Duet: A Collaborative User Driven Recommendation System for Edge Devices

Vidushi Goyal
University of Michigan, Ann Arbor
vidushi@umich.edu

Valeria Bertacco
University of Michigan, Ann Arbor
valeria@umich.edu

Reetuparna Das
University of Michigan, Ann Arbor
reetudas@umich.edu

Abstract - Recommendation systems are the backbone for numerous user applications on edge devices. However, the compute and memory-intensive nature of recommendation models renders them unsuitable for edge devices. Nevertheless, by decoupling the model fraction related to user history (e.g., past visited pages, liked posts) and user attributes (such as age, gender), we can offload partial recommendation models onto local edge devices. Hence, we present Duet, a novel collaborative edge-cloud recommendation system that intelligently decomposes the recommendation model into two smaller models – user and item models – that execute simultaneously on the edge device and cloud before coming together to deliver final recommendations. Further, we propose a lightweight Duet architecture to support user models on resource-constrained edge devices. Duet reduces the average latency by 6.4× and improves energy efficiency by 4.6× across five recommendation models.

1 Introduction

Recommendation systems are an integral part of popular mobile applications, including social media (e.g., Instagram), online shopping (e.g., Amazon), movie recommendation (e.g., Netflix), fitness (e.g., Fitbit), and e-learning (e.g., Coursera). Netflix attributes 80% of its streaming hours [1] to recommendation system and 35% of Amazon purchases [2] result from the recommendation algorithm. Recommendation systems are the top consumer of AI compute cycles in production-scale datacenters [3]. Until now, recommendation systems have been considered to be datacenter-bound because of their large memory footprint. A recommendation *query* consists of several inferences (100s) to rank candidates, where each candidate inference computes DNN-based MLP layers and looks up multiple embedding tables. We make two important observations in this work. First, we observe that a significant portion of a candidate inference is specific to user-data, which is *common* across all inferences in a query, as shown in Figure 1. This common computation can be performed once per query and reused across all candidate inferences. Second, we find an opportunity to perform the user-specific inference on edge devices, which have immediate access to user data and can efficiently process it locally. This trait is in contrast with the complexities of managing up-to-date user states for millions of users at datacenters, which is impractical at scale.

Deriving from these insights, we present *Duet*, which decouples the monolithic recommendation model into two smaller concurrent models, the user model and the item model, distributed on edge and cloud, respectively. The user model operates on the user data at the local edge device and transfers its output to the datacenter. Simultaneously, the item model computes item features for all candidates to be ranked. It then combines item model outputs with the user model output to provide final recommendations. More importantly, unlike the item model that computes item features for all the candidates to be ranked, the user model is computed only once for a recommendation query. Thus, we reuse the user-specific computation across all candidates being ranked for a user query.

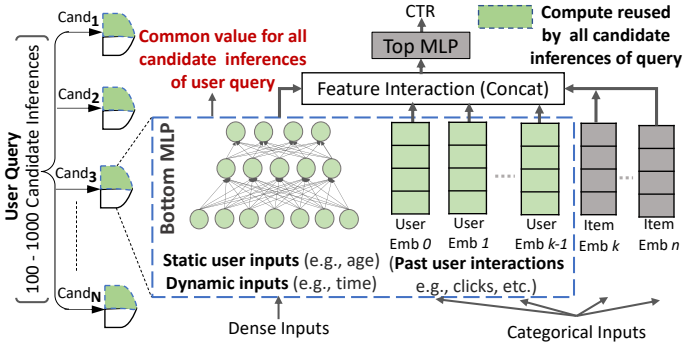

Figure 1. Recommendation model, comprising bottom MLP, embeddings, and top MLP, computes 100s of inferences to rank all candidates of a query. Common computation pertaining to user-specific information can be reused across all inferences.

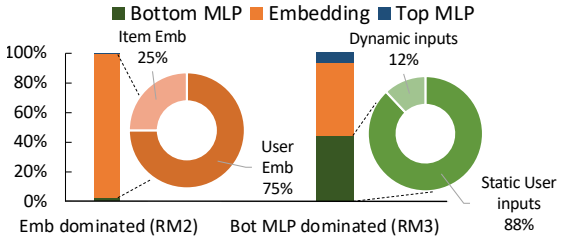

Figure 2. Detailed Timing Breakdown in three components and a further breakdown into user and item compute for a query with 1024 candidate inferences for two recommendation models (RM).

Further, as the decomposed user model is computed on a small resource-constrained edge device, we present a lightweight Duet architecture to overcome its energy limitations. We propose a hardware unit with a small user-specific scratchpad to capture up-to-date user states in hardware. While feasible on individual user edge devices, having dedicated scratchpads for millions of users at datacenter scale is unfeasible. Further, we extend support for memoization and heterogeneous lower precision (qint8) format for MLPs on the edge. These hardware optimizations reduce memory accesses, computational intensity, and data-transfer overhead from the edge to the cloud. We evaluate *Duet* across five recommendation models and boost the performance by 6.4× and energy efficiency by 4.6× with the accuracy impact limited to <= 0.06%.

2 Background and Motivation

A recommendation query infers hundreds to thousands of candidates (termed as query size) utilizing a heavy DNN-based ranking model to recommend only the top candidates with the highest Click Through Rates (CTRs) to the user, which makes this task very compute/memory intensive. These recommendation ranking models are built using embedding tables and MLP layers that comprise three different components: bottom MLP stack to process continuous inputs, embedding tables to process categorical sparse inputs,

DAC '24, June 23–27, 2024, San Francisco, CA, USA

ACM ISBN 979-8-4007-0601-1/24/06…$15.00
https://doi.org/10.1145/3649329.3656225

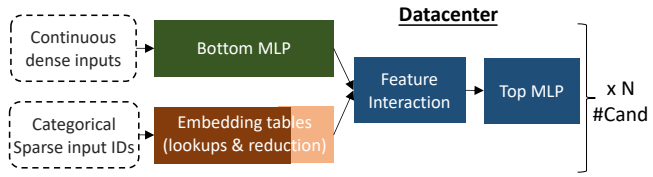

Figure 3. Data flow for the state-of-the-art recommendation system at the datacenter computing N inferences for N candidates.

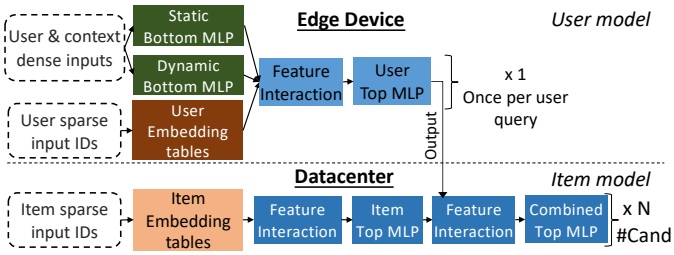

Figure 4. Data flow for our collaborative edge-cloud recommendation system. The monolithic model is decoupled into two concurrent models: User model at edge device and item model at datacenter. The edge model is computed once per query. This computation is reused by N candidate inferences performed by the small datacenter model.

and top MLP stack to predict the CTR. In Figure 2, we illustrate the time consumed by the three components for two model definitions from the prior works [3, 4]. We observe 75% of the embedding time is consumed by user embeddings, while item embeddings consume the remaining 25%. Similarly, as much as 88% of the bottom MLP inputs correspond to static user information. Our findings yield two discoveries. *First, we identify an opportunity to compute a common value per query, pertaining to user-information, which can be reused by all candidate inferences of a query.* Memory accesses to user embedding tables, corresponding to user history, are common for all inferences performed to rank multiple candidates of a user query. Further, the stable user profile information is common to all candidates of a user query. Thus, user embeddings and static user profile inputs can be computed once per query and shared by all the candidate inferences. *Second, we discovered that user-specific information can be processed separately on the user edge device.* We can decouple the user state, which includes user-specific embeddings and user profile information, from the entire model. This user state processing can be offloaded to the local edge device, which has immediate access to user data, instead of maintaining individual user states for all users at datacenters, a task that demands dedicated hardware resources and complex software management at scale.

3 Collaborative Edge-Cloud System

The current recommendation model's data flow, depicted in Figure 3, encompasses three components: bottom MLP processing continuous dense inputs, embedding tables handling sparse input IDs, and feature interaction blocks concatenating the output vectors from embedding tables and bottom MLP to yield a final feature vector, which then undergoes processing by top MLP layers to predict CTR (Click-Through Rate). This recommendation inference is iterated N times for each input candidate to predict CTR for all candidates. We propose a novel user-aware approach that splits this monolithic model into user and item models, as outlined in Figure 4.

User model: The user model at the edge handles diverse inputs: user-history sparse IDs are computed by user embedding tables, user dense inputs by the static bottom MLP, and context dense inputs by the dynamic bottom MLP. The concatenated output is processed by the user top MLP, generating the user model's output vector transferred to the datacenter for further processing by the item model.

Compute Reuse: The user information processed by the user model is common across all N candidate ranking inferences. Hence, this common computation is processed once per query on the edge, and is reused across all N candidate inferences calculated by the item model in the cloud, significantly reducing the overall embedding table memory accesses and MLP computation.

Data transfer: The size of the data transmitted to the cloud is critical due to the limited upload speed of 5G, capped at 24 Mbps. To reduce payload size, we employ two strategies. First, by decreasing the dimensionality of the edge model's output vector—achieved by offloading a portion of the Top MLP (User Top MLP) to the edge, thus reducing the vector size generated by the feature interaction layer. Second, by utilizing lower precision (qint8) when processing the User Top MLP. These optimizations substantially minimize data transfer overheads, as evident in Figure 10, where data transfer time accounts for only 1% of total query latency, enhancing overall efficiency.

Item model: In parallel to the user model, the item model at the datacenter handles each of the N candidate's sparse item input IDs, extracting their respective features. Then, the received edge model's output vector is combined individually with the N candidate feature vectors. This aggregated data is processed by a combined top MLP to determine the CTR for all N candidates. While the datacenter still computes N candidate inferences, the workload is reduced as the decomposed item model is smaller compared to the original monolithic model.

Accuracy: To limit the accuracy impact of decomposition, we retrain the decomposed model once, while offline in the datacenter. The decomposed Duet model achieves a comparable accuracy with an accuracy drop limited to <= 0.03% across the models (Table 1). Once the user top MLP adopts a quantized int8 precision, we achieve accuracy within 0.06% of the base accuracy, thus, still limiting the accuracy impact at a lower precision.

Model	Base Accuracy (%)	Duet w/o qint8 Accuracy (%)	Duet Accuracy (%)
RM1 [3, 4]	78.71	78.68	78.64
RM2 [3, 4]	78.67	78.66	78.62
RM3 [3, 4]	78.75	78.71	78.69
RM5 [5]	68.74	68.97	68.97

Table 1. Accuracy comparison between Duet & baseline models

4 Model Decomposition

In this section, we describe in detail the decomposition of three components of recommendation models across edge and cloud.

Decoupled Embedding Tables: The increasing size and number of embedding tables present challenges, especially with user embeddings, which constitute a significant portion and capture user history like past likes, favorites, and clicks. For example, n-1 out of the n embedding tables lookups pertain to past user-interaction [6]. Recent work [7] estimates user embeddings to occupy ≈ 75% of total embedding memory space. A user-agnostic model, which doesn't differentiate between user and item embedding access, repeatedly accesses user embedding tables to rank each candidate. However, since past user interactions are common across all candidates ranked for a user query, they can be computed once and reused across all candidate inferences, as shown in Figure 4.

Moreover, user embedding tables at the datacenter (in GBs) store cumulative histories for millions of users with diverse preferences. However, user embeddings memory accesses corresponding to *one* user history (in KBs) is required for a user query. Maintaining up-to-date user-specific embedding tables for each of the million users is not feasible at scale in the datacenters. Therefore, we propose

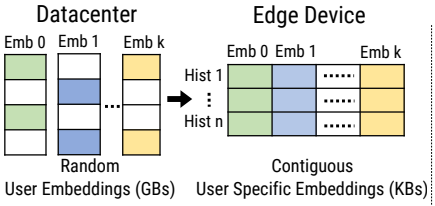

Figure 5. Conversion of large user embeddings to small user-specific embeddings.

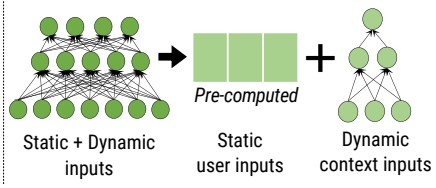

Figure 6. Decomposition of the bottom MLP stack into pre-computed static and smaller dynamic MLP stack.

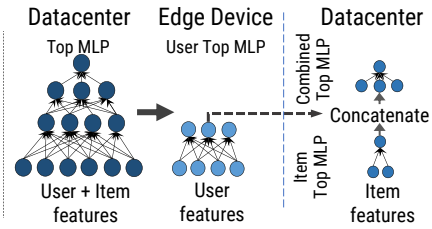

Figure 7. Top MLP stack decomposed into user top MLP, item top MLP, and combined Top MLP.

storing user-specific embedding tables pertaining to user-history in a scratchpad on the local edge device, updated constantly with recent user interactions by Duet architecture. This shrinks the embedding table size from an order of GBs to KBs, which is feasible on a small edge device. Further, as shown in Figure 5, we store them in a contiguous format rather than randomly accessing the entire embedding table.

Static-Dynamic bottom MLP split: Edge devices face challenges with compute-intensive kernels like the bottom MLP. However, a significant portion of the dense inputs to bottom MLP layers consists of stable user information—details like user age, gender, demographics, device-related metrics (such as the number of apps, device class, operating system, and device IP) that don't change frequently. Consequently, we divide the bottom MLP into two parts: the static bottom MLP and the dynamic bottom MLP, as depicted in Figure 6. The static bottom MLP pre-computes the stable user inputs and caches them using Duet's hardware, reducing computational load during execution. On the other hand, the dynamic bottom MLP computes contextual inputs for every query, representing parameters like time of day, day of the week, or time since the last login. These inputs are a smaller portion of the total dense inputs, requiring a proportionally smaller MLP. This approach decreases the real-time workload on the edge device. We design the proposed split of the bottom MLP by proportionally distributing its layers based on the number of static and dynamic inputs. Splitting the original model and using it as is, degrades the accuracy; hence, we train the complete model offline with decomposed bottom MLP stack from scratch. The new model with decomposed MLP incurs a minimal accuracy impact, as illustrated in Table 1.

User-Item top MLP decomposition: We also aim to divide the top MLP to reduce and distribute the remaining portion of the workload between the edge and the datacenter, as depicted in Figure 4. Since the top MLP plays a crucial role in predicting accurate CTR by capturing interaction between user and item features, it can't be entirely split like the bottom MLP. As shown in Figure 7, we split only the initial one or two layers of the top MLP between the edge's user model and the cloud-based item model, considering the impact on accuracy. The remaining layers of the top MLP remain in the cloud as the combined top MLP. The user top MLP processes user features, reducing the dimensionality of the output user vector and subsequently decreasing the payload size. Furthermore, after splitting the top MLP, we have a smaller item top MLP responsible for computing all candidate inferences at the datacenter, thereby reducing the workload. To ensure minimal accuracy impact from the top MLP split, we train the new decomposed model from scratch, along with the previously discussed bottom MLP split.

5 Duet Architecture

The decomposed user model is offloaded on the edge device because it is impractical to maintain individual user state at the datacenter

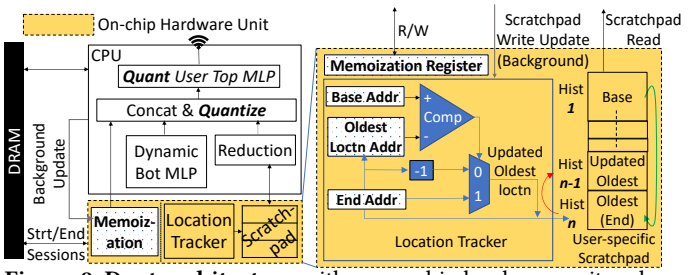

Figure 8. Duet architecture with an on-chip hardware unit and quantized int8 precision support to efficiently process the user model at edge.

scale. However, computing the decomposed user model on the device without hardware optimizations leads to significant energy implications. In this work, we propose a specialized hardware unit, which is placed near the CPU and directly communicates with it, as shown in Figure 8.

5.1 Hardware Unit

The specialized hardware unit is responsible for storing all the relevant local user information required by the recommendation query. It comprises memoization support to store the output of pre-computed static bottom MLP and user-specific scratchpad with a location tracker. The memoization register stores the partially computed result of bottom MLP, corresponding to static bottom MLP, which are reused across queries. During execution, the memoization register value is read and concatenated with other model components. If inputs to the static bottom MLP change, the register value is updated in the background without interfering with the query execution flow. User-specific Scratchpad: Decomposing embedding tables into user and item embedding allows us to store only user-specific embedding tables on the device. However, we still have to access off-chip memory to fetch user embeddings into the CPU. Since the user-specific embedding table size is in the order of KBs that stores only a *fixed length* of history, we propose a user-specific on-chip scratchpad residing in the hardware unit to eliminate the DRAM accesses altogether. Scratchpad should contain only the most up-to-date history, and thus we have a mechanism to populate and update the scratchpad as described below.

Scratchpad Population: User embeddings are related to past user activities, such as apps installed on the device, past liked movie genres, past clicked movies, etc. At the start of a session, the scratchpad is pre-loaded with the user embedding table. As the session unfolds, we update the scratchpad with user embeddings corresponding to recent user interactions of the current session. For our system, the embedding entry data is relayed along with the metadata attached to the candidates being recommended to the user. A movie recommended to a user, for example, will come along with the particular movie embedding entry and its genre embedding entry. Each embedding entry is only a few bytes (128 bytes). If the

Model	Bottom MLP	Top MLP	#EMB tables	#Lookups per Emb	EMB size (#entries)
RM1 [3, 4]	128-64-32	256-64-1	8	80	4M
RM2 [3, 4]	256-128-64	128-64-1	32	120	500K
RM3 [3, 4]	2560-1024-256-32	512-256-1	10	20	2M
RM4 [synthetic]	512-256-32	2560-1024-1	10	20	2M
RM5 [5]	-	200-80-2	3	1-200-200	1M

Table 2. Model configurations used for evaluation.

	Samsung S10e (Snapdragon 855)	X86 Server
Cores	1×A76 @2.8GHz, 3×A76 @2.4GHz, 4×A55 @1.78GHz	18 @3.7Ghz
L1 cache	1×128KB, 3×128KB, 4x128KB	32KB
L2 cache	1×512KB, 3×256KB, 4×128KB	1MB
L3 cache	2MB	25MB
DRAM	6GB, 34.1GB/s	90GB, 80GB/s

Table 3. Architectural specifications

user clicks the movie, we update the scratchpad with its associated embeddings. There will never be a scratchpad miss because the history size is fixed to the past n lookups. For instance, consider n to be 10, and scratchpad containing past 10 liked movie entries. When a new movie 11 is liked by the user, the oldest entry in the scratchpad (movie 1) is removed, and it is populated with movie 11 embeddings. Simultaneously, movie 2 becomes the oldest entry in the scratchpad.

We maintain an updated user history on the small scratchpad with the help of a lightweight location tracker, as shown in Figure 8. It points to the oldest embedding location address on the scratchpad, which is overwritten on a scratchpad update to keep a fixed history of length n. Tracker simultaneously also updates the oldest location address to the next oldest location, which will be written next.

5.2 Heterogeneous precision with int8

Splitting MLP stacks across platforms allows for varied precision, crucial for resource-constrained edge devices favoring int8 precision for efficiency. User top MLP is a significant energy/time consumer at edge; thus, by reducing it's precision from fp32 to quantized int8, energy/time spent on MLP computation as well as data transfer diminishes due to smaller payload size. Post-training static quantization is employed to convert the user top MLP to qint8 precision for inference. The concatenated output of bottom MLP and embeddings is quantized at execution, processed by the quantized user top MLP, producing a qint8 output sent to the cloud. Quantization significantly curtails computing and communication energy/time.

6 Methodology

We evaluate our solution for five recommendation models listed in Table 2. RM1 and RM2 are embedding dominated, RM3 is bottom MLP dominated, and RM4 is top MLP dominated. RM5 is also embedding dominated but does not have dedicated user or item embedding tables. The last two embedding tables are shared by user and item embeddings, i.e., out of 200 embedding table lookups, 199 correspond to the user history and one corresponds to the candidate information. The first embedding table is for user_ids; thus, an incoming user query with a unique user_id will perform one lookup on the user_id embedding table.

The baseline evaluation is conducted on a server-class machine, while mobile performance is assessed on a Samsung 10E platform (Table 3). We leverage an open-source caffe2-based benchmarking framework [4] and enable multi-threading for server models to maximize multi-core capabilities. For mobile support, we adapt the framework to convert x86 caffe2 models to ARMv8 caffe2 models, utilizing QNNPACK for quantized int8 precision. To simulate scratchpad behavior on the Samsung S10E, we store user-specific embeddings in the L3 cache. Additionally, the location tracker is

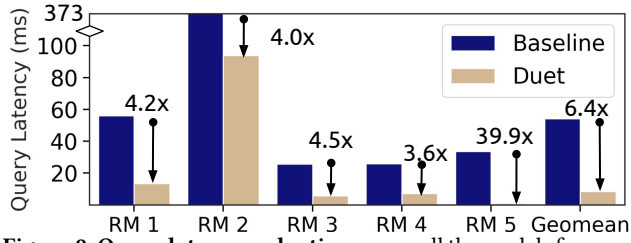

Figure 9. Query latency reduction across all the models for query size of 1024 inferences.

synthesized separately using a 15nm commercial library. We discuss overall hardware unit's overhead in Section 7.4.

To study accuracy impact, we train DLRM models RM1-3 [8] on the Criteo Kaggle dataset [9] and DIN model RM5 [5] on the Amazon dataset [10]. For RM1-3, the majority of inputs (88%) correspond to static user profile information, while 12% are dynamic inputs, based on input definitions of Alibaba's taobao dataset [11]. Thus, we assume a similar % split for the bottom MLPs in the static-dynamic input ratio of 87.5%-12.5%. User-item embedding tables follow a 75%-25% ratio [7]. Furthermore, we also split the first two layers of the top MLP stack into the user and the item model; the last layer is not divided and it is part of the combined top MLP. The RM5 model and its dataset are more descriptive about feature definitions, so we can exactly decouple the user-history access, and the candidate-specific accesses for the last two shared embedding tables. We split the first layer of the top MLP stack with a ratio of 60%-40%. The last two MLP layers are not divided and are part of the combined top MLP.

The energy estimates are extracted using the python RAPL library for the server computation; we utilize battery state dumps comprising charge counters and voltage measurements for the mobile platform. The maximum data upload speed of 24 Mbps is obtained from a speed test on a mobile phone over the 5G network, and the energy estimate is obtained from a recent 5G characterization work [12].

7 Evaluation

7.1 Performance

Figure 9 demonstrates a 6.4× decrease in average latency for a query size of 1024 compared to the baseline. Our comprehensive approach addresses all components of the recommendation model, yielding consistent improvements across bottlenecked models. These gains result from three factors: 1) decomposing the monolithic model into two smaller concurrent models and processing the user model once per query, reducing DRAM access and MLP computation; 2) optimizing the edge-based user model with the lightweight Duet architecture; and 3) overlapping computations between user and item models. The Duet architecture employs an user-specific scratchpad, static bottom MLP memoization, and qint8 precision for the user top MLP, preventing the user model from being a bottleneck by reducing computation and memory demands. Next, we provide detailed latency breakdown of various components on the edge and datacenter.

7.2 Latency Breakdown

We show the latency breakdown in Figure 10 for one of the embedding-dominated models RM2, the bottom MLP-dominated model RM3, and the top MLP-dominated model RM4.

In Figure 10(a), the baseline RM2 model allocates 73% of its time to user embeddings, 24% to item embeddings, 2% to the bottom MLP, and 1% to the top MLP. Storing only user-specific embeddings in the on-chip scratchpad at the edge device offers two benefits.

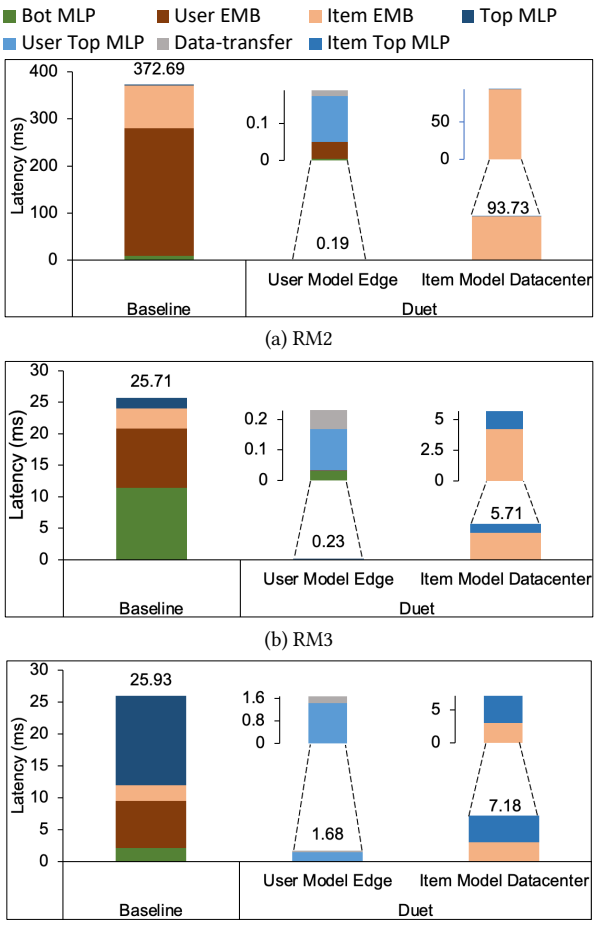

Figure 10. Latency breakdown for RM2-4 models across all components for both the edge and the datacenter platforms.

Firstly, it accelerates retrieval compared to random DRAM access in the datacenter. Secondly, by computing this significant portion of user embeddings once for a query, we reuse the computation across all query inferences. Duet's architectural optimizations reduce the combined execution time of bottom MLP, top MLP, and data transfer to less than 1% of the total execution time. Eventually, overall performance is limited by datacenter computation post-decomposition.

In the bottom MLP-focused RM3 model, 44% of the baseline execution time is dedicated to processing the bottom MLP (Figure 10(b)). The remaining 37% for user embeddings, 12% for item embeddings, and 7% for the top MLP. We optimize the bottom MLP by storing precomputed results in the hardware unit's memoization register and user embeddings by computing them once per query on the lightweight Duet architecture at edge, thus significantly reducing the individual component's execution time. The top MLP's execution time also decreases because of model decomposition, which reduces the computation demand in the datacenter.

In Figure 10(c), the breakdown for the top MLP-focused RM4 model reveals 54% of time on the top MLP, 29% on user embeddings, 10% on item embeddings, and 8% on the bottom MLP. Duet's architecture optimizes the top MLP by splitting it into user, item, and combined top MLP. The user top MLP on edge is computed once with qint8 precision, overlapping with datacenter computation. Moreover, post-split, a smaller datacenter's top MLP completes all 1024 inferences sooner than the baseline. Duet's optimizations also

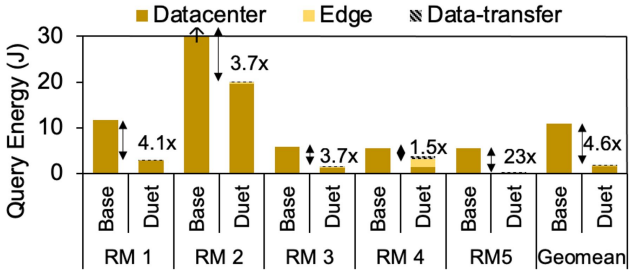

Figure 11. Energy consumption for query size of 1024 inferences.

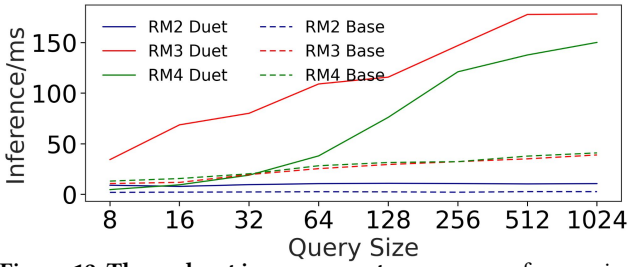

Figure 12. Throughput improvement over a range of query sizes

reduce user embedding and bottom MLP fraction. While Duet introduces additional data-transfer time, we mitigate it by minimizing payload size through user top MLP processing and quantization.

7.3 Energy

Figure 11 demonstrates Duet's energy efficiency gains. Our solution considers datacenter, edge computation, and data-transfer energy consumption. We observe energy reduction of 4.6× for 1024-inference query size. The datacenter's energy drops by 7× due to Duet's model decomposition at the expense of an additional small fraction of energy consumption on edge. Further, Duet architecture's energy-centric optimizations reduces the edge device's energy consumption.

Embedding-focused RM1 and RM2 models spend 97% of Duet's energy in the datacenter, with the remaining 3% is spent on user model processing at the edge and data transfer. For bottom MLP-driven RM3, the datacenter consumes 75% of Duet's energy, while the edge and data transfer take 25%. Despite RM3's heavier edge computation, because the static bottom MLP is pre-computed, we lower the query energy by 3.7×. In the top MLP-driven RM4 model, the datacenter consumes 36% of Duet's energy, with the edge and data-transfer using 64%. Despite quantization reducing the user top MLP size, computations on the edge device with limited cache lead to higher energy usage – a heavy user top MLP of 1.9 MB is computed on the device with 2MB of L3 cache. While it doesn't fully eliminate DRAM overheads, compared to the fp32 model, the quantized version significantly cuts edge and data-transfer energy, reducing query energy by 1.5×. We also show a significant energy reduction of 23× for embedding dominated RM5 model because the major fraction of the embedding lookups, attributed to the user history, are computed once on the edge in an optimized manner.

7.4 Hardware Unit Silicon Overheads

Table 4 presents the on-chip scratchpad size required for storing user-specific embeddings by the different models. These are derived from a recent SRAM design in 10nm technology node [13]. The location tracker is synthesized separately using a 15nm commercial library at clock frequency of 2.8Ghz. The post synthesis timing is just 0.33ns@0.66mW and occupies $250um^2$. Thus, our location tracker is an extremely lightweight unit. Majority area of the hardware unit is occupied by scratchpad. Embedding heavy RM2 requires the largest scratchpad space of 720 KB, corresponding

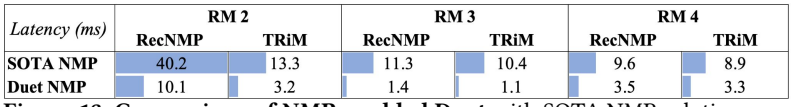

Figure 13. Comparison of NMP-enabled Duet with SOTA NMP solutions.

Model	RM1	RM2	RM3	RM4	RM5
Scratchpad Size (KBs)	60	720	17.5	17.5	28.2
Scratchpad Area (mm^2)	0.020	0.238	0.006	0.006	0.009

Table 4. Scratchpad Overhead

to $0.24mm^2$ of silicon area. In contrast, all the other models require only a few tens of KB of scratchpad and $<0.1mm^2$ area. Thus, Duet is a high-performance and energy-efficient solution with negligible area overheads.

7.5 Sensitivity to Query Size

Figure 12 shows that Duet improves throughput by 3× over a range of query sizes (8-1024) for embedding-dominated RM2, bottom MLP-dominated RM3, and top MLP-dominated RM4 models. While RM2 and RM3 offer throughput gains for all query sizes, RM4 experiences decreased throughput at smaller query sizes (8, 16, and 32). This behavior is because the RM4 user top MLP's execution time at the edge device is higher than the total time taken to process all items of a query, sized 8-32, in the datacenter; hence, edge model becomes a bottleneck. The heavy user top MLP (1.9MB) of RM4 cannot be fully cached, resulting in higher latency. RM2 and RM3 models remain lightweight even at small query sizes, thanks to pre-computed static bottom MLP and user-specific scratchpad.

7.6 Comparison to NMP solutions

In Figure 13, we compare the NMP enabled Duet with the two solutions – RecNMP [14] and TRiM [15] – for RM2, RM3, and RM4 models. NMP techniques are complementary to our solution; therefore, we also utilize these techniques for our item model computation at the datacenter for a fair comparison. Meanwhile, the user model is computed by the lightweight Duet architecture. Duet-NMP decreases the average query latency by 4.18× and 4.2× over RecNMP and TRiM because the decomposition of models reduces the amount of work that is completed at the datacenter. Thus, at the datacenter, Duet-NMP completes a lesser amount of work faster than the state-of-the-art (SOTA) NMP solutions.

8 Related Work

To the best of our knowledge, this is the first effort to incorporate edge computation in recommendation engines. Prior work EdgeREC[16] shows the benefit of using extra user inputs available exclusively on a user device, like scroll speed and exposure duration, to improve user-engagement/accuracy. Our solution incorporates user edge devices to improve performance and energy of recommendation. While recent studies explored user-specific machine learning for computer vision on edge devices [17] and partitioning DNN models for various applications [18], our methodology introduces user-aware partitioning specifically tailored for recommendation models.

Many recent works have accelerated recommendation inference using specialized hardware, like GPUs [19], FPGAs [20], and systolic arrays [21], or optimal schedulers [4]. While prior works have focused on the hardware aspect, we present a hardware-software co-design approach to leverage user information on CPUs with minimal silicon area overhead of Duet architecture. We can complement these accelerator solutions to further improve the datacenter model performance. Further, prior works like [14, 15, 19] have proposed solutions to overcome the memory bottleneck of embedding tables with two primary techniques: 1) caching hot embeddings on the CPU to bypass DRAM access, and 2) performing embedding lookups and reduction operation in memory using NMP. Duet is complementary to such prior works, as shown in the evaluation. [21] employs a multi-stage recommendation engine with a lightweight frontend model to trim down candidates for the complex model. Here, we fragment the complex model into smaller concurrent models. The frontend filtering step can be further added to boost the efficiency of Duet.

9 Conclusion

Recommendation models process multiple candidates to recommend only a few to the user. They exhibit non-uniformity in their inputs, which can be categorized as user inputs and item inputs. This paper explores the opportunity to decouple the two inputs and operate on user inputs at the local edge device. We present Duet, a novel collaborative edge-cloud recommendation system, which decomposes the giant monolithic model into two smaller concurrent models – user model on the edge and item model on the datacenter – that come together to deliver final recommendations. The user model is computed once per query by our new energy-efficient Duet architecture on the resource-constrained edge device, and its output is reused by all the candidates computed by the item model on the datacenter for the query. We demonstrate that our proposed lightweight Duet architecture reduces query latency by an average of 6.4× and lowers average energy consumption by 4.6×.

Acknowledgements. This work was supported by the Applications Driving Architectures (ADA) Research Center, a JUMP Center co-sponsored by SRC and DARPA.

References

[1] "Deep dive into netflix's recommender system." https://towardsdatascience.com/deep-dive-into-netflixs-recommender-system-341806ae3b48.
[2] "How retailers can keep up with consumers." https://www.mckinsey.com/industries/retail/our-insights/how-retailers-can-keep-up-with-consumers.
[3] U. Gupta *et al.*, "The architectural implications of facebook's dnn-based personalized recommendation," in *HPCA*, 2020.
[4] U. Gupta, S. Hsia, V. Saraph, X. Wang, B. Reagen, G.-Y. Wei, H.-H. S. Lee, D. Brooks, and C.-J. Wu, "Deeprecsys: A system for optimizing end-to-end at-scale neural recommendation inference," in *ISCA*, 2020.
[5] G. Zhou *et al.*, "Deep interest network for click-through rate prediction," in *SIGKDD*, 2018.
[6] B. Acun *et al.*, "Understanding training efficiency of deep learning recommendation models at scale," in *HPCA*, 2021.
[7] A. Eisenman *et al.*, "Bandana: Using non-volatile memory for storing deep learning models," *MLSys*, 2019.
[8] M. Naumov *et al.*, "Deep learning recommendation model for personalization and recommendation systems," *CoRR*, vol. abs/1906.00091, 2019.
[9] "Criteo kaggle advertising dataset." https://ailab.criteo.com/ressources/.
[10] J. McAuley, C. Targett, Q. Shi, and A. Van Den Hengel, "Image-based recommendations on styles and substitutes," in *SIGIR*, 2015.
[11] "Ad display/click data on taobao.com." https://tianchi.aliyun.com/dataset/dataDetail?dataId=56.
[12] A. Narayanan *et al.*, "A variegated look at 5g in the wild: performance, power, and qoe implications," in *SIGCOMM*, 2021.
[13] Z. Guo *et al.*, "A 23.6-mb/mm2 sram in 10-nm finfet technology with pulsed-pmos tvc and stepped-wl for low-voltage applications," *JSSC*, 2018.
[14] L. Ke *et al.*, "Recnmp: Accelerating personalized recommendation with near-memory processing," in *ISCA*, pp. 790–803, 2020.
[15] J. Park *et al.*, "Trim: Enhancing processor-memory interfaces with scalable tensor reduction in memory," in *MICRO*, 2021.
[16] Y. Gong *et al.*, "Edgerec: recommender system on edge in mobile taobao," in *CIKM*, 2020.
[17] V. Goyal, V. Bertacco, and R. Das, "Myml: User-driven machine learning," in *DAC*, 2021.
[18] Y. Kang *et al.*, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," *ASPLOS*, 2017.
[19] Y. Kwon, Y. Lee, and M. Rhu, "Tensordimm: A practical near-memory processing architecture for embeddings and tensor operations in deep learning," in *MICRO*, 2019.
[20] R. Hwang *et al.*, "Centaur: A chiplet-based, hybrid sparse-dense accelerator for personalized recommendations," in *ISCA*, 2020.
[21] U. Gupta *et al.*, "Recpipe: Co-designing models and hardware to jointly optimize recommendation quality and performance," in *MICRO*, 2021.