Memory-Efficient GPU Acceleration for Recommendation Systems with Large-Scale Top-K Retrieval

Ana Luisa Gaitan
Department of Computer Science, George Mason University, Fairfax, VA, USA.
analuisagaitan694@gmu.edu

Germana Bernetti
Department of Computer Science and Engineering, University at Buffalo, Buffalo, NY, USA.
germanabernetti03@buffalo.edu

Abstract

Large-scale recommender systems increasingly depend on accelerated retrieval pipelines to convert massive candidate sets into small ranked lists under strict latency and cost constraints. The central systems challenge is not merely scoring items but performing top-K retrieval over hundreds of millions or billions of embeddings while respecting limited GPU memory, bandwidth, and energy budgets. This paper examines memory-efficient GPU acceleration for recommendation systems with large-scale top-K retrieval. It argues that retrieval performance must be understood as a socio-technical property shaped by algorithmic choices, hardware architecture, deployment topology, governance policies, and sustainability commitments. The discussion first reviews the evolution from classical recommendation models to deep and retrieval-oriented architectures, then analyzes memory bottlenecks such as embedding tables, candidate indexes, intermediate score buffers, and kernel launch overheads. It explores compression, quantization, graph-based indexing, row-wise selection, multi-stage retrieval, and heterogeneous scheduling as mutually dependent design levers. The paper then connects these technical choices to infrastructure governance, fairness, robustness, and environmental accountability. It suggests that memory-efficient top-K acceleration is not a single optimization but a negotiated balance among recall, latency, fairness, reproducibility, and operational resilience. Evaluation practices must therefore include end-to-end system metrics, tail-latency behavior, workload skew, and drift-aware monitoring. The conclusion emphasizes that future recommendation platforms require co-designed retrieval kernels, memory hierarchies, policy controls, and sustainable deployment strategies rather than isolated GPU micro-optimizations.

Keywords

recommender systems, GPU acceleration, top-K retrieval, memory efficiency, approximate nearest neighbor search, systems architecture, fairness, sustainability.

1. Introduction

Recommendation systems have become core infrastructure for digital platforms, shaping what users encounter in media, commerce, social communication, and information discovery. Early systems relied on matrix factorization, collaborative filtering, and shallow ranking models, but contemporary platforms increasingly combine deep neural networks, embedding-based retrieval, and multi-stage ranking pipelines [1], [3]. In these pipelines, the retrieval stage must

reduce an enormous item corpus to a manageable candidate set before more expensive ranking models are applied. The top-K operation is therefore not a peripheral utility; it sits at the boundary between representation learning and user-facing latency. When K is small but the candidate space is vast, the computational difficulty shifts from accurate scoring alone to memory-efficient selection under tight response-time constraints [2], [4].

This paper focuses on memory-efficient GPU acceleration for large-scale top-K retrieval in recommender systems. The central premise is that GPU acceleration cannot be understood only as parallel arithmetic. It is also a memory hierarchy problem, a scheduling problem, a distributed-systems problem, and a governance problem. Recommendation workloads often have highly skewed item popularity, long-tail distributions, dynamic user behavior, and periodic retraining cycles. These properties interact with GPU memory capacity, cache behavior, interconnect bandwidth, and kernel occupancy in ways that are not captured by simple throughput benchmarks [5], [10]. A retrieval design that excels on a static benchmark may fail in production when embedding tables grow, traffic spikes, or fairness constraints alter candidate eligibility.

The paper contributes a system-level synthesis. It reviews the structural trade-offs that arise when top-K retrieval is accelerated on GPUs, including the tension between exact and approximate search, between dense and compressed representations, and between monolithic and disaggregated deployment. It then examines governance and policy implications, because retrieval systems influence visibility, economic opportunity, and access to information. Memory efficiency is not merely an engineering virtue; it determines which models can be deployed, which populations receive low-latency service, and which environmental costs are accepted as normal. The following sections develop this argument through architectural analysis, deployment considerations, robustness and fairness concerns, sustainability questions, and evaluation practices.

2. Background and Problem Context

Modern recommendation architectures typically separate candidate generation from ranking. Candidate generation may use collaborative embeddings, content features, or graph-based signals, while ranking uses richer feature interactions and sequential models. Deep learning recommendation models have expanded the parameter space dramatically, especially through large embedding tables that map users and items to dense vectors [2], [3]. Factorization machines and related approaches showed that sparse feature interactions can be modeled efficiently, but neural extensions multiply both representational capacity and memory pressure [4]. As item catalogs grow into the millions or billions, the retrieval problem becomes a vector search problem coupled with a ranking problem.

Approximate nearest neighbor search has become a dominant strategy for large-scale retrieval because exact search over high-dimensional embeddings is often too slow or too memory intensive. Hierarchical navigable small world graphs, inverted file indexes, product quantization, and GPU-accelerated similarity search provide different trade-offs among recall, memory footprint, and query latency [5], [6]. Graph-based methods can offer high recall but may require irregular memory access patterns that are difficult to optimize on GPUs. Quantization reduces memory use but can distort distances and introduce recall loss that must be compensated elsewhere in the pipeline. Recent work on anisotropic vector quantization and learned routing illustrates that index structure and learned representation are deeply intertwined [7], [8].

The top-K selection step itself has received less attention than embedding generation or approximate search, yet it is often a bottleneck in GPU inference. After similarity scores are computed, the system must identify the largest K values across a large candidate set. Naive sorting is wasteful when K is small, and parallel selection can suffer from memory contention, synchronization overhead, and poor utilization when candidate lists are unevenly distributed. Row-wise top-K selection is especially relevant for neural acceleration because many recommendation and language models produce row-wise score matrices that must be reduced efficiently [18]. The broader lesson is that memory-efficient acceleration requires co-design across index construction, score computation, and selection kernels.

3. Memory Bottlenecks in Large-Scale Top-K Retrieval

The first memory bottleneck is the embedding table. In industrial recommenders, user and item embeddings can consume tens or hundreds of gigabytes, sometimes exceeding the capacity of a single GPU. Sharding across GPUs helps, but it introduces communication costs and load imbalance because popular items are queried far more often than long-tail items. Caching helps for hot embeddings, but cache policies must account for temporal locality, user session behavior, and fairness constraints that may require exposure of less popular items. If the retrieval stage relies on approximate indexes rather than raw embeddings, the index itself becomes a memory object with its own compression and access patterns [5], [7].

The second bottleneck is the candidate score buffer. Even if embeddings are compressed, computing similarities for a large candidate set can produce intermediate score arrays that stress high-bandwidth memory. A system may attempt to stream candidates in tiles, but tiling introduces synchronization and may reduce arithmetic intensity. The memory hierarchy matters: registers, shared memory, L2 cache, and high-bandwidth memory have different capacities and access costs. GPU kernels that perform well in isolation may become memory-bound when embedded in an end-to-end retrieval pipeline. This is particularly true when top-K selection is fused with similarity computation, because fusion can reduce global memory traffic but increase register pressure and reduce occupancy [10], [18].

The third bottleneck is metadata and control state. Recommendation pipelines maintain filters, eligibility rules, freshness constraints, diversity constraints, and business rules. These controls consume memory and introduce branching that can disrupt GPU execution. A pure vector search kernel may achieve high throughput, but a production retrieval system must often exclude items, boost categories, or enforce policy constraints before final selection. The result is a hybrid workload that combines dense computation with sparse metadata lookups. Memory-efficient acceleration therefore requires careful partitioning of work between GPU and CPU, between fast and slow storage, and between precomputed and on-the-fly structures [11], [13].

4. GPU Acceleration Strategies Under Memory Constraints

GPU acceleration for top-K retrieval typically begins with parallel similarity computation. Matrix multiplication kernels, batched dot products, and tensor-core operations can compute scores rapidly when data layout is favorable. However, memory efficiency depends on how embeddings and queries are laid out. Row-major layouts may suit one kernel while column-major layouts suit another. Quantized representations reduce bandwidth but may require dequantization on the fly. Mixed-precision arithmetic can increase throughput, but it may reduce ranking stability if not carefully validated. The choice of precision is not merely

numerical; it affects fairness when small score differences determine which items enter the candidate set [12], [14].

The next strategy is index-level compression. Product quantization, scalar quantization, and binary hashing reduce memory footprint and can accelerate distance estimation. These methods are attractive for memory-constrained GPUs, but they shift complexity to training and calibration. The index must be rebuilt when item distributions drift, and the compression scheme must preserve enough ranking fidelity for downstream models. Graph-based indexes can be more memory-efficient than exhaustive search but often have irregular traversal patterns. GPU-friendly graph layouts, cooperative traversal, and cache-aware scheduling are active areas of systems research [6], [8]. A practical system may combine multiple indexes, using a coarse quantized index for broad filtering and a dense reranking stage for precision.

Top-K selection kernels are another critical lever. Instead of sorting all scores, a kernel can maintain a small local heap or threshold per row and update it as scores stream through. This reduces memory writes and avoids full sort complexity. Row-wise selection is particularly challenging on GPUs because each row may have different score distributions and candidate counts. Recent work emphasizes ultra-fast row-wise top-K selection for neural network acceleration, suggesting that specialized kernels can materially improve end-to-end inference [18]. Yet kernel-level gains must be evaluated alongside occupancy, register pressure, and interaction with upstream similarity computation. A faster selection kernel may be neutralized if it forces excessive data movement or prevents fusion with score generation [10], [12].

5. System Architecture and Deployment Trade-Offs

Deployment architecture determines whether memory-efficient kernels translate into production benefits. A monolithic GPU server may simplify programming but limits scalability and fault isolation. A disaggregated architecture separates embedding storage, index serving, and ranking into independent services, allowing each to scale and fail independently. However, disaggregation increases network traffic and tail latency, especially when top-K retrieval requires multiple round trips. The classic tail-at-scale problem applies directly: a request that touches many shards is only as fast as its slowest shard [11]. Memory efficiency at the kernel level can be undermined by network and orchestration overhead at the system level.

Multi-stage retrieval is a common architectural response. The first stage uses a compact index to retrieve a broad candidate set with high recall but low precision. The second stage uses richer features or higher-precision vectors to rerank. The final top-K is selected after policy filters and diversity adjustments. Each stage has different memory and latency characteristics. Allocating too much memory to the first stage may starve reranking, while allocating too little may harm recall. The optimal balance depends on workload, service-level objectives, and user expectations. Systems that treat top-K retrieval as a fixed pipeline rather than an adaptive resource allocation problem often overprovision some stages and bottleneck others [13], [15].

Heterogeneous deployment is increasingly important. CPUs may handle sparse metadata, filters, and fallback paths, while GPUs handle dense similarity and selection. Edge devices may perform lightweight retrieval when connectivity is poor, and data centers may handle heavy ranking. This heterogeneity complicates reproducibility and governance because different users may receive different retrieval quality depending on device or network conditions. A memory-efficient GPU design must therefore specify not only peak throughput but also fairness across deployment contexts. If low-end devices are consistently served by

compressed, lower-recall retrieval, the system may create a two-tier recommendation experience [14], [16].

6. Robustness, Fairness, and Governance

Robustness in top-K retrieval concerns both failure modes and distributional shifts. A GPU may fail, a shard may become unavailable, or an index may become stale after a sudden item surge. The system should degrade gracefully by falling back to approximate or cached results rather than failing entirely. Yet fallback behavior can introduce hidden biases. If fallback retrieval favors popular items, then during outages the long tail becomes even less visible. Robustness therefore requires explicit policy about what happens when memory budgets are exceeded, shards are unavailable, or latency targets are missed. These are governance decisions disguised as engineering defaults [13], [14].

Fairness in recommendation retrieval is closely tied to memory and candidate generation. If the system retrieves only from a compressed index that underrepresents certain categories, no downstream ranker can recover those missing items. Memory efficiency can thus become an exclusion mechanism. Fairness-aware retrieval may require maintaining separate indexes for protected or underrepresented groups, allocating candidate quotas, or adjusting thresholds to ensure diverse exposure. These interventions consume memory and computation, creating a trade-off between efficiency and equity. The trade-off is not purely technical; it reflects organizational priorities and regulatory expectations. A platform that treats fairness as a post-processing step may find that retrieval has already eliminated the relevant candidates [14], [16].

Governance also involves observability and accountability. Retrieval systems should log not only final recommendations but also candidate sets, index versions, compression settings, and fallback events. Without such records, it is difficult to audit why a user saw a particular item or why a category was systematically absent. GPU acceleration can make observability harder because kernels operate at high speed and may aggregate data in ways that obscure intermediate decisions. Governance frameworks must therefore require inspectable pipelines, versioned indexes, and reproducible evaluation environments. Memory-efficient design should not come at the cost of explainability or contestability [13], [17].

7. Sustainability and Infrastructure Policy

The environmental cost of large-scale recommendation infrastructure is significant. Training and inference consume energy, water, and hardware resources. GPU acceleration can improve energy efficiency per query, but it can also encourage larger models and more frequent retrieval, leading to rebound effects. A memory-efficient kernel may reduce per-query energy while increasing total system energy if it enables more complex services or higher traffic. Sustainability assessment must therefore consider total cost of ownership, utilization rates, embodied carbon, and data center cooling. The choice between exact and approximate retrieval, between dense and quantized embeddings, and between centralized and edge deployment all have environmental implications [15], [17].

Policy plays a role in shaping these choices. Data localization rules, privacy regulations, and content moderation requirements may restrict where embeddings can be stored or how user data can be used. These constraints affect memory architecture because they determine whether caches can be shared across regions, whether indexes can be replicated, and whether retrieval can be offloaded to third-party infrastructure. A globally distributed recommender may need region-specific indexes and policy filters that increase memory overhead. Memory-

efficient GPU acceleration must therefore be designed with regulatory modularity in mind, allowing policy constraints to be applied without duplicating the entire model stack [16], [17].

Sustainable deployment also depends on hardware lifecycles. GPUs have limited lifetimes, and memory capacity improvements often arrive through new generations rather than incremental upgrades. Systems that hard-code assumptions about memory size may become fragile when hardware changes. Modular kernels, portable memory abstractions, and adaptive compression can extend the useful life of infrastructure. From a policy perspective, procurement decisions should consider not only peak performance but also maintainability, repairability, and energy intensity. The most memory-efficient design may be one that can run on older or lower-power accelerators without sacrificing fairness or robustness [10], [15].

8. Evaluation Considerations and Case Illustrations

Evaluating memory-efficient GPU top-K retrieval requires more than reporting queries per second. End-to-end latency distributions, tail latency, recall at K, memory bandwidth utilization, energy per query, and fairness metrics should be measured together. Benchmarks should include skewed popularity distributions, cold-start items, dynamic filters, and adversarial traffic patterns. A system that performs well on uniform random queries may fail under realistic skew. Likewise, a system that achieves high recall may still be unusable if its memory footprint prevents co-location with ranking models. Evaluation must therefore be workload-aware and deployment-aware [5], [10], [11].

Case illustrations from large-scale platforms show that retrieval is often constrained by memory capacity rather than raw compute. In video and commerce recommendation, embedding tables dominate memory, and sharding strategies determine network overhead. In social and news recommendation, freshness and policy filters require frequent index updates, which can conflict with GPU-resident indexes. In multilingual or multi-region services, duplicated indexes increase memory pressure but may be necessary for privacy and latency. These cases suggest that no single top-K kernel or index structure is universally optimal. The best design depends on the relative cost of memory, bandwidth, latency, and policy compliance [1], [2], [17].

Recent advances in row-wise top-K selection illustrate the value of specialized acceleration, but they also show the limits of isolated optimization. A faster selection kernel can reduce inference time for neural models, yet its benefit depends on data layout, batch size, and integration with similarity search [18]. If the surrounding pipeline is memory-bound, kernel improvements may yield marginal gains. If the pipeline is latency-bound, they may be transformative. This conditional value is why system-level evaluation and co-design are essential. The field needs reproducible benchmarks that combine recommendation workloads with GPU memory constraints and policy requirements [12], [18].

9. Conclusion

Memory-efficient GPU acceleration for large-scale top-K retrieval is a multidimensional systems problem. It spans embedding storage, index compression, similarity computation, selection kernels, deployment topology, governance, fairness, and sustainability. The technical core is the reduction of memory traffic while preserving ranking fidelity and latency targets, but the broader challenge is designing retrieval systems that remain robust, equitable, and accountable under real-world constraints. Approximate search and quantized representations can expand scale, yet they also introduce recall loss and distributional risk that

must be governed explicitly. Row-wise top-K selection and fused kernels can improve efficiency, but their benefits depend on architecture and workload.

Future progress will require co-design across layers. Hardware architects must provide memory hierarchies and programming models that support irregular retrieval patterns. Algorithm designers must create indexes and selection methods that are both compressible and fair. Platform engineers must build observability and fallback mechanisms that make degradation visible. Policymakers and organizational leaders must treat retrieval infrastructure as consequential public-facing systems rather than neutral utilities. If these communities collaborate, memory-efficient GPU acceleration can support recommendation systems that are faster, more sustainable, and more just. If they do not, optimization may simply reproduce existing inequities at greater scale.

References

1. Covington, P., Adams, J., & Sargin, E. (2016). Deep neural networks for YouTube recommendations. In Proceedings of the 10th ACM Conference on Recommender Systems (pp. 191-198). Association for Computing Machinery.

2. Naumov, M., Mudigere, D., Shi, H. J. M., Huang, J., Sundaraman, N., Park, J., Wang, X., Gupta, U., Wu, C. J., Azzolini, A. G., Dzhulgakov, D., Mallevich, A., Cherniavskii, I., Lu, Y., Krishnamoorthi, R., Yu, A., Kondratenko, V., Pereira, S., Chen, X., ... Smelyanskiy, M. (2019). Deep learning recommendation model for personalization and recommendation systems. arXiv preprint arXiv:1906.00091.

3. Cheng, H. T., Koc, L., Harmsen, J., Shaked, T., Chandra, T., Aradhye, H., Anderson, G., Corrado, G., Chai, W., Ispir, M., Anil, R., Haque, Z., Hong, L., Jain, V., Liu, X., & Shah, H. (2016). Wide and deep learning for recommender systems. In Proceedings of the 1st Workshop on Deep Learning for Recommender Systems (pp. 7-10). Association for Computing Machinery.

4. Rendle, S. (2010). Factorization machines. In 2010 IEEE International Conference on Data Mining (pp. 995-1000). IEEE.

5. Johnson, J., Douze, M., & Jégou, H. (2021). Billion-scale similarity search with GPUs. IEEE Transactions on Big Data, 7(3), 535-547.

6. Malkov, Y. A., & Yashunin, D. A. (2020). Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. IEEE Transactions on Pattern Analysis and Machine Intelligence, 42(4), 824-836.

7. Guo, R., Sun, P., Lindgren, E., Geng, Q., Simcha, D., Chern, F., & Kumar, S. (2020). Accelerating large-scale inference with anisotropic vector quantization. In Proceedings of the 37th International Conference on Machine Learning (pp. 3887-3896). PMLR.

8. Baranchuk, D., Persiyanov, D., Sinitsin, A., & Babenko, A. (2019). Learning to route in similarity graphs. In Proceedings of the 36th International Conference on Machine Learning (pp. 475-484). PMLR.

9. Chen, T., Li, M., Li, Y., Lin, M., Wang, N., Wang, M., Xiao, T., Xu, B., Zhang, C., & Zhang, Z. (2015). MXNet: A flexible and efficient machine learning library for heterogeneous distributed systems. arXiv preprint arXiv:1512.01274.

10. Jouppi, N. P., Young, C., Patil, N., Patterson, D., Agrawal, G., Bajwa, R., Bates, S., Bhatia, S., Boden, N., Borchers, A., Boyle, R., Cantin, P., Chao, C., Clark, C., Coriell, J.,

Daley, M., Dau, M., Dean, J., Gelb, B., ... Yoon, D. H. (2017). In-datacenter performance analysis of a tensor processing unit. In Proceedings of the 44th Annual International Symposium on Computer Architecture (pp. 1-12). Association for Computing Machinery.

11. Dean, J., & Barroso, L. A. (2013). The tail at scale. Communications of the ACM, 56(2), 74-80.

12. Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D. G., Steiner, B., Tucker, P., Vasudevan, V., Warden, P., ... Zheng, X. (2016). TensorFlow: A system for large-scale machine learning. In 12th USENIX Symposium on Operating Systems Design and Implementation (pp. 265-283). USENIX Association.

13. Sculley, D., Holt, G., Golovin, D., Davydov, E., Phillips, T., Ebner, D., Chaudhary, V., Young, M., Crespo, J. F., & Dennison, D. (2015). Hidden technical debt in machine learning systems. In Advances in Neural Information Processing Systems 28 (pp. 2503-2511). Curran Associates.

14. Mehrabi, N., Morstatter, F., Saxena, N., Lerman, K., & Galstyan, A. (2021). A survey on bias and fairness in machine learning. ACM Computing Surveys, 54(6), 1-35.

15. Strubell, E., Ganesh, A., & McCallum, A. (2019). Energy and policy considerations for deep learning in NLP. In Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (pp. 3645-3650). Association for Computational Linguistics.

16. Wu, C. J., Brooks, D., Chen, K., Chen, D., Choudhury, S., Dukhan, M., Hazelwood, K., Isaac, E., Jia, Y., Jia, B., Leyvand, T., Lu, H., Lu, Y., Qiao, L., Ranganathan, P., Shi, J., Wang, A., Wang, Y., Wei, W., ... Zhang, P. (2019). Machine learning at Facebook: Understanding inference at the edge. In 2019 IEEE International Symposium on High Performance Computer Architecture (pp. 331-344). IEEE.

17. Hazelwood, K., Bird, S., Brooks, D., Chintala, S., Diril, U., Dzhulgakov, D., Fawzy, M., Jia, B., Jia, Y., Kalro, A., Law, J., Lee, K., Lu, J., Noordhuis, P., Smelyanskiy, M., Xiong, L., & Wang, X. (2018). Applied machine learning at Facebook: A datacenter infrastructure perspective. In 2018 IEEE International Symposium on High Performance Computer Architecture (pp. 620-629). IEEE.

18. Xie, X., Luo, Y., Peng, H., & Ding, C. (2025, May). RTop-K: ultra-fast row-wise top-K selection for neural network acceleration on GPUs. In International Conference on Learning Representations (Vol. 2025, pp. 81200-81214).