Trinity: Disaggregating Vector Search for Prefill-Decode Disaggregation in LLM Serving

Yi Liu, Chen Qian
University of California Santa Cruz

Abstract

Prefill and decode (PD) disaggregation separates prompt prefill and token-by-token decode stages into distinct GPU pools and has become the dominant architecture for large-scale LLM serving in industry. Also, retrieval tasks via vector search remains entangled with the model inference process, like heterogeneous RAG requests and prompt answer caches, inflating tail latency. We are motivated to investigate how vector search should be orchestrated along with PD disaggregation with a dedicated deployment architecture without violating SLOs in various retrieval workloads. We present **Trinity**, a practical framework that consolidates all retrieval into a single, shared vector-search GPU pool and make it work with PD disaggregated LLM serving in match. **Trinity** introduces (1) a novel architecture for deploying GPU-based vector search service in PD disaggregation. (2) Continuous batching for vector search that make full used of GPUs under heterogeneous queries; (3) Stage-aware scheduling that preempts vector search requests between both decode and prefill tasks. Our analysis demonstrates that an independent vector search pool can serve diverse vector retrievals while sustaining high throughput and low tail latency for LLM serving with PD disaggregation.

1 Introduction

Retrieval-augmented generation (RAG) mitigates hallucinations and enhances output quality by retrieving semantically relevant document chunks during inference [2, 8, 9, 11, 12, 16, 20, 23]. Vector search [3, 7, 19, 22] serves as the foundation of RAG services and prompt cache (e.g., GPTCache [4]), enabling efficient nearest neighbor search from large-scale embeddings. Since exact kNN search is computationally prohibitive at scale, production systems adopt approximate nearest-neighbor (ANN) indexes, such as IVF/IMI with PQ or OPQ compression, or graph-based structures like HNSW, to trade slight recall loss for substantial gains in throughput and memory efficiency. To further accelerate large-scale vector retrieval, recent GPU-based ANN schemes [6, 8, 18, 21, 24] are exploited to batch distance computations, leverage high-bandwidth HBM, and overlap data transfer, search, and re-ranking. These optimizations enable sub-millisecond query latency even under high-throughput workloads. For example, CAGRA [18], a GPU-optimized graph ANN implementation in cuVS, adopts fixed-degree graphs and warp-efficient traversal to sustain high recall while maintaining low tail latency with batched execution.

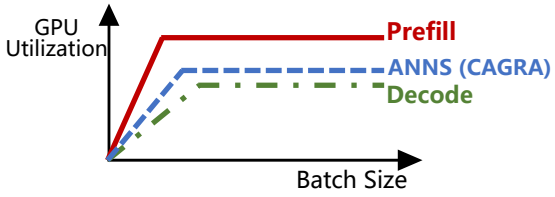
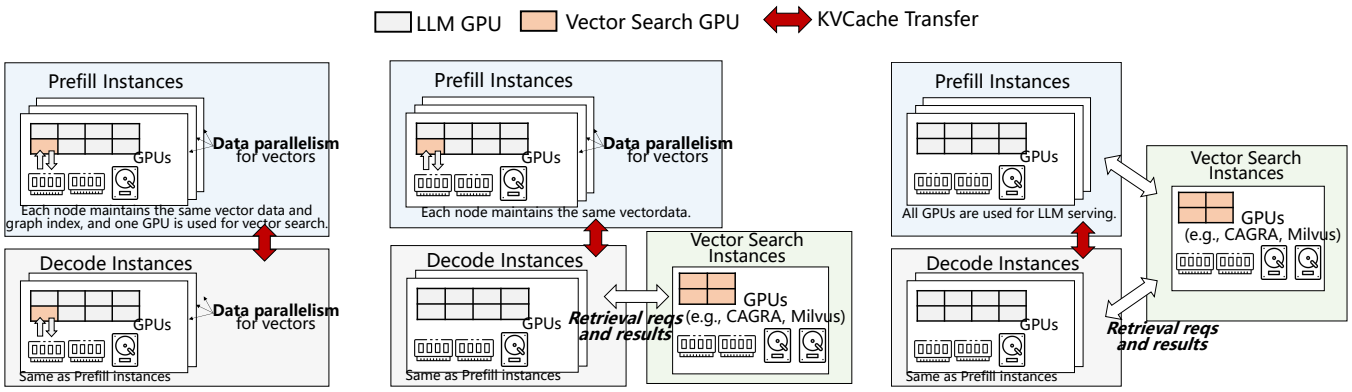


Figure 1: Roofline of Vector Search, Prefill and Decode in LLM Serving.

PD disaggregation [1, 10, 13, 14, 17, 25, 26] has emerged as a state-of-the-art paradigm for LLM serving in industry. Traditional monolithic deployments execute both stages on the same GPU, resulting in inefficient resource utilization since prefill and decode exhibit fundamentally different characteristics: the **prefill** stage is compute-intensive and benefits from high parallelism across long input sequences, while the **decode** stage is memory-bound for generating tokens sequentially with billion-scale parameters of experts [1, 5, 15] and MLP. By decoupling these stages into distinct GPU or node pools, PD disaggregation enables independent scaling, higher hardware utilization, and lower latency, and it has become central to modern LLM infrastructures with low cost.

While numerous studies [9, 10, 18, 23] have explored how to integrate vector search into existing LLM inference, the interaction between GPU-based vector retrieval and current PD disaggregation remains largely underexplored. In a disaggregated serving architecture, the prefill stage typically performs a single retrieval to initialize context at the beginning, whereas the decode stage may require multiple rounds of retrieval as generation progresses to maintain topical rele-



(a) Vector search GPU is coupled with LLM GPU in the same servers.

(b) Vector search GPU is decoupled from Decode instances, but being together with Prefill Instance.

(c) Vector search is offloaded to an independent GPU pool.

Figure 2. Architectures of serving GPU-based vector search instances in PD disaggregation.

vance or inject new knowledge. This asymmetry introduces unique challenges in coordinating retrieval frequency, managing cross-stage communication, and optimizing end-to-end latency across disaggregated resources.

In this work, we explore several GPU disaggregation architectures for incorporating vector search service into large-scale LLM serving systems and analyze how different designs coordinate these GPUs efficiently. Building on these insights, we design **Trinity**, a disaggregated serving framework with three key innovations. (1) To reduce document retrieval latency, we explore and analysis three potential architectures for running vector search either coupled with P/D GPUs or decoupled into shared GPU pools under PD disaggregation. (2) To improve GPU utilization for vector search, we introduce a continuous batching mechanism that executes graph-based vector search efficiently on GPUs with a dynamically running batch of concurrent queries. (3) To guarantee SLOs for both prefill and decode instances (e.g., time-to-first-token (TTFT)), **Trinity** employs an adaptive scheduling and preemption scheme using multiple priority queues to balance latency and throughput across serving stages.

2 Motivation

In this section, we contrast the computation and storage need for vector search, prefill, and decode, and show that each has a different optimal batch size. We observe that both decode and vector search are primarily *memory-bound*: greedy graph traversal repeatedly loads neighbor vectors and indices at each traversal step, resulting in low Arithmetic Intensity (AI). Then, we analyze the GPU utilization roofline of each stage to identify their plateaus and the batch sizes at which they saturate, as shown in Fig. 1.

Let $u(\cdot) \in [0,1]$ be GPU utilization. Let $\mathcal{P}_{\text{peak}}$ be peak compute (FLOP/s) for the chosen datatype, $\mathcal{B}_{\text{mem}}$ the effective memory bandwidth (B/s), and AI the arithmetic intensity (FLOP/Byte). Let B be batch size (sequences) for LLM. For ANN search let Q be the number of concurrent queries. Define the generic roofline as:

$$u_{\max} \triangleq \min\Big(1, \frac{\text{AI}\cdot\mathcal{B}_{\text{mem}}}{\mathcal{P}_{\text{peak}}}\Big).$$

We model the pre-plateau rise with a saturation scale X_{sat} and an optional sublinearity exponent $\alpha \in (0,1]$:

$$u(X) \approx \min\Big(u_{\max}, \big(\tfrac{X}{X_{\text{sat}}}\big)^{\alpha}\Big), \quad X \in \{B, Q\}.$$

The pre-plateau slope at small X is $\frac{du}{dX}\big|_{X \ll X_{\text{sat}}} \approx \alpha / X_{\text{sat}}$.

(1) Prefill. Large GEMMs operations means high AI, which indicates it is compute-bound.

$$u_{\text{prefill}}(B) \approx \min\Big(1, \big(\tfrac{B}{B_{\text{sat,p}}}\big)^{\alpha_p}\Big), \qquad u^{\max}_{\text{prefill}} \approx 1.$$

$B_{\text{sat,p}}$ is the smallest batch size at which the prefill phase (big GEMMs) effectively saturates the GPU. Beyond this point, increasing B yields negligible utilization gains. Thus, its plateau can reach $\approx 100\%$ once tensor cores are filled.

(2) Decode. One new token per request, multi-layer KV-Cache reads will be incurred means it is low AI (memory-bound).

$$u_{\text{decode}}(B) \approx \min\Big(u^{\max}_{\text{decode}}, \big(\tfrac{B}{B_{\text{sat,d}}}\big)^{\alpha_d}\Big), \quad u^{\max}_{\text{decode}} \approx \frac{\text{AI}_{\text{dec}}\cdot\mathcal{B}_{\text{mem}}}{\mathcal{P}_{\text{peak}}}.$$

Plateau: flat, bandwidth-limited (often well below 100%).

(3) GPU ANN (e.g., CAGRA in cuVS). This phase is dominated by graph traversal and distance computations, because per-vector bytes dominate, AI is low. Thus, it is memory-bound, even with only a small compute-leaning re-rank stage when queries are batched.

$$u_{\text{cagra}}(Q) \approx \min\left(u^{\max}_{\text{cagra}}, \left(\frac{Q}{Q_{\text{sat}}}\right)^{\alpha_a}\right), \quad u^{\max}_{\text{cagra}} \approx \frac{\text{AI}_{\text{graph}} \cdot \mathcal{B}_{\text{mem}}}{\mathcal{P}_{\text{peak}}}.$$

Plateau: bandwidth roof, which is similar order to decode at equal precision.

3 Methodology

3.1 Architecture

From the motivation, we know that vector search and the prefill/decode phases have different optimal batch sizes, suggesting they should run on separate GPUs. To reduce the retrieval latency, the most intuitive way is co-locate a vector search GPU on the P/D servers, thus, fast intra-node links (e.g., NVLink) can reduce the data transfer latency for retrieval. The trade-off is that P/D server has to reserve a GPU for vector search, which reduces available computation capacity for LLM inference and can introduce bandwidth contention. In the following, we discuss co-location by trading off SLO improvements on vector search against lost inference throughput and contention to see when co-location is worthy.

Coupled placement of vector search and LLM GPUs. Fig. 2a shows a design where each prefill and decode server co-locates one vector-search GPU with the LLM GPUs. The vector database (embeddings and graph index) is replicated per node and sharded across the local vector GPU via Data Parallelism (DP). At query time, both prefill and decode workers send retrievals to the in-node vector GPU over NVLink, minimizing retrieval latency and avoiding network hops. Also, the per-node replication of the graph/index inflates memory footprint and complicates updates, we can also make several LLM nodes to share one vector search GPU.

However, in the decode stage we typically use expert parallelism (EP): each GPU stores parameters for a subset of MLP experts and participates in dispatch/combine kernels. If one GPU on a node is reserved for vector search, an additional GPU must be provisioned elsewhere to host the displaced experts, pushing part of the EP traffic inter-node. The extra network latency during dispatch/combine can outweigh the latency saved by in-node retrieval.

Decoupled vector search from Decode but co-located with Prefill. The Fig. 2b depicts a layout where vector search GPUs are running in a separated in a independent pool only for serving retrieval requests from decode instances, but some of them are co-located with the prefill servers with DP. Prefill sends retrieval requests to the in-node vector GPUs over NVLink/PCIe, while decode reaches the vector pool over the network.

Co-locating vector search with prefill removes bandwidth contention between decode and vector search , and gives prefill the shortest retrieval path with intra-node fabric, reducing retrieval latency for prefill instances. However, even though prefill is compute-bound, tensor parallelism (TP) introduces collective synchronization that sits on the critical path. In particular, the Q/K/V projections and subsequent attention blocks require all-reduces to merge partial GEMM results across TP shards. The saved microseconds from in-node document retrieval are typically smaller than the added or unchanged collective latency for merging these GEMMs, especially at higher TP degrees. Consequently, the fast retrieval path cannot compensate for the step-time dominated by TP collectives, so this architecture gains are limited despite co-location with prefill.

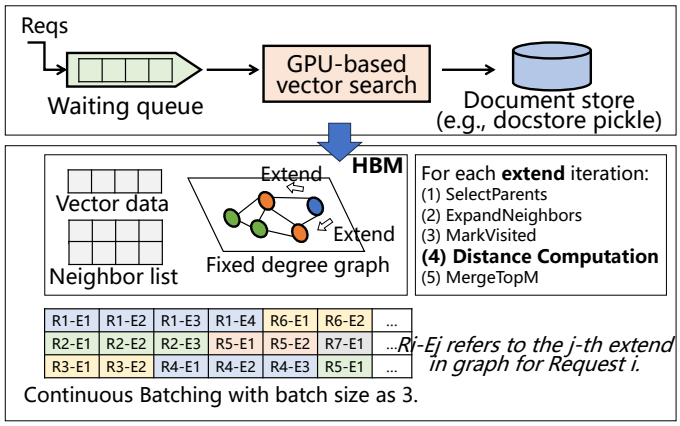


Figure 3: Continuous batching for vector search in **Trinity**.

Offloading vector search to an independent GPU pool. Based on the preceding analysis of rooflines and saturation scales, we derive the design shown in Fig. 2c, vector search is offloaded to a separate GPU pool, while all GPUs on prefill and decode servers are dedicated to LLM serving. Retrieval requests and results traverse the network between the LLM tiers and the vector-search tier.

The vector-search cluster shards embeddings and the graph index across its own GPUs (e.g., CAGRA, Milvus). Prefill and decode instances issue retrieval RPCs to this pool and receive top-k document chunks. No GPU on the LLM side is reserved for vector search, thus, also eliminating intra-node and inter-node bandwidth contention for both prefill and decode.

3.2 Continuous batching on vector search.

To improve GPU utilization and solve latency jitter that arise when CAGRA executes each request in batches, we design a continuous-batch execution that treats one *extend* step on graph as the unit and interleaves many requests precisely at that granularity. As illustrated in Fig. 3, the GPU keeps the database vectors $\mathbf{X} \in \mathbb{R}^{N \times d}$ resident in HBM together with a fixed-degree neighbor list $G \in \mathbb{N}^{N \times D}$, so distance and expansion are pure device-memory operations. Each request maintains compact device-side state: an internal `topM` (ids and distances) used for storing candidates by far, a `visited` table used to admit only first-seen candidates to computation. A scheduling loop advances all active requests through one ***extend***: for each request we select up to p non-expanded parents

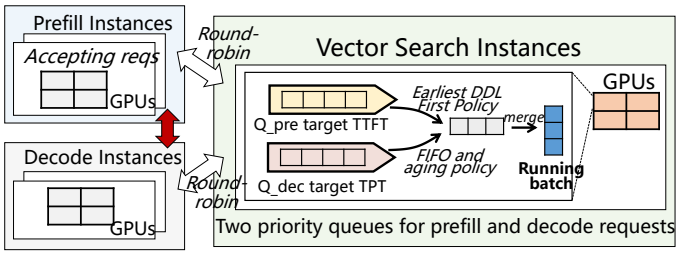


Figure 4: Priority queue-based scheduling for prefill/decode instances in **Trinity**.

from its `topM`, read D neighbors per parent from the neighbor list, filter by `visited`, and emit the surviving ids as *distance tasks* into a global, cross-request task array. We then evaluate all accumulated distance tasks with a single fixed-shape kernel that uses warp-splitting teams with high SM occupancy and cuda graph. If the available tasks are fewer than the captured kernel size, we round up with masked dummies to preserve a stable operator shape. Then, the computed $(\mathrm{id}, \mathrm{dist})$ are scattered back to their origin requests, locally merged with `topM`, and the consumed parents are marked `expanded`. Early-stop is decided independently per request when the candidates in `topM` list are not changed, so completed requests exit immediately and their device buffers are recycled, while new arrivals are admitted and begin contributing tasks to the next distance batch without idling the GPU.

Compared to the original per-request batching, continuous batching yields a distinct set of advantages in a single, compatible design. First, cross-request aggregation converts many tiny, shape-varying distance evaluations into a single fixed-shape operator that maintains near-peak HBM throughput and SM utilization even when individual requests are short, uneven, or bursty. Second, using a small number of fixed-shape launches (or a captured CUDA Graph) amortizes CPU/driver overhead and reduces variance, producing smoother P50/P95 and predictable capacity at scale. Third, the kernel's stable shape with fixed number of degree D and fixed captured batch size with safe padding simplifies memory planning. All finished queries vacate resources immediately, and newcomers join the very next distance batch with only a short, configurable flush timeout bounding additional wait. Since CAGRA maintains all vectors and the fixed-degree graph in HBM, our design keeps vector search results accuracy/recall behavior intact while turning the global distance stage into a steady, high-throughput engine that better matches GPU hardware characteristics.

3.3 Latency-aware requests scheduling for prefill/decode instances.

To meet TTFT for prefill and TPT for decode with best efforts, we add a scheduler in the vector pool that coordinates search requests from both prefill and decode stages. For prefill instances, it will make latency-critical RAG requests to retrieve prompt-relevant context upon it receives batched requests,, whereas decode emits periodic RAG probes every Δ tokens[1] to preserve perplexity and relevance during new token generation. A naïve policy that always favors vector search requests from prefill side will cause decode to wait on retrieval, it will leave token generation idling for context on decode side, thereby reducing average token throughput of ranks even if TTFT appears healthy. Conversely, always favoring decode will delay prefill stage, push TTFT beyond the latency budget, and fails to fill the KVCache transfer link (e.g., Mooncake) bandwidth between prefill and decode pools, enough to keep decode workers busy, and the PD disaggregation pipeline never saturates. Thus, we design an adaptive scheduler that works with two requests queues, and feed the running batch with vector requests from prefill and decode sides flexibly, so that latency SLOs will be met and the GPU utilization achieved by PD disaggregation will be preserved.

As shown in Fig. 4, we use two priority queues to schedule vector search requests: a high priority prefill queue Q_{pre} and a lower priority decode queue Q_{dec}. When we select request candidates from two queues into the current running batch, we build a request buffer of length N, the scheduler sets N to match the number of available slots in the current batch, reserves a fraction $r \in [r_{\min}, r_{\max}]$ for Q_{pre} so that at least rN entries come from prefill, and fills the remainder with Q_{dec}. If Q_{pre} cannot supply enough entries at that moment, its unused share is immediately given to Q_{dec}. If $n_{\text{pre}} + n_{\text{dec}} < N$, the builder pads with masked dummy tasks to preserve the kernel's fixed shape.

For Q_{pre} we apply Earliest Deadline First (EDF) policy and slack-driven scheduling approach: each request carries a deadline as $ddl = t_{\text{arr}} + L_{\text{pre,max}}$ and an estimate of remaining extends $\tilde{E}$ in vector search, and we rank by $slack = ddl - (t_{\text{now}} + \tilde{E} \cdot T_{\text{ext}})$, where T_{ext} is the measured average latency per extend. Furthermore, we assign Q_{pre} a short flush timeout τ_{pre}, ensuring urgent prefill search request can be appended into running batch without waiting for the global timeout.

For Q_{dec} we use FIFO to select request candidates. Together with the prefill reservation r and the short prefill timeout τ_{pre}, FIFO allows decode to steadily consume the remaining capacity while prefill meets TTFT and keeps the KV cache pipeline saturated.

The scheduler materializes a pair $(n_{\text{pre}}, n_{\text{dec}})$ with $n_{\text{pre}} \geq rN$ and $n_{\text{pre}} + n_{\text{dec}} = N$, launches one fixed shape distance kernel with warp teams, and scatters results back for per request `topM` merge. A lightweight control loop runs every few hundred milliseconds and adjusts r and τ_{pre} using real-time feedback to improve the KVCache link bandwidth B_{kv} toward its target $B^{\star}_{\text{kv}}$, reduce the prefill P95 wait (a proxy for TTFT), and lower the fraction of decode time stalled by RAG. When $u_{\text{kv}} < u^{\star}_{\text{kv}}$ the scheduler increases r or shortens τ_{pre} in order to accelerate prefill, while rising decode stalls decrease r so that Q_{dec}

[1]There are many of different RAG search triggering policies, here we use a fixed-tokens policy as an example.

occupies more of N.

This design intentionally makes decode as absorption point and gives prefill first class latency protection. Prefill receives earliest deadline first with step aware slack, a reserved batch share r, and a short τ_{pre}, therefore the KV cache is primed quickly and the KV link remains saturated rather than oscillating.

References

[1] Ernie 4.5 technical report. https://ernie.baidu.com/blog/publication/ernie_technical_report.pdf.

[2] Langchain: The platform for reliable agents. https://github.com/langchain-ai/langchain.

[3] ADAMS, P., LI, M., ZHANG, S., TAN, L., CHEN, Q., LI, M., LI, Z., RISVIK, K., AND SIMHADRI, H. V. Distributedann: Efficient scaling of a single diskann graph across thousands of computers. *arXiv preprint arXiv:2509.06046* (2025).

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

[5] DEEPSEEK-AI, D. G., YANG, D., ZHANG, H., SONG, J., ZHANG, R., XU, R., ET AL. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning." arxiv. *Preprint posted online on 22* (2025), 13–14.

[6] GUI, Y., YIN, P., YAN, X., ZHANG, C., ZHANG, W., AND CHENG, J. Pilotann: Memory-bounded gpu acceleration for vector search. *arXiv preprint arXiv:2503.21206* (2025).

[7] JAYARAM SUBRAMANYA, S., DEVVRIT, F., SIMHADRI, H. V., KRISHNAWAMY, R., AND KADEKODI, R. Diskann: Fast accurate billion-point nearest neighbor search on a single node. *Advances in neural information processing Systems 32* (2019).

[8] JIANG, W., LI, S., ZHU, Y., DE FINE LICHT, J., HE, Z., SHI, R., RENGGLI, C., ZHANG, S., REKATSINAS, T., HOEFLER, T., ET AL. Co-design hardware and algorithm for vector search. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (2023), pp. 1–15.

[9] JIANG, W., SUBRAMANIAN, S., GRAVES, C., ALONSO, G., YAZDANBAKHSH, A., AND DADU, V. Rago: Systematic performance optimization for retrieval-augmented generation serving. In *Proceedings of the 52nd Annual International Symposium on Computer Architecture* (2025), pp. 974–989.

[10] JIANG, W., ZELLER, M., WALEFFE, R., HOEFLER, T., AND ALONSO, G. Chameleon: a heterogeneous and disaggregated accelerator system for retrieval-augmented language models. *arXiv preprint arXiv:2310.09949* (2023).

[11] JIANG, W., ZHANG, S., HAN, B., WANG, J., WANG, B., AND KRASKA, T. Piperag: Fast retrieval-augmented generation via adaptive pipeline parallelism. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 1* (2025), pp. 589–600.

[12] JIN, C., ZHANG, Z., JIANG, X., LIU, F., LIU, S., LIU, X., AND JIN, X. Ragcache: Efficient knowledge caching for retrieval-augmented generation. *ACM Transactions on Computer Systems* (2024).

[13] JIN, Y., WANG, T., LIN, H., SONG, M., LI, P., MA, Y., SHAN, Y., YUAN, Z., LI, C., SUN, Y., ET AL. P/d-serve: Serving disaggregated large language model at scale. *arXiv preprint arXiv:2408.08147* (2024).

[14] KWON, W., LI, Z., ZHUANG, S., SHENG, Y., ZHENG, L., YU, C. H., GONZALEZ, J. E., ZHANG, H., AND STOICA, I. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles* (2023).

[15] LIU, A., FENG, B., XUE, B., WANG, B., WU, B., LU, C., ZHAO, C., DENG, C., ZHANG, C., RUAN, C., ET AL. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437* (2024).

[16] LIU, S., ZENG, Z., CHEN, L., AINIHAER, A., RAMASAMI, A., CHEN, S., XU, Y., WU, M., AND WANG, J. Tigervector: Supporting vector search in graph databases for advanced rags. In *Companion of the 2025 International Conference on Management of Data* (2025), pp. 553–565.

[17] LIU, Z., CHENG, S., TAN, G., YOU, Y., AND TAO, D. Elasticmm: Efficient multimodal llms serving with elastic multimodal parallelism. *arXiv preprint arXiv:2507.10069* (2025).

[18] OOTOMO, H., NARUSE, A., NOLET, C., WANG, R., FEHER, T., AND WANG, Y. Cagra: Highly parallel graph construction and approximate nearest neighbor search for gpus. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)* (2024), IEEE, pp. 4236–4247.

[19] PIKTUS, A., PETRONI, F., KARPUKHIN, V., OKHONKO, D., BROSCHEIT, S., IZACARD, G., LEWIS, P., OGUZ, B., GRAVE, E., YIH, W., AND RIEDEL, S. The web is your oyster - knowledge-intensive NLP against a very large web corpus. *CoRR abs/2112.09924* (2021).

[20] SHENG, Y., ZHENG, L., YUAN, B., LI, Z., RYABININ, M., CHEN, B., LIANG, P., RÉ, C., STOICA, I., AND ZHANG, C. Flexgen: High-throughput generative inference of large language models with a single gpu. In *International Conference on Machine Learning* (2023), PMLR, pp. 31094–31116.

[21] TIAN, B., LIU, H., TANG, Y., XIAO, S., DUAN, Z., LIAO, X., JIN, H., ZHANG, X., ZHU, J., AND ZHANG, Y. Towards high-throughput and low-latency billion-scale vector search via {CPU/GPU} collaborative filtering and re-ranking. In *23rd USENIX Conference on File and Storage Technologies (FAST 25)* (2025), pp. 171–185.

[22] WANG, J., YI, X., GUO, R., JIN, H., XU, P., LI, S., WANG, X., GUO, X., LI, C., XU, X., ET AL. Milvus: A purpose-built vector data management system. In *Proceedings of the 2021 international conference on management of data* (2021), pp. 2614–2627.

[23] YU, W., LIAO, N., LUO, S., AND LIU, J. Ragdoll: Efficient offloading-based online rag system on a single gpu. *arXiv preprint arXiv:2504.15302* (2025).

[24] ZHAO, W., TAN, S., AND LI, P. Song: Approximate nearest neighbor search on gpu. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)* (2020), IEEE, pp. 1033–1044.

[25] ZHENG, L., YIN, L., XIE, Z., SUN, C. L., HUANG, J., YU, C. H., CAO, S., KOZYRAKIS, C., STOICA, I., GONZALEZ, J. E., ET AL. Sglang: Efficient execution of structured language model programs. *Advances in neural information processing systems 37* (2024), 62557–62583.

[26] ZHONG, Y., LIU, S., CHEN, J., HU, J., ZHU, Y., LIU, X., JIN, X., AND ZHANG, H. {DistServe}: Disaggregating prefill and decoding for goodput-optimized large language model serving. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)* (2024), pp. 193–210.