Disaggregating Embedding Recommendation Systems with FlexEMR

Yibo Huang, Zhenning Yang, Jiarong Xing[†], Yi Dai[⋆], Yiming Qiu
Dingming Wu[*], Fan Lai[◇], Ang Chen

University of Michigan [†]Rice University [⋆]Fudan University
[*]Unaffiliated [◇]University of Illinois Urbana-Champaign

ABSTRACT

Efficiently serving embedding-based recommendation (EMR) models remains a significant challenge due to their increasingly large memory requirements. Today's practice splits the model across many monolithic servers, where a mix of GPUs, CPUs, and DRAM is provisioned in fixed proportions. This approach leads to suboptimal resource utilization and increased costs. Disaggregating embedding operations from neural network inference is a promising solution but raises novel networking challenges. In this paper, we discuss the design of FlexEMR for optimized EMR disaggregation. FlexEMR proposes two sets of techniques to tackle the networking challenges: Leveraging the temporal and spatial locality of embedding lookups to reduce data movement over the network and designing an optimized multi-threaded RDMA engine for concurrent lookup subrequests. We outline the design space for each technique and present initial results from our early prototype.

1 INTRODUCTION

Embedding-based Recommendation (EMR) models, widely used in e-commerce, search engines, and short video services, dominate AI inference cycles in production datacenters, such as those at Meta [12, 28]. They process user queries using both continuous and categorical features, transforming categorical features into dense vectors via embedding table lookups, and finally, combining them with the continuous features for neural network (NN) scoring.

Serving EMR models at scale leads to pressing memory requirements [32] as embedding tables can grow to terabytes in size, accounting for over 99% of model parameters [4]. In practice, we need to partition an EMR model and distribute it across multiple monolithic servers with a mix of GPUs, CPUs, and DRAM [28, 33, 45, 54]. On a specific server, existing advances propose to decouple the embedding lookup from the NN computation and use DRAM for embedding store [30]. Recent work [45] further enhances this approach by employing an embedding cache on GPUs to optimize lookup performance. However, this monolithic approach has limitations in scalability and total cost of ownership (TCO) in practice. Recommendation workloads need a mix of resources—memory for embedding store and GPUs for NN computation, and this mixture varies across models and evolves over time. Monolithic servers that provision resources in a fixed portion is hard to achieve both performance and cost efficiency. Recent studies show that it can lead to idle resources and wasted costs of up to 23.1% [24].

A promising approach to achieve performant and cost-efficient large EMR model serving is to disaggregate embedding storage and NN computation into independent servers. Specifically, using CPU-based servers to store embedding tables in memory while utilizing GPU nodes for NN computations. These components are interconnected via high-speed networks, such as remote direct memory access (RDMA) [1, 9, 40]. This decouples the memory and GPU resources and allows them to scale independently, improving the total resource efficiency and reducing the TCO. Disaggregation also increases system robustness, as failures are isolated to individual components.

However, disaggregating EMR model serving raises novel networking challenges. First, remote embedding lookup involves extensive data transmission over the network. For example, an 8-byte categorical feature index could generate a returned embedding vector with hundreds or even thousands of float values [37, 55]. Worse still, each lookup needs to query multiple such indices, and each batch contains up to thousands of lookups [18, 33]. This can be efficiently handled by local GPU memory with high memory bandwidth in a monolithic design. However, decoupling embedding storage and computation shifts this pressure to the network, with a much smaller bandwidth, potentially causing network bottlenecks. On the other hand, intensive data transmission imposes stringent performance requirements on the network layer. Unfortunately, today's RDMA systems [7, 48] are not designed for EMR disaggregation. For instance, the single-thread RDMA I/O models that are commonly used in regular applications [5, 22] will suffer from high software queuing latency for EMR serving. The recent design on disaggregated EMR systems [24] mainly focuses on resource provisioning but overlooks the above networking challenges.

In this paper, we design an optimized disaggregated EMR system called FlexEMR. FlexEMR optimizes the disaggregation by proposing two classes of techniques to tackle the challenges discussed above. The first set of techniques explores the *temporal* and *spatial* locality of embedding lookup. While existing works [45] implement embedding caches on GPUs to leverage temporal locality, we observe that such caches could compete with NN computation for limited GPU memory, and propose mechanisms to dynamically adjust caching strategy

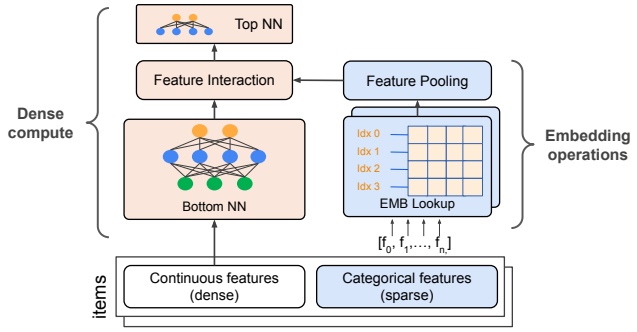

Figure 1: A representative EMR model—Deep Learning Recommendation Model (DLRM).

to avoid contention. Given that multiple lookup subrequests could point to the same embedding server, we further investigate the benefit of spatial locality. we design a hierarchical embedding pooling strategy that partially offloads pooling operations into the embedding servers, utilizing their available CPU cycles. This reduces embedding movement in the network and mitigate pressure on the rankers, while putting CPU resources to good use.

The second set of techniques aims at optimizing the RDMA I/O engines used for remote embedding lookups. First, we explore the design of a contention-free multi-threaded RDMA service, allowing concurrent RDMA threads to post lookup requests to different embedding servers in parallel. This approach significantly reduces queuing latency compared to commonly used single-threaded RDMA solutions. Additionally, we handle skewed access patterns by periodically migrating connections across embedding servers, and deploy credit-based flow control to mitigate response congestions. These optimizations collectively enhance the performance of remote embedding lookups in FlexEMR.

Working together, these two sets of techniques enable an efficient, flexible, and cost effective EMR model serving architecture. We validate the key ideas of FlexEMR using micro-benchmarks and present preliminary results in §4.

2 OVERVIEW

In this section, we provide background on EMR model serving, describe the motivation and challenges for disaggregated EMR serving, and present an overview of our solutions.

2.1 Background: EMR models

An EMR model handles two types of input features: *categorical* features (*sparse*) representing discrete categories or groups and *continuous* features (*dense*) representing measurements or quantities that are continuous in nature [33]. For example, in a video recommendation system, categorical features could include video IDs, genres, or user IDs, while continuous features could include user age or watch time. The categorical features often have very high cardinality, as each feature can consist of millions of instances (e.g., numerous specific IDs for users in feature "user IDs"). EMR models convert these high-dimensional categorical features into dense vector representations via *embedding tables*.

Figure 1 illustrates the key components and workflow of a representative EMR model. It takes candidate items as input, including both categorical and continuous features from upstream. For those accessed instances (e.g., IDs) in a batch, EMR retrieves their associated embeddings (dense vectors), which will be aggregated into a single fixed-size embedding vector through pooling operations such as sum or average. Meanwhile, the continuous features are processed by a bottom neural network (*bottom NN*) which is typically a multilayer perceptron (MLP) to generate high-dimensional dense vectors. The feature interaction process combines the dense vectors from categorical and continuous input features through operations such as element-wise multiplication or concatenation. The combined result is fed into a top neural network (*top NN*) to compute user-item scores for top-k ranking. The items with the highest scores are presented to the user.

2.2 Motivation: Disaggregated EMR serving

State-of-the-art EMR models consist of hundreds of sparse features, each associated with an embedding table with potentially millions of embedding rows [10, 34]. Indeed, production-level EMR models could have TB-level embedding tables (e.g., Meta uses 50TB DLRM model [32]).

The large-size embeddings have presented significant challenges for EMR serving because they cannot be stored on a single GPU. Therefore, EMR embeddings are often partitioned and scattered across multiple servers, each server has a combination of GPUs, CPUs, and DRAM. Considering that EMR workloads require two distinct types of resources—large memory for embedding storage and GPUs for NN computation—researchers propose decoupling them for better flexibility. Specifically, this approach leverages DRAM and CPUs for embedding storage and lookup, while utilizing GPUs on the same servers for NN computation. As a further improvement, an embedding cache is employed in GPU memory to cache the "hot" entries to optimize the lookup performance [45].

The embedding-NN decoupling enables more flexible EMR serving, but doing that on monolithic servers has several limitations. Monolithic servers provision GPU, CPU, and DRAM resources in fixed proportions, but the demands for these resources by EMR workloads can evolve across models and change over time due to varied recommendation workloads. Scaling up the whole server for the most demanding resource or the peak workload will lead to low resource utilization and waste of costs. A recent study [24] has found that fixed resource provision on monolithic servers can result in wasted costs of up to 23.1%. Therefore, more resource- and cost-efficient EMR serving are urgently needed.

Building upon the trend of disaggregation in datacenters [6, 13], a promising solution is to fully disaggregate the embedding layer and dense NN compute into network-interconnected CPU embedding servers and GPUs (rankers), respectively.

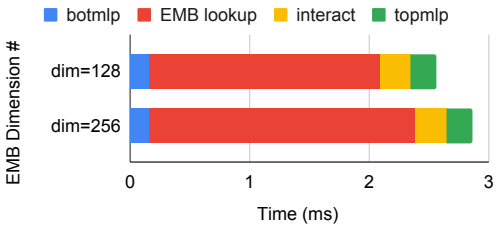

Figure 2: Embedding layer dominates EMR serving.

The rankers access embeddings stored on embedding servers over high-speed networks, such as RDMA. This new EMR serving paradigm offers multifold benefits including: (1) Flexible scalability. It allows each component to scale independently, e.g., allocating additional memory to accommodate larger embedding tables. (2) Cost efficiency. It allows rankers to multiplex many embeddings streamed from CPU embedding servers, greatly improving resource utilization and reducing the number of rankers needed for deploying EMR models. (3) Improved robustness. It enhances system robustness because the embedding operations and NN computation failures can be perfectly isolated.

EMR disaggregation is an emerging direction that remains underexplored. The most closely related work, DisaggRec [24], shares similar disaggregation concepts with us but primarily focuses on resource provisioning and scheduling post-disaggregation. However, EMR disaggregation introduces several networking challenges, as we will discuss next, that have not yet been thoroughly studied.

2.3 Key research challenges

Disaggregated EMR serving involves a large volume of data movements over the network. Typically, given some categorical feature indices as input, the ranker first fetches all corresponding embedding vectors from remote embedding servers and then performs feature pooling operations. As a result, the network bandwidth between embedding servers and rankers becomes a major bottleneck (as shown in Figure 2), presenting several domain-specific challenges.

(1) Contention in GPU memory. To reduce data movement over the network, existing works [17, 21, 23, 25, 27, 29, 46, 47] attempt to cache frequently accessed embedding entries in GPU memory. However, we observe that embedding cache is far from a perfect solution. Using precious GPU memory for caching could significantly reduce serving throughput, especially when the NN model size and request batch size are large. Essentially, NN inference also requires a large amount of GPU memory, and the existing caching strategy could cause serious resource contention between the two tasks.

(2) Large-scale fan-out pattern. Remote embedding lookup generates large-scale fan-out subrequests. For example, an 8-byte categorical feature index could generate a returned embedding vector with hundreds or even thousands of bytes in dimension size. Moreover, each lookup needs to query multiple such indices, and each batch contains up to thousands of lookups. Unlike local memory, the network bandwidth is significantly lower. Hence, issuing hundreds or thousands of concurrent batched embedding lookups can lead to severe network contention and degraded performance.

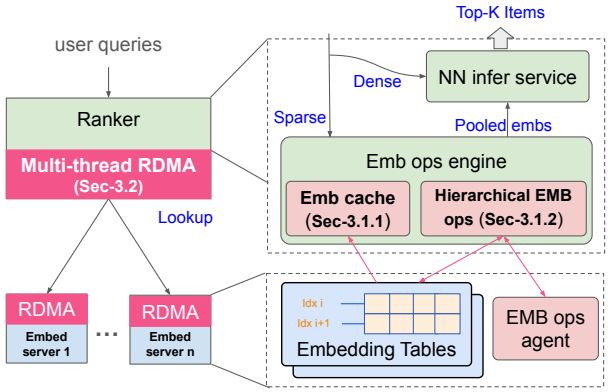

Figure 3: FlexEMR architecture overview.

(3) RDMA engine efficiency. RDMA is commonly used for remote data access. Most existing RDMA applications employ single-threaded RDMA I/O models, which send out RDMA read requests to different target machines using one thread. This leads to extended queuing latency in our scenario. We need to design a more efficient RDMA I/O engine capable of handling concurrent embedding lookup requests and results, while effectively re-balancing skewed workload patterns across distributed embedding servers.

2.4 Our solution: FlexEMR

In this paper, we propose FlexEMR—an EMR serving system that aims at addressing the aforementioned challenges. Figure 3 illustrates the envisioned system architecture. At a high level, the rankers initiate embedding lookups. Each lookup contains multiple subrequests, which firstly go through an adaptive embedding cache on rankers serving as a lookup fast path for reduced latency. The requests are then sent to embedding servers via a set of optimized RDMA engines. Once the corresponding embedding vectors are found, FlexEMR initiates a hierarchical pooling process to retrieve results without causing network contentions.

Our design is primarily based on two sets of techniques. First, we reduce embedding movement over the network by exploiting *temporal* and *spatial* locality across embedding lookups and subrequests. Temporal locality means that (i) a non-negligible portion of embeddings (e.g., 10%~15%) are the most frequently accessed in a period (i.e. *hot embeddings* [10, 11, 46]), and (ii) some subrequests often appear together in the same lookup (i.e. *embedding co-occurrence* [49]). Existing work has leveraged temporal locality to design embedding caches on GPUs, but we argue that the caching design should be dynamically adjusted to avoid GPU memory contention. Spatial locality means that multiple embedding tables/shards often co-locate in the same embedding server, so many subrequests in a lookup will be sent to the same destinations. As such, we propose to push-down lightweight pooling operations onto embedding servers. This leverages

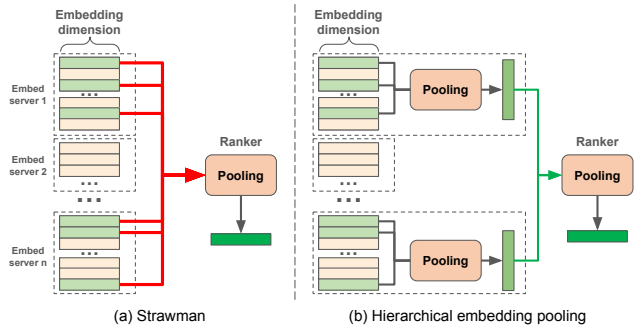

Figure 4: Hierarchical EMB pooling. Pooling computation handled solely by the ranker can cause network contention (4a). Performing pooling hierarchically, sending only the intermediate results to the ranker can reduce network traffic (4b).

the fact that embedding servers also contain CPU resources, but they are under-utilized at runtime [19].

Second, we improve the networking layer of FlexEMR serving with a fleet of RDMA optimizations. Specifically, we propose to use multi-thread RDMA for embedding lookups. It allows multiple RDMA IO threads to concurrently handle lookups into different embedding servers, thus reducing the queuing latency. We address the RNIC resource contention problem across concurrent RDMA requests using a mapping-aware RDMA IO engine, and envision a load-balanced live migration mechanism to overcome the problem of skewed requests across embedding servers. A new credit-based flow control mechanism is further implemented to avoid head-of-line blocking caused by traffic bursts.

3 FLEXEMR DESIGN

In this section, we outline a potential system design and optimizations. We first highlight the design of an adaptive caching mechanism and a hierarchical EMB pooling architecture in section 3.1. Next, in section 3.2, we discuss how multi-threaded RDMA further optimizes the embedding lookup.

3.1 Locality-enhanced disaggregation

3.1.1 Adaptive EMB caching. A common practice to reduce embedding lookup latency is to leverage the *temporal locality* across requests and cache "hot" lookup or pooling results in GPU memory [45]. However, because the embedding caches share GPU memory with NN computation, an enlarged cache inevitably leads to a smaller batch size for NN computing due to GPU memory contention, thereby degrading overall throughput. In this work, we explore an approach to adaptively adjust the size of cache: when the system is overloaded, FlexEMR reduces cache size automatically to preserve overall throughput; otherwise, it expands the cache to improve latency.

Tracing temporal dynamics. The first step towards an adaptive caching strategy is to capture the workload temporal dynamics (Figure 5). In reality, the ranker often uses a task queue to receive batches of requests from upstream, then feeds them into downstream EMR models. FlexEMR could monitor the size of these batches, then apply a sliding window algorithm to determine whether the system is under high load.

Adjusting cache size. Once a decision is made to enlarge or shrink cache size, we need to consider how to enforce these actions accordingly. This involves two sub-tasks: Firstly, we need to determine the updated cache size. Our observation here is that, given the incoming batch size and the EMR model architecture, it is possible to build a model to estimate the memory size required by NN computation. The ideal cache size is the difference between GPU memory capacity and the parts reserved for NN. The second task is to swap embeddings into or out of GPU memory For the *swap in* action, FlexEMR could initiate RDMA reads from the ranker to asynchronously fetch the hot embeddings from embedding servers in a transparent manner. For the *swap out* action, FlexEMR should remove part of embedding cache lines based on a LRU algorithm, and free up the corresponding GPU memory.

3.1.2 Hierarchical EMB pooling. Apart from temporal locality, *spatial locality* is also prevalent in EMR serving systems. In a disaggregated architecture, embedding tables are placed onto a set of remote embedding servers. Given an embedding lookup request from the ranker, a typical workflow is shown in Figure 4(a): First, the ranker sends sub-requests to remote embedding servers and asks them to return corresponding embedding vectors. The ranker then aggregates these results through *pooling* operations. This communication leads to extensive embedding movement over the network and increased latency.

Hierarchical pooling leveraging spatial locality. We seek to reduce the embedding movements between the ranker and embedding servers for higher throughput under bounded latency. Our finding here is that the CPUs in embedding servers could be utilized to perform *partial pooling* operations. If an embedding server contains multiple required vectors (i.e. spatial locality), then it could aggregate them first before sending to the ranker. Motivated by this finding, we envision a hierarchical pooling architecture, as shown in 4(b). For each embedding lookup, FlexEMR first invokes embedding server CPUs to perform partial pooling whenever possible, then asks the ranker to retrieve their outputs and perform *global pooling* to obtain the final results. Unlike existing works [57], FlexEMR is the first to explore parallel operator push-downs (i.e., pushing pooling operations down to embedding servers). This design could potentially generalize to other workloads with large-scale fan-out patterns.

Routing table for identifying co-located embeddings. An important question here is how to identify the embedding *spatial locality* among embedding servers—i.e., given as input a set of sparse feature indices, we need to identify which indices are co-located at where. A naïve solution is to maintain a routing table storing the <*feature indice, dest embedding server*> mapping pairs. It then queries all corresponding embedding

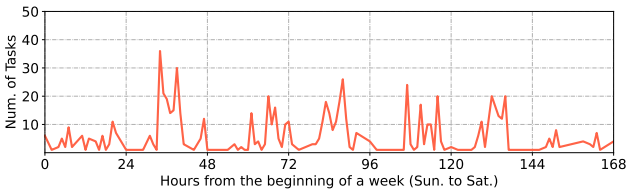

Figure 5: Distribution of inference workloads in Alibaba PAI platform over one week.

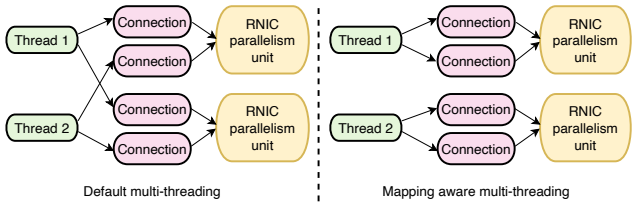

Figure 6: FlexEMR multi-threaded embedding lookup. Mapping awareness eliminates inter-thread contentions.

servers and aggregates the sparse feature indices into multiple groups depending on their embedding servers. However, this leads to huge memory footprints due to numerous sparse feature spaces. We observe a large embedding table is often partitioned into multiple shards in a row-wise manner, and each shard corresponds to an embedding range containing *start* and *end* indices. Based on this, we envision a range-based routing table in ranker where we store <*(start index, end index), dest embedding server*> pairs for each embedding shard. For a list of sparse indices, FlexEMR only needs to use the range to which each indice belongs to get the target embedding server efficiently.

3.2 EMB lookup with Multi-threaded RDMA

We next discuss how to optimize the RDMA I/O engine for remote embedding lookup. Given a batch of embedding lookup requests, each containing a large amount of fan-out subrequests, rankers in FlexEMR require an efficient RDMA I/O engine to forward subrequests to remote embedding servers. Since the completion time of an embedding lookup is dominated by the slowest subrequest, the overall pipeline is very sensitive to tail latency. The single-threaded RDMA IO model used for most existing RDMA applications [5] becomes a major bottleneck, since it incurs high queuing latency overhead between the embedding and transport layers.

A promising solution is to use *multi-threaded RDMA*, where RDMA connections to embedding servers are assigned to multiple I/O threads, and embedding subrequests are distributed to these connections according to corresponding destinations. However, naïvely using multi-threaded RDMA introduces non-negligible contentions due to limited RNIC parallelism resources (e.g., user access regions [43]): Figure 8 (left) shows that it can lead to up to 62% throughput drop in our microbenchmark.

Contention-free multi-threaded embedding lookup. To understand the root cause of contentions under concurrent lookup subrequests, we delve deep into the architecture of multi-threaded RDMA. As shown in Figure 6, we find that each RDMA engine contains a dedicated I/O thread, and each thread encompasses multiple RDMA connections. The RNIC parallelism units are allocated to each newly created connection in a round-robin manner, resulting in a one-to-many mapping between RDMA parallelism units and connections. However, the I/O threads for remote embedding lookup are not aware of such mappings, thereby enforcing multiple RDMA connections belonging to different I/O threads to access the same parallelism unit simultaneously. To coordinate different I/O threads, each parallelism unit must implement a complex locking mechanism, which could introduce significant performance overhead.

To solve this problem, we envision a mapping-aware multi-threaded RDMA engine, capable of transparently generating one-to-one mapping between I/O threads and RNIC parallelism units, as shown in Figure 6 (right). The key-enabling technique is the *resource domain* feature provided by RDMA [2, 31], which exposes the mapping between connections and RNIC parallelism units to the application layer. As such, FlexEMR could ensure that all connections assigned to the same parallelism unit are allocated to the same RDMA engine, thus preventing contention from concurrent threads. Essentially, in the cluster initialization stage, FlexEMR firstly creates RDMA connections between embedding servers and rankers, then identifies their resource domains. Since there is a static mapping between the resource domain and parallelism unit, FlexEMR could subsequently aggregate connections into different RDMA engines according to the resource domain, so that each RDMA engine points to a dedicated parallelism unit.

Live connection migratation among RDMA engines. Another common problem in practice is skewed subrequest patterns. Connections to different embedding servers might experience vastly different utilization rate, which leads to imbalanced loads among RDMA engines. Since an RDMA engine can manage multiple connections used for different embedding servers, a strawman solution is to live-migrate connections in overloaded engines to under-utilized engines: Periodically, FlexEMR traces the number of queued subrequests in each connection. When a connection becomes overloaded, FlexEMR selects the least loaded RDMA engine and initiate the migration process. However, this workflow brings back the RDMA multi-thread contention problem, because the migrated connection still points to the old parallelism unit. FlexEMR aims at a live migration strategy without RDMA contention concerns. The key idea is to *re-associate* the migrated connection with the resource domain used by the new RDMA engine. Notably, FlexEMR detaches the connection from the old domain, and then attaches it to the resource domain of the new one.

Fast credit-based flow control with RDMA QoS. FlexEMR pipelines pooling computation and remote embedding lookup

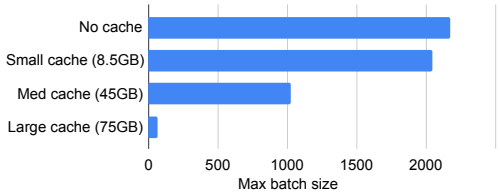

Figure 7: GPU caching significantly limits the maximum batch size for inferences due to EMB-NN contentions.

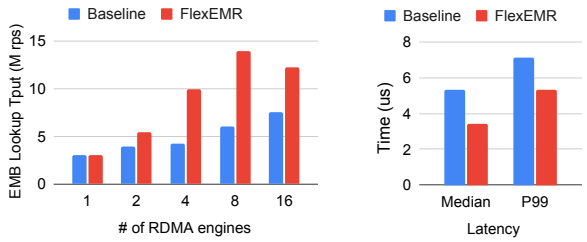

Figure 8: Performance comparison between baseline and FlexEMR: Multi-threaded EMB lookup (left) and credit flow control for lookup (right).

to further reduce serving latency. To do that, FlexEMR introduces a per-connection task queue between remote embedding servers and RDMA engines, so that the responses (i.e., embedding vectors) from embedding servers can be pushed into their respective queues asynchronously. However, without careful flow control, concurrent subrequests could result in response bursts, which might overflow corresponding task queues and drastically increase tail latency. A strawman solution is to leverage existing credit-based flow control [15, 44]—specifically, the ranker controls the size of each task queues via *credits* and proactively piggybacks the credits in lookup requests. However, since these credit messages share channels with regular lookup messages, the latter could easily introduce head-of-line blocking and delay reception of the former. As a result, embedding servers won't be able to adjust sending rate of responses in time. To mitigate such head-of-line blocking, we envision a fast credit control channel with higher priority. By leveraging the hardware feature of connection-level quality of service (QoS) offered by RDMA, FlexEMR can create a dedicated RDMA connection with a higher service level for each <*ranker, embedding server*> pair. At runtime, FlexEMR can use such connections as fast path to transfer credits timely even under high load pressure.

4 PRELIMINARY RESULTS

Even though FlexEMR is still work-in-progress, we showcase initial evidence around its major components, including adaptive embedding caching (§3.1.1) and multi-threaded RDMA embedding lookup (§3.2). We use the popular MLPerf framework [36] and a set of production-scale embedding lookup traces released by Meta [37] to synthesize inference workloads. Our testbed includes two interconnected Intel Xeon servers each equipped with 32 CPU cores, 128GB memory, and a 100Gbps Mellanox RDMA NIC. One of them is equipped with a Nvidia A100 GPU with 80GB of memory.

Naïve caching leads to GPU contentions. To understand the benefit of adaptive EMB caching, we analyze a pure GPU caching-based solution on a representative RMC2 model [10, 19]. For the GPU caching baseline, we vary the size of EMB caches and observe the changes on supported batch sizes. As Figure 7 shows, as we increase the GPU cache size, the caching-based solution has to settle with smaller batch size due to contention on GPU memory capacity, resulting in decreased inference throughput and wasted GPU compute cycles. FlexEMR on the other hand aims to achieve the highest batch size through an adaptive embedding caches, as proposed in §3.1.1, mitigating memory contention in most scenarios.

FlexEMR outperforms naïve RDMA-based embedding lookup in efficiency. Next, we compare the lookup performance of a naïve multi-threaded RDMA baseline against our FlexEMR prototype. As Figure 8 illustrates, with mapping aware multi-threading, FlexEMR achieves higher throughput than baseline by up to 2.3x. Moreover, FlexEMR achieves 35% lower latency on credits transmission, which further reduces possible congestion between rankers and embedding servers. This demonstrates the importance of an efficient multi-threaded RDMA engine (§3.2).

5 RELATED WORK

Many existing works treat EMR as generic deep learning models and adopt GPU-centric approaches for their deployment [28, 33, 45, 54], leading to under-utilized GPU resources. Recent projects apply a variety of caching mechanism[17, 21, 23, 25, 27, 29, 46, 47] to speed up embedding lookups. However, these solutions suffer from low cache hit rate in production environments [19]. Specialized hardware such as FPGAs has also been explored to enhance recommendation systems [14, 20, 53], but we strive for a generic solution with commodity hardware. Compression [3, 8, 41, 50] and sharding [38, 39, 42, 54, 56] are common optimizations to embedding table lookup. These works are complementary to ours, as the proposed techniques can be integrated seamlessly into FlexEMR for further improved performance. DisaggRec [24] proposed a similar disaggregated memory system. However, the resource distribution is fixed and determined through an exhaustive search. This approach introduces overhead and fails to capture serving dynamics.

6 CONCLUSION & FUTURE WORK

Embedding-based recommendation (EMR) model serving consumes the majority of AI inference cycles in production datacenters due to its unique embedding-dominated characteristics and stringent service-level objectives. However, prior serving systems for EMR models struggle to achieve high performance at low cost. We propose FlexEMR, a fast and efficient system that disaggregates embedding table lookups from NN computation. FlexEMR uses a set of locality-enhanced optimizations atop a multi-threaded RDMA engine to ensure performance and resource efficiency. We envision FlexEMR to improve user experience of Internet-scale recommendation

services while driving down costs for their providers. Furthermore, we believe this paradigm can benefit other ML workloads, including large language models (LLM) [26, 52], multimodal models [16], and mixture-of-expert (MoE) [35, 51], which we will also investigate in future works.

REFERENCES

[1] Wei Bai, Shanim Sainul Abdeen, Ankit Agrawal, Krishan Kumar Attre, Paramvir Bahl, Ameya Bhagat, Gowri Bhaskara, Tanya Brokhman, Lei Cao, Ahmad Cheema, et al. 2023. Empowering azure storage with {RDMA}. In *NSDI*.

[2] NVIDIA Corporation. 2024. Resource Domain. https://docs.nvidia.com/networking/display/rdmacore50/resource+domain

[3] Aditya Desai, Li Chou, and Anshumali Shrivastava. 2022. Random Offset Block Embedding (ROBE) for compressed embedding tables in deep learning recommendation systems. In *MLSys*.

[4] Aditya Desai and Anshumali Shrivastava. 2022. The trade-offs of model size in large recommendation models: 100GB to 10MB Criteo-tb DLRM model. In *NeurIPS*.

[5] Aleksandar Dragojević, Dushyanth Narayanan, Miguel Castro, and Orion Hodson. 2014. FaRM: Fast Remote Memory. In *NSDI*.

[6] Mohammad Ewais and Paul Chow. 2023. Disaggregated Memory in the Datacenter: A Survey. *IEEE Access* (2023).

[7] Yixiao Gao, Qiang Li, Lingbo Tang, Yongqing Xi, Pengcheng Zhang, Wenwen Peng, Bo Li, Yaohui Wu, Shaozong Liu, Lei Yan, et al. 2021. When cloud storage meets {RDMA}. In *NSDI*.

[8] A.A. Ginart, Maxim Naumov, Dheevatsa Mudigere, Jiyan Yang, and James Zou. 2021. Mixed Dimension Embeddings with Application to Memory-Efficient Recommendation Systems. In *ISIT*.

[9] Chuanxiong Guo, Haitao Wu, Zhong Deng, Gaurav Soni, Jianxi Ye, Jitu Padhye, and Marina Lipshteyn. 2016. RDMA over commodity ethernet at scale. In *SIGCOMM*.

[10] Udit Gupta, Samuel Hsia, Vikram Saraph, Xiaodong Wang, Brandon Reagen, Gu-Yeon Wei, Hsien-Hsin S. Lee, David Brooks, and Carole-Jean Wu. 2020. DeepRecSys: A System for Optimizing End-To-End At-Scale Neural Recommendation Inference. In *ISCA*.

[11] Udit Gupta, Samuel Hsia, Vikram Saraph, Xiaodong Wang, Brandon Reagen, Gu-Yeon Wei, Hsien-Hsin S. Lee, David Brooks, and Carole-Jean Wu. 2020. DeepRecSys: A System for Optimizing End-To-End At-Scale Neural Recommendation Inference. In *ISCA*.

[12] Udit Gupta, Carole-Jean Wu, Xiaodong Wang, Maxim Naumov, Brandon Reagen, David Brooks, Bradford Cottel, Kim Hazelwood, Mark Hempstead, Bill Jia, et al. 2020. The architectural implications of facebook's dnn-based personalized recommendation. In *HPCA*.

[13] Bowen He, Xiao Zheng, Yuan Chen, Weinan Li, Yajin Zhou, Xin Long, Pengcheng Zhang, Xiaowei Lu, Linquan Jiang, Qiang Liu, Dennis Cai, and Xiantao Zhang. 2023. DxPU: Large-scale Disaggregated GPU Pools in the Datacenter. *ACM Trans. Archit. Code Optim.* (2023).

[14] Samuel Hsia, Udit Gupta, Bilge Acun, Newsha Ardalani, Pan Zhong, Gu-Yeon Wei, David Brooks, and Carole-Jean Wu. 2023. MP-Rec: Hardware-Software Co-design to Enable Multi-path Recommendation. In *ASPLOS*.

[15] Shuihai Hu, Wei Bai, Gaoxiong Zeng, Zilong Wang, Baochen Qiao, Kai Chen, Kun Tan, and Yi Wang. 2020. Aeolus: A Building Block for Proactive Transport in Datacenters. In *SIGCOMM*.

[16] Jun Huang, Zhen Zhang, Shuai Zheng, Feng Qin, and Yida Wang. 2024. DISTMM: Accelerating Distributed Multimodal Model Training. In *NSDI*.

[17] Mohamed Assem Ibrahim, Onur Kayiran, and Shaizeen Aga. 2021. Efficient Cache Utilization via Model-aware Data Placement for Recommendation Models. In *MEMSYS*.

[18] Rishabh Jain, Scott Cheng, Vishwas Kalagi, Vrushabh Sanghavi, Samvit Kaul, Meena Arunachalam, Kiwan Maeng, Adwait Jog, Anand Sivasubramaniam, Mahmut Taylan Kandemir, et al. 2023. Optimizing cpu performance for recommendation systems at-scale. In *ISCA*.

[19] Rishabh Jain, Scott Cheng, Vishwas Kalagi, Vrushabh Sanghavi, Samvit Kaul, Meena Arunachalam, Kiwan Maeng, Adwait Jog, Anand Sivasubramaniam, Mahmut Taylan Kandemir, and Chita R. Das. 2023. Optimizing CPU Performance for Recommendation Systems At-Scale. In *ISCA*.

[20] Wenqi Jiang, Zhenhao He, Shuai Zhang, Kai Zeng, Liang Feng, Jiansong Zhang, Tongxuan Liu, Yong Li, Jingren Zhou, Ce Zhang, and Gustavo Alonso. 2021. FleetRec: Large-Scale Recommendation Inference on Hybrid GPU-FPGA Clusters. In *KDD*.

[21] Hongju Kal, Seokmin Lee, Gun Ko, and Won Woo Ro. 2021. SPACE: Locality-Aware Processing in Heterogeneous Memory for Personalized Recommendations. In *ISCA*.

[22] Anuj Kalia, Michael Kaminsky, and David Andersen. 2019. Datacenter RPCs can be General and Fast. In *NSDI*.

[23] Liu Ke, Udit Gupta, Benjamin Youngjae Cho, David Brooks, Vikas Chandra, Utku Diril, Amin Firoozshahian, Kim Hazelwood, Bill Jia, Hsien-Hsin S Lee, et al. 2020. Recnmp: Accelerating personalized recommendation with near-memory processing. In *ISCA*.

[24] Liu Ke, Xuan Zhang, Benjamin Lee, G. Edward Suh, and Hsien-Hsin S. Lee. 2022. DisaggRec: Architecting Disaggregated Systems for Large-Scale Personalized Recommendation.

[25] Daniar H. Kurniawan, Ruipu Wang, Kahfi S. Zulkifli, Fandi A. Wiranata, John Bent, Ymir Vigfusson, and Haryadi S. Gunawi. 2023. EVStore: Storage and Caching Capabilities for Scaling Embedding Tables in Deep Recommendation Systems. In *ASPLOS*.

[26] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient Memory Management for Large Language Model Serving with PagedAttention. In *SOSP*.

[27] Youngeun Kwon, Yunjae Lee, and Minsoo Rhu. 2019. Tensordimm: A practical near-memory processing architecture for embeddings and tensor operations in deep learning. In *MICRO*.

[28] Fan Lai, Wei Zhang, Rui Liu, William Tsai, Xiaohan Wei, Yuxi Hu, Sabin Devkota, Jianyu Huang, Jongsoo Park, Xing Liu, Zeliang Chen, Ellie Wen, Paul Rivera, Jie You, Chun cheng Jason Chen, and Mosharaf Chowdhury. 2023. AdaEmbed: Adaptive Embedding for Large-Scale Recommendation Models. In *OSDI*.

[29] Yejin Lee, Seong Hoon Seo, Hyunji Choi, Hyoung Uk Sul, Soosung Kim, Jae W. Lee, and Tae Jun Ham. 2021. MERCI: efficient embedding reduction on commodity hardware via sub-query memoization. In *ASPLOS*.

[30] Michael Lui, Yavuz Yetim, Özgür Özkan, Zhuoran Zhao, Shin-Yeh Tsai, Carole-Jean Wu, and Mark Hempstead. 2021. Understanding capacity-driven scale-out neural recommendation inference. In *ISPASS*.

[31] Mellanox. 2017. Verbs: Introduce resource domain. https://patchwork.kernel.org/project/linux-rdma/patch/1505648922-21346-1-git-send-email-yishaih@mellanox.com/

[32] Dheevatsa Mudigere, Yuchen Hao, Jianyu Huang, Zhihao Jia, Andrew Tulloch, Srinivas Sridharan, Xing Liu, Mustafa Ozdal, Jade Nie, Jongsoo Park, et al. 2022. Software-hardware co-design for fast and scalable training of deep learning recommendation models. In *ISCA*.

[33] Maxim Naumov, Dheevatsa Mudigere, Hao-Jun Michael Shi, Jianyu Huang, Narayanan Sundaraman, Jongsoo Park, Xiaodong Wang, Udit Gupta, Carole-Jean Wu, Alisson G Azzolini, et al. 2019. Deep learning recommendation model for personalization and recommendation systems. *arXiv preprint arXiv:1906.00091* (2019).

[34] Zaifeng Pan, Zhen Zheng, Feng Zhang, Ruofan Wu, Hao Liang, Dalin Wang, Xiafei Qiu, Junjie Bai, Wei Lin, and Xiaoyong Du. 2024. RECom: A Compiler Approach to Accelerating Recommendation Model Inference with Massive Embedding Columns. In *ASPLOS*.

[35] Samyam Rajbhandari, Conglong Li, Zhewei Yao, Minjia Zhang, Reza Yazdani Aminabadi, Ammar Ahmad Awan, Jeff Rasley, and Yuxiong He. 2022. DeepSpeed-MoE: Advancing Mixture-of-Experts Inference and Training to Power Next-Generation AI Scale. In *ICML*.

[36] Vijay Janapa Reddi, Christine Cheng, David Kanter, Peter Mattson, Guenther Schmuelling, Carole-Jean Wu, Brian Anderson, Maximilien

Breughe, Mark Charlebois, William Chou, Ramesh Chukka, Cody Coleman, Sam Davis, Pan Deng, Greg Diamos, Jared Duke, Dave Fick, J. Scott Gardner, Itay Hubara, Sachin Idgunji, Thomas B. Jablin, Jeff Jiao, Tom St. John, Pankaj Kanwar, David Lee, Jeffery Liao, Anton Lokhmotov, Francisco Massa, Peng Meng, Paulius Micikevicius, Colin Osborne, Gennady Pekhimenko, Arun Tejusve Raghunath Rajan, Dilip Sequeira, Ashish Sirasao, Fei Sun, Hanlin Tang, Michael Thomson, Frank Wei, Ephrem Wu, Lingjie Xu, Koichi Yamada, Bing Yu, George Yuan, Aaron Zhong, Peizhao Zhang, and Yuchen Zhou. 2020. MLPerf Inference Benchmark.

[37] Meta Research. 2023. GitHub - Set of datasets for the deep learning recommendation model (DLRM). https://github.com/facebookresearch/dlrm_datasets

[38] Geet Sethi, Bilge Acun, Niket Agarwal, Christos Kozyrakis, Caroline Trippel, and Carole-Jean Wu. 2022. RecShard: statistical feature-based memory optimization for industry-scale neural recommendation. In *ASPLOS*.

[39] Geet Sethi, Pallab Bhattacharya, Dhruv Choudhary, Carole-Jean Wu, and Christos Kozyrakis. 2023. FlexShard: Flexible Sharding for Industry-Scale Sequence Recommendation Models. *arXiv preprint arXiv:2301.02959* (2023).

[40] Yizhou Shan, Yutong Huang, Yilun Chen, and Yiying Zhang. 2018. Legoos: A disseminated, distributed {OS} for hardware resource disaggregation. In *OSDI*.

[41] Hao-Jun Michael Shi, Dheevatsa Mudigere, Maxim Naumov, and Jiyan Yang. 2020. Compositional Embeddings Using Complementary Partitions for Memory-Efficient Recommendation Systems. In *KDD*.

[42] Hao-Jun Michael Shi, Dheevatsa Mudigere, Maxim Naumov, and Jiyan Yang. 2020. Compositional embeddings using complementary partitions for memory-efficient recommendation systems. In *KDD*.

[43] Mellanox Technologies. 2016. Mellanox Adapters Programmer's Reference Manual (PRM). https://network.nvidia.com/files/doc-2020/ethernet-adapters-programming-manual.pdf

[44] Zhe Wang, Teng Ma, Linghe Kong, Zhenzao Wen, Jingxuan Li, Zhuo Song, Yang Lu, Guihai Chen, and Wei Cao. 2022. Zero Overhead Monitoring for Cloud-native Infrastructure using RDMA. In *USENIX ATC*.

[45] Yingcan Wei, Matthias Langer, Fan Yu, Minseok Lee, Jie Liu, Ji Shi, and Zehuan Wang. 2022. A GPU-specialized inference parameter server for large-scale deep recommendation models. In *RecSys*.

[46] Mark Wilkening, Udit Gupta, Samuel Hsia, Caroline Trippel, Carole-Jean Wu, David Brooks, and Gu-Yeon Wei. 2021. RecSSD: near data processing for solid state drive based recommendation inference. In *ASPLOS*.

[47] Minhui Xie, Youyou Lu, Jiazhen Lin, Qing Wang, Jian Gao, Kai Ren, and Jiwu Shu. 2022. Fleche: an efficient GPU embedding cache for personalized recommendations. In *EuroSys*.

[48] Jilong Xue, Youshan Miao, Cheng Chen, Ming Wu, Lintao Zhang, and Lidong Zhou. 2019. Fast distributed deep learning over rdma. In *EuroSys*.

[49] Haojie Ye, Sanketh Vedula, Yuhan Chen, Yichen Yang, Alex Bronstein, Ronald Dreslinski, Trevor Mudge, and Nishil Talati. 2023. GRACE: A Scalable Graph-Based Approach to Accelerating Recommendation Model Inference. In *ASPLOS*.

[50] Chunxing Yin, Bilge Acun, Carole-Jean Wu, and Xing Liu. 2021. TT-Rec: Tensor Train Compression for Deep Learning Recommendation Models. In *MLSys*.

[51] Dianhai Yu, Liang Shen, Hongxiang Hao, Weibao Gong, Huachao Wu, Jiang Bian, Lirong Dai, and Haoyi Xiong. 2024. MoESys: A Distributed and Efficient Mixture-of-Experts Training and Inference System for Internet Services. *IEEE Transactions on Services Computing* (2024).

[52] Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. 2022. Orca: A Distributed Serving System for Transformer-Based Generative Models. In *OSDI*.

[53] Chaoliang Zeng, Layong Luo, Qingsong Ning, Yaodong Han, Yuhang Jiang, Ding Tang, Zilong Wang, Kai Chen, and Chuanxiong Guo. 2022. {FAERY}: An {FPGA-accelerated} Embedding-based Retrieval System. In *OSDI*.

[54] Daochen Zha, Louis Feng, Bhargav Bhushanam, Dhruv Choudhary, Jade Nie, Yuandong Tian, Jay Chae, Yinbin Ma, Arun Kejariwal, and Xia Hu. 2022. Autoshard: Automated embedding table sharding for recommender systems. In *KDD*.

[55] Daochen Zha, Louis Feng, Qiaoyu Tan, Zirui Liu, Kwei-Herng Lai, Bhargav Bhushanam, Yuandong Tian, Arun Kejariwal, and Xia Hu. 2022. Dreamshard: Generalizable embedding table placement for recommender systems. *NeurIPS* (2022).

[56] Daochen Zha, Louis Feng, Qiaoyu Tan, Zirui Liu, Kwei-Herng Lai, Bhargav Bhushanam, Yuandong Tian, Arun Kejariwal, and Xia Hu. 2022. DreamShard: Generalizable Embedding Table Placement for Recommender Systems. In *NeurIPS*.

[57] Qizhen Zhang, Xinyi Chen, Sidharth Sankhe, Zhilei Zheng, Ke Zhong, Sebastian Angel, Ang Chen, Vincent Liu, and Boon Thau Loo. 2022. Optimizing data-intensive systems in disaggregated data centers with teleport. In *SIGMOD*.