that ignore the error.

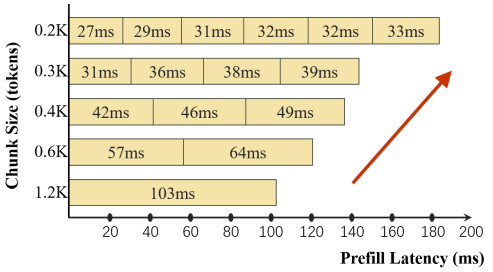


Fig. 9. *Chunked-prefills* increases the total latency: A context of 1.2k tokens is divided into several chunks for prefill. This test is based on Gemma-2B and RTX 3090.

prefill latency. We further test the extra cost of *chunked-prefills* in Figure 9 and demonstrate the trend of extra latency increasing with the number of chunks. Results indicate that when dividing 1.2K input tokens into six equal-sized chunks, the overall prefill latency increased by 1.78 times.

4 AquaPipe Design

4.1 Overview

This section proposes three corresponding designs to address the above challenges. As illustrated in Figure 10, the overall architecture of **AquaPipe** can be conceptualized as a producer-consumer model. ANNS serves as a producer and inserts the prefetched results into Pipeline Pool; LLM acts as a consumer and fetches the results from pipeline pool for prefill. Moreover, ANNS also identifies and removes elements incorrectly inserted into pipeline pool. Two key elements that collect the multiple design components include:

- **Pipeline Pool** (*PP*): A pool to store the prefetched results to support pipeline strategy, supporting three operations: produce, consume and remove. As shown in Figure 10.
- **Result Set** (*R*): A list containing all visited nodes with their precise distance (see §2.1). Unless otherwise stated, the *R* mentioned in this section will be reranked in each iteration.

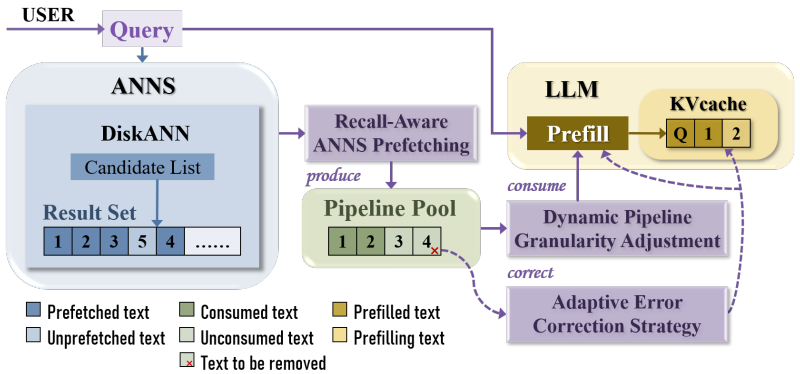


Fig. 10. **AquaPipe Design Overview.**

Based on this architecture, we employ Recall-Aware ANNS Prefetching(§4.2) to balance prefetching accuracy and performance; develop Adaptive Error Correction Strategy(§4.3) to minimize the cost of correcting erroneous pipelines; and leverage Dynamic Pipeline Granularity Adjustment(§4.4) to select the optimal prefill granularity to build efficient pipelines.

4.2 Recall-Aware ANNS Prefetching

There are two major issues in designing a prefetching strategy: the **time** and **quantity** of prefetching, which are both challenged by the corresponding impact of recall loss(§3.2.1). However, recall loss can only be obtained given the ground truth set, which is not applicable unless a brutal force search is performed. Therefore, we utilize the dynamic historical retrieval information of R during the retrieval process to adjust the time (§4.2.1) and quantity (§4.2.2 and §4.2.3) of prefetching in real-time, and employ local accuracy of PP as a reward-punishment mechanism(§4.2.4) to balance the prefetching speed.

4.2.1 ***Size of Pipeline Pool: Determining when to perform prefetching.*** The timing of prefetching significantly impacts the performance and retrieval quality of RAG pipelines, which is challenged by the unstable growth speed of ANNS recall rate. Fortunately, graph based ANNS is affected by marginal effects, the last retrieved nearest neighbor usually requires a larger retrieval range and greater retrieval latency [41]. Thus, the recall rate growth generally follows a pattern of rapid initial growth followed by a slower pace. Consequently, prefetching strategy is designed to align with this recall rate growth trend by adhering to the following guidelines: prefetch more frequently during the early stages of ANNS and less frequently during the later stages.

Specifically, when performing the ANNS task of retrieving the top-k results, assuming that the n_{th} prefetching operation occurs in the Opt_n iteration, then the occurrence time of the $n{+}1_{th}$ prefetching operation can be calculated by Equation 1. In this equation, the real-time prefetching progress, defined as $PP.size/\text{top-}k$ (where < 1), is used to determine whether the current ANNS is in the early or late stage. This value is multiplied by the current prefetching quantity, $PP.size$, to serve as the basis for the prefetching step, thereby constructing a nonlinear function that better complies with the aforementioned guidelines. Additionally, to ensure $Opt_{n+1} > Opt_n$, a basic increment of 1 is added. A real-time evaluation factor, $Balancer_n$, which characterizes the prefetching accuracy (as mentioned in Section 4.2.4) is introduced to dynamically balance the prefetching speed. When this factor is positive, indicating high accuracy of the current prefetching, it can be subtracted to increase the prefetching frequency; otherwise, the prefetching frequency needs to be reduced.

$$Opt_{n+1} = Opt_n + \lfloor (PP.size/top\text{-}k) * PP.size + 1 - Balancer_n \rfloor \tag{1}$$

In summary, this strategy dynamically adjusts the prefetching frequency based on the prefetching progress and real-time accuracy to balance the performance and accuracy of prefetching. First, Equation 1 ensures that as the size of the Pipeline Pool (i.e., the number of prefetched results) increases, the frequency of prefetching gradually decreases. Then, the *Balancer* in the Equation 1 provides an opportunity to adjust the prefetching frequency based on the accuracy of the current prefetching behavior.

4.2.2 ***Stability of Result Set: Determining the max quantity of prefetching.*** The analysis in this section will be based on a fundamental fact: due to the real-time sorting of elements in R, the more stable the top-K elements are, the more likely R is to contain more correct results. To quantify this stability, we designed the following indicators by comparing the positional variations of elements in R between every two prefetching operations:

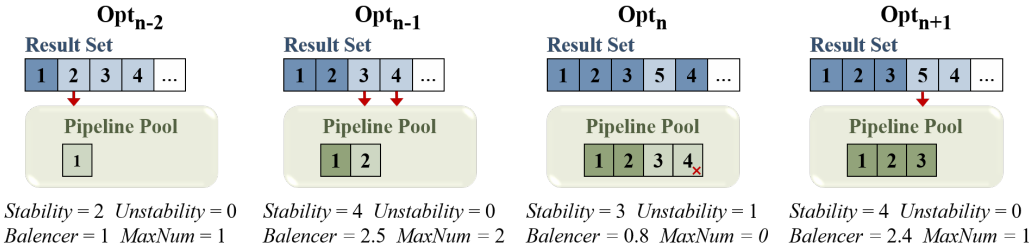


Fig. 11. Demonstration of prefetching strategy (top-k=4). Opt_n is calculated by Equation1; *Stability* and *Unstability* is explained in §4.2.2, while *MaxNum* is calculated by Equation2; *Balencer* is calculated by Equation3. The red arrow indicates the actual prefetching operation, determined by the Historical location in §4.2.3

- ***Stability*** = The number of elements with equal positions in R between two prefetching operations.
- ***Unstability*** = The number of elements with different positions in R between two prefetching operations, and these elements have already been prefetched.

We further elaborate the meanings of *Stability* and *Unstability* in Figure 11: The positions of the top-4 elements in R remain unchanged between Opt_{n-2} and Opt_{n-1}, so *Stability* of Opt_{n-1} is equal to 4. In Opt_n, the newly inserted element 5 changes the position of element 4 in R, resulting in *Unstability* being equal to 1, while *Stability* being reduced to 3.

Considering the quantified stability of R, we set the maximum number of each prefetching operation as ***MaxNum***. The following equation calculates this crucial value, which plays a significant role in preserving an acceptable recall rate.

$$MaxNum = \lfloor (Stability - PP.size + Balancer_n)/2 \rfloor \tag{2}$$

In Equation(2), $Stability - PP.size$ represents the potential number of opportunities for a prefetching operation; *Balancer*(§4.2.4) refers to the reward or punishment given to the max number of prefetching based on the current prefetching accuracy; Finally, to ensure the accuracy of prefetching, we divide the value of *MaxNum* by 2 to eliminate the influence of boundary conditions. Based on this design, we observe that in the example shown in Figure 11, *MaxNum* dynamically changes with the stability and *Balancer*, thus achieving the goal of automatically adjusting the number of prefetches to balance prefetching speed and accuracy.

4.2.3 ***Historical location: Final determination of prefetching or removing.*** After calculating the *MaxNum* of prefetching based on the overall stability of R, we also need to carefully consider whether each element is worth prefetching. Specifically, AquaPipe will record the historical position of each element in R after each prefetching operation, with a retention time of several prefetching intervals. Then, when determining whether a certain text should be prefetched, it is only necessary to check whether the historical and current positions of the element are equal, otherwise, it cannot be prefetched. In addition, if any element in the *PP* is found to have a positional change during iterations, it should be removed from *PP*.

For example, as the red arrow shown in Figure 11, from Opt_n to Opt_{n+1}, the positions of elements 5 and 4 remain stable. Considering the *MaxNum* of Opt_{n+1} is equal to 1, we can only prefetch element 5 and insert it to *PP*. And for removing incorrect elements, element 4 was removed from *PP* at Opt_n due to its positional change between Opt_{n-1} and Opt_n.

4.2.4 ***Accuracy of Pipeline Pool: Balancing the accuracy and speed of prefetching.*** Although the size of Pipeline Pool (§4.2.1) and the stability of Result Set (§4.2.2) can adjust the speed of prefetching dynamically, they are still not sensitive enough to the growth of Recalls for various queries. In queries with fast Recall@k growth, we should increase the prefetching speed to build a more efficient pipeline; In queries with slower Recall@k growth, we should slow down the prefetching speed to reduce the Recall Loss. For this, we further designed a balance factor $\boldsymbol{Balancer_n}$ based on the current accuracy of PP, which can be characterized by $Stability$ and $Unstability$ mentioned above.

$$Balancer_n = (Stability - 4 * Unstability + Balancer_{n-1})/2 \tag{3}$$

Equation(3) provides the calculation method for $Balancer_n$. Overall, the calculation of $Balancer$ is influenced by the historical $Balancer$ values, preventing excessive oscillations of the prefetching speed during the iteration process. As shown in Figure 10, from $Opt_{n-1}$to Opt_n, the accuracy of PP decreases due to incorrectly prefetching element 4, resulting in a decrease in $Balancer$. This further extends the time interval of prefetching operations, reduces the maximum number of prefetching, and finally improves the recall rate.

4.3 Adaptive Error Correction Strategy

Given the inherent nature of ANNS prefetching, it is crucial to have a result correction mechanism in place to handle inappropriate early returns. The adaptive error correction strategy provides two solutions once it discovers that the executed prefill task has contained the error elements:

- **Re-prefill:** stops the current prefill task and restarts a new one based on correct elements.
- **Remove-after-prefill:** continues the current prefill task and removes the incorrect element's $KVcache$ after the prefill task is completed.

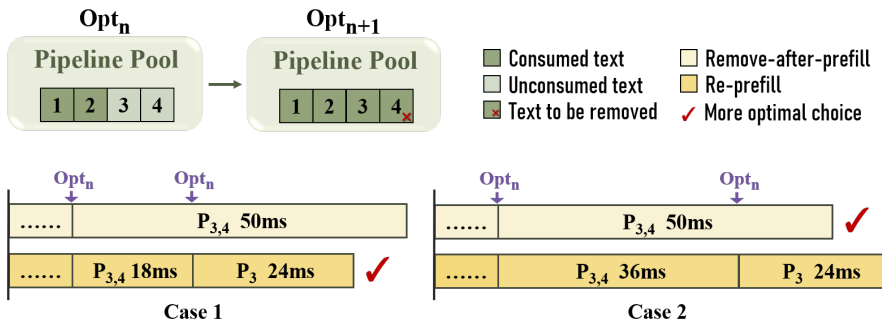


Fig. 12. Dynamic correcting solutions: After starting the prefill task, if errors are detected earlier(i.e., Case-1), it is more appropriate to adopt a Re-prefill strategy; If errors are discovered later(i.e., Case-2), it is more appropriate to adopt a Remove-after-prefill strategy.

Adopting the above two solutions depends on when an error is detected. If the error is detected early, in other words, only a few of the top-k have returned, we opt to drop the existing $KVcache$ and perform re-prefill. On the other hand, if the error is detected late and many of the elements have already been prefilled into the $KVcache$, we opt to preserve existing results and remove incorrect elements after the prefill phase. As shown in Figure 12, the text-3 and text-4 are prefetched in Opt_n and are taken to execute prefill. Unfortunately, we found that text-4 is incorrectly prefetched in Opt_{n+1}, so we have to withdraw the prefill task for text-4. At this time, the Re-prefill strategy is

more optimized in Case-1, where the Opt_{n+1} prefetching occurs immediately after the Opt_n; The Remove-after-prefill strategy is more optimized in Case-2, where the Opt_{n+1} prefetching occurs long after the Opt_n. Therefore, we will comprehensively consider the two error correction strategies and adaptively select the optimal solution by calculating their cost and benefits in real time.

4.4 Dynamic Pipeline Granularity Adjustment

Due to the additional cost of *chunked-prefills*, simply starting prefilling whenever ANNS prefetches a text leads to an increased number of prefill subtasks, which in turn reduces the overall performance. Specifically, we have two strategies for scheduling pipelines:

- **Fine-grained pipeline** starts prefilling immediately after receiving new prefetched results, which effectively overlaps ANNS latency while incurring higher costs of *chunked-prefills*;
- **Coarse-grained pipeline** waits for a batch of prefetched results to initiate batch prefilling, but it reduces the additional cost of excessive chunking.

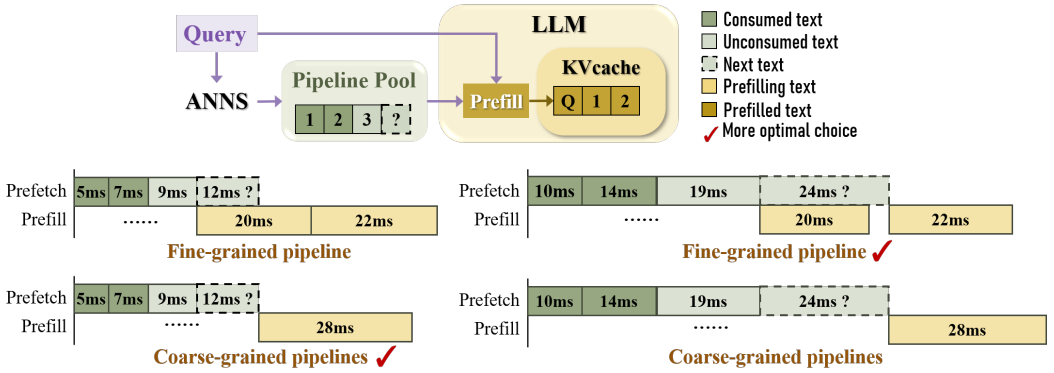


Fig. 13. Dynamic pipeline granularity: Adopting coarse-grained pipelines for ANNS with faster prefetching speed and fine-grained pipelines for ANNS with slower prefetching speed can achieve the best latency overlapping effect.

The latency of per-chunk prefill remains reliably stable, given a fixed chunk size under specific hardware and software environments. This predictability allows us to measure the latency of *chunked-prefills* in advance, instilling confidence in the system's performance. It further makes it possible to optimize the pipeline strategy by dynamically balancing the granularity of the pipeline. The principle to follow is straightforward and manageable: calculate the optimal overlap granularity based on the speed of prefetch and the latency of *chunked-prefills*.

Figure 13 shows the state of the RAG system after Opt_n in Figure 12, where text-1,text-2 have already been prefilled, and text-3 are in an unfilled state. At this point, The coarse-grained pipeline strategy will wait for the next text before executing batch prefilling. The fine-grained pipeline strategy will immediately perform prefilling on text-3, then the next prefilling will be initiated after the prefilling of text-3 and prefetching of the subsequent text. Case-1 and Case-2 further illustrate that to achieve better optimization results, ANNS with faster prefetching speed should employ a coarse-grained pipeline, while ANNS with slower prefetching speed should utilize a fine-grained pipeline.

5 Evaluation

In this section, we will describe the implementation of AquaPipe and evaluate its efficiency on various datasets, retrieval workloads, and LLMs. Firstly, we compared AquaPipe with various baselines and evaluated the response latency TTFT of RAG (§5.2.1) and the independent latency of ANNS (§5.2.2) to demonstrate the overall performance improvement. Then we evaluated the effectiveness of each design: AquaPipe can effectively balance the speed and accuracy of prefetching (§5.3.1), adaptively reduce the cost of correcting erroneous pipelines (§5.3.2), and dynamically adjust the granularity of prefill to improve the pipeline performance (§5.3.3).

Evaluation results show that AquaPipe can greatly improve various performance metrics. Specifically, AquaPipe can effectively reduce the Independent latency of disk-based ANNS by 56% to 99%, while reducing the response time (i.e., TTFT) of RAG by 1.3× to 2.6× under various conditions. Meanwhile, AquaPipe can effectively control the recall loss of the prefetching strategy at about 1%, reduce the average error correction cost by 55.6% and 32.6%, and further reduce the TTFT of pipeline by 11.2% and 13.5% under dynamic pipeline granularity.

5.1 Experimental Setup

5.1.1 ***Hardware settings.*** We use an NVIDIA RTX3090 GPU server with two GPUs to run LLMs. The server is equipped with 256GB of memory, two Intel(R) Xeon(R) Platinum 8336C CPUs, and two INTEL SSDPE2KE032T8 3.2TB SSDs.

5.1.2 ***Implementation and Porting Overhead.*** AquaPipe is lightweight and capable of dynamically adapting to varying datasets and ever-changing system statuses. It can be incorporated into offerings as a value-added proposition by knowledge base and LLM providers.

On the one hand, the Recall-Aware ANNS Prefetching strategy (§4.2) can be widely applied to various ANNS algorithms. Although this article focuses on describing the prefetching strategy in the context of DiskANN[53], other ANNS algorithms such as SPANN[12] also exhibit similar search marginal effects (§4.2.1) and temporarily store the search results in a specific result set (§4.1), which means the implementation of the prefetching strategy is not limited to specific algorithms as long as these two conditions are met. We further implemented the prefetching strategy in DiskANN and SPANN at a low porting cost, with over 200 lines of C++ code, respectively.

On the other hand, *chunked-prefills* can be implemented with 900+ lines of Python code in the source code of LLMs: Gemma [55], OPT [69] and Llama [56]. Accordingly, the proposed pipeline framework can be realized with 200+ lines of Python code to construct a complete RAG system. Furthermore, AquaPipe could be transparently and directly leveraged by customers, adding significant optimization without requiring substantial modifications.

5.1.3 ***Tasks and Datasets.*** We evaluate AquaPipe by constructing ANNS index on external Knowledge Bases and test it with real Question Answering (QA) task sets. Notably, all the texts are encoded into vector representations using the same embedding model Sentence-BERT [50] for performing ANNS retrieval.

- **Knowledge Base**: We choose Wikipedia [38] as the external knowledge base, which is a preprocessed text set containing approximately 20 million text chunks of 100 tokens.
- **Test Task**: We test the performance of the RAG system by four QA tasks, namely SQuAD[49], WQ[10], TQA[34], NQ[39]. In each set, there are thousands to tens of thousands of questions like "In what country is Normandy located?".

5.1.4 ***Models.*** To build a complete RAG system to evaluate the performance of AquaPipe, we selected Gemma-2B [55] and OPT-1.3B [69], both of which are LLMs that support the feature of *chunked-prefills*.

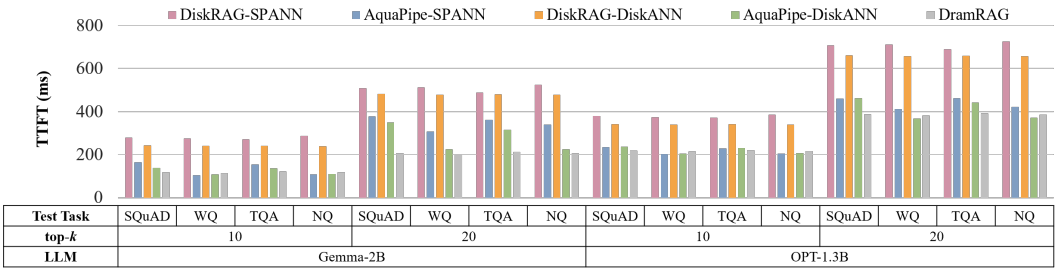


Fig. 14. TTFT under different conditions, demonstrating the improvement of AquaPipe in response speed.

5.1.5 ***Baselines.*** We compare AquaPipe to the following baselines implemented by us:

- **DiskRAG-DiskANN**, a naive RAG that executes disk-based ANNS (i.e., DiskANN [53]) and LLM inference in series.
- **DiskRAG-SPANN**, a naive RAG that executes disk-based ANNS (i.e., SPANN [12]) and LLM inference in series.
- **DramRAG**, a naive RAG that executes dram-based ANNS (i.e., HNSW in Faiss [17]) and LLM inference in series.

5.1.6 ***Metrics.*** We use TTFT (§2.2.2) as the major evaluation metric to characterize the response speed of RAG systems. We also adopted recall@k (§2.1.3) to quantitatively evaluate the accuracy of ANNS under the pipeline strategy.

5.2 Overall Improvement

In this section, we compare AquaPipe against the baseline methods **DiskRAG** and **DramRAG**, where both DiskRAG and the corresponding AquaPipe are implemented using two ANNS methods. In summary, we demonstrate the overall performance improvement of AquaPipe by testing the response latency of RAG (§5.2.1) and the Independent latency of ANNS (§5.2.2), resulting in significantly better performance than DiskRAG.

5.2.1 ***Response latency of RAG: TTFT.*** In Figure 14, we demonstrate that AquaPipe effectively improves the response speed of RAG systems by measuring TTFT.

Firstly, we demonstrate the variations in TTFT of AquaPipe when confronted with query sets exhibiting different distributions. The comprehensive observation of Figure 7 and Figure 14 indicates that AquaPipe actively detected the slower growth in retrieval accuracy of SQuAD and TQA during ANNS, and provided them with a longer retrieval path, resulting in a relative increase in TTFT. Conversely, for WQ and NQ, the trend is reversed. This indicates that AquaPipe dynamically and adaptively balances the speed and accuracy of ANNS, as further elaborated in §5.3.1.

Secondly, the experimental results have shown that compared to DiskRAG, AquaPipe reduces the response time of RAG by 1.3× to 2.2× under DiskANN and 1.3× to 2.6× under SPANN. Our tests also show that despite utilizing disk-based ANNS, in most cases, AquaPipe is very close to or even better than DramRAG in terms of response speed. Thi is attributed to AquaPipe's ability to fully overlap the sequential "ANNS retrieval" and "LLM prefilling" tasks performed by both DiskRAG and DramRAG, enabling it to introduce disk-based ANNS to increase knowledge base capacity while effectively control the growth of response latency.

5.2.2 ***Independent latency of ANNS.*** To better demonstrate the performance improvement of ANNS, we define the unoverlapped running time of ANNS as the "Independent Latency", which is equal to $TTFT - Latency_{prefill}$, where $Latency_{prefill}$ represents the overall latency of prefill phase.

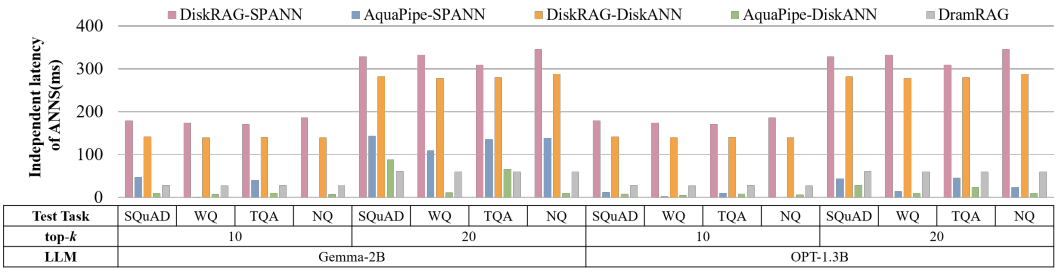


Fig. 15. Independent latency of ANNS under different conditions, proving that AquaPipe can fully mask the latency of ANNS.

As shown in Figure 15, the experimental results have shown that compared to DiskRAG, AquaPipe significantly reduces the Independent Latency of ANNS by 68% to 96% under DiskANN and by 56% to 99% under SPANN. Furthermore, the ANNS Independent Latency of AquaPipe is smaller than that of DramRAG even in most of the cases. These results further indicate that AquaPipe has made disk-based ANNS no longer the main bottleneck of RAG systems.

5.3 Ablation Studies

In this section, we evaluate the effectiveness of AquaPipe's three sub-designs and the impact of various environmental configurations on AquaPipe through more microscopic tests. Unless otherwise specified, all the DiskRAG and AquaPipe mentioned in the evaluations of this section are constructed based on DiskANN.

5.3.1 ***Effect of Recall-Aware Prefetching.*** The biggest challenge of prefetching is that returning partial retrieval results in advance will reduce the accuracy of ANNS, as mentioned in §3.2.1. Therefore, we will demonstrate the effective control of recall loss by Recall-Aware ANNS Prefetching in this subsection.

Firstly, we selected two test tasks with different recall growth rates to characterize the prefetching operation trajectory when top-k=10, as shown in Figure 16. This figure demonstrates that the Recall-Aware prefetching strategy of AquaPipe effectively distinguishes tasks with different recall growth rates and adopts a matching prefetching speed. More importantly, the error correction capability (§4.2.3) of the Recall-Aware prefetching strategy can promptly correct the prefetching results when the prefetching accuracy drops too quickly. For example, in the SQuAD task shown in Figure 16, before iterating to point A, the prefetching speed (blue solid line) is significantly faster than the growth of accurate results (red solid line), resulting in a high recall loss of prefetching. However, our Recall-Aware prefetching strategy promptly detects and removes the incorrect prefetching results at point A, thereby improving the prefetching accuracy of ANNS.

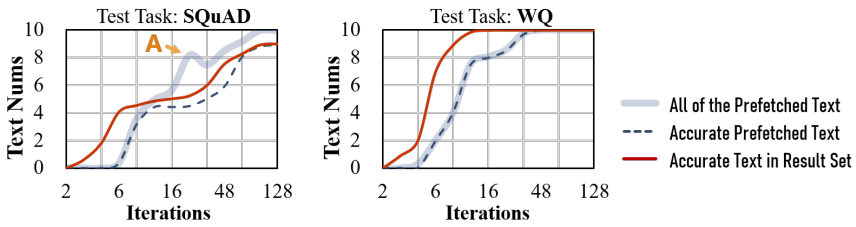


Fig. 16. Prefetching trajectory of AquaPipe, indicating that the prefetching speed is sensitive to recall rate.

Test Task	top-k	Extra Recall Loss		
		Plan A	Plan B	AquaPipe
SQuAD	10	38.97%	<0.01%	0.40%
	20	40.08%	<0.01%	0.65%
WQ	10	0.08%	<0.01%	<0.01%
	20	0.02%	<0.01%	0.04%
TQA	10	32.71%	<0.01%	1.06%
	20	41.17%	7.6%	1.53%
NQ	10	1.77%	<0.01%	0.17%
	20	2.93%	0.55%	0.64%

Table 2. Extra Recall Loss of AquaPipe and other static prefetching plans under different conditions

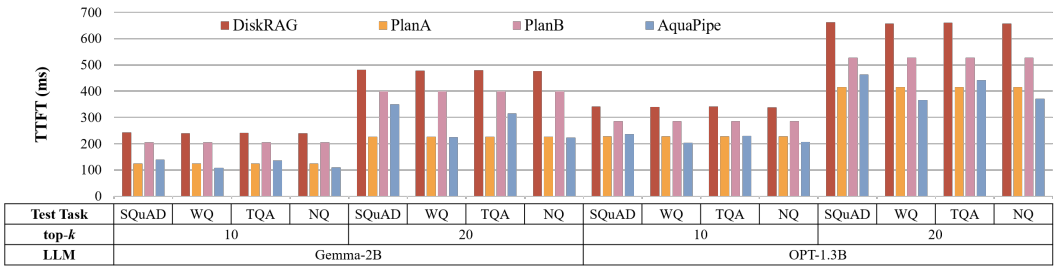


Fig. 17. Performance comparison under different prefetching strategy.

We further test the extra recall loss and performance impact of AquaPipe and other static prefetching plans (§3.2.1) in various conditions and list them in Table2. Assuming that $Origin\_Recall@k$ is the recall rate of DiskANN without prefetching, and $Prefetch\_Recall@k$ is the recall rate of prefetched results, then the extra recall loss is calculated using $Origin\_Recall@k - Prefetch\_Recall@k$. The results show that the extra recall loss of AquaPipe is secured between 0.00% and 1.5%, which is much lower than the radical static Plan A and not significantly higher than the extremely conservative static Plan B. Considering the performance of pipeline demonstrated in §5.2, the extremely low Extra Recall Loss of AquaPipe proves that the Recall-Aware ANNS Prefetching effectively balances the performance and accuracy of prefetching. Figure 17 compares the performance of different strategies. The results indicate that since Plan A sacrifices the accuracy, it performs better than Plan B on all tests. Although Plan B achieves the best result accuracy, it reduces the overall performance in most cases. Our AquaPipe achieves acceptable performance with an accuracy comparable to Plan B, given that, in some cases, it performs even better than Plan A.

5.3.2 ***Effect of Adaptive Error Correction.*** To better illustrate the effectiveness of Adaptive Error Correction, we compared the error correction time of AquaPipe with two static methods mentioned in §4.3, and also statistically analyzed the improvement of retrieval accuracy by error correction strategies.

As shown in Figure 18, in various testing conditions, Re-prefill and Remove-after-prefill both have a probability to be more optimal than each other. However, by adaptively selecting error correction mechanisms, AquaPipe's error correction time is on average 55.6% and 32.6% lower than these two static error correction methods, respectively. This further indicates that AquaPipe's Adaptive Error Correction Strategy can effectively reduce error correction time and quickly repair the disrupted pipelines.

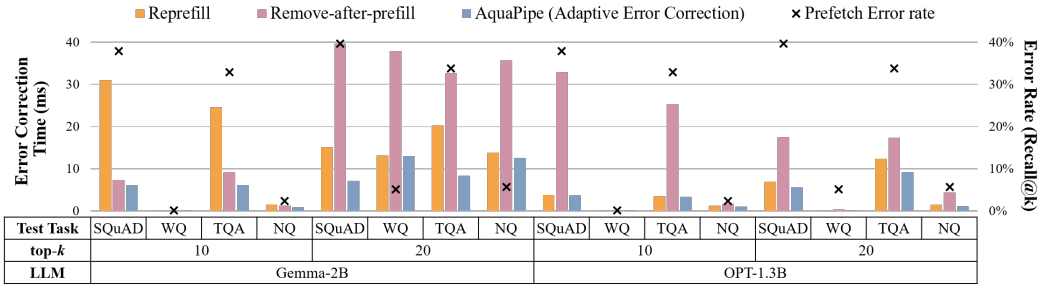


Fig. 18. The error correction time and prefetch error rate of AquaPipe, demonstrating the efficiency of its Adaptive Error Correction strategy.

Furthermore, the quality of early results is one of the key factors that may impact the benefit and penalty of launching the LLM prefill process early. As illustrated in Figure 18, an increase in the accuracy of prefetched results leads to a decrease in the cost of error correction mechanisms. Notably, even under conditions of high prefetching error rates, the elevated cost associated with Adaptive Error Correction does not significantly impair AquaPipe's overall performance. For instance, in the case of the SQuAD test set, the average error rate for prefetching reached as high as 38.75%, yet the average time required to rectify these errors amounted to only 3.20% of the TTFT, demonstrating AquaPipe's robust performance in the face of varying prefetching accuracy.

5.3.3 ***Effect of Dynamic Pipeline Granularity.*** To demonstrate the effectiveness of Dynamic Pipeline Granularity Adjustment in detail, we compared the chunk size and TTFT of AquaPipe with two static methods mentioned in §4.4. As shown in Figure 19, from fine-grained to coarse-grained pipelines, the chunk size of prefill gradually increases, while AquaPipe efficiently strikes a balance between them. Figure 20 further illustrates that, compared to the static pipeline granularity strategies, AquaPipe reduced TTFT by an average of 11.2% and 13.5%, respectively. This further indicates that AquaPipe's Dynamic Pipeline Granularity can effectively reduce the extra cost of *chunked-prefills* and the benefits of overlapping, thereby accelerating the speed of the pipeline.

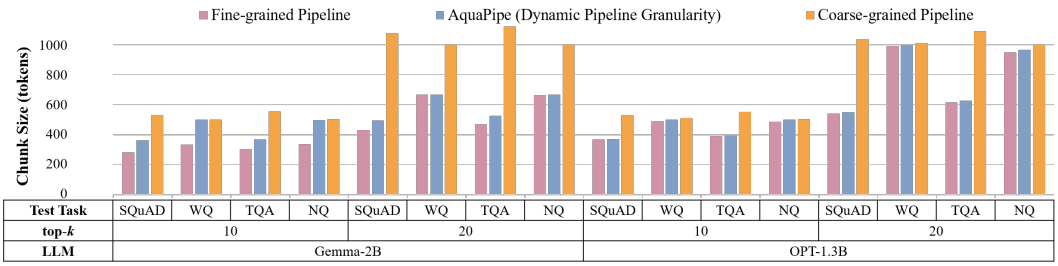


Fig. 19. The average chunk size under pipelines of different granularity strategies.

5.3.4 ***Benefit decomposition of AuqaPipe.*** To conduct a quantitative analysis of the two major sources of benefit for AquaPipe as illustrated in Figure 2, we designate the design leveraging "overlapping ANNS with Prefill" as ***AquaPipe-1*** and the design exploiting "early terminate ANNS" as ***AquaPipe-2***. It is worth noting that the counterpart to "early termination" is the traditional ANNS approach, which performs retrieval using fixed parameters, such as preset search list lengths, rendering it insensitive to the growth rate of recall for queries during retrieval execution. In Figure 21,

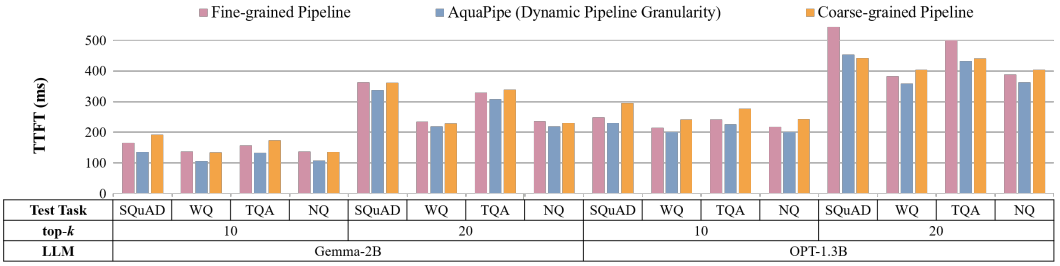


Fig. 20. The TTFT of RAG under pipelines of different granularity strategies, demonstrating the acceleration of pipeline by AquaPipe's Dynamic Pipeline Granularity.

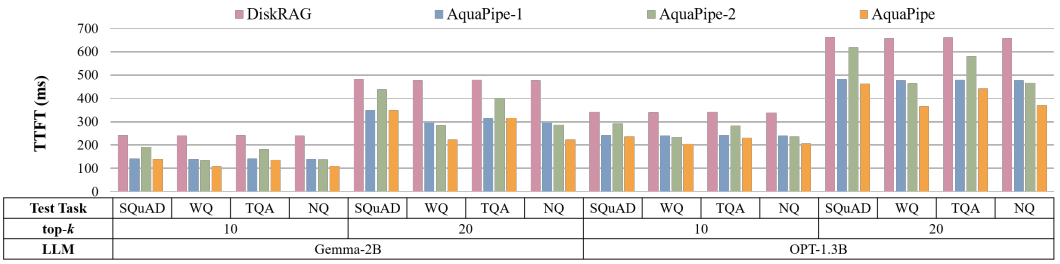


Fig. 21. The quantitative decomposition of AquaPipe's improvement, proving that the sub-benefits represented by AquaPipe-1 and AquaPipe-2 are both significant.

we further evaluated the TTFT of DiskRAG, AquaPipe-1, AquaPipe-2, and complete AquaPipe under various configurations. Specifically, compared to DiskRAG, AquaPipe-1 and AquaPipe-2 effectively reduced TTFT by 27% to 42% and 7% to 44% respectively, validating that both of these benefits are significant in contributing to overall performance optimization of AquaPipe.

5.3.5 ***Impact of model size.*** We increased the size of the LLMs to analyze the applicability of AquaPipe. To control for variables other than model size, we consistently utilize Wikipedia [38], containing approximately 20 million texts, as the knowledge base for this evaluation, thereby maintaining a relatively stable retrieval latency. At this point, expanding the size of the LLM from 1.3B to 30B will result in its prefill latency shifting from comparable to the retrieval latency to being an order of magnitude higher. As shown in Figure 22, our evaluations first demonstrated that AquaPipe exhibited positive optimization for the response time(i.e., TTFT) of the RAG system across various model sizes. Furthermore, our evaluation also indicates that when the prefill latency of LLM(e.g., Llama-30B) substantially exceeded the knowledge retrieval latency, the optimization effect of AquaPipe will weaken due to the sharp decrease in the proportion of retrieval latency in TTFT. Given that the Wikipedia knowledge base employed in the experiment contained only 20 million texts, if a larger knowledge base is utilized and results in higher retrieval latency, the above conclusions can be further generalized to larger-scale LLMs.

5.3.6 ***Impact of ANNS index cache ratio.*** To assess the impact of hybrid storage architectures on AquaPipe, we adjusted the caching ratio of ANNS index for further evaluation. Specifically, we varied the caching ratio of index data (1%, 10%, and 50%) and quantified AquaPipe's optimization effects on RAG systems. As illustrated in Figure 23, the increase in cache ratio reduced the TTFT of RAG systems by reducing the I/O latency of ANNS, although this relationship is non-linear.

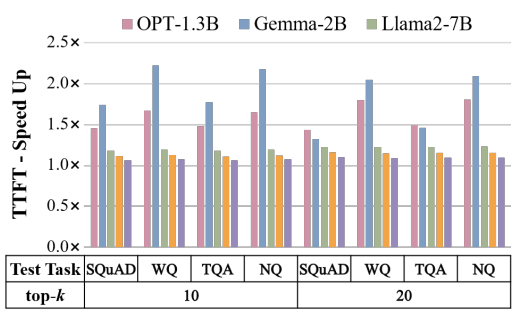


Fig. 22. The impact of different model sizes on the TTFT speed up ratio of AquaPipe(compared to DiskRAG) under fixed knowledge base capacity.

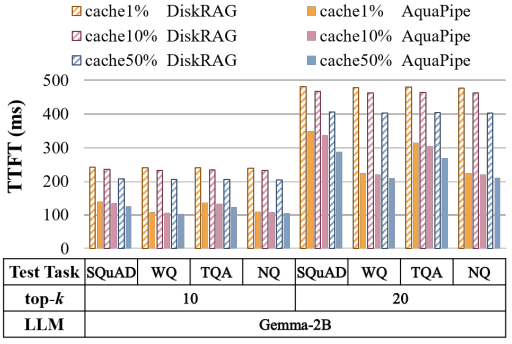


Fig. 23. The impact of ANNS index cache ratio on AuqaPipe accelerating system response time.

This non-linearity arises primarily from two factors: 1) both the latency of ANNS and prefill are significant components of TTFT, and 2) both I/O latency and computational latency are crucial components of ANNS latency. Consequently, ANNS implemented with a hybrid storage architecture exhibits limitations in optimizing overall performance of both DiskRAG and AquaPipe; among them, since AquaPipe already significantly masks most of the ANNS latency through pipelining, the overall performance of the RAG systems at lower cache rates did not exhibit notable degradation, thereby further reducing the storage cost of the RAG systems.

6 Related Work

Query-sensitive ANNS: Many recent works have noticed the insensitivity of traditional ANNS to queries and pointed out that executing searches with fixed termination conditions to all queries brings a huge retrieval overhead [14, 22, 41, 64, 73]. They developed various query-sensitive ANNS techniques to terminate the search for ANNS adaptively. AdaptiveEarlyTermination [41] collects the dynamic execution information of ANNS to determine the termination condition of ANNS through machine learning. Tao [64] employs a learning framework to predict when ANNS will end based on static features of the query. Different from all these works, AquaPipe is the first to perform joint optimization between knowledge retrieval and LLM, thus addressing the challenges of developing an efficient execution pipeline without sacrificing result accuracy. On the one hand, by carefully examining the dynamic retrieval characteristics of graph indexes, our formalized recall-aware prefetching achieves high accuracy with significantly lower costs compared with learning-based schemes [41, 64, 73]. On the other hand, in addition to adaptive early termination, our work also leverages the perception of recall to achieve the early return of intermediate ANNS results, enabling the overlapped execution of LLM prefill.

RAG Pipelines: Recently, many works have optimized the RAG systems through pipelined execution to improve performance [71]. First, Adaptive Retrieval determines the necessity of retrieval based on rules, ensuring that external information is retrieved only when essential, thereby reducing unnecessary retrieval [8, 16, 27, 32, 37, 46, 61]. Second, Iterative RAG adopts multiple iterations, alternating between retrieval and generation, to gradually refine the results [13, 51, 62, 66]. These works regard knowledge retrieval as an indivisible process, which is incapable of hiding the high retrieval latency when the dataset is out of memory. On the contrary, AquaPipe dynamically splits knowledge retrieval (ANNS) at a finer level and then overlaps it with LLM inference, allowing

these two main modules of the RAG system to be pipelined and parallelized, thereby greatly reducing the system's response time.

7 Conclusions

This paper proposes AquaPipe, a novel quality-aware pipeline for RAG systems that overlaps the execution of ANNS and LLM prefill to effectively mask the latency of disk-based knowledge retrieval, allowing for the expansion of storage capacity without affecting RAG's response time. To improve pipeline efficiency without incurring high recall loss, we design recall-aware prefetching, adaptive error correction, and dynamic granularity adjustment. We further analyze the efficient portability of the AquaPipe's core design and demonstrated that the adoption of pipeline execution is a optimization opportunity for any SOTA ANNS-based RAG systems. Extensive experimental results demonstrate that AquaPipe successfully masks latency of disk-based ANNS by 56% to 99%, thus resulting in a 1.3× to 2.6× reduction of the response time of the RAG (i.e., TTFT), while the extra recall loss caused by prefetching is limited to approximately 1%.

Acknowledgments

We would like to thank our shepherd and the anonymous reviewers for their valuable feedback. This work was supported in part by the National Key RD Program of China under Grant 2022YFB4501100; in part by the National Natural Science Foundation of China under Grant No.61821003, No.62372197, No. U22A2071, No.62102155; in part by the 111 Project (No. B07038).

References

[1] [n. d.]. Weaviate: The AI-native database for a new generation of software. https://weaviate.io/.
[2] 2023. NeurIPS'23 Competition Track: Big-ANN. https://big-ann-benchmarks.com/neurips23.html.
[3] Amey Agrawal, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav S. Gulavani, and Ramachandran Ramjee. 2023. SARATHI: Efficient LLM Inference by Piggybacking Decodes with Chunked Prefills. *ArXiv* abs/2308.16369 (2023). https://api.semanticscholar.org/CorpusID:261395577
[4] Alexandr Andoni and Piotr Indyk. 2006. Near-Optimal Hashing Algorithms for Approximate Nearest Neighbor in High Dimensions. In *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*. 459–468. doi:10.1109/FOCS.2006.49
[5] Fabien André, Anne-Marie Kermarrec, and Nicolas Le Scouarnec. 2015. Cache locality is not enough: high-performance nearest neighbor search with product quantization fast scan. *Proc. VLDB Endow.* 9, 4 (dec 2015), 288–299. doi:10.14778/2856318.2856324
[6] Akhil Arora, Sakshi Sinha, Piyush Kumar, and Arnab Bhattacharya. 2018. HD-Index: Pushing the Scalability-Accuracy Boundary for Approximate kNN Search in High-Dimensional Spaces. *Proc. VLDB Endow.* 11 (2018), 906–919. https://api.semanticscholar.org/CorpusID:5045956
[7] Daman Arora, Anush Kini, Sayak Ray Chowdhury, Nagarajan Natarajan, Gaurav Sinha, and Amit Sharma. 2023. GAR-meets-RAG Paradigm for Zero-Shot Information Retrieval. arXiv:2310.20158 [cs.CL]
[8] Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. 2023. Self-RAG: Learning to Retrieve, Generate, and Critique through Self-Reflection. *ArXiv* abs/2310.11511 (2023). https://api.semanticscholar.org/CorpusID:264288947
[9] Jon Louis Bentley. 1975. Multidimensional binary search trees used for associative searching. *Commun. ACM* 18, 9 (sep 1975), 509–517. doi:10.1145/361002.361007
[10] Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. 2013. Semantic Parsing on Freebase from Question-Answer Pairs. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, David Yarowsky, Timothy Baldwin, Anna Korhonen, Karen Livescu, and Steven Bethard (Eds.). Association for Computational Linguistics, Seattle, Washington, USA, 1533–1544. https://aclanthology.org/D13-1160
[11] Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George van den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, Diego de Las Casas, Aurelia Guy, Jacob Menick, Roman Ring, T. W. Hennigan, Saffron Huang, Lorenzo Maggiore, Chris Jones, Albin Cassirer, Andy Brock, Michela Paganini, Geoffrey Irving, Oriol Vinyals, Simon Osindero, Karen Simonyan, Jack W. Rae, Erich Elsen, and L. Sifre. 2021. Improving language models by retrieving from trillions of tokens. In *International Conference on Machine Learning*. https://api.semanticscholar.org/CorpusID:244954723

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

[13] Xin Cheng, Di Luo, Xiuying Chen, Lemao Liu, Dongyan Zhao, and Rui Yan. 2023. Lift Yourself Up: Retrieval-augmented Text Generation with Self Memory. *ArXiv* abs/2305.02437 (2023). https://api.semanticscholar.org/CorpusID:258479968

[14] P. Ciaccia and M. Patella. 2000. PAC nearest neighbor queries: Approximate and controlled search in high-dimensional and metric spaces. In *Proceedings of 16th International Conference on Data Engineering (Cat. No.00CB37073)*. 244–255. doi:10.1109/ICDE.2000.839417

[15] Zhuyun Dai and Jamie Callan. 2019. Deeper Text Understanding for IR with Contextual Neural Language Modeling. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM. doi:10.1145/3331184.3331303

[16] Hanxing Ding, Liang Pang, Zihao Wei, Huawei Shen, and Xueqi Cheng. 2024. Retrieve Only When It Needs: Adaptive Retrieval Augmentation for Hallucination Mitigation in Large Language Models. *ArXiv* abs/2402.10612 (2024). https://api.semanticscholar.org/CorpusID:267740648

[17] Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazar'e, Maria Lomeli, Lucas Hosseini, and Herv'e J'egou. 2024. The Faiss library. *ArXiv* abs/2401.08281 (2024). https://api.semanticscholar.org/CorpusID:267028372

[18] Cong Fu, Chao Xiang, Changxu Wang, and Deng Cai. 2019. Fast approximate nearest neighbor search with the navigating spreading-out graph. *Proc. VLDB Endow.* 12, 5 (jan 2019), 461–474. doi:10.14778/3303753.3303754

[19] Bin Gao, Zhuomin He, Puru Sharma, Qingxuan Kang, Djordje Jevdjic, Junbo Deng, Xingkun Yang, Zhou Yu, and Pengfei Zuo. 2024. Cost-Efficient Large Language Model Serving for Multi-turn Conversations with CachedAttention. In *2024 USENIX Annual Technical Conference (USENIX ATC 24)*. USENIX Association, Santa Clara, CA, 111–126. https://www.usenix.org/conference/atc24/presentation/gao-bin-cost

[20] Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, Shawn Presser, and Connor Leahy. 2020. The Pile: An 800GB Dataset of Diverse Text for Language Modeling. *ArXiv* abs/2101.00027 (2020). https://api.semanticscholar.org/CorpusID:230435736

[21] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Qianyu Guo, Meng Wang, and Haofen Wang. 2023. Retrieval-Augmented Generation for Large Language Models: A Survey. *ArXiv* abs/2312.10997 (2023). https://api.semanticscholar.org/CorpusID:266359151

[22] Anna Gogolou, Theophanis Tsandilas, Themis Palpanas, and Anastasia Bezerianos. 2019. Progressive Similarity Search on Time Series Data. In *EDBT/ICDT Workshops*. https://api.semanticscholar.org/CorpusID:68228902

[23] Siddharth Gollapudi, Neel Karia, Varun Sivashankar, Ravishankar Krishnaswamy, Nikit Begwani, Swapnil Raz, Yiyong Lin, Yin Zhang, Neelam Mahapatro, Premkumar Srinivasan, Amit Singh, and Harsha Vardhan Simhadri. 2023. Filtered-DiskANN: Graph Algorithms for Approximate Nearest Neighbor Search with Filters. In *Proceedings of the ACM Web Conference 2023* (Austin, TX, USA) *(WWW '23)*. Association for Computing Machinery, New York, NY, USA, 3406–3416. doi:10.1145/3543507.3583552

[24] Rentong Guo, Xiaofan Luan, Long Xiang, Xiao Yan, Xiaomeng Yi, Jigao Luo, Qianya Cheng, Weizhi Xu, Jiarui Luo, Frank Liu, Zhenshan Cao, Yanliang Qiao, Ting Wang, Bo Tang, and Charles Xie. 2022. Manu: a cloud native vector database management system. *Proc. VLDB Endow.* 15, 12 (aug 2022), 3548–3561. doi:10.14778/3554821.3554843

[25] Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Ming-Wei Chang. 2020. REALM: retrieval-augmented language model pre-training. In *Proceedings of the 37th International Conference on Machine Learning (ICML'20)*. JMLR.org, Article 368, 10 pages.

[26] Kiana Hajebi, Yasin Abbasi-Yadkori, Hossein Shahbazi, and Hong Zhang. 2011. Fast approximate nearest-neighbor search with k-nearest neighbor graph. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Volume Two* (Barcelona, Catalonia, Spain) *(IJCAI'11)*. AAAI Press, 1312–1317.

[27] Junxian He, Graham Neubig, and Taylor Berg-Kirkpatrick. 2021. Efficient Nearest Neighbor Language Models. In *Conference on Empirical Methods in Natural Language Processing*. https://api.semanticscholar.org/CorpusID:237452184

[28] Jae-Pil Heo, Zhe L. Lin, and Sung eui Yoon. 2019. Distance Encoded Product Quantization for Approximate K-Nearest Neighbor Search in High-Dimensional Space. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 41 (2019), 2084–2097. https://api.semanticscholar.org/CorpusID:206769363

[29] Cunchen Hu, Heyang Huang, Liangliang Xu, Xusheng Chen, Jiang Xu, Shuang Chen, Hao Feng, Chenxi Wang, Sa Wang, Yungang Bao, Ninghui Sun, and Yizhou Shan. 2024. Inference without Interference: Disaggregate LLM Inference for Mixed Downstream Workloads. *ArXiv* abs/2401.11181 (2024). https://api.semanticscholar.org/CorpusID:267068930

[30] Qiang Huang, Jianlin Feng, Yikai Zhang, Qiong Fang, and Wilfred Ng. 2015. Query-aware locality-sensitive hashing for approximate nearest neighbor search. *Proc. VLDB Endow.* 9, 1 (sep 2015), 1–12. doi:10.14778/2850469.2850470

[31] Gautier Izacard and Edouard Grave. 2021. Leveraging Passage Retrieval with Generative Models for Open Domain Question Answering. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, Paola Merlo, Jorg Tiedemann, and Reut Tsarfaty (Eds.). Association for Computational

Linguistics, Online, 874–880. doi:10.18653/v1/2021.eacl-main.74
[32] Zhengbao Jiang, J. Araki, Haibo Ding, and Graham Neubig. 2020. How Can We Know When Language Models Know? On the Calibration of Language Models for Question Answering. *Transactions of the Association for Computational Linguistics* 9 (2020), 962–977. https://api.semanticscholar.org/CorpusID:235078802
[33] Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2021. Billion-Scale Similarity Search with GPUs. *IEEE Transactions on Big Data* 7, 3 (2021), 535–547. doi:10.1109/TBDATA.2019.2921572
[34] Mandar Joshi, Eunsol Choi, Daniel S. Weld, and Luke Zettlemoyer. 2017. TriviaQA: A Large Scale Distantly Supervised Challenge Dataset for Reading Comprehension. arXiv:1705.03551 [cs.CL]
[35] Herve Jégou, Matthijs Douze, and Cordelia Schmid. 2011. Product Quantization for Nearest Neighbor Search. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 33, 1 (2011), 117–128. doi:10.1109/TPAMI.2010.57
[36] Yannis Kalantidis and Yannis Avrithis. 2014. Locally Optimized Product Quantization for Approximate Nearest Neighbor Search. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*. 2329–2336. doi:10.1109/CVPR.2014.298
[37] Nikhil Kandpal, H. Deng, Adam Roberts, Eric Wallace, and Colin Raffel. 2022. Large Language Models Struggle to Learn Long-Tail Knowledge. In *International Conference on Machine Learning*. https://api.semanticscholar.org/CorpusID:253522998
[38] Vladimir Karpukhin, Barlas Oğuz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen tau Yih. 2020. Dense Passage Retrieval for Open-Domain Question Answering. arXiv:2004.04906 [cs.CL]
[39] Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, Kristina Toutanova, Llion Jones, Matthew Kelcey, Ming-Wei Chang, Andrew M. Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. 2019. Natural Questions: A Benchmark for Question Answering Research. *Transactions of the Association for Computational Linguistics* 7 (2019), 452–466. doi:10.1162/tacl_a_00276
[40] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Proceedings of the 34th International Conference on Neural Information Processing Systems* (Vancouver, BC, Canada) *(NIPS '20)*. Curran Associates Inc., Red Hook, NY, USA, Article 793, 16 pages.
[41] Conglong Li, Minjia Zhang, David G. Andersen, and Yuxiong He. 2020. Improving Approximate Nearest Neighbor Search through Learned Adaptive Early Termination. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data* (Portland, OR, USA) *(SIGMOD '20)*. Association for Computing Machinery, New York, NY, USA, 2539–2554. doi:10.1145/3318464.3380600
[42] Jimmy Lin, Rodrigo Nogueira, and Andrew Yates. 2021. Pretrained Transformers for Text Ranking: BERT and Beyond. arXiv:2010.06467 [cs.IR]
[43] Yuhan Liu, Hanchen Li, Yihua Cheng, Siddhant Ray, Yuyang Huang, Qizheng Zhang, Kuntai Du, Jiayi Yao, Shan Lu, Ganesh Ananthanarayanan, Michael Maire, Henry Hoffmann, Ari Holtzman, and Junchen Jiang. 2023. CacheGen: KV Cache Compression and Streaming for Fast Language Model Serving. https://api.semanticscholar.org/CorpusID:268378938
[44] Kejing Lu, Mineichi Kudo, Chuan Xiao, and Y. Ishikawa. 2021. HVS: Hierarchical Graph Structure Based on Voronoi Diagrams for Solving Approximate Nearest Neighbor Search. *Proc. VLDB Endow.* 15 (2021), 246–258. https://api.semanticscholar.org/CorpusID:246653946
[45] Yu A. Malkov and D. A. Yashunin. 2020. Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 42, 4 (2020), 824–836. doi:10.1109/TPAMI.2018.2889473
[46] Alex Troy Mallen, Akari Asai, Victor Zhong, Rajarshi Das, Hannaneh Hajishirzi, and Daniel Khashabi. 2022. When Not to Trust Language Models: Investigating Effectiveness and Limitations of Parametric and Non-Parametric Memories. *ArXiv* abs/2212.10511 (2022). https://api.semanticscholar.org/CorpusID:260443047
[47] Yu Pan, Jianxin Sun, and Hongfeng Yu. 2023. LM-DiskANN: Low Memory Footprint in Disk-Native Dynamic Graph-Based ANN Indexing. *2023 IEEE International Conference on Big Data (BigData)* (2023), 5987–5996. https://api.semanticscholar.org/CorpusID:267105237
[48] Zhibin Pan, Liangzhuang Wang, Yang Wang, and Yuchen Liu. 2020. Product quantization with dual codebooks for approximate nearest neighbor search. *Neurocomputing* 401 (2020), 59–68. doi:10.1016/j.neucom.2020.03.016
[49] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100,000+ Questions for Machine Comprehension of Text. arXiv:1606.05250 [cs.CL]
[50] Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *Conference on Empirical Methods in Natural Language Processing*. https://api.semanticscholar.org/CorpusID:201646309
[51] Zhihong Shao, Yeyun Gong, Yelong Shen, Minlie Huang, Nan Duan, and Weizhu Chen. 2023. Enhancing Retrieval-Augmented Large Language Models with Iterative Retrieval-Generation Synergy. In *Findings of the Association for*

Computational Linguistics: EMNLP 2023, Houda Bouamor, Juan Pino, and Kalika Bali (Eds.). Association for Computational Linguistics, Singapore, 9248–9274. doi:10.18653/v1/2023.findings-emnlp.620

[52] Aditi Singh, Suhas Jayaram Subramanya, Ravishankar Krishnaswamy, and Harsha Vardhan Simhadri. 2021. FreshDiskANN: A Fast and Accurate Graph-Based ANN Index for Streaming Similarity Search. arXiv:2105.09613 [cs.IR]

[53] Suhas Jayaram Subramanya, Devvrit, Rohan Kadekodi, Ravishankar Krishaswamy, and Harsha Vardhan Simhadri. 2019. DiskANN : Fast Accurate Billion-point Nearest Neighbor Search on a Single Node. https://api.semanticscholar.org/CorpusID:209392043

[54] Kento Tatsuno, Daisuke Miyashita, Taiga Ikeda, Kiyoshi Ishiyama, Kazunari Sumiyoshi, and Jun Deguchi. 2024. AiSAQ: All-in-Storage ANNS with Product Quantization for DRAM-free Information Retrieval. arXiv:2404.06004 [cs.IR]

[55] Gemma Team. 2024. Gemma: Open Models Based on Gemini Research and Technology. *ArXiv* abs/2403.08295 (2024). https://api.semanticscholar.org/CorpusID:268379206

[56] Hugo Touvron, Louis Martin, Kevin R. Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Daniel M. Bikel, Lukas Blecher, Cristian Cantón Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony S. Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel M. Kloumann, A. V. Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, R. Subramanian, Xia Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zhengxu Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023. Llama 2: Open Foundation and Fine-Tuned Chat Models. *ArXiv* abs/2307.09288 (2023). https://api.semanticscholar.org/CorpusID:259950998

[57] Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. 2023. Interleaving Retrieval with Chain-of-Thought Reasoning for Knowledge-Intensive Multi-Step Questions. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (Eds.). Association for Computational Linguistics, Toronto, Canada, 10014–10037. doi:10.18653/v1/2023.acl-long.557

[58] Ashish Vaswani, Noam M. Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Neural Information Processing Systems*. https://api.semanticscholar.org/CorpusID:13756489

[59] Jianguo Wang, Xiaomeng Yi, Rentong Guo, Hai Jin, Peng Xu, Shengjun Li, Xiangyu Wang, Xiangzhou Guo, Chengming Li, Xiaohai Xu, Kun Yu, Yuxing Yuan, Yinghao Zou, Jiquan Long, Yudong Cai, Zhenxiang Li, Zhifeng Zhang, Yihua Mo, Jun Gu, Ruiyi Jiang, Yi Wei, and Charles Xie. 2021. Milvus: A Purpose-Built Vector Data Management System. In *Proceedings of the 2021 International Conference on Management of Data* (Virtual Event, China) *(SIGMOD '21)*. Association for Computing Machinery, New York, NY, USA, 2614–2627. doi:10.1145/3448016.3457550

[60] Mengzhao Wang, Xiaoliang Xu, Qiang Yue, and Yuxiang Wang. 2021. A comprehensive survey and experimental comparison of graph-based approximate nearest neighbor search. *Proc. VLDB Endow.* 14, 11 (jul 2021), 1964–1978. doi:10.14778/3476249.3476255

[61] Yile Wang, Peng Li, Maosong Sun, and Yang Liu. 2023. Self-Knowledge Guided Retrieval Augmentation for Large Language Models. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, Houda Bouamor, Juan Pino, and Kalika Bali (Eds.). Association for Computational Linguistics, Singapore, 10303–10315. doi:10.18653/v1/2023.findings-emnlp.691

[62] Zihao Wang, Anji Liu, Haowei Lin, Jiaqi Li, Xiaojian Ma, and Yitao Liang. 2024. RAT: Retrieval Augmented Thoughts Elicit Context-Aware Reasoning in Long-Horizon Generation. *ArXiv* abs/2403.05313 (2024). https://api.semanticscholar.org/CorpusID:268297389

[63] Yubao Wu, Ruoming Jin, and Xiang Zhang. 2014. Fast and unified local search for random walk based k-nearest-neighbor query in large graphs. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data* (Snowbird, Utah, USA) *(SIGMOD '14)*. Association for Computing Machinery, New York, NY, USA, 1139–1150. doi:10.1145/2588555.2610500

[64] Kaixiang Yang, Hongya Wang, Bo Xu, Wei Wang, Yingyuan Xiao, Ming Du, and Junfeng Zhou. 2021. Tao: A Learning Framework for Adaptive Nearest Neighbor Search using Static Features Only. *ArXiv* abs/2110.00696 (2021). https://api.semanticscholar.org/CorpusID:238259044

[65] Zhihang Yuan, Yuzhang Shang, Yang Zhou, Zhen Dong, Chenhao Xue, Bingzhe Wu, Zhikai Li, Qingyi Gu, Yong Jae Lee, Yan Yan, Beidi Chen, Guangyu Sun, and Kurt Keutzer. 2024. LLM Inference Unveiled: Survey and Roofline Model Insights. *ArXiv* abs/2402.16363 (2024). https://api.semanticscholar.org/CorpusID:268032253

[66] Fengji Zhang, B. Chen, Yue Zhang, Jin Liu, Daoguang Zan, Yi Mao, Jian-Guang Lou, and Weizhu Chen. 2023. RepoCoder: Repository-Level Code Completion Through Iterative Retrieval and Generation. In *Conference on Empirical Methods in*

Natural Language Processing. https://api.semanticscholar.org/CorpusID:257663528

[67] Minjia Zhang and Yuxiong He. 2018. Zoom: SSD-based Vector Search for Optimizing Accuracy, Latency and Memory. arXiv:1809.04067 [cs.CV]

[68] Minjia Zhang and Yuxiong He. 2019. GRIP: Multi-Store Capacity-Optimized High-Performance Nearest Neighbor Search for Vector Search Engine. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management* (Beijing, China) *(CIKM '19)*. Association for Computing Machinery, New York, NY, USA, 1673–1682. doi:10.1145/3357384.3357938

[69] Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona T. Diab, Xian Li, Xi Victoria Lin, Todor Mihaylov, Myle Ott, Sam Shleifer, Kurt Shuster, Daniel Simig, Punit Singh Koura, Anjali Sridhar, Tianlu Wang, and Luke Zettlemoyer. 2022. OPT: Open Pre-trained Transformer Language Models. *ArXiv* abs/2205.01068 (2022). https://api.semanticscholar.org/CorpusID:248496292

[70] Yue Zhang, Yafu Li, Leyang Cui, Deng Cai, Lemao Liu, Tingchen Fu, Xinting Huang, Enbo Zhao, Yu Zhang, Yulong Chen, Longyue Wang, Anh Tuan Luu, Wei Bi, Freda Shi, and Shuming Shi. 2023. Siren's Song in the AI Ocean: A Survey on Hallucination in Large Language Models. *ArXiv* abs/2309.01219 (2023). https://api.semanticscholar.org/CorpusID:261530162

[71] Penghao Zhao, Hailin Zhang, Qinhan Yu, Zhengren Wang, Yunteng Geng, Fangcheng Fu, Ling Yang, Wentao Zhang, and Bin Cui. 2024. Retrieval-Augmented Generation for AI-Generated Content: A Survey. *ArXiv* abs/2402.19473 (2024). https://api.semanticscholar.org/CorpusID:268091298

[72] Xi Zhao, Bolong Zheng, Xiaomeng Yi, Xiaofang Luan, Charles Xie, Xiaofang Zhou, and Christian S. Jensen. 2023. FARGO: Fast Maximum Inner Product Search via Global Multi-Probing. *Proc. VLDB Endow.* 16 (2023), 1100–1112. https://api.semanticscholar.org/CorpusID:257384908

[73] Bolong Zheng, Ziyang Yue, Qi Hu, Xiaomeng Yi, Xiaofang Luan, Charles Xie, Xiaofang Zhou, and Christian S. Jensen. 2023. Learned Probing Cardinality Estimation for High-Dimensional Approximate NN Search. *2023 IEEE 39th International Conference on Data Engineering (ICDE)* (2023), 3209–3221. https://api.semanticscholar.org/CorpusID:260171174

[74] Yifan Zhu, Ruiyao Ma, Baihua Zheng, Xiangyu Ke, Lu Chen, and Yunjun Gao. 2024. GTS: GPU-based Tree Index for Fast Similarity Search. *Proc. ACM Manag. Data* 2, 3, Article 142 (may 2024), 27 pages. doi:10.1145/3654945

Received July 2024; revised September 2024; accepted November 2024