ENMC: Extreme Near-Memory Classification via Approximate Screening

Liu Liu*
liu_liu@ucsb.edu
UC Santa Barbara
USA

Jilan Lin*
jilan@ucsb.edu
UC Santa Barbara
USA

Zheng Qu
zhengqu@ucsb.edu
UC Santa Barbara
USA

Yufei Ding
yufeiding@cs.ucsb.edu
UC Santa Barbara
USA

Yuan Xie
yuanxie@ece.ucsb.edu
UC Santa Barbara
USA

ABSTRACT

Extreme classification (XC) is the essential component of large-scale Deep Learning Systems for a wide range of application domains, including image recognition, language modeling, and recommendation. As classification categories keep scaling in real-world applications, the classifier's parameters could reach several thousands of Gigabytes, way exceed the on-chip memory capacity. With the advent of near-memory processing (NMP) architectures, offloading the XC component onto NMP units could alleviate the memory-intensive problem. However, naive NMP design with limited area and power budget cannot afford the computational complexity of full classification. To tackle the problem, we first propose a novel screening method to reduce the computation and memory consumption by efficiently approximating the classification output and identifying a small portion of key candidates that require accurate results. Then, we design a new extreme-classification-tailored NMP architecture, namely ENMC, to support both screening and candidates-only classification. Overall, our approximate screening method achieves 7.3× speedup over the CPU baseline, and ENMC further improves the performance by 7.4× and demonstrates 2.7× speedup compared with the state-of-the-art NMP baseline.

CCS CONCEPTS

• **Computer systems organization** → **Neural networks**; • **Hardware** → *Memory and dense storage.*

KEYWORDS

Near-memory processing, Extreme classification

ACM Reference Format:
Liu Liu, Jilan Lin, Zheng Qu, Yufei Ding, and Yuan Xie. 2021. ENMC: Extreme Near-Memory Classification via Approximate Screening. In *MICRO'21: 54th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO '21), October 18–22, 2021, Virtual Event, Greece.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3466752.3480090

*Both authors contributed equally to this research.

MICRO '21, October 18–22, 2021, Virtual Event, Greece

ACM ISBN 978-1-4503-8557-2/21/10.
https://doi.org/10.1145/3466752.3480090

1 INTRODUCTION

Recent advances in many machine intelligence areas, such as natural language processing (NLP) [30, 37, 40], image recognition [27, 39, 50], and recommendation[24, 35, 47], involve tackling the extreme classification problem, where classification category size is extreme large. For example, in the NLP domain, making predictions is basically classifying the words with high probabilities. Similarly, for image recognition tasks and recommendation tasks, the features generated from hidden neural network layers need to go through the classification layer to output predictions. As shown in Fig. 1, extreme classification is the essential component to deal with large-scale problems.

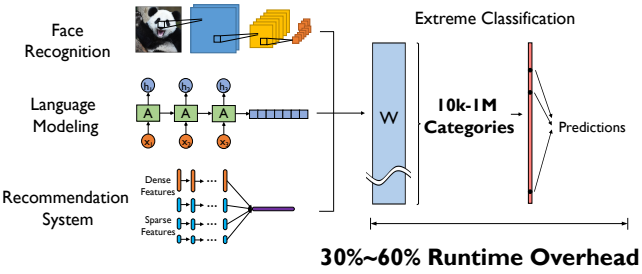

Figure 1: Extreme Classification serves as the common component of large-scale Deep Learning applications. The classifier processes with hidden representations from application-specific hidden layers and generates predictions as used in recognition, language, and recommendation.

As classification categories keep scaling in real-world applications, the classifier's parameters could reach hundreds of gigabytes, far beyond the on-chip memory capacity. For large-scale NLP models, the vocabulary sizes are in the range of hundreds of thousands, contributing hundreds of megabytes data [40, 42]. For recommendation systems, using commodity datasets to solve industry-level problems would require classification on the scale of 100M categories [27, 39], consuming around 190GB memory.

Due to the large memory footprint of extreme classification, accessing system memory for the classifier's weight data becomes

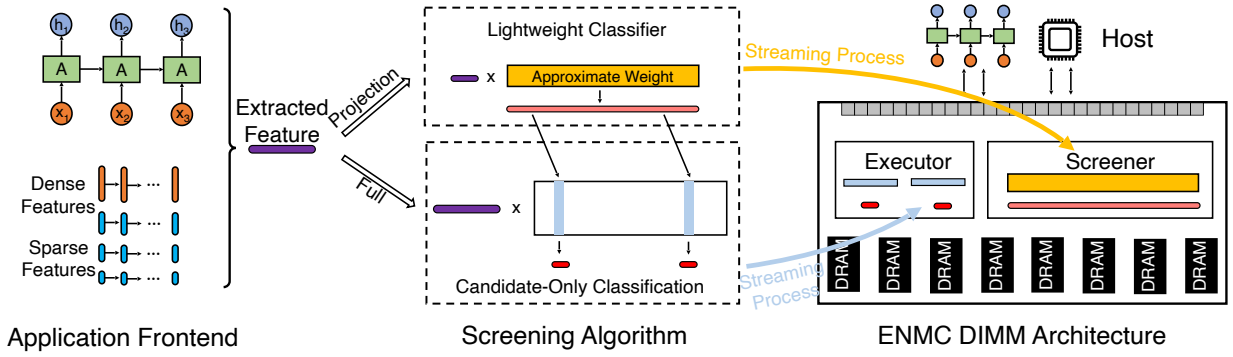

Figure 2: The overview of our Approximate Screening algorithm and NMP architecture co-design. Instead of full classification, our co-design essentially performs candidates-only classification, where the candidates are based on the screening method. Our NMP architecture design features a Screener and an Executor to collaboratively process candidates-only classification.

the bottleneck of system performance. We characterize the state-of-the-art Transformer-based language model [33] and show that the final classification layer consumes 50% of overall model inference time. While GPUs and specialized accelerators can boost the performance of DNN layers [6, 14], they suffer from inter-device data movements when executing the memory-intensive classification layer, as the memory usage exceeds device memory capacity.

Emerging Near-Memory Processing (NMP) technologies [15, 20] have the potential to address the memory-bound problem of extreme classification. However, naive NMP designs cannot support the computational complexity of full classification due to the area and power limitations. Even the classifier weight data are stored and processed near-memory, the low operational intensity of linear transformation, which is basically matrix-vector multiplication, is still causing performance degradation when accessing weight data from DRAM modules.

Therefore, we propose the first end-to-end solution to address the memory-bound problem of extreme classification with NMP architecture. Fig. 2 gives an overview of the proposed software and hardware co-design. To reduce the overhead of classification, we propose an approximate screening algorithm that directly reduces the required computations and data access involved in linear transformations. As demonstrated in Fig. 2, given the extracted feature vectors from the application front-end, a learned lightweight classifier firstly performs approximate classification to efficiently identify the set of important candidates in the category space. Afterwards, the classifier will trigger candidates-only computation to generate accurate classification results, while the rest can directly utilize the approximate results computed from the screening phase. Therefore, a large amount of computations and data loading of classification are saved. Our experiments (Section 7.1) show that the proposed screening method achieves better trade-off for classification accuracy and computation saving, compared with conventional low-rank approximation-based method [37].

To fully leverage the approximate screening method, we further propose the Extreme Near-Memory Classification architecture, namely *ENMC*. Here we highlight the key features of our ENMC design as follows: Firstly, as shown in Fig. 2, we deploy a dual-module architecture that contains a Screener module and an Executor module that run in parallel. The Screener performs approximate screening efficiently, as described in Section 5, and predicts the classification candidates in advance. For each candidate selected in a batch, the ENMC controller will generate instructions for accurate computations handled by the Executor. The computing modules are deployed at the rank level such that there is no need to invade the DRAM chips. Secondly, we design the ENMC instruction set to facilitate the workloads accommodation between host processor and ENMC, and support the communications between the Screener and the Executor. We define the instruction format by leveraging the reserved command space so that it is compatible with the commodity DDR interface. Thus, our ENMC DIMM can also support regular memory requests. Finally, we provide the system-level design, including application workflow and program compiler support, to make the ENMC architecture cooperate with the software framework. Our design could be easily extended no matter the host processors are CPUs, GPUs, or domain-specific accelerators.

Our contributions are as follows:

- We study extreme classification in different applications and identify the memory-bound problem (Section 3).
- We present the approximate screening method to significantly reduce the computational complexity of extreme classification by selecting key candidates and avoiding full classification (Section 4). The evaluation results show that our approximate screening method boosts the classification performance by 7.3×.
- We propose the near-memory architecture design, i.e., *ENMC*, with support for the Screener and the Executor (Section 5).
- We build a cycle-accurate simulator that interfaced with Ramulator [18] to evaluate the performance of ENMC, and it provides 7.4× average speedup over the CPU baseline and 2.7× speedup compared with the state-of-the-art NMP baseline.

We introduce the preliminary background in Section 2 and discuss our evaluation methodology and experimental results in Section 6 and Section 7.

2 PRELIMINARIES

In this section, we first introduce the preliminary background for future discussions, including extreme classification workloads, system architectures for inference, and near-memory processing techniques.

2.1 Extreme Classification

The Extreme Classification problem refers to multi-class or multi-label classification with extremely large category volume. Many large-scale NLP and recommendation applications can be modeled as a feature extraction part with an extreme classifier. For example, in NLP applications, the typical sequence-to-sequence modeling consists of a stack of encoders, a stack of decoders, and a final classification layer [29, 40, 44]. Each encoder and decoder is a type of DNN layer, such as Transformer layers [42] and recurrent neural networks [44]. The encoders process input embeddings into hidden representations repeatedly. The decoders that attend over all hidden states from the encoder stack process queries from the previous decoder layer and output decoded hidden vectors. The final classification layer turns the hidden vector from the last decoder layer into a translated word as in translation tasks or probabilities as in language modeling tasks. The classification layer consists of a large linear layer followed by a softmax layer. One way to interpret the linear layer is performing the inner-products of the hidden vector from the decoder stack and a number of weight vectors, which correspond to the target vocabulary size. The softmax function then normalizes the inner-products into probabilities.

Also, in large-scale recommendation systems such as commodity product recommendation and webpage recommendation, extreme classification refers to the problem of multi-class prediction [4, 24, 27, 39]. First, the hidden layers, e.g., DNNs, take dense features and sparse features from users as input. Then, the classification layer maps the output of the last hidden layer, usually through softmax normalization, to a probability distribution. For real-world scenarios and next-generation applications, the final classification layer is becoming even more challenging as the computational complexity and the memory usage grows linearly with the category size.

2.2 System Architecture

GPUs and domain-specific accelerators are widely used to process compute-intensive models in the front-end, such as CNNs, RNNs, and Transformers [42]. In contrast, for memory-intensive front-ends like recommendation models and embedding look-ups, CPUs are more favored because of the larger memory capacity. In these scenarios, the processing units (CPU/GPU/accelerator) typically allocate the classification parameters in the local memory, as shown in the Figure 3(a).

As we point out in the Section 2.1, the tremendous classification categories essentially need enormous memory capacity. For example, the largest dataset in an academic extreme classification repository [4] consists of 3 million categories, while industries have reported 50 million to 100 million categories used in classification [27, 39]. With the hidden size of 512, the memory usage of classification alone is reaching 190GB. The need for memory is increasing with the scaling of problem size in applications, easily exceeds the device memory capacity and even system memory capacity.

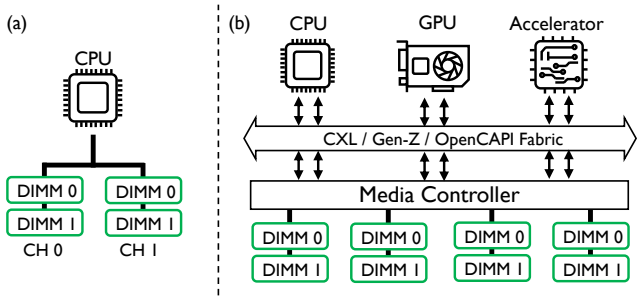

Figure 3: The system architecture of classification workloads: (a) Host-only system; (b) A pooled memory architecture to extend the memory capacity.

Therefore, we also consider the system architecture employing a memory pool to store the classification parameters, as shown in Figure 3(b). Facilitated by emerging memory protocols such as Gen-z [19], GPUDirect [36], CXL [41], etc., the pooled memory could easily stack from 1TB to 10TB DRAMs to tackle the application requirement.

2.3 Near-Memory Processing

As DNNs now appear to overwhelm almost every domain in our daily life and such applications are increasingly bandwidth-hungry, near-memory processing (NMP) technique is getting growing attention to accelerate these workloads. Leveraging the large internal bandwidth provided by rank parallelism or inside the memory chips, conventional NMPs put customized computation logic beside the data and saves the system bandwidth and memory access latency.

Different NMP techniques can be categorized by the distance between the computation logic and DRAM cell array. Here we generally refer two types of NMP techniques for a DRAM-based memory subsystem: intrusive and non-intrusive NMP. The intrusive NMP hacks the architecture inside the DRAM device, and the computation logic could be placed at the logic die for a 3D-stacking DRAM module [2, 16, 17, 31, 49], or directly beside the DRAM banks to gain higher bandwidth [10, 11, 22, 38, 46]. The non-intrusive NMP makes use of the rank-level parallelism in the current memory hierarchy. It tries to leverage commodity DRAM chips and places the processing unit at each rank on the DIMM, and thus higher bandwidth can be achieved with multiple ranks in a memory channel [15, 20, 21]. Our *ENMC* design takes the non-intrusive NMP approach since it requires minimized hardware changes in existing DRAM technology and does not need the support from the DRAM vendors.

3 MOTIVATION

As discussed in Section 2, the classification layer is the essential component in NLP tasks and large-scale recommendation systems. In Figure 4, we show the breakdown of model parameters and operations into classification and non-classification, i.e., input embedding and hidden layers. For the three NLP tasks, classifiers consume a significant amount of parameters and operations. When

classification category sizes scale to millions as in large-scale recommendation, classification layers become the major bottleneck. We observe similar breakdown on execution time.

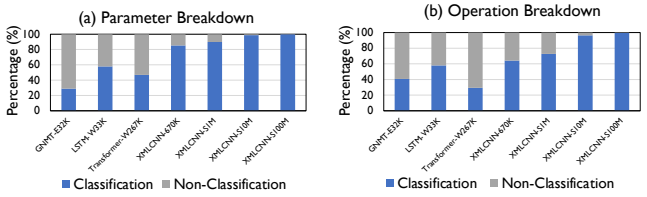

Figure 4: The breakdown of parameters and operations into classification and non-classification. Classification layers consume a large portion and become the bottleneck when categorize sizes scale.

3.1 Opportunity

The root cause for extreme classification being the bottleneck is from the large memory footprint and the low operational intensity. We show in Fig. 5(a) that classifiers consume memory in the order of hundreds megabytes or even gigabytes, far beyond the on-chip memory capacity of modern GPUs or NPUs. The execution time of classification increases linearly with category size and hidden dimensions. From the perspective of DL practitioners and algorithm developers, using larger vocabulary or category and hidden dimensions is almost always a way to improve model quality. However, the increasing memory usage will worsen the memory-bounded execution problem. For recommendation systems, the increasing need for an enormous number of items results in even more challenging requirement to accommodate the classifier.

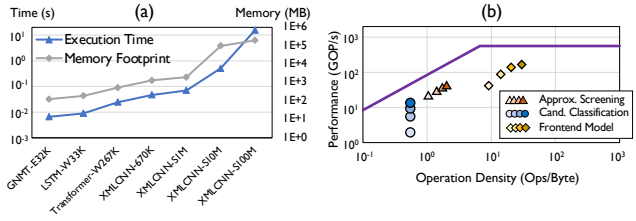

Figure 5: (a) The memory footprint and the execution time on CPU of classification layers scale linearly with the number of categories. (b) Roofline analysis of the major components. Darker color indicates larger batch size.

Opportunity of approximation: In extreme classification, outputs from classifier are probabilities. While we should compute all the outputs of the linear transformation using all classifier parameters, many applications require only the probabilities of the most top words. For example, in neural machine translation, we only use the top-K values of softmax-normalized probabilities to select the translated words, where K is the beam search size when applied. Therefore, we could have only the top-K probabilities to be accurate, then having the rest to be approximate, aiming at significantly reduced computations and data accesses. In the next section, we explore the opportunity of using approximation to achieve efficient extreme classification.

Opportunity of NMP: Although approximation can greatly reduce the computation amount in extreme classification, approximate screening is still bounded by the memory bandwidth. As shown in Fig. 5(b), we plot the data points for our approximate screening, candidate-only classification, and front-end neural networks in a CPU's roofline model. Both screening and classification exhibit low operation intensity after we eliminate redundant computations and reduce hidden dimensions. Therefore, different from the front-end models that are often bounded by computation capability, approximate screening and candidate-only classification can benefit from the large bandwidth of NMP architectures.

3.2 Limitations of Existing NMP

As mentioned above, due to the memory-bounded execution pattern of XC, NMP-enabled systems could leverage the near-data capability to avoid significant amount of off-chip memory traffic. However, existing NMPs often employ a homogeneous architecture equipped with unified floating-point and integer compute units [3, 9, 20]. Our proposed screening method explores a heterogeneous computation pattern that includes a low-precision approximate screening phase and a full-precision candidate-only classification phase. Therefore, our NMP architecture features a dedicate resource management of both phases and a customized pipeline design.

4 APPROXIMATE SCREENING

In Section 3, we discuss the potential of using NMP to alleviate the memory pressure of executing extreme classification. However, the limited computing capability of NMP logic cannot afford the computations of extreme classification. In other words, the execution of full classification on NMP core becomes the bottleneck.

We find that not all computations in classification are useful. In fact, only a small portion of classification results contribute to model predictions. For example, in language modeling tasks, only output probabilities of the most important words need to be accurate. Thus, we propose an efficient approximation method that can estimate the subset of output probabilities that need accurate computations and then populate the rest probabilities with approximate results. Similarly, for other classification-involved tasks, we only need accurate computations for a small number of key candidates and use approximate results for the remaining outputs.

4.1 Screening Method Overview

Given a d-dimensional vector ($h \in \mathbb{R}^d$) from hidden DNN layers, where d is the hidden dimension, the softmax classification transforms the hidden vector h to a l-dimensional probability space. We denote the output probability vector as $z \in \mathbb{R}^l$, where l is the vocabulary size. The transformation is essentially matrix-vector multiplication as

$$z = Wh + b \tag{1}$$

where $W \in \mathbb{R}^{l \times d}$ is the classifier weight matrix and $b \in \mathbb{R}^l$ is the bias vector. Then, the softmax function normalize the output vector

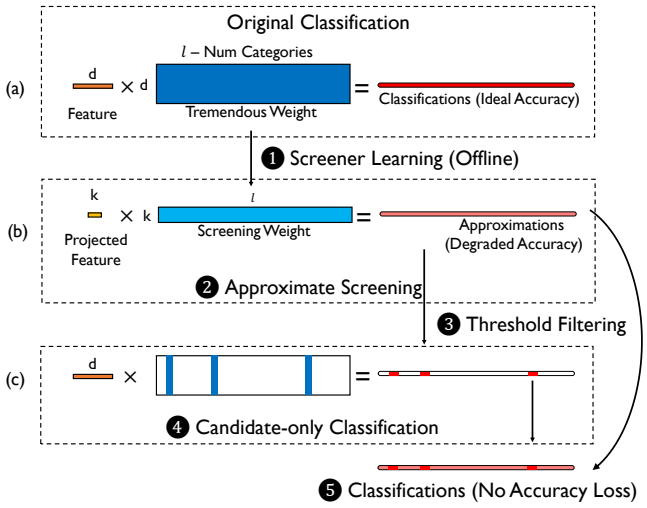

Figure 6: Illustration of approximate screening: (1) the screener learns from full classifier at the offline learning phase; (2) the screening step computes approximate results, involving lightweight Screener weights and the projection matrix, and selects candidates among approximate results; (3) the threshold filtering step selects key candidates; (4) only the corresponding vectors in the full classification weights are used to compute candidates-only accurate results; (5) the final results before softmax normalization combine both approximate and accurate results.

z into probability distribution as

$$p_i = softmax(z_i) = \frac{exp(z_i)}{\sum_j exp(z_j)} \tag{2}$$

where p_i is the i-th element of output probability vector p. The probability vector is then used to perform next word predictions as in language modeling or translation. While softmax is the most common normalization function used in classification, our method is capable to other non-linear functions used in classification such as sigmoid [24].

As discussed in Section 3, the memory-intensive transformation is a good candidate of NMP architectures. However, the computational complexity is not affordable for NMP. Our proposal seeks redundancy in extreme classification and uses low-cost approximated computations to mitigate the computational burden. We introduce a low-dimensional and low-precision screening module that can approximate the original classifier. We first discuss how to reduce computations at inference time given the screening module. After that, we explain the learning process to obtain such a screening module.

4.2 Inference Process

As shown in Figure 6(a), the standard classification is essentially matrix-vector multiplication followed by softmax normalization. The execution is bounded by accessing W from DRAM modules.

We construct the approximate screening module with a projection matrix P and a reduced-hidden-dimension weight matrix $\tilde{W}$. The initialization of the projection matrix is according to standard sparse random projection [1], and the overhead is negligible (less than 0.1%) compared with classifier weights as the projection matrix P can be represented in 2-bit format. The process of computing approximate results can be expressed as

$$\tilde{z} = \tilde{W}Ph + \tilde{b} \tag{3}$$

where $\tilde{W} \in \mathbb{R}^{l \times k}$ and $P \in \sqrt{\frac{3}{k}} \cdot \{-1, 0, 1\}^{k \times d}$.

Figure 6(b) illustrates the process: the d-dimensional hidden vector h is first projected to a lower k-dimensional space, and the low-dimensional vector multiplies $\tilde{W}$ to get approximated output $\tilde{z}$. Compared with full classification, the accessed approximate weight volume is significantly reduced since $k << d$. Furthermore, we can reduce the precision of running the screening module to further reduce accessed data.

After obtaining the approximate results, i.e., $\tilde{z}$, we estimate the importance of all l values and select the most important m values, referred as candidates, that require accurate computations. The estimation can be done with top-m searching or thresholding, where the threshold value can be tuned on validation sets.

Only for the candidates that need accurate computations, our method then need to access full classifier weights W, i.e., a small portion of totally l weight vectors. These weight vectors then multiply with the original hidden vector to produce the accurate results for the candidates, as shown in Figure 6(c). The final outputs before softmax function is a mixed vector with approximate values from screening and accurate values from full W.

Algorithm 1: Training algorithm for the parameters of the Screener

Data: Batched context vectors $\{h_i\}_{i=1}^{S}$, where $h_i \in \mathbb{R}^d$ from hidden layers; trained classifier weights $W \in \mathbb{R}^{l \times d}$ and bias $b \in \mathbb{R}^l$; projection matrix P.
Result: Screener weights $\tilde{W} \in \mathbb{R}^{l \times k}$ and bias $\tilde{b} \in \mathbb{R}^l$.

```
Initialize projection matrix P ∈ sqrt(3/k) · {−1, 0, 1}^{k×d};
for it ∈ all iterations do
    Compute loss according to Eq. (4);
    Update W̃, b̃ with SGD(min Loss);
end
```

4.3 Learning Algorithm

Here, we discuss the learning procedure to obtain screening module. The goal for screening is to approximate the classifier well. Therefore, we regard the outputs z from full classifier as the learning target and train the screening module weights $\tilde{W}$ to fit. The optimization objective function is

$$L = \frac{1}{s} \sum_{s} ||(Wh + b) - (\tilde{W}Ph + \tilde{b})||_2^2 \tag{4}$$

where s is the mini-batch size of training samples. During training, the classifier parameters, i.e., W and b, as well as the parameters of hidden layers are fixed and will not be changed. We only update the screening module's parameters $\tilde{W}$ and $\tilde{b}$. The projection matrix P

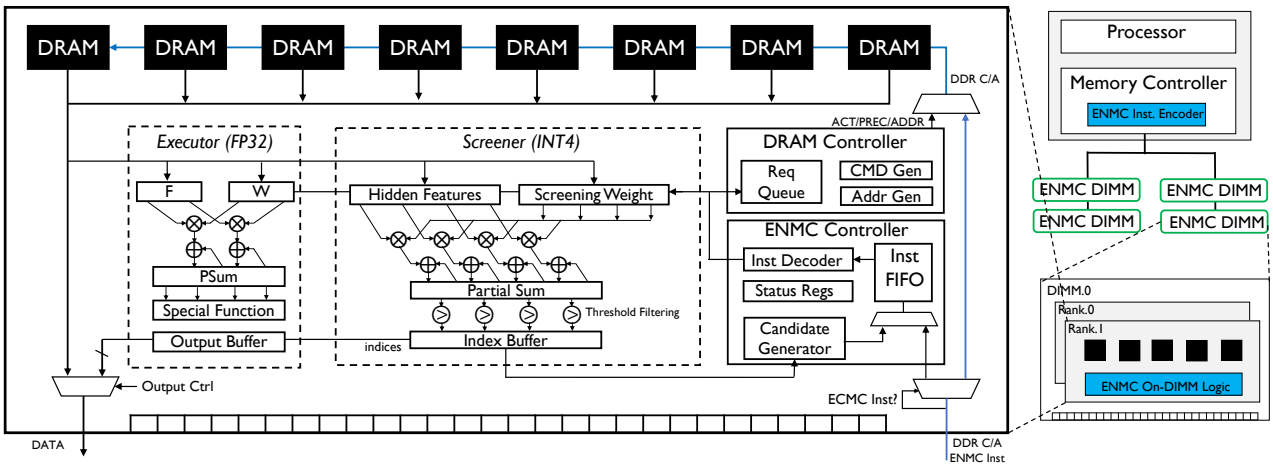

Figure 7: The architecture overview of an ENMC DIMM. The ENMC logic is located at each rank to leverage the large rank-level bandwidth. The ENMC mainly consists of a controller to decode instructions, a DRAM controller to generate DDR C/A commands, a screener to perform approximation, and an executor to process full-precision classification.

is constructed and initialized before distillation and stays constant during distillation and inference.

Our learning algorithm uses the default training and validation datasets and does not need extra training data. The convergence happens in a several training epochs, much faster than original model training. Algorithm 1 gives the overall training of screening parameters.

5 ENMC ARCHITECTURE

In this section, we introduce the architecture design of the ENMC. We first give a glimpse of the design overview, followed by the microarchitecture details. Then, we present the ENMC instruction set and system-level design.

5.1 Design Overview

We have yet exploited the opportunity of eliminating the redundancy in the extremely-large weight and forecasting the classification results with much smaller overhead using lightweight screening algorithm. Although the computation bottleneck is alleviated with our proposed approximate screening framework, the tremendous classification dimension is still bandwidth-hungry, and conventional processor-memory systems are hardly able to overcome the memory throughput wall. Therefore, in this section, we further co-architect the near-data processing subsystem, Extreme Near-Memory Classifier (ENMC), to facilitate the processor computing the extreme classification. The design goal of such near-data architecture is to leverage the large bandwidth provided by rank-level parallelism in a DRAM channel, and process the classification in data stream through dedicate on-DIMM hardware.

Specifically, we highlight the features of our ENMC design as follows:

First, we deploy a dual-module architecture that contains a Screener module and an Executor module that runs in parallel. The Screener performs fixed-point screening as described in Section 4, and predicts the classification candidates in advance. Since the classification weight is low-dimensional and quantized, the Screener is able to process the data in a streaming manner, such that the large rank-level bandwidth can be leveraged. For each candidate found in a batch, the ENMC Controller will generate instructions for further full-precision computations which are completed by the Executor. We put these computation logic at the rank level such that there is no need to invade the DRAM chips.

Second, we design the ENMC instruction set to facilitate the workloads accommodation from host processors and support the communications between the Screener and Executor modules. We define the instruction format by leveraging the unused address line and data line in the PRECHARGE command to ensure the compatibility with the commodity DDR interface. Thus, regular memory requests can also be served with our ENMC DIMM.

Third, we provide the system-level design, including the program compiler support and application workflow , to make the ENMC architecture cooperate with the software framework. Our design could be easily extended to support different scenarios where the host processors could be CPU, GPU, or domain-specific accelerators.

5.2 ENMC Microarchitecture

We now introduce the microarchitecture of ENMC. We first present the design overview, followed by the implementation details of each component.

Overview. We put ENMC on the DIMM board between the DRAM devices and the DDR PHY, such that the host processor could interface with ENMC through standard memory channels. Fig. 7 illustrates the details of the proposed ENMC architecture. The host processor contains several memory channels, which are deployed as the ENMC DIMMs. The ENMC logic locates at each rank of a ENMC DIMM, and thus enjoys scaling bandwidth offered by larger number of ranks. The on-DIMM ENMC architecture consists of a ENMC controller, a DRAM controller, and two processing units: the Executor and the Screener. The ENMC controller buffers

the instruction from the host processor for approximate screening. It also generates instructions for full-precision computation according to the candidate indices provided by the Screener. Then, it decodes the formatted instructions to generate control signals for data access, computation, and output transmission. The DRAM controller works as a simplified memory controller that processes data access requests in ENMC instructions and generates the standard DDR C/A signals to the DRAM chips. The Screener and Executor take charge of the approximate screening and the full-precision computation as described in Section 4.2, respectively. The Screener performs dimension-reduced INT4 computations to efficiently approximate the classifier's output. A preloaded threshold is used to filter out the important candidates based on the approximate results. Apart from floating-point arithmetic, the Executor is also equipped with a special-function unit to process the non-linear activation in the final layer. The two computation modules works in parallel and write results to the output buffer that returns them to the host processor asynchronously.

ENMC Controller. The ENMC controller has two main functionalities: processing the instructions from host processor (i.e., screening computation) and generating instructions for the Executor (i.e., candidate-only computation). It is made of status register files, an instruction buffer, an instruction decoder, and an instruction generator. The status register files are used for ENMC initialization and stores information such as addresses and sizes of input features, vocabulary, and screening weight. It also includes the instruction counter. The instruction buffer is a FIFO, and both the host processor and instruction generator could push instructions into it. The instruction decoder sequentially reads from the FIFO and generates control signals to corresponding ENMC components. For example, an instruction of accessing a piece of tiled screening weight would result in a read request to the DRAM controller and a select signal to the top DEMUX that chooses the integer weight buffer. Meanwhile, a full-precision computation instruction would lead to a triggering signal to the floating-point MAC array, which reads data from two input buffers and writes results to the partial sum (PSUM) buffer. The instruction generator receives the indices of classification candidates from the Screener (*batch_id*, *candidate_id*) , and then reads the constant reg to generate corresponding instruction for candidate-only computation in full-precision.

DRAM Controller. The DRAM controller employs a similar architecture as the host-side memory controller and consists of a request queue, a command generator, and an address generator. The request buffer takes memory request from the ENMC controller. The command and address generators initiate standard DDR4 C/A signals that are sent to all the DRAM chips. For hardware simplicity, we do not deploy unnecessary features like queue prioritizing, request coalescing, etc.

Screener. The Screener processes the approximate screening phase in the approximate screening algorithm with fixed-point precision. We put two input buffers (feature buffer and screening weight buffer), a fixed-point multiply-accumulate (MAC) array, a partial sum (PSUM) buffer, a threshold filter, and an instruction translator in the Screener. The MAC array performs the screening computation over the two input buffers and accumulates with the intermediate results in the PSUM buffer. After a tiled screening is finished, the data in the PSUM buffer are filtered with a comparator array. The indices of values larger than the threshold are buffered and later sent to the ENMC controller.

Executor. The Executor computes candidate-only classification under full-precision. Compared with the Screener, it applies floating-point MAC array and has an extra special-function unit that performs the non-linear activation such as Softmax and Sigmoid. We also put an output buffer below the special-function unit, which caches both the results from the Screener and the Executor. The output buffer keeps the state of the data with status reg files and notifies the ENMC controller (by pushing a RETURN instruction) when finishing a batched/tiled data.

5.3 ENMC Instruction Set

The design goal of the ENMC instruction set is to make the host processor able to communicate with ENMC DIMM through standard DDR4 memory channels. Inspired by FIRDRAM [22], we issue ENMC instructions from the memory controller with PRECHARGE command combining special addresses and data. For example, according to the DDR4 JEDEC specification, for a 4Gb DIMM with 8 ×8 DRAM chips, the row address space consumes 14 bits, i.e., A0-A13 in the C/A bus, and the data bus is 64-bit. Normal PRECHARGE command sets all the row address bits to be low, since no row information is needed. Therefore, an ENMC instruction could be accommodated with sending a PRECHARGE command but turning on the row address signals. Given this insight, we design the ENMC instruction formatted in 13-bit command and 64-bit data that transmits through signal A0-A12 and D/Q bus. With that, we first present the instruction specification and explains the instruction in details. Then, we define the instruction format.

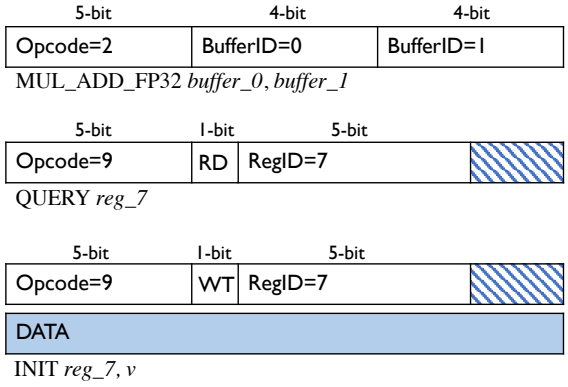

Figure 8: Instruction Format

Instruction Specification. As shown in Table 1, the ENMP instruction set consists of four types of instructions: Initialization, Data Transfer, Compute, and Control. (a) *Initialization.* The initialization instruction is used to write the status reg files in the ENMC controller, in order to initiate the parameters of a classification task. It specifies which reg to write and the corresponding value. (b) *Data Transfer.* The data transfer instructions are used to access the on-DIMM buffers, such as loading data to the input feature buffer or writing back the results to the PSUM buffer with specific addresses. Also, we use the instruction MOVE to transfer data in

Table 1: The ENMP instruction set

ENMC Instruction Set		
Type	Instruction	Description
Initialization	INIT reg, data	Initialize the ENMC module by writing a particular register

Data Transfer	LDR buffer, addr
	STR buffer, addr
	MOVE buffer1, buffer2
		load/store the quantized feature data into/from the INT4 feature buffer (weight buffer, with specified address addr

Compute	ADD_INT4 buffer1, buffer2 MUL_INT4 buffer1, buffer2 ADD_FP32 buffer1, buffer2 MUL_FP32 buffer1, buffer2	add/multiply the data in two specified buffer buffer1, buffer2
	MUL_ADD_INT4 MUL_ADD_FP32	multiply the data in feature buffer and weight buffer, and accumulate they with the partial sum buffer
	FILTER buffer	filter the data in the specific buffer and write the results to the index buffer
	SIGMOID, SOFTMAX	special functions such as Sigmoid and Softmax that run on specialized hardware for the data in the FP32 partial sum buffer
Control	BARRIER, NOP	synchronization and bubble instruction to let the controller wait for memory accesses, compute operation, data movement, etc.
	QUERY reg	query the value in the specific reg
	RETURN	return the data in the output buffer
	CLR	clear and reset all buffers and registers

two buffers, such as storing results in the PSUM buffer to the output buffer. (c) *Compute.* The compute instructions corresponds to the computation operations in the two computing units, including ADD, MUL, MUL_ADD, and denotes the operation precision. FILTER instruction is used to filter out the candidates. There are also instruction for special functions such as SOFTMAX and SIGMOID that operate on the PSUM buffer in the Executor. (d) *Control.* The control instructions include BARRIER for synchronization, NOP for stalling, RETURN to send back the output buffer data, and CLR to reset the ENMC. We also design a QUEUE instruction for the host processor to pull the status counters in each component.

Instruction Format. As shown in the Fig. 8, a typical ENMC command without data or address takes 13 bits, where the opcode is 5 bits and the rest 8 bits are used to specify which buffer to operate on. For example, Fig. 8(a) shows the instruction format for performing multiply-accumulate in the Screener. For the status register accessing instruction, QUERY and INIT shares the same opcode, and we use one bit after opcode to specify the read or write operation, and 5 bits to specify the register index, as shown in Fig. 8(b). Moreover, for instructions that involves values (i.e., data or address) that exceeds the length of row addresses, the DQ bus is further utilized. For example, when the host processor tries to write the status reg in the ENMC controller, the command address bus specifies the write operand and the ID of target regwith INIT instruction, and the DQ bus transmits the desired data in burst manner following the ENMC command.

5.4 System Design

In this subsection, we further architect the system-level design to facilitate existing software solutions running on the ENMC memory. We first present the programming support that wraps up ENMC instructions into high-level APIs such that a program could call the ENMC kernels directly. Second, we show the execution flow to demonstrate how the host processor interacts with the ENMC DIMM.

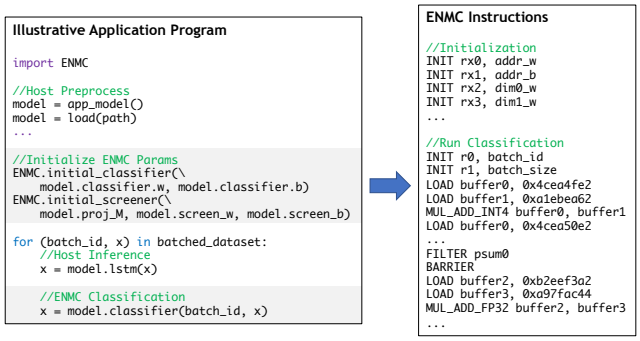

Figure 9: An illustrative example of programming support of ENMC. The ENMC APIs are wrapped as high-level function libraries, which are further compiled into ENMC instructions.

Programming Support. Following previous NMP solutions [15, 20], we divide the application code into kernels running on the host processor and ENMC in a heterogeneous manner. Therefore, the host processor calls the provided APIs to offload specific classification tasks. Fig. 9(a) shows an illustrative application code in Python style. We wrap up the functions that runs on ENMC DIMM into a Python package, such as initializing the Screener and screening-based classification. Therefore, a programmer could build a machine-learning model transparently using the ENMC package. Inside an ENMC object of classifier, we implement the approximate screening algorithm in the forward function with pretrained projection matrix and screening weight. Furthermore, when translating the applications into ENMC instructions, the compiler tiles the operation with initialized parameters and hardware configurations and executes the instruction in a loop. The ENMC instructions are further packed into a memory request packet and routed to the

Table 2: Evaluated models and datasets.

Application	Dataset	Dataset Type	Num. Categories	Inference Model	Model Type	Hidden Size	Abbr.
NLP	Wikitext-2	Language Modeling	33,278	LSTM	RNN	1500	LSTM-W33K
NLP	Wikitext-103	Language Modeling	267,744	Transformer	DNN	512	Transformer-W268K
NMT	WMT16, en-de	Translation	32,317	GNMT	DNN	1024	GNMT-E32K
Recommendation	Amazon-670k	Multi-label Classification	670,091	XMLCNN	CNN	512	XMLCNN-670K

memory controller, which transmits them to the ENMC DIMM, as shown in Fig. 9(b).

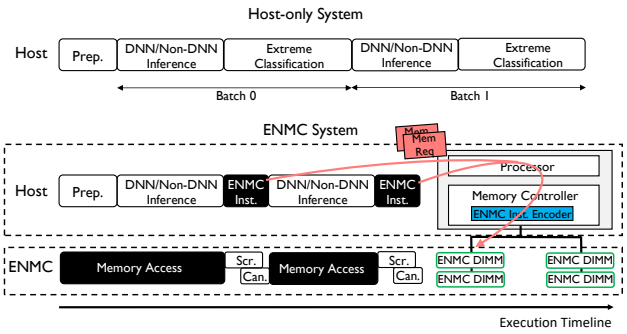

Figure 10: The ENMC workflow compared with a host-only system. ENMC offloads the classification tasks to the ENMC DIMMs by sending the instructions as memory requests through the memory controller.

Execution Flow. Fig. 10 presents the ENMC workflow compared with a host only system. The execution of front-end feature extraction (DNN-based or non-DNN-based) and the classification can be treated in a decoupled way. To be more specific, the host in the ENMC system is dedicated to run the feature extraction and offloads the classification tasks to the ENMC memory. The ENMC memory works as a regular main memory for data accessing in the first phase, and performs screening approximation and candidate-only classification in the second phase.

6 EVALUATION METHODOLOGY

In this section, we discuss the methodology of evaluating the ENMC co-design, including the implementation details and performance metrics.

6.1 Software Evaluation

We implement the approximate screening algorithm on top of existing pre-trained models in the PyTorch machine learning framework [34]. The screening parameters are trained under mean-square-error (MSE) loss using the original training and validation datasets till convergence. Both the input features and the screening parameters are further quantized at inference time. We set the number of candidates, screening parameters size, and quantization precision adjustable for sensitivity studies.

Workloads. We evaluate our method on different tasks including Language Modeling (LM) [28], Neural Machine Translation (NMT) [40], and product-to-product recommendation [26]. For LM, we use the Wikitext-2 and Wikitext-103 datasets [29] and evaluate on both long short-term memory networks (LSTM) and Transformer networks. For NMT, we use the WMT16 English-to-German dataset and evaluate on Google's Neural Machine Translation System (GNMT) [44]. For product recommendation, we use the Amazon670K dataset [4] and evaluate on a Convolutional Neural Network based model [24]. Table 2 lists the applications, the models, and the datasets used in our evaluation, as well as the number of categories and the hidden dimensions. We also synthesize three larger datasets with 1 million, 10 million, and 100 million categories to study the scalability of ENMC (namely S1M, S10M, and S100M). For detailed and reproducible implementation, we will submit our implementation for artifact evaluation and open-source our repository after the anonymous review process.

Baselines. For comparison, we include two other approximation methods for classification: SVD-softmax [37] and FGD [48]. The SVD-softmax method leverages singular value decomposition (SVD) to approximate the classification weight with principle singular values; the FGD method uses graph-based nearest neighbor search to approximate top-k classification results. We implement both baselines in our PyTorch-based framework.

Table 3: ENMC Configurations

DRAM Configuration			
Spec	DDR4-2400MHz	DRAM Chip	8Gb×8
Channels	8	Ranks/CH	8
Queue	64-entry	Capacity/CH	64GB
Timing	CL-tRCD-tRP: 16-16-16 tRC=55, tCCD=4, tRRD=4, tFAW=6		
ENMC Configuration			
Tech Node	28nm	Frequency	400MHz
Executor Buffer	256B+256B	Screener Buffer	256B+256B
FP32 MAC	16	INT4 MAC	128

6.2 Hardware Evaluation

We implement the ENMC logic in RTL and synthesize it with Design Compiler for hardware parameters including timing, power, and area. We build a cycle-accurate simulator for the ENMC DIMM that interfaces with Ramulator [18] to derive the DRAM timing information. Since the host processor and the ENMC DIMM execute the feature extraction phase and the classification phase separately without complicated feature interactions in between, we simulate a simple host model that only issues ENMC instructions regularly according to the status registers.

Configurations. As shown in Table 3, the ENMC DIMM is based on DDR4-2400 specifications. Each rank consists of 8×8 DRAM

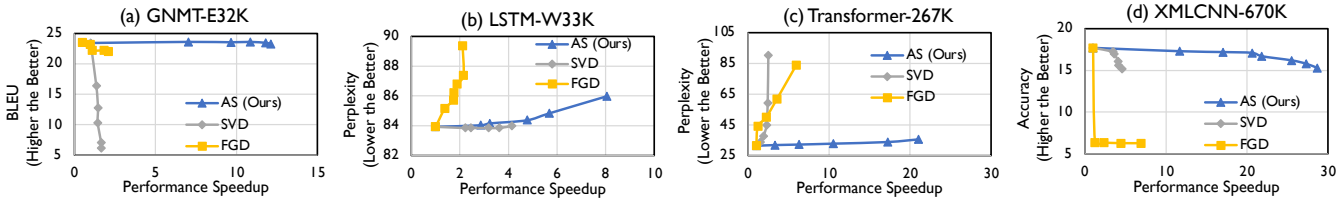

Figure 11: Quality vs. Speedup trade-off of Approximate Screening (AS) and two baselines: SVD and FGD.

chips that add to a total capacity of 8Gb. We put 8 memory channels for the host processor, and there are 8 ranks per channel, contributing to 64GB capacity and 21.3 GB/s bandwidth per channel. In addition, we synthesize our ENMC logic with TSMC 28nm technology, running on the frequency of 400MHz. The two input buffers and accumulation buffer in both Screener and Executor are 256B. We put 64 INT4 MACs and 16 FP32 MACs on each DIMM. For non-linear activations in the executor, we approximate the exponential function with Taylor expansion to the 4^{th} order.

Table 4: Comparing ENMC with three NMP baselines, all configured with similar area and power budget.

NMP Designs	Configuration	Est. Area - mm^2	Est. Power - mW
NDA [9]	4*4 Functional Units + 1KB Memory	0.445	293.6
Chameleon [3]	4*4 Systolic Array + 1KB Memory	0.398	249.0
TensorDIMM [20]	16-lane VPU + 512B Queue * 3	0.457	303.5
ENMC (Ours)	FP32 * 16 + INT4 * 128 + 256B Buffer * 4	0.442	285.4

Baselines. We compare ENMC with CPU and other NMP architectures, and all of them are equipped with the approximate screening algorithm. The CPU baseline is Intel Xeon Platinum 8280 @ 2.7GHz. It has 28 physical cores and 6 DDR4-2666 memory channels, contributing to a total memory capacity of 512 GB and 128GB/s ideal bandwidth. Three state-of-the-art DRAM-based NMP architectures are also selected for evaluation:

NDA [9] provides a near-data acceleration solution by stacking coarse-grain reconfigurable accelerators (CGRA) with DRAM devices. The CGRA mainly consists of functional units, switches, and memory.

Chameleon [3] is similar to NDA by employing a 2D architecture and focusing on how to integrate the accelerator with commercial DRAM. As Chameleon could work with any programmable compute unit, we put a systolic array as the accelerator core to distinguish it from NDA.

TensorDIMM [20] is a NMP architecure for deep learning applications, especially for recommendation workloads. It leverages the VPU to accelerate the embedding operations in recommendation systems.

For a fair comparison, we configure the ENMC and three baselines with approximately the same area and power budget, as shown in Table 4; the control logic and DRAM device controller are excluded.

7 EVALUATION

In this section, we evaluate the screening method for extreme classification and the micro-architecture of near-memory processing cores. For the method, we show the trade-offs between inference quality and speedup to CPU execution time of full classification. Then, we present the speedup of classification enabled by NMP co-design and the system performance improvements.

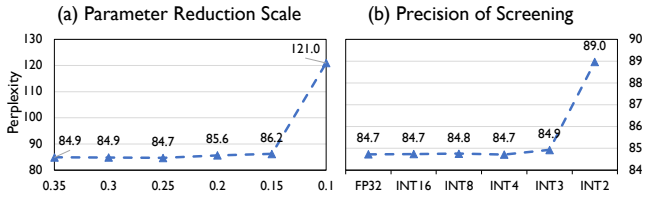

Figure 12: Comparing different (a) parameter reduction scales and (b) quantization levels of AS.

7.1 Algorithm-level Evaluation

Overall model quality. We post the hypothesis that extreme classification can afford approximation. Here we provide experimental results to support the hypothesis. Overall, our method can achieve significant computation saving with negligible model quality degradation. We can trade-off model inference quality to the acceptable extend for more computation reduction.

As shown in Fig. 11(a), compared with using full classification as in NMT tasks, our method can achieve speedup of 11.8× without any loss in translation quality measured by BLEU score. As for LM tasks, the speedups can reach 5.7× to 6.3× while preserving perplexity results, as shown in Fig. 11(b) and (c). Similarly, for product recommendation, our method can achieve 17.4× speedup with only 0.5% drop in accuracy, as shown in Fig. 11(d).

Because of the well approximation that our method achieves, the screening phase can effectively select the key candidates for classification. Using the NMT task as an example, at every decoding step, we want the most likely word or a few words if using beam search. With Approximate Screening, we can identify the key candidates and compute the accurate probabilities of these words for translation, saving redundant computations for the remaining words in the vocabulary. We set the overhead of Approximate Screening to be 3.1% of full classification.

Compared with two other approximation methods, our method achieves better quality-speedup trade-off, as shown in Figure 11. Besides, the computation overhead of SVD-based approximation is

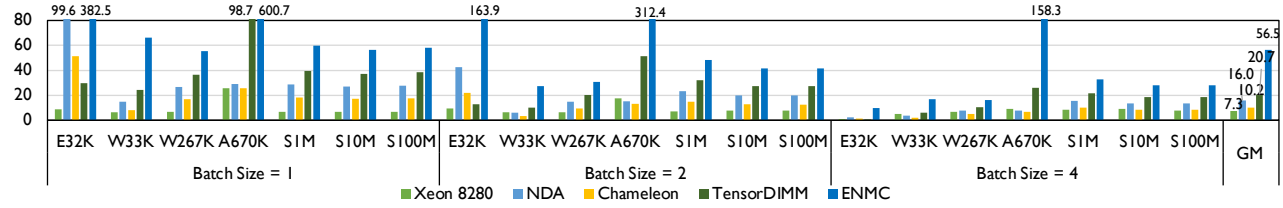

Figure 13: The performance results of ENMC, CPU, NDA, Chameleon, and TensorDIMM, normalized to vanilla CPU; all schemes are equipped with approximate screening.

4× more than ours. We infer that the improvement of our method is due to the learning-based approximation and no strong requirement for classifier weights to be low-rank.

Sensitivity on Approximate Screening. Intuitively, better approximation costs larger computation and data overhead, while achieving better model quality with screening. We show different parameter sizes of the screening module and the corresponding quality. Fig. 12(a) shows different parameter reduction scales of the screening module vs. full classifier; we choose the scale to be 0.25 as the good quality preserving. As shown in Fig. 12(b), we use 4-bit fixed-point quantization on the screening module as this quantization level maintains approximation as using single floating-point precision.

7.2 Architecture-level Evaluation

Performance. As described in Section 6, we compare ENMC with four baselines. As shown in Fig. 13, we take the batch size of 1, 2, 4 and normalize the performance results to the full-classification CPU baseline for each workload, and arrange the results according to the size of classification across the x-axis. Our approximate screening demonstrates 7.3× performance speedup on average in CPU baseline, and the ENMC offers a total 56.5× speedup over the CPU. Also, 3.5×, 5.6×, and 2.7× average are observed when compared with NDA, Chameleon, and TensorDIMM respectively. First, we find ENMC provides significant speedups of 55.5×-600.7× when we do low-latency inference with batch size of 1, because ENMC processes the inference in a streaming manner over the lightweight classification. The huge performance gain in XMLCNN-670K workload is because we considerably reduces the number of candidates by 50×. Second, the three NMP baselines benefit from large internal bandwidth and offer 10.2-20.7× speedup over the CPU baseline. However, our ENMC could further boost their performance by 2.7-5.6× with heterogeneous resource management and dataflow customization. This result aligns our assumption that the performance of naive NMP solutions is bounded by the limited on-DIMM buffers and computation resources. Because they employ homogeneous FP32 computation units and hardly meets the throughput requirement in the screening phase. ENMC eliminates the redundant computation and needs only a small portion of FP32 computations. The entire screening phase is processed with lightweight INT4 units in stream.

Energy Consumption. We evaluate the energy results of ENMC against TensorDIMM and TensorDIMM-Large for a fair comparison. As shown in Fig. 14, we reduce the average energy cost by 5.0× and 8.4× compared with TensorDIMM and TensorDIMM-Large,

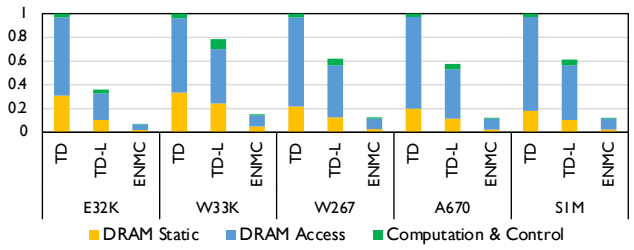

Figure 14: Energy breakdown by DRAM static cost, DRAM access, and computation & control logic, normalized to TensorDIMM.

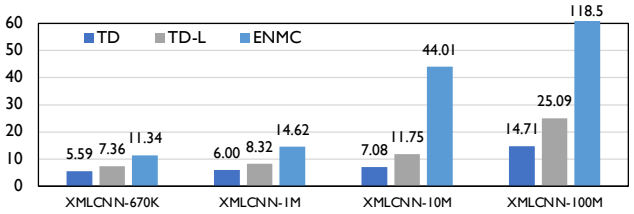

Figure 15: The end-to-end performance scalability compared with TensorDIMM and TensorDIMM large.

respectively. Particularly, we breakdown the energy consumed by the DRAM static cost, DRAM access, and on-DIMM computation and control logic. We observe that the significant energy reduction of ENMC comes from two facts: First, the co-designed approximation algorithm greatly reduces the DRAM accesses in ENMC. In ENMC, we perform INT4 and low-dimensional screening during the classification phase, while TensorDIMM and TensorDIMM-Large need to operate over the full classification weight. Moreover, due to the limited logic-side buffer size, TensorDIMM cannot store the intermediate results in a matrix multiplication operation. Thus, the buffer overflow results in frequent DRAM memory accesses. Second, the reduced execution time leads to the background energy reduction of the DRAM modules. As the DRAM takes a noticeably portion of power for refreshing, ENMC reduces the DRAM static energy consumption by 9.3× and 4.8× compared with TensorDIMM and TensorDIMM-Large.

End-to-End Scalability. We evaluate the scalability of performance considering the end-to-end performance over large synthetic datasets. As shown in Fig. 15, we restrict the application to the same front-end model of XMLCNN, and the performance of TensorDIMM,

TensorDIMM-Large, and ENMC is normalized to the CPU baseline. For comparison, ENMC achieves 4.7× and 2.9× speedup over TensorDIMM and TensorDIMM-Large. Particularly, for the two smaller datasets, ENMC achieves 2.2× and 1.6× speedups, while for the two tremendous datasets, ENMC achieves 7.1× and 4.2× speedups, compared with TensorDIMM and TensorDIMM-Large, respectively. The excellent scalability of ENMC comes from the fact that the ENMC processes the lightweight classification in stream and does not need to buffer large intermediate results back to DRAM.

Table 5: Area and Power Estimation.

	Area (mm^2)	Power (mW)		Area (mm^2)	Power (mW)
INT4 MAC	0.013	10.4	**FP32 MAC**	0.145	58.0
Compute Buffer	0.061	56.8	**Control Buffer**	0.053	49.3
ENMC Ctrl	0.035	32.9	**DRAM Ctrl**	0.135	78.0
Total Area 0.442mm^2; Total Power 285.4mW					

Area and Power. Table 5 shows the breakdown area and power estimation of ENMC. The total area of ENMC logic is $0.388mm^2$, and the total power is 264.6mW, which are comparable to prior NMP architectures such as RecNMP [15]. Specifically, the compute unit (INT4 and FP32 MAC arrays) takes 40.8% of the total area and 25% of the total power. The buffers made of register files in the Screener and the Executor compose of 23.5% of the total area and 32.2% of the total power. Finally, the ENMC controller and DRAM controller takes 9.0% and 34.8% of the area, and 12.4% and 29.5% of the power, respectively.

8 RELATED WORK

Near-memory processing (NMP). NMP techniques are widely used to solve the memory-bound problems, due to the large internal bandwidth provided [2, 10, 11, 16, 17, 22, 31, 38, 46, 49]. The most related work to our ENMC are the non-intrusive DRAM-based NMP architectures [3, 9, 15, 20, 21]. NDA [9] and Chameleon [3] are general-purpose NMP designs that make effort on communication and interconnection between the accelerator cores and DRAM devices. TensorDIMM [20], TensorCasting [21], and RecNMP [15] particularly focus on the recommendation applications and identify the performance bottleneck as embedding operations. Therefore, they design specialized architecture for inference or training phases for recommendation systems. Our ENMC distinguishes from the above work as we propose an algorithm-architecture co-design for extreme classification workloads. Prior NMP designs are for the embedding layers in DNN models, but those work fall short in handling full-precision matrix multiplications as in extreme classification workloads.

On the other hand, intrusive NMP architectures are also broadly studied based on 3D-stacking memory technology, such as HBM and HMC [2, 10, 16, 17, 22, 31, 49]. These NMP designs tend to enjoy larger bandwidth by connecting accelerators and DRAM dies with through-silicon vias (TSVs). Recently, Samsung announced the first industry's HBM2-PIM, which takes the advantage of the AI engine directly inside memory banks. However, 3D memory often suffers from limited capacity and hardly meets the scalability demand in extreme classification tasks.

Algorithms for efficient classification. Prior studies propose approximation methods for softmax classification as typically used in NLP applications [5, 37, 48]. However, these methods only demonstrate effectiveness at small-scale vocabulary sizes and cannot keep up with the increasingly scaling of vocabulary size. Without NMP architectures, the performance improvements from approximation saturate as the memory-bounded problem being neglected. Model compression techniques are orthogonal to our method and can be integrated to further reduce data access of classification [7, 8, 12, 13, 32, 43, 45, 51, 52].

Methods for training extreme classification use softmax variants, such as KNN softmax and selective softmax [39], to select active classes from training samples. However, active class selection requires the knowledge of true labels of training samples, making it impractical for inference usage. MACH [27] uses a divide-and-conquer method to search over the entire classes in parallel, but the work cannot mitigate overall memory usage much and suffers from classification accuracy drop.

We suggest collaborative design considerations from both ML experts and hardware designers. From the algorithm side, pure approximation methods do not mitigate the memory-intensive nature of extreme classification and also suffer from accuracy drop. From the architecture side, emerging NMP architectures are suitable for mitigating the memory-intensive extreme classification by bring the computations near memory. However, we need careful dataflow and customization to make it practical. In the context of distributed inference [23, 25], our design can scale-out from single-node to distributed nodes, where each node keeps an approximate screener. We envision ENMC co-design would support and enable future research on efficient methods for extreme classification.

9 CONCLUSION

In this paper, we address the extreme classification problem with NMP-based software-hardware co-design. We propose an approximate screening algorithm to reduce the computational complexity and memory consumption of extreme classification. We further design a near-memory architecture to utilize efficient candidates-only classification enabled by our screening method. Finally, our approximate screening method achieves 7.3× speedups, and the ENMC architecture further improves the performance by 7.4× and demonstrates 2.7× speedup compared with the state-of-the-art NMP baseline.

ACKNOWLEDGMENTS

We thank the anonymous reviewers of MICRO 2021 for their helpful comments and suggestions. We also thank Shuangchen Li, Peng Gu, Fengbin Tu, and Zhenyu Gu for their insightful feedback of this research. This work was supported in part by NSF 1925717. Use was made of computational facilities purchased with funds from the National Science Foundation (OAC-1925717) and administered by the Center for Scientific Computing (CSC). The CSC is supported by the California NanoSystems Institute and the Materials Research Science and Engineering Center (MRSEC; NSF DMR 1720256) at UC Santa Barbara.

REFERENCES

[1] Dimitris Achlioptas. 2001. Database-friendly random projections. In *Proceedings of the twentieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. ACM, 274–281.

[2] Junwhan Ahn, Sungpack Hong, Sungjoo Yoo, Onur Mutlu, and Kiyoung Choi. 2015. A scalable processing-in-memory accelerator for parallel graph processing. In *Proceedings of the 42nd Annual International Symposium on Computer Architecture*. 105–117.

[3] Hadi Asghari-Moghaddam, Young Hoon Son, Jung Ho Ahn, and Nam Sung Kim. 2016. Chameleon: Versatile and practical near-DRAM acceleration architecture for large memory systems. In *2016 49th annual IEEE/ACM international symposium on Microarchitecture (MICRO)*. IEEE, 1–13.

[4] K. Bhatia, K. Dahiya, H. Jain, P. Kar, A. Mittal, Y. Prabhu, and M. Varma. 2016. The extreme classification repository: Multi-label datasets and code. http://manikvarma.org/downloads/XC/XMLRepository.html

[5] Pei Hung Chen, Si Si, Sanjiv Kumar, Yang Li, and Cho Jui Hsieh. 2019. Learning to screen for fast softmax inference on large vocabulary neural networks. In *International Conference on Learning Representations (ICLR)*. arXiv:1810.12406 https://openreview.net/forum?id=ByeMB3Act7

[6] Yu-Hsin Chen, Tushar Krishna, Joel S Emer, and Vivienne Sze. 2017. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE Journal of Solid-State Circuits* 52, 1 (2017), 127–138.

[7] Jungwook Choi, Zhuo Wang, Swagath Venkataramani, Pierce I-Jen Chuang, Vijayalakshmi Srinivasan, and Kailash Gopalakrishnan. 2018. Pact: Parameterized clipping activation for quantized neural networks. *arXiv preprint arXiv:1805.06085* (2018).

[8] Xiaoliang Dai, Hongxu Yin, and Niraj K Jha. 2018. Grow and prune compact, fast, and accurate LSTMs. *arXiv preprint arXiv:1805.11797* (2018).

[9] Amin Farmahini-Farahani, Jung Ho Ahn, Katherine Morrow, and Nam Sung Kim. 2015. NDA: Near-DRAM acceleration architecture leveraging commodity DRAM devices and standard memory modules. In *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 283–295.

[10] Peng Gu, Xinfeng Xie, Yufei Ding, Guoyang Chen, Weifeng Zhang, Dimin Niu, and Yuan Xie. 2020. iPIM: Programmable in-memory image processing accelerator using near-bank architecture. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 804–817.

[11] Peng Gu, Xinfeng Xie, Shuangchen Li, Dimin Niu, Hongzhong Zheng, Krishna T Malladi, and Yuan Xie. 2020. DLUX: a LUT-based Near-Bank Accelerator for Data Center Deep Learning Training Workloads. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2020).

[12] Song Han, Huizi Mao, and William J Dally. 2015. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149* (2015).

[13] Song Han, Jeff Pool, John Tran, and William Dally. 2015. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*. 1135–1143.

[14] Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, Rick Boyle, Pierre-luc Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Mike Daley, Matt Dau, Jeffrey Dean, Ben Gelb, Tara Vazir Ghaemmaghami, Rajendra Gottipati, William Gulland, Robert Hagmann, C. Richard Ho, Doug Hogberg, John Hu, Robert Hundt, Dan Hurt, Julian Ibarz, Aaron Jaffey, Alek Jaworski, Alexander Kaplan, Harshit Khaitan, Daniel Killebrew, Andy Koch, Naveen Kumar, Steve Lacy, James Laudon, James Law, Diemthu Le, Chris Leary, Zhuyuan Liu, Kyle Lucke, Alan Lundin, Gordon MacKean, Adriana Maggiore, Maire Mahony, Kieran Miller, Rahul Nagarajan, Ravi Narayanaswami, Ray Ni, Kathy Nix, Thomas Norrie, Mark Omernick, Narayana Penukonda, Andy Phelps, Jonathan Ross, Matt Ross, Amir Salek, Emad Samadiani, Chris Severn, Gregory Sizikov, Matthew Snelham, Jed Souter, Dan Steinberg, Andy Swing, Mercedes Tan, Gregory Thorson, Bo Tian, Horia Toma, Erick Tuttle, Vijay Vasudevan, Richard Walter, Walter Wang, Eric Wilcox, and Doe Hyun Yoon. 2017. In-Datacenter Performance Analysis of a Tensor Processing Unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture (ISCA '17)*. Association for Computing Machinery, New York, NY, USA, 1–12. https://doi.org/10.1145/3079856.3080246

[15] Liu Ke, Udit Gupta, Benjamin Youngjae Cho, David Brooks, Vikas Chandra, Utku Diril, Amin Firoozshahian, Kim Hazelwood, Bill Jia, Hsien-Hsin S. Lee, Meng Li, Bert Maher, Dheevatsa Mudigere, Maxim Naumov, Martin Schatz, Mikhail Smelyanskiy, Xiaodong Wang, Brandon Reagen, Carole-Jean Wu, Mark Hempstead, and Xuan Zhang. 2020. RecNMP: Accelerating Personalized Recommendation with Near-Memory Processing. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. 790–803. https://doi.org/10.1109/ISCA45697.2020.00070

[16] Chad D Kersey, Hyesoon Kim, and Sudhakar Yalamanchili. 2017. Lightweight SIMT core designs for intelligent 3D stacked DRAM. In *Proceedings of the International Symposium on Memory Systems*. 49–59.

[17] Gwangsun Kim, Niladrish Chatterjee, Mike O'Connor, and Kevin Hsieh. 2017. Toward standardized near-data processing with unrestricted data placement for GPUs. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–12.

[18] Yoongu Kim, Weikun Yang, and Onur Mutlu. 2015. Ramulator: A fast and extensible DRAM simulator. *IEEE Computer architecture letters* 15, 1 (2015), 45–49.

[19] Patrick Knebel, Dan Berkram, Al Davis, Darel Emmot, Paolo Faraboschi, and Gary Gostin. 2019. Gen-Z Chipsetfor Exascale Fabrics. In *2019 IEEE Hot Chips 31 Symposium (HCS)*. IEEE Computer Society, 1–22.

[20] Youngeun Kwon, Yunjae Lee, and Minsoo Rhu. 2019. 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*. 740–753.

[21] Youngeun Kwon, Yunjae Lee, and Minsoo Rhu. 2020. Tensor Casting: Co-Designing Algorithm-Architecture for Personalized Recommendation Training. *arXiv preprint arXiv:2010.13100* (2020).

[22] Young-Cheon Kwon, Suk Han Lee, Jaehoon Lee, Sang-Hyuk Kwon, Je Min Ryu, Jong-Pil Son, O Seongil, Hak-Soo Yu, Haesuk Lee, Soo Young Kim, Youngmin Cho, Jin Guk Kim, Jongyoon Choi, Hyun-Sung Shin, Jin Kim, BengSeng Phuah, HyoungMin Kim, Myeong Jun Song, Ahn Choi, Daeho Kim, SooYoung Kim, Eun-Bong Kim, David Wang, Shinhaeng Kang, Yuhwan Ro, Seungwoo Seo, JoonHo Song, Jaeyoun Youn, Kyomin Sohn, and Nam Sung Kim. 2021. 25.4 A 20nm 6GB Function-In-Memory DRAM, Based on HBM2 with a 1.2TFLOPS Programmable Computing Unit Using Bank-Level Parallelism, for Machine Learning Applications. In *2021 IEEE International Solid- State Circuits Conference (ISSCC)*, Vol. 64. 350–352. https://doi.org/10.1109/ISSCC42613.2021.9365862

[23] Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. 2020. Gshard: Scaling giant models with conditional computation and automatic sharding. *arXiv preprint arXiv:2006.16668* (2020).

[24] Jingzhou Liu, Wei Cheng Chang, Yuexin Wu, and Yiming Yang. 2017. Deep learning for extreme multi-label text classification. In *SIGIR 2017 - Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*. Association for Computing Machinery, Inc, New York, NY, USA, 115–124. https://doi.org/10.1145/3077136.3080834

[25] Michael Lui, Yavuz Yetim, Özgür Özkan, Zhuoran Zhao, Shin-Yeh Tsai, Carole-Jean Wu, and Mark Hempstead. 2021. Understanding capacity-driven scale-out neural recommendation inference. In *2021 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 162–171.

[26] Julian McAuley and Jure Leskovec. 2013. Hidden factors and hidden topics: understanding rating dimensions with review text. In *Proceedings of the 7th ACM conference on Recommender systems*. 165–172.

[27] Tharun Medini, Qixuan Huang, Yiqiu Wang, Vijai Mohan, and Anshumali Shrivastava. 2019. Extreme classification in log memory using count-min sketch: A case study of amazon search with 50m products. In *Neural Information Processing Systems (NeurIPS)*. arXiv:1910.13830

[28] Stephen Merity, Nitish Shirish Keskar, and Richard Socher. 2017. Regularizing and optimizing LSTM language models. *arXiv preprint arXiv:1708.02182* (2017).

[29] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2016. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843* (2016).

[30] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Distributed representations of words and phrases and their compositionality. *arXiv preprint arXiv:1310.4546* (2013).

[31] Lifeng Nai, Ramyad Hadidi, He Xiao, Hyojong Kim, Jaewoong Sim, and Hyesoon Kim. 2018. CoolPIM: Thermal-aware source throttling for efficient PIM instruction offloading. In *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 680–689.

[32] Sharan Narang, Erich Elsen, Gregory Diamos, and Shubho Sengupta. 2017. Exploring sparsity in recurrent neural networks. *arXiv preprint arXiv:1704.05119* (2017).

[33] Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. 2019. fairseq: A Fast, Extensible Toolkit for Sequence Modeling. In *Proceedings of NAACL-HLT 2019: Demonstrations*.

[34] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (Eds.). Curran Associates, Inc., 8024–8035. http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[35] Yashoteja Prabhu, Anil Kag, Shrutendra Harsola, Rahul Agrawal, and Manik Varma. 2018. Parabel: Partitioned label trees for extreme classification with application to dynamic search advertising. In *Proceedings of the 2018 World Wide Web Conference*. 993–1002.

[36] Davide Rossetti and S Team. 2015. GPUDIRECT: Integrating the GPU with a Network Interface. In *GPU Technology Conference*.

[37] Kyuhong Shim, Minjae Lee, Iksoo Choi, Yoonho Boo, and Wonyong Sung. 2017. SVD-softmax: Fast softmax approximation on large vocabulary neural networks.

In *Neural Information Processing Systems (NeurIPS)*. 5464–5474.

[38] Hyunsung Shin, Dongyoung Kim, Eunhyeok Park, Sungho Park, Yongsik Park, and Sungjoo Yoo. 2018. McDRAM: Low latency and energy-efficient matrix computations in DRAM. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 37, 11 (2018), 2613–2622.

[39] Liuyihan Song, Pan Pan, Kang Zhao, Hao Yang, Yiming Chen, Yingya Zhang, Yinghui Xu, and Rong Jin. 2020. Large-Scale Training System for 100-Million Classification at Alibaba. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2909–2917. https://doi.org/10.1145/3394486.3403342 arXiv:2102.06025

[40] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. *arXiv preprint arXiv:1409.3215* (2014).

[41] S Van Doren. 2019. HOTI 2019: Compute Express Link. In *2019 IEEE Symposium on High-Performance Interconnects (HOTI)*. IEEE Computer Society, 18–18.

[42] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *arXiv preprint arXiv:1706.03762* (2017).

[43] Peiqi Wang, Xinfeng Xie, Lei Deng, Guoqi Li, Dongsheng Wang, and Yuan Xie. 2018. HitNet: hybrid ternary recurrent neural network. In *Advances in Neural Information Processing Systems*. 604–614.

[44] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. 2016. Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. (2016). arXiv:cs.CL/1609.08144

[45] Chen Xu, Jianqiang Yao, Zhouchen Lin, Wenwu Ou, Yuanbin Cao, Zhirong Wang, and Hongbin Zha. 2018. Alternating multi-bit quantization for recurrent neural networks. *arXiv preprint arXiv:1802.00150* (2018).

[46] Amir Yazdanbakhsh, Choungki Song, Jacob Sacks, Pejman Lotfi-Kamran, Hadi Esmaeilzadeh, and Nam Sung Kim. 2018. In-dram near-data approximate acceleration for gpus. In *Proceedings of the 27th International Conference on Parallel Architectures and Compilation Techniques*. 1–14.

[47] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. 2018. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 974–983.

[48] Minjia Zhang, Xiaodong Liu, Wenhan Wang, Jianfeng Gao, and Yuxiong He. 2018. Navigating with Graph Representations for Fast and Scalable Decoding of Neural Language Models. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems (NIPS'18)*. Curran Associates Inc., Red Hook, NY, USA, 6311–6322.

[49] Mingxing Zhang, Youwei Zhuo, Chao Wang, Mingyu Gao, Yongwei Wu, Kang Chen, Christos Kozyrakis, and Xuehai Qian. 2018. GraphP: Reducing communication for PIM-based graph processing with efficient data partition. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 544–557.

[50] Xingcheng Zhang, Lei Yang, Junjie Yan, and Dahua Lin. 2018. Accelerated training for massive classification via dynamic class selection. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 32.

[51] Chenzhuo Zhu, Song Han, Huizi Mao, and William J Dally. 2016. Trained ternary quantization. *arXiv preprint arXiv:1612.01064* (2016).

[52] Michael Zhu and Suyog Gupta. 2017. To prune, or not to prune: exploring the efficacy of pruning for model compression. *arXiv preprint arXiv:1710.01878* (2017).