Cross-Stack Workload Characterization of Deep Recommendation Systems

Samuel Hsia[1], Udit Gupta[1,2], Mark Wilkening[1],
Carole-Jean Wu[2], Gu-Yeon Wei[1], David Brooks[1]

[1]Harvard University [2]Facebook Inc.

shsia@g.harvard.edu

***Abstract*—Deep learning based recommendation systems form the backbone of most personalized cloud services. Though the computer architecture community has recently started to take notice of deep recommendation inference, the resulting solutions have taken wildly different approaches – ranging from near memory processing to at-scale optimizations. To better design future hardware systems for deep recommendation inference, we must first systematically examine and characterize the underlying systems-level impact of design decisions across the different levels of the execution stack. In this paper, we characterize eight industry-representative deep recommendation models at three different levels of the execution stack: algorithms and software, systems platforms, and hardware microarchitectures. Through this cross-stack characterization, we first show that system deployment choices (i.e., CPUs or GPUs, batch size granularity) can give us up to $15\times$ speedup. To better understand the bottlenecks for further optimization, we look at both software operator usage breakdown and CPU frontend and backend microarchitectural inefficiencies. Finally, we model the correlation between key algorithmic model architecture features and hardware bottlenecks, revealing the absence of a single dominant algorithmic component behind each hardware bottleneck.**

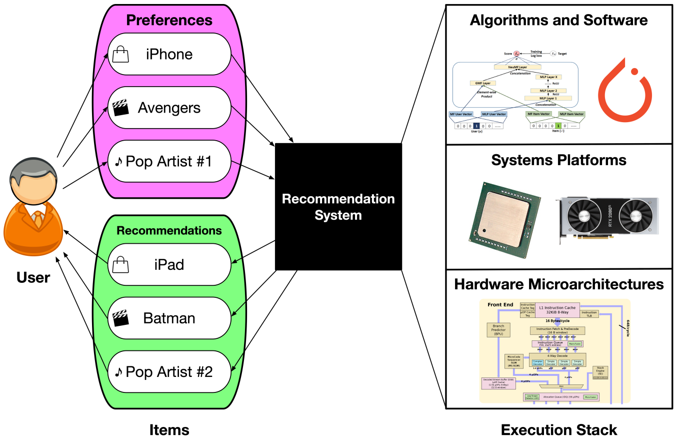


Fig. 1: *Recommendation systems* leverage information about a user's preferences to recommend new content. The three main components include users, items, and the recommendation system itself. This paper analyzes recommendation inference at three different levels of the computing stack: algorithms and software, systems platforms, and microarchitectures.

I. Introduction

Recommendation systems recommend items to users based on the users' personal preferences. E-commerce marketplaces (e.g., Amazon, Alibaba) that recommend relevant goods for purchase [1]–[3], social media platforms (e.g., Facebook, Twitter) that regularly update user feeds with new multimedia content [4], and entertainment services (e.g., Netflix, YouTube) that promote new playlists [5], [6] all heavily rely on high-accuracy recommendation systems to maintain quality of service and positive user experience. To achieve the highest possible recommendation accuracies, these recommendation algorithms have evolved from using classical information filtering techniques [7], [8] to state-of-the-art deep-learning based solutions. Figure 1 depicts the general workflow of a typical recommendation system in the context of the aforementioned internet-based applications – these algorithms are programmed using different deep learning frameworks and executed on systems of different architectures.

While deep-learning solutions provide high quality recommendations, they also require high infrastructure overheads to run efficiently. For instance, Facebook's recommendation use cases require more than 10× the datacenter inference capacity compared to computer vision and natural language processing tasks [9]. As a result, over 80% of machine learning inference cycles on Facebook's datacenter fleet are devoted to recommendation filtering and ranking [10]. The model training process tells a similar story — over 50% of the training demands are attributed to deep recommendation models [11]. Similar capacity demands can be found at other companies like Google [12], Amazon [13], Alibaba [2], [3], and Baidu [14].

Given the importance of recommendation models, exploring custom hardware solutions will be important for further workload acceleration and infrastructure efficiency. Recent work has demonstrated that some recommendation models – given their large memory capacity requirements and irregular memory access patterns – pose unique performance bottlenecks compared to other deep learning based workloads (e.g., CNNs and RNNs) [15]. Thus, existing proposals for accelerating datacenter scale CNNs and RNNs do not directly apply to recommendation models. By exploiting these unique compute characteristics, researchers have demonstrated that novel memory systems can provide significant speedup on a specific class of recommendation models dominated by table lookup operations [16]–[18]. However, there still exists large algorithmic diversity across recommendation use cases [15], [19], [20]. Because of this diversity, holistic architectural bottleneck analysis – for both CPU microarchitecture and heterogeneous hardware – will enable hardware customization for improving recommendation inference efficiency.

In this paper, we perform a detailed workload characterization of recommendation inference at three different levels of the execution stack: algorithms and software, systems platforms, and hardware microarchitectures. First, we evaluate the eight recommendation networks on a variety of server class CPUs and AI accelerators (i.e., GPUs). This evaluation shows the optimal system – between CPUs and GPUs and between microarchitecture generations (i.e., Intel Broadwell versus Cascade Lake and NVIDIA Pascal versus Turing) – varies based on the model architectures and inference batch sizes. In addition to heterogeneous system evaluation, we demonstrate that Caffe2 software operator usage breakdowns – which vary across model architectures, batch sizes, and underlying hardware – expose performance bottlenecks and opportunities for architectural optimizations. Finally, we perform a detailed microarchitectural performance analysis on server class Intel Broadwell and Cascade Lake CPUs using TopDown [21]. Our microarchitectural analysis reveals that recommendation inference suffers from a variety of microarchitecture inefficiencies related to frontend decoders, backend functional units and memory systems, and branch speculation.

The main contributions of this paper are:

1) While conventional wisdom and recent research indicates that AI accelerators, such as GPUs, readily accelerate deep learning workloads, this work shows the optimal hardware system (i.e., CPU versus GPU) varies based on recommendation use cases (Section IV).
2) In addition to overall execution times, we analyze software operator usage at the algorithm level. Previous work classifies recommendation models based on their operator breakdowns at fixed use cases on CPUs. In contrast, our analysis shows that varying the model architecture, batch-size, and hardware platform can alter the target operators for future hardware optimization (Section V).
3) Based on the detailed TopDown results from a server class Intel Broadwell CPU, we see that pipeline bottlenecks vary based on model architecture features. FC-intensive models (i.e., RM3, WnD, MT-WnD) suffer from insufficient functional units; embedding-intensive models (i.e., RM1, RM2) suffer from ineffective frontend decoders; attention-based models (i.e., DIN, DIEN) suffer from instruction cache misses (Section VI).
4) We compare the microarchitectural performance characteristics of Broadwell and Cascade Lake CPUs. The Cascade Lake microarchitecture improves recommendation inference performance across all use cases with its wider Single Instruction Multiple Data (SIMD) execution units and enhanced speculation capabilities. With these optimizations, the performance bottleneck on Cascade Lake shifts to the backend memory subsystem (Section VI).

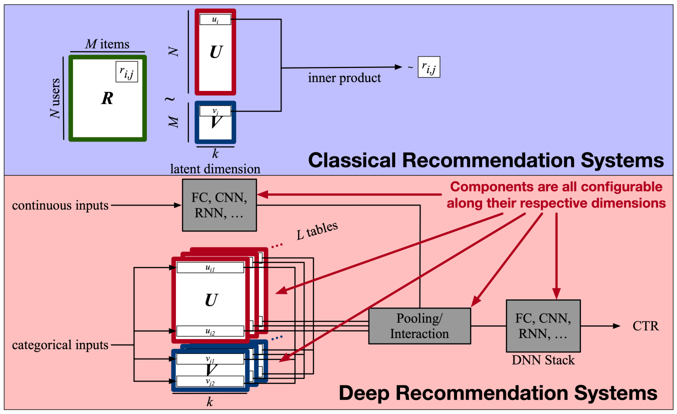


Fig. 2: **Evolution of Recommendation Systems.** Traditional collaborative filtering (**top**) models user preferences as two (user and item) embedding tables and inner product between table entries. Deep learning based methods (**bottom**) still leverage embedding tables – though the number of tables, number of lookups per table, latent dimension, and subsequent matrix operations (gray blocks) are all highly configurable.

II. Recommendation Background

Recommendation is the task of suggesting new content to users based on their preferences and prior content interactions. Recommendation is used in many popular internet services (e.g., search, entertainment streaming, e-commerce) to improve user experience by enabling personalization. The main challenge for recommendation systems is accurately modeling users' preferences based on sparse training data — users often explore only a tiny fraction of all available items. This section explores how recommendation systems have evolved over time. More specifically, we discuss the semantics of classical recommendation approaches and important algorithmic components of state-of-the-art deep learning methods.

A. Classical Recommendation Systems

Classical recommendation systems are generally categorized as either content-based filtering or collaborative filtering. Content-based filtering recommends content based on a user's personal preferences and item interaction history while collaborative filtering exploits preference similarity across users. Content-based filtering models each user as feature vector $\mathbf{u}_i \in \mathbb{R}^k$ and each item as feature vector $\mathbf{v}_j \in \mathbb{R}^k$, where k is the number of learned features (i.e., latent dimension) for each user and item.

Collaborative filtering represents historical interactions between N users and M items as a partially-observed matrix $R \in \mathbb{R}^{N \times M}$, where r_{ij} is the historical interaction between user i and item j. To compensate for unobserved entries, collaborative filtering approximates R with matrix factorization

Model	Application Domain (Evaluation)	Unique Requirement/Use Case	Model Architecture Insight
NCF	Movies (MovieLens)	Small amount of required training data (see # of embedding tables)	Small model with only four embedding tables
RM1	Social Media (Facebook)	Early stage filtering (i.e., low run-time requirements)	Small model with medium amount (80) of lookups per embedding table
RM2	Social Media (Facebook)	Late stage ranking (i.e., high accuracy requirements) targeting categorical features	Large model with large amount (120) of lookups per embedding table
RM3	Social Media (Facebook)	Late stage ranking (i.e., high accuracy requirements) targeting continuous features	Large model with large FC stacks and immediate continuous input processing
WnD	Smartphone Applications (Google Play Store)	Generic large-scale regression and classification problems with categorical features	Medium model with large FC stacks
MT-WnD	Video (YouTube)	Evaluation of multiple objectives (e.g., likes, ratings)	Large model with multiple parallel FC stacks on top of WnD
DIN	E-Commerce (Alibaba)	Model evolving user preferences (i.e., time-series nature of dataset)	Large model with local activation weights for large amount (750) of lookups from user behavior embedding tables
DIEN	E-Commerce (Alibaba - Taobao)	Model evolving user preferences (i.e., time-series nature of dataset)	Medium model with interaction GRUs to replace large amount of lookups found in DIN

TABLE I: Summary of eight industry-representative recommendation models and their important architectural insights.

(MF) as a user matrix $U \in \mathbb{R}^{N \times k}$ and item matrix $V \in \mathbb{R}^{M \times k}$, where k is again the latent dimension:

$$R \approx \hat{R} \equiv UV^T \tag{1}$$

As shown in Figure 2 (top), to predict the interaction score of user i with item j (r_{ij}) we would have to find the inner product of the i^{th} row of U and the j^{th} row of V:

$$r_{ij} \approx \hat{r_{ij}} \equiv \mathbf{u}_i{}^T \mathbf{v}_j \tag{2}$$

B. Deep Recommendation Systems

In order to leverage the abundance of user data and model complex user-item interactions, recommendation systems have shifted from the aforementioned techniques, e.g., [7], [8], to deep-learning based approaches [19]. Figure 2 (bottom) shows the general deep-learning based model architecture; the embedding tables and components shaded in gray (i.e., DNN-stacks and pooling/interaction layers) are highly configurable:

- **Embedding Tables.** Deep recommendation systems rely on embedding tables (Figure 2, red and blue outline) to encode information about users and items. Every input sample has categorical components (e.g., movies a user likes) – represented as multi-hot encoded vectors – that are processed as table lookups. Embedding tables are configured based on the number of lookups, number of entries per table, number of tables, and latent dimension of embedding vectors. As table sizes increase ($\sim$GBs), access patterns become increasingly sparse – leading to irregular memory accesses that make system optimizations challenging.
- **DNN Stacks.** In addition to categorical features, input samples also include continuous features (e.g., user age) that are processed directly by DNN stacks. DNN stacks range from vanilla fully-connected (FC) layers to more complicated architectures (e.g., autoencoders, CNNs, RNNs).
- **Feature Interaction Layers.** To unify outputs from embedding table lookups and DNN stacks, deep recommendation systems have feature interaction layers that range from simple concatenation to attention mechanisms.

III. Methodology

Machines	Xeon E5-2697A	Xeon Gold 6242	GTX 1080 Ti	T4
Microarchitecture	Broadwell	Cascade Lake	Pascal	Turing
Frequency	2.6 GHz	2.8 GHz	1.48 GHz	0.58 GHz
Cores (SM Count)	16	16	(28)	(40)
SIMD (CUDA Capability)	AVX-2	AVX-512	(6.1)	(7.5)
L1 Cache Size	32 KB	32 KB	48 KB	64 KB
L2 Cache Size	256 KB	1 MB	2.75 MB	4 MB
L3 Cache Size	40 MB	22 MB	N/A	N/A
L2/L3 (L1/L2) Cache Inclusion Policy	Inclusive	Exclusive	(Inclusive)	(Inclusive)
DRAM Capacity	256 GB	384 GB	11 GB	16 GB
DDR Type	DDR4	DDR4	GDDR5X	GDDR6
DDR Frequency	2400 MHz	2933 MHz	1376 MHz	1250 MHz
DDR Bandwidth	77 GB/s	131 GB/s	484.4 GB/s	320 GB/s
TDP	145 W	150 W	250 W	70 W

TABLE II: Summary of hardware platforms studied.

Personalized recommendation systems run a diverse collection of state-of-the-art deep-learning models across heterogeneous datacenter hardware. To understand the impact of algorithmic model diversity on inference performance, we characterize eight industry-representative, publicly-available recommendation models. The model implementations are from the open-sourced DeepRecSys repository and are also not pre-trained as this study focuses solely on inference compute requirements [15]. We characterize the recommendation model performance on server class CPUs (i.e., Intel Broadwell and Cascade Lake) and GPU-based AI accelerators (i.e., NVIDIA 1080 Ti and T4). This section describes the models and system platforms used in this work.

A. Deep Recommendation Models

Figure 2 (bottom) provides a generalization of deep recommendation model architectures. Building on the general model architecture, our characterization studies eight industry-representative deep recommendation models with unique network parameters, as shown in Table I [15].

1) **Neural Collaborative Filtering (NCF)** extends matrix factorization with multi-layer perceptrons (MLPs) and non-linearities:

$$r_{ij} \approx \hat{r_{ij}} \equiv \phi(\mathbf{w}^T(\mathbf{u}_i \circ \mathbf{v}_j)) \tag{3}$$

where ϕ and $\mathbf{w}$ are the activation function and weights respectively. Although NCF has only four embedding tables, it has shown success with the MovieLens dataset [5].

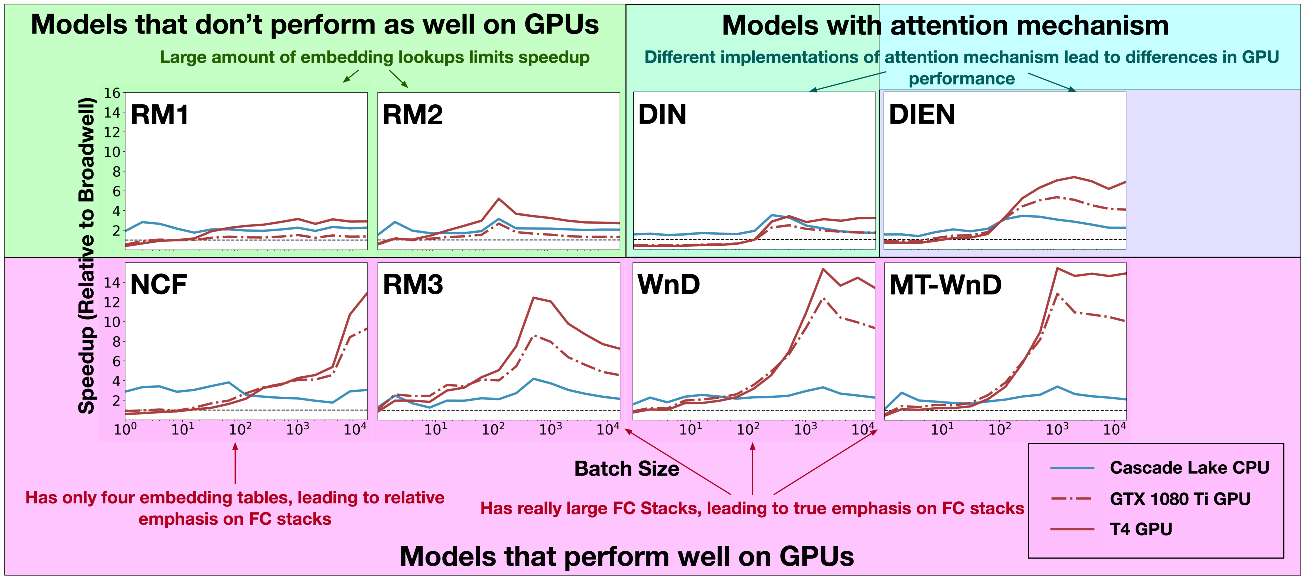


Fig. 3: **Systems performance evaluation** represented as speedup over Broadwell CPU across models, batch-sizes, and hardware platforms. Models are grouped into three primary, overlapping categories – ones that perform well on GPUs, ones that have comparable performance on CPUs and GPUs, and attention-based models with varying implementations.

2) **Deep Learning Recommendation Model (DLRM RM1, RM2, RM3)** is a highly configurable model with multi-hot encoded embedding lookups. Outputs of embedding lookups are aggregated with the output of DNN stacks that process continuous input features. We configure three representative DLRM networks – RM1, RM2, RM3 – with varying ratios of FC weights and embedding lookups based on Facebook's social media ranking models [4], [15], [22].
3) **Wide and Deep (WnD)** captures both the memorization and generalization benefits by concatenating outputs of one-hot encoded embedding lookups with continuous inputs. The resulting features are then processed with deep feed-forward networks. WnD has been used to rank applications in Google's Play Store [23].
4) **Multi-Task Wide and Deep (MT-WnD)** expands upon WnD by adding parallel output FC layers on top of WnD to evaluate multiple objectives. While the other models predict a single engagement objective such as click-through rate (CTR), MT-WnD evaluates multiple objectives such as likes and ratings. MT-WnD provides high quality next video recommendations on YouTube [6].
5) **Deep Interest Network (DIN)** addresses evolving user preferences by implementing the attention mechanism with local activation units for embedding table lookups. User embedding tables process a small number of lookups while item embedding tables process hundreds of lookups. DIN has been deployed to great success by Alibaba in its online marketplace for display advertising [2].
6) **Deep Interest Evolution Network (DIEN)** also addresses evolving user preferences but uses multi-layered gated recurrent units (GRUs) to explicitly separate user preferences from user interaction history. For item embedding tables, this leads to fewer lookups per table as more of the information processing is offloaded to the GRU layers. Like DIN, DIEN has been deployed successfully by Alibaba on its display advertising services (specifically on Taobao) [3].

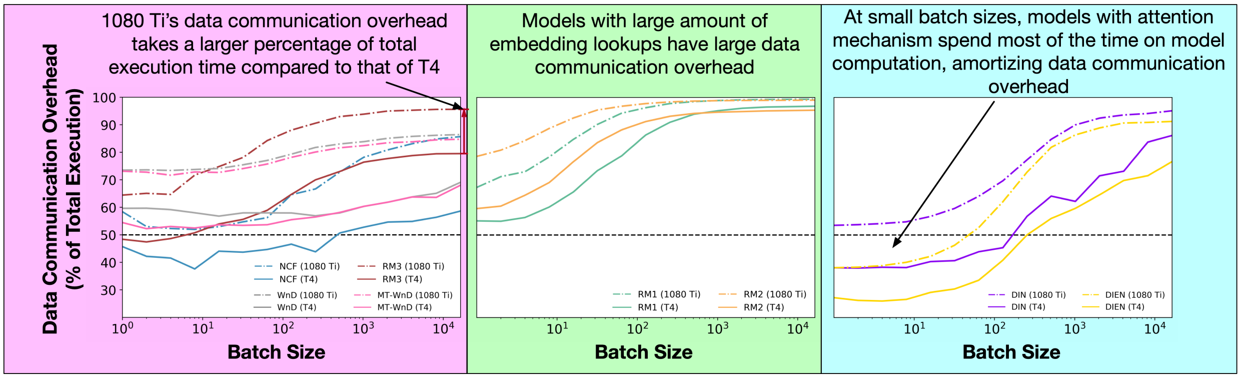
B. *Systems Platforms*

State-of-the-art recommendation models are deployed across heterogeneous hardware systems in datacenters. In fact, exploiting hardware heterogeneity to schedule inferences on optimum platforms based on use cases (i.e., model architecture, inference batch-size) significantly improves recommendation performance [15]. This work mimics this hardware heterogeneity by providing in-depth characterizations on two server class CPUs (i.e., Intel Broadwell and Cascade Lake) and two GPUs (i.e., NVIDIA GTX 1080 Ti and T4). Table II summarizes the key architectural features of the platforms. GPUs are connected to CPUs via PCIe 3.0. All results assume single-threaded inference in Caffe2, and include both data loading and model computation times to capture end-to-end recommendation inference.

IV. Systems Platforms Evaluation

This section describes the performance characteristics of the eight recommendation models at different use cases (i.e., input batch sizes and systems platforms). The range of batch sizes follows recent work that shows recommendation in datacenters runs with batch sizes from tens to thousands to meet different SLA targets [15]; the range of systems platforms exposes the effects of different generations of CPUs and GPUs. In the context of GPUs, we find that data-communication overheads limit GPU performance. Overall, the analysis shows that the optimum hardware for recommendation inference depends on both model architecture and batch size.

Figure 3 depicts the speedups of the Cascade Lake CPU, GTX 1080 Ti GPU, and T4 GPU over a baseline Broadwell


1080 Ti's data communication overhead takes a larger percentage of total execution time compared to that of T4

Models with large amount of embedding lookups have large data communication overhead

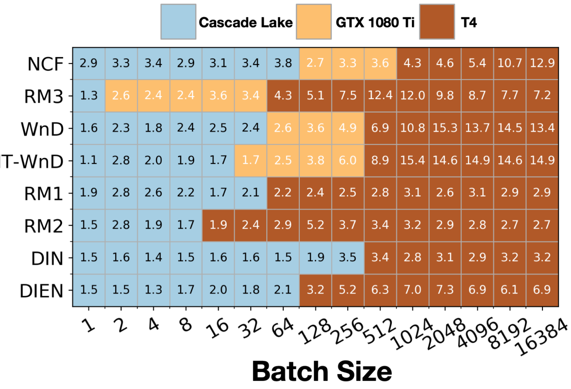
At small batch sizes, models with attention mechanism spend most of the time on model computation, amortizing data communication overhead

Fig. 4: **GPU data communication overheads** as percent of total execution time. While parts of this overhead can be attributed to software implementation overhead, the majority is due to GPU data loading (i.e., CPU-GPU communication overheads).

CPU server. The results are organized by recommendation model, across batch-sizes 1 to 16384, and consider end-to-end execution times (i.e., model-computation plus data-communication). The important observations are:

1) Model architecture plays an important role in accelerating recommendation inference. Models in the bottom row of Figure 3 exhibit high speedup on the NVIDIA 1080 Ti and T4 GPUs. For larger batch sizes ($\sim 10^3$), the GPUs provide *an order of magnitude* speedup over the baseline Broadwell CPU; for smaller batch sizes ($< 10^2$), we observe a $2-4\times$ speedup. The relatively high speedups observed come from the models (i.e., NCF, RM3, WND, MT-WND, RM3) sharing an important algorithmic characteristic (Table I): whether it is fewer embedding tables in NCF, large FC stacks for continuous inputs in RM3, or large FC stacks to output final probability scores in WnD and MT-WnD, each of these models relies on FC stacks to model user preferences. As GPUs readily accelerate matrix operations, they outperform CPUs on NCF, RM3, WND, and MT-WND.

Compared to the models with large FC components, RM1 and RM2 exhibit relatively lower speedups – less than 4× – when deployed on GPUs (Figure 3 top left). In fact, at small batch sizes, Cascade Lake consistently outperforms the 1080 Ti GPU (and by at least 2× at small batch sizes) and offers speedups within 10% of the T4. This is a result of RM1 and RM2 having a large number of lookups per embedding table – 80 and 120 lookups respectively. In comparison, the remaining models have fewer than 20 lookups per table. The larfe number of lookups shifts the performance bottleneck towards embedding operations that comprise of irregular memory accesses (see Section V for details). Depending on the input batch-size, CPUs and GPUs perform comparably on these models dominated by irregular memory accesses.



	1	2	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384
NCF	2.9	3.3	3.4	2.9	3.1	3.4	3.8	2.7	3.3	3.6	4.3	4.6	5.4	10.7	12.9
RM3	1.3	2.6	2.4	2.4	3.6	3.4	4.3	5.1	7.5	12.4	12.0	9.8	8.7	7.7	7.2
WnD	1.6	2.3	1.8	2.4	2.5	2.4	2.6	3.6	4.9	6.9	10.8	15.3	13.7	14.5	13.4
MT-WnD	1.1	2.8	2.0	1.9	1.7	1.7	2.5	3.8	6.0	8.9	15.4	14.6	14.9	14.6	14.9
RM1	1.9	2.8	2.6	2.2	1.7	2.1	2.2	2.4	2.5	2.8	3.1	2.6	3.1	2.9	2.9
RM2	1.5	2.8	1.9	1.7	1.9	2.4	2.9	5.2	3.7	3.4	3.2	2.9	2.8	2.7	2.7
DIN	1.5	1.6	1.4	1.5	1.6	1.6	1.5	1.9	3.5	3.4	2.8	3.1	2.9	3.2	3.2
DIEN	1.5	1.5	1.3	1.7	2.0	1.8	2.1	3.2	5.2	6.3	7.0	7.3	6.9	6.1	6.9

Fig. 5: **Optimal hardware for each model architecture and batch size:** each grid cell show the speedup over Broadwell when using the optimum hardware (color).

2) Different model architecture implementations of an algorithmic feature like the attention mechanism can have different hardware implications. Algorithmically, both DIN and DIEN use attention to learn users' evolving interests over time. DIN implements attention with local activation units and small FC layers followed by concatenation operations for aggregation while DIEN implements attention using multilayered gated-recurrent units (GRUs). For DIN, Broadwell machines outperform GPUs at batch-sizes less than 100. At larger batch-sizes, GPU speedup saturates below 4×. The lower speedups are a direct result of DIN implementing attention with heavy concatenation operations that perform poorly on GPUs. In comparison, DIEN achieves up to 7× speedup on GPUs compared to Broadwell, as GRUs translate to matrix multiplications that perform well on GPUs.

3) Compared to Broadwell, Cascade Lake improves performance across all models and batch sizes. Across all use cases, encompassing models and batch-sizes, Cascade Lake achieves higher performance than Broadwell CPUs. Following Table II, the improved performance is a result of various micro-architectural features such as wider SIMD width for FC-focused models, larger L2 cache capacity, and higher DRAM frequency. Section VI details the micro-architectural features that enable higher performance on Cascade Lake.

4) Compared to GTX 1080 Ti, T4 improves performance for specific models and batch-sizes. For NCF, RM3, WnD, MT-WnD, and DIEN, T4 outperforms the 1080 Ti at batch sizes larger than $\sim 10^3$, offering higher speedups due to higher streaming multiprocessor (SM) count. However, for RM1 and RM2, T4 becomes advantageous at smaller batch sizes – due to the increase in GDDR5X to GDDR6 frequency. This is

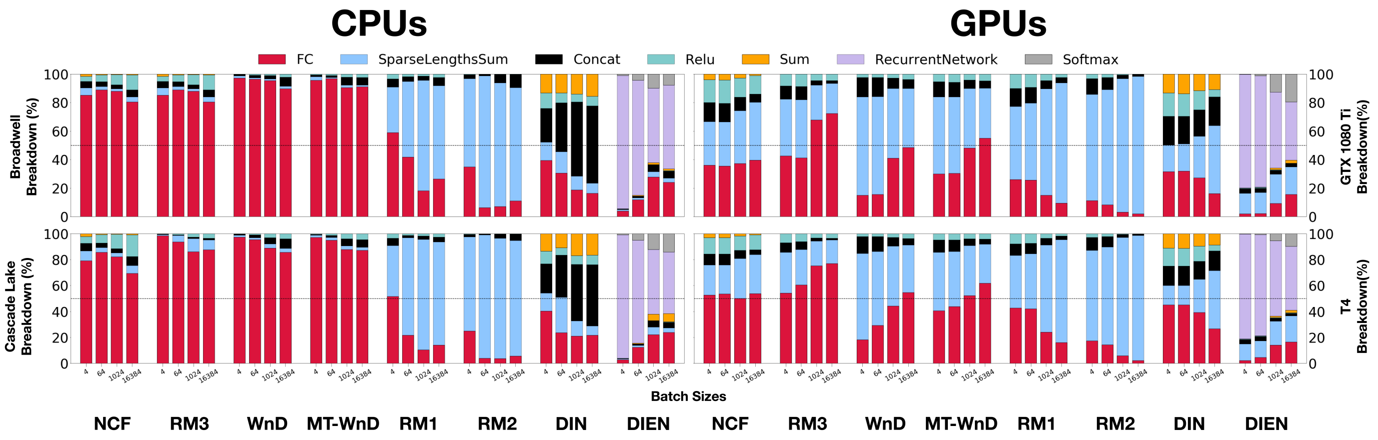


Fig. 6: **Caffe2 operator breakdowns** with CPUs (left) and GPUs (right). Models readily accelerated by GPUs are dominated by matrix operations (i.e., FC in red and recurrent layers in purple). Operator breakdowns between CPUs and GPUs vary significantly — models dominated by FC execution time on CPUs spend a large fraction of time on other operators on GPUs.

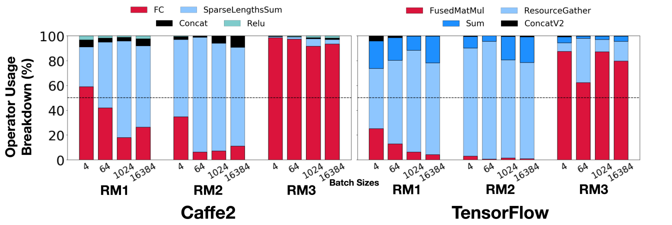


Fig. 7: **Comparison of Caffe2 and TensorFlow operator breakdowns** for DLRM-based recommendation models. Operators that comprise the majority of execution time are similar across both frameworks. Note, embedding table operations correspond to `SparseLengthsSum` in Caffe2 and the combination of `ResourceGather` and `Sum` in TensorFlow.

important as for latency-critical applications with strict SLA targets, input samples must run at small batch sizes.

5) GPU speedup over CPU is limited by data communication overheads. Figure 4 quantifies the fraction of time spent on data communication for different models and batch sizes. Data communication overheads come from offloading both continuous and categorical inputs via PCIe. For all models, the fraction of time spent on data communication scales with batch size as compute operations are readily accelerated (sub-linear) but data communication is not. Exact percentage of time spent on data-communication still depends on the model architecture; models that rely on embedding lookups suffer most. Given the high data-communication overheads, we conclude that running recommendation models out of the box on GPUs underutilizes the GPUs' compute resources.

Figure 5 summarizes the results of Section IV by showing how the optimal system platform (color coded) and speedup (number inside cell) vary across the use cases (models across the rows and batch sizes across the columns). While this section provides a high level intuition on the tradeoffs between CPUs and GPUs, the following sections dive deeper into the heterogeneity behind Figure 5.

V. Algorithms and Software Characterization

To better understand the system performance trends, in this section, we provide an algorithmic characterization of the different models and use cases. More specifically, we breakdown Caffe2 operators' usage for inference across the eight models and batch sizes. Furthermore, we show that the algorithmic bottlenecks are consistent across deep-learning frameworks (i.e., Caffe2 and Tensorflow), demonstrating that the performance trends are fundamental to model architectures.

A. Operator Breakdown

Operator usage breakdowns allow us to quantify the importance of each operator and compare them across model architectures in a *unified* manner. This is extremely important as model architectures for deep recommendation systems are rapidly evolving. In fact, these model architectures often mirror recent advances in deep learning [19]. Figure 6 shows the operator breakdown of the eight models – implemented in Caffe2 – across four different batch sizes on Broadwell, Cascade Lake, GTX 1080 Ti, and T4 machines. We see that different generations of hardware (top versus bottom rows) and classes of hardware (left versus right columns) alter operator usage breakdown. The important observations are:

1) GPUs accelerate models dominated by `FC` operators on CPUs but struggle with those bottlenecked by `SparseLengthsSum` on CPUs. Since GPUs contain large arrays of SMs that execute matrix multiplication efficiently, models with `FC`-dominated runtimes show the most acceleration (see: NCF, RM3, WnD, and MT-WnD). On the other hand, models with CPU runtimes bottlenecked by the `SparseLengthsSum` operator do not perform as well on GPUs. The `SparseLengthsSum` operator itself consists of both looking up a specified number of embedding vectors from each table and a subsequent partial sum. This becomes an issue for GPUs when the number of lookups per table and the table counts increase, leading to irregular memory access patterns.

2) In addition to the impact of varying model architecture, batching inference requests across different hardware

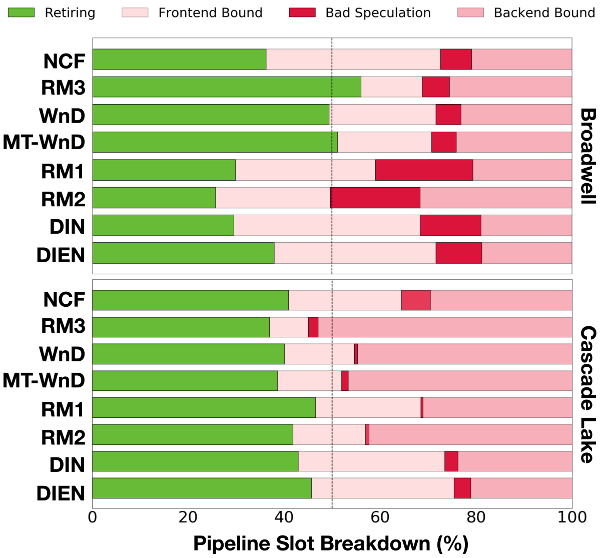


Fig. 8: **TopDown pipeline slot breakdowns. (Top)** On Broadwell, models that rely on matrix operations (i.e., NCF, RM3, WnD, MT-WnD), fill most of their pipeline slots with retiring instructions; the remaining models are either frontend bound or backend bound. **(Bottom)** On Cascade Lake, models with large FC components (i.e., RM3, WnD, MT-WnD) have fewer retiring pipeline slots due to wider SIMD width. The remaining models exhibit an increase in retiring mainly due to a reduction in bad speculation pipeline slots.

platforms uncovers additional opportunities for hardware optimization. Previous work has categorized recommendation models into three types: *MLP*-, *Embedding*-, or *Attention*-dominated models based on using a Broadwell CPU at a fixed batch size of 64 [15]. While this offers an efficient grouping for high-level discussions, analyzing operator breakdowns across all possible use cases reveals even more optimization points for designing future hardware. For example, on RM1, varying batch sizes from 4 to 64 will shift the dominant operator bottleneck from `FC` to `SparseLengthsSum`. Classifying the recommendation models based on their GPU performance also leads to different conclusions. For example, WnD, an FC-heavy model on CPUs, is dominated by the `SparseLengthsSum` operator at small batch sizes on GPUs. Identifying these shifting bottlenecks is important in order to thoroughly explore optimization opportunities. Designing efficient hardware that specializes for low-latency targets (i.e., smaller batch sizes), high-throughput (i.e., larger batch sizes), or other specific cases will require revisiting the operator breakdowns at target use cases.

3) Different generations of the same platform type (i.e., CPU/GPU) affect exact operator usages but retain general trends. On the left subplots of Figure 6 are Broadwell and Cascade Lake breakdowns and on the right are GTX 1080 Ti and T4 breakdowns. Inter-generation microarchitectural changes (i.e. Broadwell to Cascade Lake) affect operator breakdowns (e.g., for RM1 and RM2, time spent on FC layers is reduced) – Section VI goes more in depth on how microarchitectural differences lead to this.

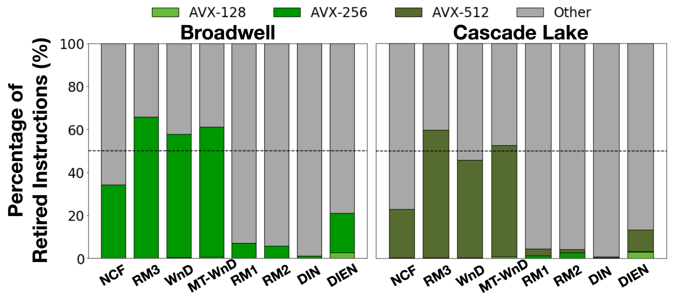


Fig. 9: **Instruction vectorization.** (**Left**) Broadwell AVX instructions constitute over 60% of retired instructions for models with larger FC layers (i.e., RM3, WnD, MT-WnD). (**Right**) Cascade Lake's wider SIMD-width results in shorter execution time despite reduced AVX instruction footprint.

B. Effects of Different Deep Learning Frameworks

Figure 7 compares operator breakdowns between Caffe2 and TensorFlow for DLRM-based models. As the operator breakdowns are similar, we know the optimization targets will be, to first order, the same regardless of differences in software frameworks. The mapping of the operator responsible for FC stacks is straightforward: `FC` in Caffe2 maps to `FusedMatMul` in TensorFlow. However, the `SparseLengthsSum` operator in Caffe2 maps to the combination of `ResourceGather` (lookup) and `Sum` (pool) operators in TensorFlow.

VI. CPU Microarchitectural Characterization

Complementing the operator breakdowns, in this section we present a detailed CPU microarchitectural characterization that provides additional insights into the performance trends for recommendation inference. In order to better understand the architectural bottlenecks in general purpose processors, we use TopDown-based performance measurement unit (PMU) analysis for server-class Broadwell and Cascade Lake CPUs (Table II) [21]. This analysis shows the important microarchitectural components that form the performance bottlenecks for different recommendation models on Broadwell and how the bottlenecks change for Cascade Lake CPUs.

A. TopDown analysis

Following TopDown performance analysis [21], we break down the CPU pipeline into four major portions: frontend, speculation, backend, and retiring. The frontend fetches instructions from memory and converts them into micro-operations (μops); speculation realizes predictive optimizations; backend schedules and executes the μops; retiring commits the μops. In order to optimize the performance of a processor we must maximize instructions per cycle (IPC). Generally, IPC can be improved by increasing the fraction of processor cycles devoted to retiring as opposed to stalled in the frontend, speculation, or backend portions. Recent work uses TopDown analysis to better understand the fraction of cycles in

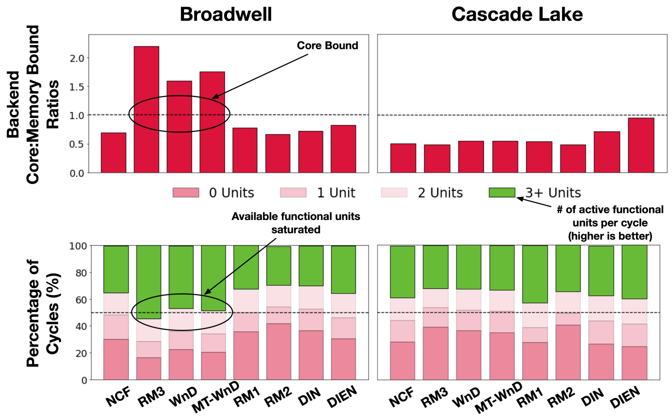


Fig. 10: (**Top**) **Ratio of Core:Memory Backend Bound cycles**. Majority of stalls come from functional units on Broadwell and from memory subsystem on Cascade Lake. (**Bottom**) **Functional unit usage**. RM3, WnD, and MT-WnD saturate Broadwell's functional units more than other models with a large fraction of cycles that use 3+ units out of 8. Cascade Lake decreases the pressure on functional units.

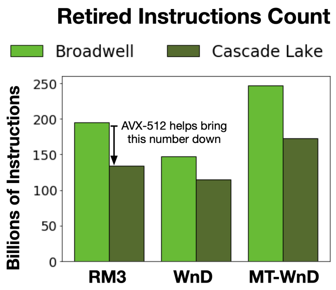


Fig. 11: **Retired Instructions Count** decreases from Broadwell to Cascade Lake due to the introduction of the more efficient AVX-512 VNNI instructions.

each pipeline portion for server and datacenter workloads [21], [24], [25].

B. Recommendation performance using TopDown

Figure 8 shows the TopDown breakdown of the eight deep recommendation models, with a batch-size of 16, on Broadwell and Cascade Lake CPUs. Generally, on Broadwell, models with larger FC layers (i.e., RM3, WnD, and MT-WND) spend the majority of their cycles in retiring. On the other hand, the remaining models (i.e., NCF, RM1, RM2, DIN, and DIEN) suffer from a variety of frontend, backend, and bad speculation bottlenecks. Following are notable observations:

1) On Broadwell, larger FC-dominated models benefit from vector execution but remain limited with insufficient functional units. On Broadwell, models that rely on FC layers (i.e., NCF, RM3, WnD, MT-WND) spend a large percentage of pipeline slots on retiring instructions. Thus, the natural next step would be to investigate the *degree* of instruction vectorization for these models (Figure 9). On Broadwell, over 60% of all retired instructions for RM3, WnD, and MT-WnD are Advanced Vector Instructions (AVX) (Figure 9 (Left)). This is a result of machine learning frameworks, like Caffe2, translating FC layers to vectorized matrix operations.

Despite this high degree of vectorization, the larger FC-dominant models still spend a significant fraction of pipeline slots backend bound – highlighting the need for improved CPU backend pipelines for faster μops consumption. Backend bound cycles can be further classified as either core bound or memory bound. Figure 10 (Left) quantifies the core-bound nature of these backend-bound models on a Broadwell machine in two ways. The top row shows the breakdown of the backend bound slots as a core:memory bound ratio, where RM3, WnD, and MT-WnD all show numbers > 1. In the case of RM3, where the ratio ~ 2, there are twice as many functional units-induced stall cycles as memory subsystem-induced stall cycles. We see that for WnD and MT-WnD, this ratio is > 1.5. Thus, despite wide vector execution, larger FC-dominant models remain backend core-bound on Broadwell machines.

Figure 10 (Bottom) details the cycle-level utilization of functional units. Recall that Broadwell CPUs have eight functional units: four arithmetic units, two load units, and two store units. Figure 10 (left, bottom) shows nearly 50% of cycles in RM3, WnD, and MT-WnD require more than three functional units: this high functional unit utilization underscores the core-bound bottleneck. This illustration of the core-bound bottleneck also corroborates the source of GPU speedups for RM3, WnD, and MT-WnD. Since these models are bottlenecked by the lack of more functional units, the increase in the amount of compute units (streaming multiprocessors) on GPUs alleviates this core bound issue on Broadwell.

2) On Cascade Lake, larger FC-dominated models benefit from wider SIMD width and compute capabilities, shifting the bottleneck to the memory subsystem. Figure 8 (bottom) shows the TopDown analysis of the eight recommendation models on Cascade Lake CPUs. In comparison to Broadwell, Cascade Lake enables the majority of models (e.g., NCF, RM1, RM2, DIN, DIEN) to have a larger fraction of retiring pipeline slots. This larger fraction of cycles spent on retiring instruction is the main reason why Cascade Lake provides consistent speedup over Broadwell (see Figure 3). Note that the fraction of cycles devoted to the retiring stage did not increase between Broadwell to Cascade Lake for RM3, WnD, and MT-WnD. The slight decrease in the retiring cycles is due to fewer total dynamic instructions, as shown in Figure 11; overall, the wider width AVX-512 Vector Neural Network Instructions (VNNI) improves performance for larger FC-dominated models.

Recall that RM3, WnD, and MT-WnD are core bound on Broadwell. Figure 10 (right) shows the backend TopDown analysis on Cascade Lake. Given the wider AVX512-VNNI instructions, Cascade Lake implements more sophisticated fused multiply-add hardware, which increases the compute capability of the processor. The increased compute capability reduces pressure on the functional units as shown in Figure 10 (bottom, right). Despite the reduced execution port utilization,

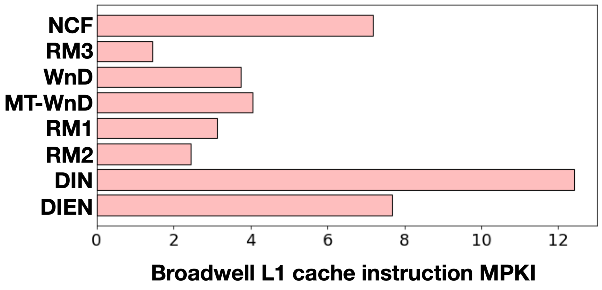


Fig. 12: **NCF and attention-based models suffer from instruction cache misses.** NCF's small size shifts the bottleneck from execution units to i-cache. Attention-based models scale each embedding vector (from irregular memory accesses) with individual weights, leading to high i-cache MPKI.

inference performance for RM3, WnD, and MT-WnD remains backend bound on Cascade Lake. As shown in Figure 10 (upper, right), the backend bottleneck has shifted from being core-bound to memory bound. The particular memory subsystem limiting performance depends on the input batch size – smaller batch sizes (i.e., less than 100) are limited by L3 cache accesses while, larger batch-sizes are limited by DRAM latency.

3) Smaller FC-dominant models and attention-based models suffer from frontend latency – especially L1 instruction cache latency. Not all FC-dominant models are core-bound. For example, on Broadwell, NCF suffers from frontend latency bottlenecks and in particular, L1 instruction cache latency. Because of its relatively small FC layers, NCF does not exhibit core-bound levels of high compute intensity.

To understand these frontend limitations, Figure 12 quantifies the L1 instruction cache miss rate. NCF, along with attention-based models like DIN and DIEN, have higher L1 instruction cache miss rates compared to the remaining models. For instance, we measure a L1 instruction misses per thousand instructions (i-MPKI) of 12.4 and 7.7 for DIN and DIEN, respectively. The high instruction cache miss rates are tied to how DIN and DIEN implement attention. Recall that for recommendation systems, attention allows networks to individually weight the importance of embedding vectors to offer higher personalization. In DIN, attention is implemented using hundreds of local concatenation and FC layers; this leads to a large number of instructions with unique reference locations (since the instruction cache does not cache opcodes but specific instructions, including the reference operand). Given the unique memory addresses for embedding table lookups, the instruction cache hit rate suffers from irregular memory accesses (i.e., lack of spatial and temporal locality). DIEN's GRU implementation more efficiently translates to matrix operations compared to DIN's implementation with local concatenation-FC *per lookup*. This offers cache friendly loops with regular operand and reference locations.

4) Models with more embedding table lookups suffer from instruction decoder bottlenecks. As the degree of embedding table lookups increases, the performance bottleneck shifts from pure decoder issues to also include DRAM bandwidth limitations. On Broadwell, RM1 and RM2 are frontend bandwidth-bound deep recommendation models. Generally, this denotes inefficiencies in the instruction decode phase as opposed to in the instruction fetch phase. Figure 13 illustrates the fraction of cycles spent on two parts of Broadwell's frontend decoder pipeline, the decoded i-cache (Decoded Stream Buffer - DSB) and the legacy decoder pipeline (Micro-Instruction Translation Engine - MITE). MITE is responsible for fetching instructions from instruction memory and decoding them into μops while the DSB caches results from MITE. For each target instruction, DSB is first queried. If the instruction is found in the DSB, the corresponding μops are directly delivered to the instruction decode queue (IDQ). If the instruction is not found, MITE is used to fetch and decode instructions and the result is added into DSB.

Figure 13 (bottom) shows Broadwell's decoder pipeline

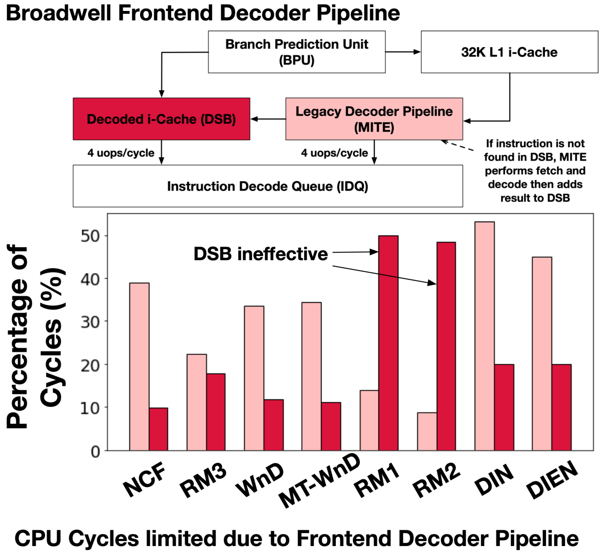


Fig. 13: **Frontend Decoder Pipeline Inefficiencies.** The two main decoder microarchitecture components are DSB and MITE. Shown are percent of cycles in which the CPU was limited by a specific decoder component (i.e., component was not supplying IDQ with optimal number of decoded instructions).

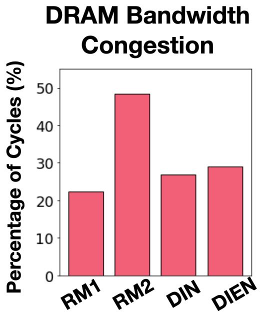


Fig. 14: **RM2 also suffers from DRAM Bandwidth Congestion** from the large number of embedding lookups.

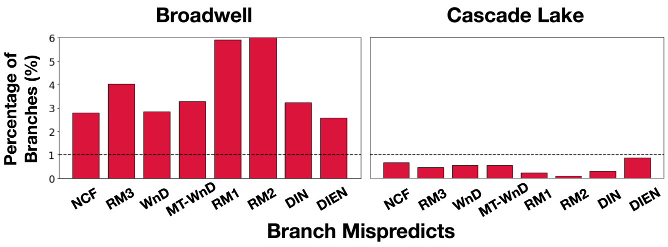


Fig. 15: **Branch Mispredicts** decrease significantly when we transition from Broadwell to Cascade Lake machines.

– including DSB and MITE. CPU cycles are analyzed to determine if either DSB or MITE could not supply IDQ with sufficient μops. For both RM1 and RM2, the frontend bandwidth bound models, TopDown analysis illustrates the bottlenecks in DSB as the main source of inefficiency.

The DSB bottleneck can be tied to algorithmic, model-architecture features of RM1 and RM2. In particular, both models require a high degree, tens to hundreds, of embedding table lookups. Combined with the irregular memory accesses coming from embedding lookups, larger instruction footprints stress the DSB. Furthermore, RM1 and RM2 spend a large fraction of their cycles on bad speculation as shown in Figure 8. Since DSB is also affected by the Branch Prediction Unit (BPU), the large amount of branch misprediction latency will degrade the performance of DSB, as the speculation stalls are primarily from branch mispredictions.

Despite the similarities between RM1 and RM2, RM2 has a unique performance bottleneck. As shown in Table I, RM2 comprises more embedding tables (32 versus 8 in RM1) and more lookups per table (120 versus 80 in RM1). Given the larger size, RM2 suffers from bottlenecks in both the frontend and backend pipeline. Figure 14 illustrates the DRAM bandwidth congestion of RM1, RM2, DIN, and DIEN. DRAM bandwidth congestion, as defined by Intel, occurs when the offcore read queue occupancy exceeds 70% of the maximum number of requests that can be served by the memory controller simultaneously; whereas when below 70% occupancy, the stall can be characterized as DRAM latency bound [26]. We find that RM2 suffers from significantly higher DRAM bandwidth congestion limitations compared to the other models. Previous work exploit this property to design near memory processing solutions for DRAM bandwidth-bound recommendation models [16], [17].

5) Cascade Lake significantly reduces the amount of pipeline slots lost to bad speculation. One of the marked differences between the TopDown breakdowns of Broadwell and Cascade Lake in Figure 8 is the decrease in pipeline slots lost to bad speculation in Cascade Lake. While the specific detail of the branch predictor designs used in Broadwell and Cascade Lake are not available, the transition from Broadwell to Skylake sees a penalty reduction for incorrect direct jump target [27]. This overall improvement shifts the Cascade Lake backend bottlenecks to the memory subsystems as discussed in Observation #2.

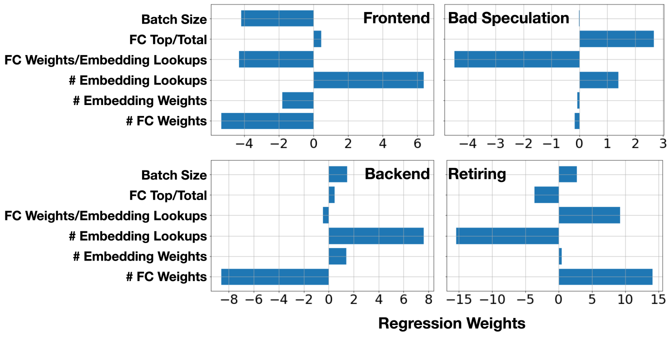


Fig. 16: **Linear Regression modeling** of algorithmic model architecture components and pipeline bottlenecks reveals that there is not a single deciding factor for each bottleneck.

C. *Tying Model Architectures to Pipeline Bottlenecks*

To tie our microarchitectural observations to the specifics of recommendation model architectures, we quantify the effects of select algorithmic model architecture features with a linear regression model.

Figure 16 summarizes our linear regression modeling; all input features have been normalized so the weight magnitude represents degree of impact. Data points are collected from running the 8 models at batch sizes from 1 to 16384. The model shows that each pipeline bottleneck is a result of a combination of different algorithmic model architecture features. For example, this model shows that a high ratio of FC to embedding weights reduces bad speculation while a top-heavy distribution of FC weights leads to increases in bad speculation. The first point explains the intuition that compute-intensive models have more predictable branches while the second point shows that more direct processing of continuous inputs is correlated with less bad speculation.

VII. Related Work

Analysis and optimizations for recommendation systems. Recommendation systems have recently come under the spotlight for computer systems researchers. As mentioned earlier, a few recent works explore near-memory processing techniques for recommendation models dominated by table lookup operations. TensorDimm evaluates near-data processing enabled custom DIMM modules on recommendation models similar to RM1-3 [16]; RecNMP evaluates a set of techniques centered around memory-side caching on production-representative embedding traces [17]. Ginart et al. and Shi et al. [28], [29] compresses embedding tables in recommendation models while maintaining the model accuracy. Centaur extends near-memory processing designs to also account for the MLP layers through a chiplet-based accelerator design [18]. Other works have explored at-scale optimizations [10]: DeepRecSys explores different optimizations at the datacenter scale; the recommendation suite evaluated throughout this paper is from DeepRecSys's open sourced implementations [15]. Other work has started to explore implications of training [11], [30]. In contrast, this paper focuses on purely characterizing the recommendation suite introduced in [15]. To the best of our knowledge, this is the first detailed microarchitectural characterization of deep recommendation systems.

DNN benchmarks and accelerator designs. Current benchmarks and characterizations for DNNs primarily focus on FC, CNNs, and RNNs [20], [31]–[35]. Building upon the performance bottlenecks derived from these studies, a variety of hardware solutions have been proposed to optimize for traditional DNNs [36]–[59]. While these DNNs share the operators introduced in Section V, recommendation models present them in unique ratios and model architecture organizations.

VIII. Conclusion

It is important to characterize deep recommendation models across different layers in the execution stack because this helps us better understand the bottlenecks that arise from our evaluations. By understanding more about these bottlenecks and how they realize themselves at different levels (i.e., as operators in Caffe2 and as inefficiencies of different CPU components), we can intelligently design future hardware that optimizes for deep recommendation inference.

IX. Acknowledgements

We would like to thank the anonymous reviewers for their thoughtful comments and suggestions. We would also like to thank Glenn Holloway and Emma Wang for their valuable feedback. This work was sponsored in part by NSF CCF-1533737 and a National Science Foundation Graduate Research Fellowship (NSFGRFP).

References

[1] G. Linden, B. Smith, and J. York, “Amazon.com recommendations: item-to-item collaborative filtering,” *IEEE Internet Computing*, vol. 7, no. 1, pp. 76–80, 2003.

[2] G. Zhou, X. Zhu, C. Song, Y. Fan, H. Zhu, X. Ma, Y. Yan, J. Jin, H. Li, and K. Gai, “Deep interest network for click-through rate prediction,” in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 2018, pp. 1059–1068.

[3] G. Zhou, N. Mou, Y. Fan, Q. Pi, W. Bian, C. Zhou, X. Zhu, and K. Gai, “Deep interest evolution network for click-through rate prediction,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 5941–5948.

[4] M. Naumov, D. Mudigere, H. M. Shi, J. Huang, N. Sundaraman, J. Park, X. Wang, U. Gupta, C. Wu, A. G. Azzolini, D. Dzhulgakov, A. Mallevich, I. Cherniavskii, Y. Lu, R. Krishnamoorthi, A. Yu, V. Kondratenko, S. Pereira, X. Chen, W. Chen, V. Rao, B. Jia, L. Xiong, and M. Smelyanskiy, “Deep learning recommendation model for personalization and recommendation systems,” *CoRR*, vol. abs/1906.00091, 2019. [Online]. Available: http://arxiv.org/abs/1906.00091

[5] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T.-S. Chua, “Neural collaborative filtering,” in *Proceedings of the 26th International Conference on World Wide Web*, ser. WWW ’17. Republic and Canton of Geneva, Switzerland: International World Wide Web Conferences Steering Committee, 2017, pp. 173–182. [Online]. Available: https://doi.org/10.1145/3038912.3052569

[6] Z. Zhao, L. Hong, L. Wei, J. Chen, A. Nath, S. Andrews, A. Kumthekar, M. Sathiamoorthy, X. Yi, and E. Chi, “Recommending what video to watch next: A multitask ranking system,” in *Proceedings of the 13th ACM Conference on Recommender Systems*, ser. RecSys ’19. New York, NY, USA: ACM, 2019, pp. 43–51. [Online]. Available: http://doi.acm.org/10.1145/3298689.3346997

[7] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, “Item-based collaborative filtering recommendation algorithms,” in *Proceedings of the 10th International Conference on World Wide Web*, ser. WWW ’01. ACM, 2001, pp. 285–295.

[8] “Netflix update: Try this at home,” https://sifter.org/simon/journal/20061211.html.

[9] K. Hazelwood, S. Bird, D. Brooks, S. Chintala, U. Diril, D. Dzhulgakov, M. Fawzy, B. Jia, Y. Jia, A. Kalro, J. Law, K. Lee, J. Lu, P. Noordhuis, M. Smelyanskiy, L. Xiong, and X. Wang, “Applied machine learning at facebook: A datacenter infrastructure perspective,” in *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, Feb 2018, pp. 620–629.

[10] U. Gupta, C. Wu, X. Wang, M. Naumov, B. Reagen, D. Brooks, B. Cottel, K. M. Hazelwood, M. Hempstead, B. Jia, H. S. Lee, A. Malevich, D. Mudigere, M. Smelyanskiy, L. Xiong, and X. Zhang, “The architectural implications of facebook’s dnn-based personalized recommendation,” in *IEEE International Symposium on High Performance Computer Architecture, HPCA 2020, San Diego, CA, USA, February 22-26, 2020*. IEEE, 2020, pp. 488–501. [Online]. Available: https://doi.org/10.1109/HPCA47549.2020.00047

[11] M. Naumov, J. Kim, D. Mudigere, S. Sridharan, X. Wang, W. Zhao, S. Yilmaz, C. Kim, H. Yuen, M. Ozdal, K. Nair, I. Gao, B.-Y. Su, J. Yang, and M. Smelyanskiy, “Deep learning training in facebook data centers: Design of scale-up and scale-out systems,” 2020.

[12] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers *et al.*, “In-datacenter performance analysis of a tensor processing unit,” in *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2017, pp. 1–12.

[13] M. Chui, J. Manyika, M. Miremadi, N. Henke, R. Chung, P. Nel, and S. Malhotra, “Notes from the ai frontier insights from hundreds of use cases,” 2018.

[14] W. Zhao, J. Zhang, D. Xie, Y. Qian, R. Jia, and P. Li, “Aibox: Ctr prediction model training on a single node,” in *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, ser. CIKM ’19. New York, NY, USA: Association for Computing Machinery, 2019, p. 319–328. [Online]. Available: https://doi.org/10.1145/3357384.3358045

[15] 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,” 2020.

[16] Y. Kwon, Y. Lee, and M. Rhu, “Tensordimm: A practical near-memory processing architecture for embeddings and tensor operations in deep learning,” in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO ’52. New York, NY, USA: Association for Computing Machinery, 2019, p. 740–753. [Online]. Available: https://doi.org/10.1145/3352460.3358284

[17] L. Ke, U. Gupta, C.-J. Wu, B. Y. Cho, M. Hempstead, B. Reagen, X. Zhang, D. Brooks, V. Chandra, U. Diril, A. Firoozshahian, K. Hazelwood, B. Jia, H.-H. S. Lee, M. Li, B. Maher, D. Mudigere, M. Naumov, M. Schatz, M. Smelyanskiy, and X. Wang, “Recnmp: Accelerating personalized recommendation with near-memory processing,” 2019.

[18] R. Hwang, T. Kim, Y. Kwon, and M. Rhu, “Centaur: A chiplet-based, hybrid sparse-dense accelerator for personalized recommendations,” 2020.

[19] S. Zhang, L. Yao, A. Sun, and Y. Tay, “Deep learning based recommender system: A survey and new perspectives,” *ACM Comput. Surv.*, vol. 52, no. 1, Feb. 2019. [Online]. Available: https://doi.org/10.1145/3285029

[20] C.-J. Wu, R. Burke, E. H. Chi, J. Konstan, J. McAuley, Y. Raimond, and H. Zhang, “Developing a recommendation benchmark for mlperf training and inference,” 2020.

[21] A. Yasin, “A top-down method for performance analysis and counters architecture,” in *2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2014, pp. 35–44.

[22] U. Gupta, X. Wang, M. Naumov, C.-J. Wu, B. Reagen, D. Brooks, B. Cottel, K. Hazelwood, B. Jia, H.-H. S. Lee *et al.*, “The architectural implications of facebook’s dnn-based personalized recommendation,” *arXiv preprint arXiv:1906.03109*, 2019.

[23] H.-T. Cheng, L. Koc, J. Harmsen, T. Shaked, T. Chandra, H. Aradhye, G. Anderson, G. Corrado, W. Chai, M. Ispir *et al.*, “Wide & deep learning for recommender systems,” in *Proceedings of the 1st workshop on deep learning for recommender systems*. ACM, 2016, pp. 7–10.

[24] S. Kanev, J. P. Darago, K. Hazelwood, P. Ranganathan, T. Moseley, G.-Y. Wei, and D. Brooks, "Profiling a warehouse-scale computer," in *Proceedings of the 42nd Annual International Symposium on Computer Architecture*, ser. ISCA '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 158–169. [Online]. Available: https://doi.org/10.1145/2749469.2750392

[25] A. Sriraman, A. Dhanotia, and T. F. Wenisch, "Softsku: Optimizing server architectures for microservice diversity @scale," in *2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA)*, 2019, pp. 513–526.

[26] Intel, "Intel® 64 and ia-32 architectures optimization reference manual," 2020.

[27] A. Fog, "The microarchitecture of intel, amd and via cpus: An optimization guide for assembly programmers and compiler makers," 2020. [Online]. Available: https://www.agner.org/optimize/microarchitecture.pdf

[28] A. Ginart, M. Naumov, D. Mudigere, J. Yang, and J. Zou, "Mixed dimension embeddings with application to memory-efficient recommendation systems," *arXiv preprint arXiv:1909.11810*, 2019.

[29] H.-J. M. Shi, D. Mudigere, M. Naumov, and J. Yang, "Compositional embeddings using complementary partitions for memory-efficient recommendation systems," *arXiv preprint arXiv:1909.02107*, 2019.

[30] D. Kalamkar, E. Georganas, S. Srinivasan, J. Chen, M. Shiryaev, and A. Heinecke, "Optimizing deep learning recommender systems' training on cpu cluster architectures," 2020.

[31] Robert Adolf, Saketh Rama, Brandon Reagen, Gu-Yeon Wei, and David Brooks, "Fathom: Reference workloads for modern deep learning methods," ser. IISWC'16, 2016. [Online]. Available: http://vlsiarch.eecs.harvard.edu/wp-content/uploads/2016/08/iiswc2016-final.pdf

[32] E. Wang, G.-Y. Wei, and D. Brooks, "Benchmarking tpu, gpu, and cpu platforms for deep learning," *arXiv preprint arXiv:1907.10701*, 2019.

[33] C. Coleman, D. Narayanan, D. Kang, T. Zhao, J. Zhang, L. Nardi, P. Bailis, K. Olukotun, C. Ré, and M. Zaharia, "Dawnbench: An end-to-end deep learning benchmark and competition."

[34] H. Zhu, M. Akrout, B. Zheng, A. Pelegris, A. Jayarajan, A. Phanishayee, B. Schroeder, and G. Pekhimenko, "Benchmarking and analyzing deep neural network training," in *Proceedings of the IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 2018, pp. 88–100.

[35] "A broad ml benchmark suite for measuring performance of ml software frameworks, ml hardware accelerators, and ml cloud platforms," https://mlperf.org/, 2019.

[36] A. Parashar, M. Rhu, A. Mukkara, A. Puglielli, R. Venkatesan, B. Khailany, J. S. Emer, S. W. Keckler, and W. J. Dally, "SCNN: an accelerator for compressed-sparse convolutional neural networks," *CoRR*, vol. abs/1708.04485, 2017. [Online]. Available: http://arxiv.org/abs/1708.04485

[37] M. Rhu, N. Gimelshein, J. Clemons, A. Zulfiqar, and S. W. Keckler, "vdnn: Virtualized deep neural networks for scalable, memory-efficient neural network design," in *The 49th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Press, 2016, p. 18.

[38] Y. Kwon and M. Rhu, "Beyond the memory wall: A case for memory-centric hpc system for deep learning," in *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2018, pp. 148–161.

[39] Y. Choi and M. Rhu, "Prema: A predictive multi-task scheduling algorithm for preemptible neural processing units," *arXiv preprint arXiv:1909.04548*, 2019.

[40] Udit Gupta, Brandon Reagen, Lillian Pentecost, Marco Donato, Thierry Tambe, Alexander M. Rush, Gu-Yeon Wei, David Brooks, "MASR: A modular accelerator for sparse rnns," in *International Conference on Parallel Architectures and Compilation Techniques*, 2019.

[41] Y.-H. Chen, T. Krishna, J. Emer, and V. Sze, "Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks," in *ISSCC*, 2016.

[42] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "EIE: efficient inference engine on compressed deep neural network," *CoRR*, vol. abs/1602.01528, 2016. [Online]. Available: http://arxiv.org/abs/1602.01528

[43] B. Reagen, P. Whatmough, R. Adolf, S. Rama, H. Lee, S. K. Lee, J. M. Hernandez-Lobato, G.-Y. Wei, and D. Brooks, "Minerva: Enabling Low-Power, Highly-Accurate Deep Neural Network Accelerators," in *ISCA*, 2016.

[44] F. Silfa, G. Dot, J.-M. Arnau, and A. Gonzàlez, "E-pur: An energy-efficient processing unit for recurrent neural networks," in *Proceedings of the 27th International Conference on Parallel Architectures and Compilation Techniques*, ser. PACT '18. ACM, 2018, pp. 18:1–18:12.

[45] K. Hegde, R. Agrawal, Y. Yao, and C. W. Fletcher, "Morph: Flexible acceleration for 3d cnn-based video understanding," in *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Oct 2018, pp. 933–946.

[46] K. Hegde, H. Asghari-Moghaddam, M. Pellauer, N. Crago, A. Jaleel, E. Solomonik, J. Emer, and C. W. Fletcher, "Extensor: An accelerator for sparse tensor algebra," in *Proceedings of the 52Nd Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '52. ACM, 2019, pp. 319–333.

[47] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun, and O. Teman, "Dadiannao: A machine-learning supercomputer," in *MICRO*, 2014.

[48] S. Zhang, Z. Du, L. Zhang, H. Lan, S. Liu, L. Li, Q. Guo, T. Chen, and Y. Chen, "Cambricon-X: An accelerator for sparse neural networks," in *49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Oct 2016, pp. 1–12.

[49] H. Sharma, J. Park, E. Amaro, B. Thwaites, P. Kotha, A. Gupta, J. K. Kim, A. Mishra, and H. Esmaeilzadeh, "Dnnweaver: From high-level deep network models to fpga acceleration."

[50] Z. Du, R. Fasthuber, T. Chen, P. Ienne, L. Li, T. Luo, X. Feng, Y. Chen, and O. Temam, "Shidiannao: Shifting vision processing closer to the sensor," in *ACM SIGARCH Computer Architecture News*, vol. 43, no. 3. ACM, 2015, pp. 92–104.

[51] D. Liu, T. Chen, S. Liu, J. Zhou, S. Zhou, O. Teman, X. Feng, X. Zhou, and Y. Chen, "Pudiannao: A polyvalent machine learning accelerator," in *ACM SIGARCH Computer Architecture News*, vol. 43, no. 1. ACM, 2015, pp. 369–381.

[52] P. Chi, S. Li, C. Xu, T. Zhang, J. Zhao, Y. Liu, Y. Wang, and Y. Xie, "Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory," in *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3. IEEE Press, 2016, pp. 27–39.

[53] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. P. Strachan, M. Hu, R. S. Williams, and V. Srikumar, "Isaac: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 14–26, 2016.

[54] D. Kim, J. Kung, S. Chai, S. Yalamanchili, and S. Mukhopadhyay, "Neurocube: A programmable digital neuromorphic architecture with high-density 3d memory," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2016, pp. 380–392.

[55] R. LiKamWa, Y. Hou, J. Gao, M. Polansky, and L. Zhong, "Redeye: analog convnet image sensor architecture for continuous mobile vision," in *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3. IEEE Press, 2016, pp. 255–266.

[56] D. Mahajan, J. Park, E. Amaro, H. Sharma, A. Yazdanbakhsh, J. K. Kim, and H. Esmaeilzadeh, "Tabla: A unified template-based framework for accelerating statistical machine learning," in *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2016, pp. 14–26.

[57] M. Gao, J. Pu, X. Yang, M. Horowitz, and C. Kozyrakis, "Tetris: Scalable and efficient neural network acceleration with 3d memory," in *ACM SIGARCH Computer Architecture News*, vol. 45, no. 1. ACM, 2017, pp. 751–764.

[58] L. Pentecost, M. Donato, B. Reagen, U. Gupta, S. Ma, G.-Y. Wei, and D. Brooks, "Maxnvm: Maximizing dnn storage density and inference efficiency with sparse encoding and error mitigation," in *Proceedings of the 52Nd Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '52. New York, NY, USA: ACM, 2019, pp. 769–781. [Online]. Available: http://doi.acm.org/10.1145/3352460.3358258

[59] J. Albericio, A. Delmás, P. Judd, S. Sharify, G. O'Leary, R. Genov, and A. Moshovos, "Bit-pragmatic deep neural network computing," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 2017, pp. 382–394.