ECSSD: Hardware/Data Layout Co-Designed In-Storage-Computing Architecture for Extreme Classification

Siqi Li
siqi_li@ucsb.edu
University of California, Santa Barbara
Santa Barbara, California, USA

Fengbin Tu
fengbintu@ust.hk
The Hong Kong University of Science and Technology
Hong Kong, China

Liu Liu
liu.liu@rpi.edu
Rensselaer Polytechnic Institute
Troy, New York, USA

Jilan Lin
jilan@ucsb.edu
University of California, Santa Barbara
Santa Barbara, California, USA

Zheng Wang
zheng_wang@ucsb.edu
University of California, Santa Barbara
Santa Barbara, California, USA

Yangwook Kang
yangwook.k@samsung.com
Samsung Semiconductor Inc.
San Jose, California, USA

Yufei Ding
yufeiding@cs.ucsb.edu
University of California, Santa Barbara
Santa Barbara, California, USA

Yuan Xie
yuanxie@gmail.com
DAMO Academy, Alibaba Group
Sunnyvale, California, USA

ABSTRACT

With the rapid growth of classification scale in deep learning systems, the final classification layer becomes extreme classification with a memory footprint exceeding the main memory capacity of the CPU or GPU. The emerging in-storage-computing technique offers an opportunity on account of the fact that SSD has enough storage capacity for the parameters of extreme classification. However, the limited performance of naive in-storage-computing schemes is insufficient to support the heavy workload of extreme classification.

To this end, we propose ECSSD, the first hardware/data layout co-designed in-storage-computing architecture for extreme classification, based on the approximate screening algorithm. We propose an alignment-free floating-point MAC circuit technique to improve the computational ability under the limited area budget of in-storage-computing schemes so that the computational ability can match SSD's high internal bandwidth. We present a heterogeneous data layout design for the 4/32-bit weight data in the approximate screening algorithm to avoid data transfer interference and further utilize the internal DRAM bandwidth of SSD. Moreover, we propose a learning-based adaptive interleaving framework to balance the access workload in each flash channel and improve channel-level bandwidth utilization. Putting them together, our ECSSD achieves 3.24-49.87x performance improvements compared with state-of-the-art baselines.

ISCA '23, June 17–21, 2023, Orlando, FL, USA

ACM ISBN 979-8-4007-0095-8/23/06.
https://doi.org/10.1145/3579371.3589093

CCS CONCEPTS

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

KEYWORDS

In-storage-computing architecture, Extreme classification

ACM Reference Format:
Siqi Li, Fengbin Tu, Liu Liu, Jilan Lin, Zheng Wang, Yangwook Kang, Yufei Ding, and Yuan Xie. 2023. ECSSD: Hardware/Data Layout Co-Designed In-Storage-Computing Architecture for Extreme Classification. In *Proceedings of the 50th Annual International Symposium on Computer Architecture (ISCA '23), June 17–21, 2023, Orlando, FL, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3579371.3589093

1 INTRODUCTION

Classification workloads are essential in many applications, such as image recognition [4, 8, 41], natural language processing [6, 13, 56], and recommendation [34, 43, 51], where the classification layer works as the final step to make predictions. As the size of classification categories keeps scaling up, the classification layer becomes larger and larger, leading to extreme classification problem which takes 30%-60% execution time overhead of the overall applications[22]. The category size of recent classification workloads can even reach 100 million, causing the classifier's parameters to consume hundreds of GBs [21, 28, 37, 42, 54], which exceeds the common main memory capacity in CPU or GPU systems [5, 24]. Thus, storage devices with much larger capacity are necessary to store the increasing amount of classification parameters, such as SSD [15, 33, 53]. However, the bandwidth from SSD to main memory is limited [16, 47, 50]. The CPU or GPU suffers from a long delay to transfer such a huge amount of data from SSD storage. As a result, the system performance is severely bound by the data

movement bottleneck if extreme classification is implemented in conventional computing architectures.

Emerging in-storage-computing technologies [19, 20, 25, 44] provide a promising solution to run the extreme classification efficiently. First, the storage capacity of SSD can reach TB level, which is sufficient to store all parameters in extreme classification. Besides, the computation units inside the storage device [20, 44] can directly utilize the much higher internal bandwidth, which both breaks down the external bandwidth limitation of conventional I/O and eliminates a large number of time-consuming data movements from the storage system to the main memory and computation units. But there exist challenges using in-storage-computing technology to deal with extreme classification. On the one hand, the computations are often huge due to the enormous classification category size, especially in full precision. On the other hand, the limited area budget makes inserting extra computation units inside SSD difficult. The mismatch between the limited compute resource and the high internal bandwidth in SSD, if not appropriately addressed, will inevitably lead to a waste of the internal bandwidth and a much longer computation delay to finish extreme classification.

One potential solution is to explore new algorithmic methods to eliminate computation redundancy. Using approximate screening algorithm [22] has been verified as an efficient method to reduce the computation amount without compromising accuracy. By approximate projection [22], we can first get a low-precision (e.g., in 4-bit) weight matrix for the huge classification layer. After first doing vector-matrix multiplication between projected input features and approximate weight matrix, we can compare the low-precision results with thresholds and decide which full-precision weight vectors will be used for further computation to get the final predictions. In this way, we can reduce the amount of floating-point computations to 10%, which helps a lot to mitigate the computation burden for in-storage-computing.

However, the remaining 10% floating-point computations in extreme classification still exceed the limited computing resources that could be added into SSD. Besides, the approximate screening algorithm also brings some new challenges to the targeted in-storage-computing scenario. If the 4-bit low-precision data and the 32-bit full-precision data used in the approximate screening algorithm are homogeneously stored in flash, the data transfer interference between these two data types could slow down the whole transfer process. Moreover, the imbalanced workloads in each flash channel caused by the discontinuous data access pattern of the algorithm will lower the actual channel-level bandwidth utilization. As shown in the roofline analysis in Fig. 1, the in-storage-computing baseline, indicated as point A, is compute-bound due to the limited computation capability of conventional floating-point MAC units. If the computation ability under the limited area budget can be promoted, the performance point will go higher to point B in Fig. 1. Then the original compute-bound problem shifts into the memory-bound scenario which is restricted by the data transfer interference and flash channel workload imbalance. We can further unleash the performance via improved utilization of internal bandwidth by mitigating data transfer interference and imbalanced accesses on flash channels, indicated as point C in Fig. 1.

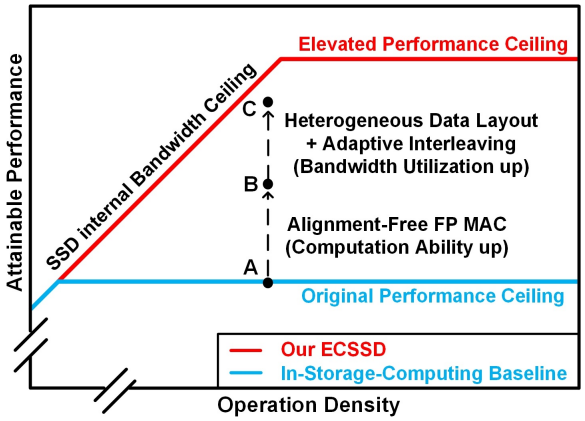

Figure 1: Roofline model analysis of our design vs. in-storage-computing baseline.

To summarize, we face three main challenges to implement the approximate-screening-based extreme classification with in-storage-computing technology:

- **Challenge 1**: Under limited chip area for extra logic circuits in SSD, the in-storage computing performance is limited and can not match SSD's high internal bandwidth, leading to under-utilization of available internal bandwidth and long processing latency.
- **Challenge 2**: If the two types of 4-bit/32-bit data introduced by approximate screening are stored homogeneously in NAND-based flash, the data transfer interference between the 4-bit and 32-bit data will slow down the whole processing.
- **Challenge 3**: The discontinuous access pattern of 32-bit weight data in the approximate screening algorithm leads to imbalanced workload on each flash channel in SSD, which is detrimental to the channel-level bandwidth utilization.

To tackle the above challenges, we propose a hardware/data-layout co-designed in-storage-computing architecture for extreme classification, based on the approximate screening algorithm. Our key insight is that instead of solving the problem purely at a single stage, we exploit the improvements at all different levels from circuit to architecture and to data layout in a synergistic manner. For the first time, we analyze and present the area budget guideline of additional logic circuits for in-storage-computing architectures. We propose a novel alignment-free floating-point MAC circuit design to significantly elevate the computational capability under the limited area budget. To avoid the data transfer interference between 4-bit and 32-bit data and fully utilize the internal bandwidth of SSD, we further propose a heterogeneous data layout and architecture design to decouple the two-type data storage, transfer, and computation. What is more, we explore balancing the accesses of each flash channel to enhance the actual channel bandwidth utilization for the in-storage-computing architecture. We creatively propose a learning-based adaptive interleaving framework to balance the access workload of each channel to further increase the channel-level bandwidth utilization and reduce total processing time.

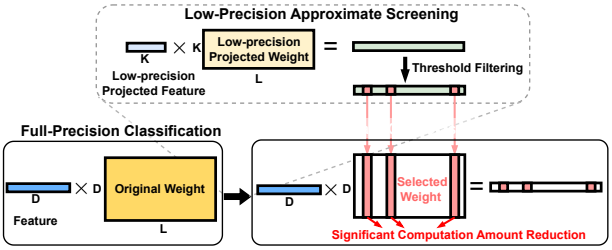

Figure 2: Approximate screening algorithm for extreme classification.

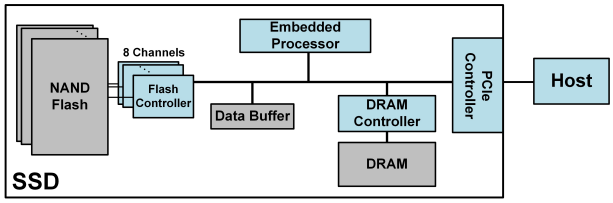

Figure 3: Original SSD structure.

Our contributions are as follows:

- We present the first in-storage-computing architecture for extreme classification, which comprehensively incorporates optimization across the circuit, architecture, and data layout levels.
- For **Challenge 1**, we propose a novel alignment-free floating-point MAC circuit technique that significantly enhances the in-storage-computing performance under limited area budget condition.
- For **Challenge 2**, we propose a heterogeneous data layout and architecture design to avoid the data transfer interference between 4-bit and 32-bit data and further utilize SSD's internal bandwidth.
- For **Challenge 3**, we present a learning-based adaptive interleaving framework to balance each flash channel's workload in SSD and further elevate the flash channel level bandwidth utilization.
- The evaluation results show our ECSSD achieves 3.24-49.87x performance improvements compared with the state-of-the-art baseline architectures.

We introduce the background in Section 2 and our motivation in Section 3. Architecture design and learning-based adaptive interleaving framework are discussed in Section 4 and 5. Evaluations are shown in Section 6.

2 BACKGROUND

In this section, we introduce the fundamental background for further discussion. We first introduce the approximate screening algorithm for extreme classification. Then, we present the technical background of the SSD organization.

2.1 Approximate Screening Algorithm

Dealing with the extreme classification [21, 37, 42] is becoming even more and more challenging for large-scale scenarios. Approximate computation [9, 23, 48, 49] is an efficient method to simplify the computation complexity of machine learning applications with acceptable accuracy drop. Recently approximation ideas are widely used in both software algorithm [40, 55] and hardware circuit [10, 11, 23]. For the extreme classification problem, approximation has also been proved as an efficient method [22].

In the extreme classification problem, it has been observed that not all the computation is important, or necessary [22]. Only a small part of the computation decides the final prediction results. For example, the final prediction results are only generated from the scope of top-k values with the highest probabilities [14]. So the approximate screening algorithm [22] is proposed to greatly reduce the amount of computation. First, a low-precision approximate screening module with shrunk hidden dimension is introduced to filter out the candidate full-precision weight vectors. Then the candidate-only classification module only computes the candidate-related classification under full precision to get the final predictions.

The approximate screening algorithm is shown in Fig. 2. There are two modules with different precision in the approximate screening algorithm. The full-precision classification module consists of a large 32-bit floating-point weight matrix with original high hidden dimension D and classification output dimension L. The approximate screening module is constructed by a projected small weight matrix with low shrunk hidden dimension K (D>K) and 4-bit integer precision, which is projected from the original 32-bit large weight matrix. For the inference process of the approximate screening algorithm, first, projected input features are multiplied with the projected low-precision weight matrix to get the approximate results. Then according to these approximate results, the importance of each floating-point weight vector is estimated. By comparing with the pre-trained threshold, the most important values of all the L approximate results are selected to decide which floating-point weight vectors are potential candidates. The selected candidate floating-point weight vectors further do full-precision computation with original input features in order to get the final accurate prediction results. With this low-precision approximate screening algorithm, the whole processing time can be reduced significantly with 10x speedup.

2.2 SSD Organization

Fig. 3 shows the internal structure of a common NAND flash based solid state drive (SSD). As a commonly-used storage device, SSD usually consists of NAND flash memory, flash controllers, embedded processor, data buffer, DRAM and a host interface controller [7, 19]. Nowadays, the capacity of SSD devices can easily reach TB level, or tens of TBs level, which is much higher than the capacity of main memory. And the huge capacity of SSD provided the most important cornerstone for dealing the large-scale extreme classification problem using the in-storage-computing scheme, specifically, the in-SSD-computing scheme.

There are multiple flash channels in the SSD. Each channel has one independent flash controller to control the data read/write of this channel. Thus, different channels can work independently and concurrently. Commonly, the hierarchical organization of NAND

flash SSD is channel, package, die, plane, block and page. The typical capacity of one page is 4kB. In SSD, the read and write operations happen at the page level. Erase operation happens at the block level.

The I/O bandwidth of SSD commonly is at single digit level GB/s, such as 4GB/s, which causes long storage access latency for huge data amount. For PCIe 3.0, the I/O bandwidth is only 1GB/s in each lane, which confines the speed of data movement from SSD to the host's main memory. The internal bandwidth of SSD is usually higher than its external I/O bandwidth. This is comprehensible that inside SSD, there is internal dataflow for many SSD management tasks. For each flash channel, it can reach 1GB/s internal bandwidth. And more flash channels can bring higher internal channel level bandwidth. For some high-end SSD products, they can have 16 flash channels.

The embedded processor with its firmware is mainly responsible for the flash translation layer, as known as FTL. The FTL implements the management functions for the whole SSD such as flash wear monitoring and garbage collection. It also executes logical to physical address mapping, which is necessary for data read/write. This address mapping function provides a foundation for our proposed data interleaving method, which is further discussed in Section 5. The data buffer in SSD is used for temporally buffering the data which is transferred between other main components in SSD such as NAND flash, embedded processor and interface controllers. The MB-level data buffer offers us a design opportunity that it can buffer the data for our inserted accelerator so that extra buffer overhead can be saved. The DRAM is used to store the SSD management data. Especially, the logical-to-physical address mapping table is maintained in DRAM. The capacity and the high bandwidth of DRAM provide a chance for our heterogeneous data layout design in Section 4.

3 MOTIVATION

In this section, we introduce the motivation for proposing our ECSSD architecture. We first analyze the limitation of the approximate screening algorithm for the in-storage-computing scheme. Then, we present the hardware opportunity and limitation of the in-storage-computing scheme.

3.1 Limitation of Approximate Screening Algorithm

Although the approximate screening algorithm can help to reduce the computation amount in extreme classification without accuracy drop, it brings new limitations to data access. Originally, all the 32-bit weight vectors are needed to compute with input features for the predictions. So the 32-bit weight data access pattern can be sequential, which is friendly to memory or storage device access. However, in the approximate screening algorithm, after low-precision screening and filtering, only a few full-precision weight vectors need to be fetched from storage for further computation. Such discontinuous 32-bit data access pattern brings the limitation to storage devices, especially the SSD: if all the 32-bit weight vectors are still simply stored continuously in the NAND flash of the SSD, then an imbalanced flash channel access pattern will be caused by the approximate screening algorithm, leading to lower channel level bandwidth utilization. Thus, to implement the approximate screening algorithm efficiently, the storage method of 32-bit weight vectors needs to be carefully designed, which is further discussed in Section 5.

3.2 Hardware Opportunity

The memory capacity needed for extreme classification parameters is very huge, which can reach hundred-GB level. If the extreme classification is implemented under conventional Von Neumann architecture, such exhausting data movement from SSD to main memory causes too long latency. Besides, state-of-the-art in-memory-computing or near-memory-computing schemes are also not suitable for dealing with extreme classification because commonly the memory capacity of these schemes is not enough to store the parameters of extreme classification.

The in-storage-computing scheme is a guaranteed method that:

- SSD's high storage capacity is enough for all parameters of extreme classification.
- Computing in SSD can directly avoid lengthy data movement out of SSD which is limited by SSD's low external I/O bandwidth.
- Computing in SSD can save the resources in the host so that the host can have enough ability to deal with other workloads.

With these guaranteed characters, the in-storage-computing scheme provides a hardware opportunity for extreme classification problems.

3.3 Limitation of In-Storage-Computing Scheme

Although directly doing computation in SSD can both take advantage of SSD's high internal bandwidth and avoid the huge amount of data movement between SSD and host's main memory, there exists limitation of the naive in-storage-computing scheme: the internal space of SSD is limited and the area budget for additional inserted computing resource is also limited. We do a comprehensive research on state-of-the-art industry storage solutions [3, 26] and academic in-storage-computing works [19, 20, 25, 44, 52] to find the acceptable and reasonable chip area budget for extra accelerator or computing logic inserted in SSD. However, since in-storage-computing field is still under development and far from maturity, until now there is no work that has explicitly clarified the area budget limitation for extra accelerator in SSD. It can be anticipated that the problem of area budget for the extra logic circuit in SSD will be more and more important as in-storage-computing accelerators become more and more complex.

To fill this gap, for the first time, we propose an area budget guideline for adding extra logic circuit into SSD to accelerate a specific application with lightweight and acceptable overhead. The area budget guideline is that: under the same manufacturing technology, the area of the additional logic circuit should not be larger than the area of the single embedded processor of the original SSD controller. There are two main reasons for choosing the area of the embedded processor as the standard of lightweight extra chip area budget. First, the embedded processor is always a necessary and major component for both industry solutions and academic architectures. Second, the embedded processor in the original SSD

architecture is a complicated logic that handles multiple SSD management tasks. Comparatively, the extra lightweight logic circuit just works as an accelerator for single specific application. For example, the ARM Cortex R5 [2] fabricated in 28nm technology is a common embedded processor used in many SSD products [26]. Under 28nm technology, 0.21mm^2 [2], the area of a single Cortex R5 processor, could be the area budget limitation standard for lightweight accelerator insertion.

With the restriction of area budget, the scale of the inserted logic circuit is limited, thus the computational ability that can be added into SSD is restricted. In order to implement the approximate screening algorithm in SSD and match the inserted computational ability to the high internal bandwidth of an 8-channel SSD, which is a common configuration for medium-end SSDs, at least 0.24mm^2 area would be needed only for the naive floating-point MAC circuit under 28nm technology. Further considering the chip area needed for other components such as control logic and INT4 MAC circuit, the total area must far exceed the 0.21mm^2 budget restriction. Thus, with the naive floating-point circuit technique, the computational ability under the limited area budget is not enough to match SSD's high internal bandwidth. Such mismatch leads to a waste of SSD's internal bandwidth resource and makes the in-storage-computing scheme for extreme classification to be a compute-bound problem, even with the help of approximate screening algorithm to reduce the computation amount. To fully utilize the internal bandwidth of SSD, improving the computational ability on limited area is necessary. Our computational ability improvement technique is discussed in Section 4.

4 ARCHITECTURE AND CIRCUIT DESIGN

In this section, we present the ECSSD architecture design. We first show the overall architecture, followed by the proposed alignment-free floating-point MAC circuit technique and heterogeneous data layout design. Then, we present the software integration and the whole workflow of the ECSSD.

4.1 Overall Architecture

To address the extreme classification problem, we propose a novel in-storage-computing architecture ECSSD. Fig. 4 shows the overall architecture of our ECSSD. Different from conventional SSD architecture in Fig. 3, we add customized accelerator near the data buffer in SSD. The INT4 MAC circuit and the comparator in the inserted accelerator are responsible for low-precision approximate screening and candidate filtering. The FP32 MAC circuit with our proposed alignment-free technique is responsible for full-precision candidate-only classification. The scheduler module controls the whole accelerator and coordinates with the FTL of ECSSD. The ECSSD architecture can work in two modes: SSD mode and accelerator mode. Simple changes are made to the original firmware of the embedded processor to support the FTL for both modes.

SSD Mode. In SSD mode, the working principle is very similar to the conventional SSD product. In this mode, the inserted accelerator is disabled and ignored. When receiving load/store commands from host, the ECSSD gets the logical address of the data and translates the logical address into the physical address in ECSSD. Then according to the physical address, the data is fetched from/programmed into the NAND flash. The embedded processor, working as FTL, controls the logical to physical address mapping, garbage collection and wear leveling. The DRAM works as a cache to store the L2P address translation table and other management information.

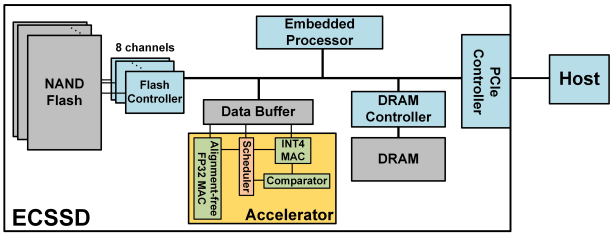

Figure 4: ECSSD architecture overview.

Accelerator Mode. In accelerator mode, the inserted accelerator is enabled and the ECSSD only works for accelerating the extreme classification application. The ECSSD only receives 4-bit/32-bit weight data and corresponding input features from host. The 4-bit/32-bit weight data is stored heterogeneously in ECSSD. The 4-bit weight data is only stored in DRAM and 32-bit weight data is only stored in NAND flash. The heterogeneous data layout design will be further discussed in Section 4.3. The original data buffer in SSD now works for the inserted accelerator to buffer the 4-bit/32-bit weight/input/result data and needed physical addresses for 32-bit candidate weight vector filtering. When inference starts, the accelerator's INT4 MAC circuit and alignment-free FP32 MAC circuit work together to finish the extreme classification with the approximate screening algorithm. The INT4 MAC circuit computes the results of the low-precision approximate screener. The comparator filters out the key candidates for further full-precision computation, according to the comparison results between pre-trained threshold and low-precision screening results. Then the candidate 32-bit weight vectors are fetched from NAND flash channels to do candidate-only classification in the alignment-free FP32 MAC circuit and finally get accurate prediction results. The proposed alignment-free FP32 MAC circuit will be discussed in detail in Section 4.2.

4.2 Alignment-free floating-point MAC Circuit

We have discussed in Section 3 that the area budget for the additional inserted logic circuit in SSD is limited and the maximum total area of inserted logic should be smaller than 0.21mm^2 under 28nm technology. Thus, optimizing the floating-point MAC circuit for ECSSD is necessary for two reasons. First, since the circuit of 32-bit floating-point MAC is much more complex than the circuit of 4-bit integer MAC, the area concern is laid on the floating-point MAC circuit. Second, if we obey the chip area budget requirement and directly implement naive floating-point MAC circuit into SSD, then the floating-point computational ability cannot match SSD's high internal bandwidth and the performance of the whole ECSSD is compute-bound.

By analyzing the area breakdown of the naive floating-point MAC circuit, we find that the proportion of alignment-related components, such as exponent comparators and mantissa shifters, is

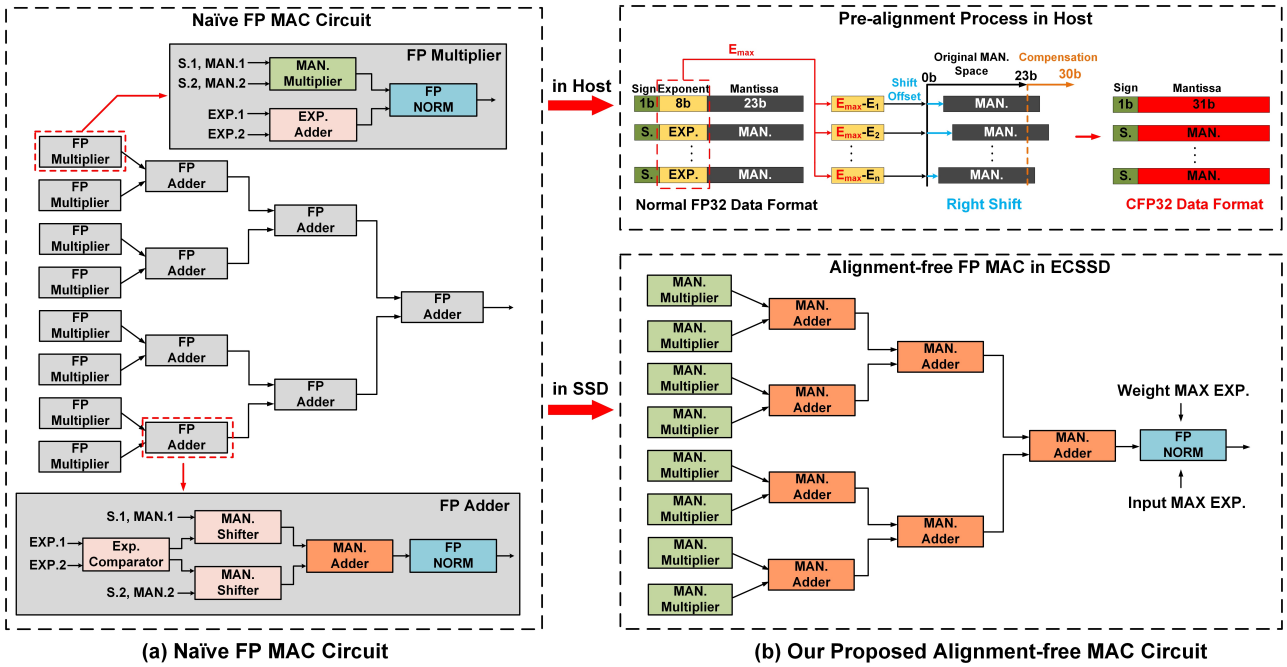

Figure 5: Naive FP MAC circuit VS. our proposed alignment-free FP MAC circuit

37.7%, which takes up significant overhead. Therefore, it is necessary to do optimization aimed at the alignment-related components to save chip area. Fig. 5(a) shows the details of the naive floating-point MAC circuit, which contains many area-consuming alignment-related components, especially mantissa shifters. For the floating-point accumulation part, it is actually an adder tree containing many separate floating-point adders. When the adder is dealing with two floating-point addends, it first compares the exponent parts of the two addends using an exponent comparator and gets the exponent difference. Then the mantissa is shifted for alignment by the mantissa shifter, according to the exponent difference. After the alignment process, the mantissa adder implements mantissa addition for the two addends. Finally, the exponent part and mantissa part of the result are normalized to satisfy the standard of floating-point number presentation.

From the whole computation process of the naive floating-point MAC circuit, we can see that there are too many alignment processes that lower the circuit area efficiency. Inspired by the state-of-the-art pre-alignment floating-point MAC circuit technique [18], we creatively propose our alignment-free methodology for the in-storage-computing scenario.

Our alignment-free floating-point MAC circuit in ECSSD associated with the pre-alignment process executed in host is shown in Fig. 5(b). First, the maximum exponent part E_{max} of each floating-point input feature vector is found. Then the mantissa parts of each floating-point data in the input vector are right-shifted according to the corresponding shifting offset (E_{max}-E). After shifting, all the floating-point values share the vector-wise maximum exponent E_{max} as their common exponent part. In this way, vector-wise alignment is achieved. There are two reasons that we implement the pre-alignment process in host. First, the floating-point input features of extreme classification are actually the intermediate computation results of previous model layers, which are executed in host. Second, although the alignment process is a severe challenge for ECSSD's inserted accelerator, it is trivial and easy for the powerful GPU, CPU, or FPGA host. We evaluate the vector-wise pre-alignment operation on an NVIDIA RTX 3090 GPU. For a typical 1x1024 floating-point feature vector, it only takes 0.005ms to finish the whole pre-alignment operation.

However, there may be a new concern that such pre-alignment will cause some information lost. Due to the right shifting, the least significant bits of original mantissa parts are dropped, which possibly does harm to classification accuracy. If compensation bits are directly added to keep the least significant bits, it will cause extra heavy data storage and transfer burden as all the floating-point data is much longer than before. To solve this problem, we present a new compensation floating-point data format Compensation FP32 (CFP32), shown in Fig. 5(b). In CFP32, instead of using the original 8-bit space in FP32 data format to store the repeated shared vector-wise maximum exponent value E_{max}, we keep the 8-bit space as the compensation bits for the 1-bit hidden one and the least significant bits that exceeds the original 23-bit mantissa scope in FP32. And the common 8-bit exponent value is stored separately and shared by all the floating-point data in the same vector. In this way, the computation accuracy can be guaranteed without extra heavy data storage or transfer overhead. The floating-point weight data is also offline pre-aligned into CFP32 data format before storing into flash.

We analyze the data distribution of real model and input feature parameters. Due to the value locality of deep learning models, with

Table 1: ECSSD API.

Type	API	Description
Preparation	ECSSD_enable/disable()	ECSSD working mode choice: SSD mode/accelerator mode
	Pre_align()	Pre-align the floating-point input/weight data into CFP32 format in host
	Weight_deploy()	Deploy the 4/32-bit weight data into ECSSD
Transmission	INT4_input_send()	Send 4-bit input vector to ECSSD
	CFP32_input_send()	Send 32-bit input vector to ECSSD
	Get_results()	Get the final classification results from ECSSD
Computation	INT4_screen()	Low-precision computation and filtering in ECSSD
	CFP32_classify()	Full-precision computation in ECSSD
	Filter_threshold()	Set the filtering threshold for low-precision screening

the 7-bit mantissa compensation, more than 95% of the floating-point data has no bit information lost. Even for the minimum outlier, the CFP32 still keeps most of its mantissa bits. We also experiment on real extreme classification benchmarks listed in Section 6. The results show that there is no classification accuracy drop, compared with the original FP32 computation method.

Associated with the proposed pre-alignment process and the CFP32 data format, the in-storage alignment-free FP MAC circuit can directly use the shifted mantissa parts of the weight and input data to implement MAC operation. In this way, original area-consuming alignment-related components are eliminated, shown in Fig. 5(b). Even though the precision of the mantissa multiplier increases from 24 bits to 31 bits, causing a little more area consumption than the original mantissa multiplier, the overall area saving achieved by the alignment-free FP MAC circuit is still significant. Detailed evaluation is shown in Section 6.4. We test on the LSTM-W33K benchmark listed in Section 6. Originally, a computational ability of 34.8GFLOPS is needed to consume the floating-point data transferred from flash channels without any delay. However, the naive FP32 MAC circuit can only achieve 29.2GFLOPS performance under the limited area budget. With our technique, we can achieve 50GFLOPS performance under the same area budget. In conclusion, under the same area budget limitation, with the help of our alignment-free FP MAC technique, the floating-point computational ability is significantly improved and able to match SSD's high internal channel level bandwidth, which overcomes the compute-bound problem. The performance of ECSSD now turns into a memory-bound problem, instead of a compute-bound scenario, shown as point A moving to point B in Fig. 1.

4.3 Heterogeneous Data Layout Design

Considering the whole data transfer process, even with the help of the approximate screening algorithm that can reduce the amount of floating-point data transfer significantly, the transfer workload of 32-bit floating-point candidate weight vectors is still much heavier than the transfer workload of 4-bit integer screener weight vectors. Besides, if we store the two kinds of weight data homogeneously that both 4-bit and 32-bit weight data are stored in the NAND flash channels, there will be data transfer interference when both 4-bit weight data and 32-bit weight data need to be transferred from the NAND flash to corresponding computing logic simultaneously. Such data transfer interference further slows down the 32-bit data movement and attenuates the system performance.

To eliminate the data transfer interference and accelerate the whole data transfer process, we propose the heterogeneous data layout design that 4-bit integer weight vectors are stored in DRAM and 32-bit floating-point weight vectors are stored in NAND flash channels. So that all the channel level bandwidth can be used for the heavy transfer workload of 32-bit floating-point weight data and extra DRAM bandwidth can be utilized for 4-bit integer weight data. And now these two kinds of data can be transferred to the accelerator simultaneously without data transfer interference. Besides, the SSD's internal bandwidth resource is further utilized.

4.4 Software Integration

We design a software library to coordinate the execution between the host and the ECSSD and take full advantage of the in-storage-computing accelerator. The major Python-style APIs for ECSSD are listed as Table 1, which could be integrated with existing machine learning frameworks flexibly. There are mainly three types of APIs that focus on working preparation, data transmission, and in-storage computation.

Preparation-related APIs are applied to select the working mode of ECSSD, pre-align the floating-point weight and input data, and deploy the 4-bit and 32-bit weight data into ECSSD. Transmission-related APIs work for input data sending and classification result gathering. Computation-related APIs are responsible for the dual-precision in-storage computation.

4.5 Workflow

To support the above proposed techniques, we design the workflow of the ECSSD. The `ECSSD_enable/disable` API is used to decide the working mode of ECSSD. For the SSD mode, as the inserted accelerator is ignored, the workflow is just similar to a conventional SSD product. For the accelerator mode, the workflow is much different. In the data preparation period of accelerator mode, the preparation-type APIs are utilized to deploy 4-bit weight data into ECSSD's DRAM from host. All the offline pre-aligned 32-bit floating-point weight data is deployed into the NAND flash channels, according to the proposed learning-based interleaving framework, discussed in Section 5. The L2P address mappings and index correspondence between 4-bit and 32-bit weight vectors are kept in DRAM. After all the 4-bit/32-bit weight data is loaded into ECSSD, the data preparation finishes and inference starts.

During inference, the transmission-type and computation-type APIs are applied. Both approximate screener and candidate-only

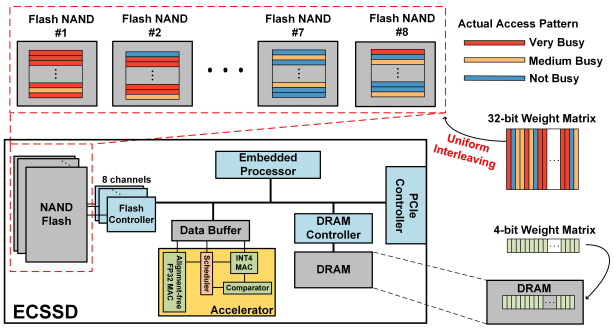

Figure 6: Uniform interleaving method. All the 32-bit weight vectors are uniformly stored into 8 flash channels.

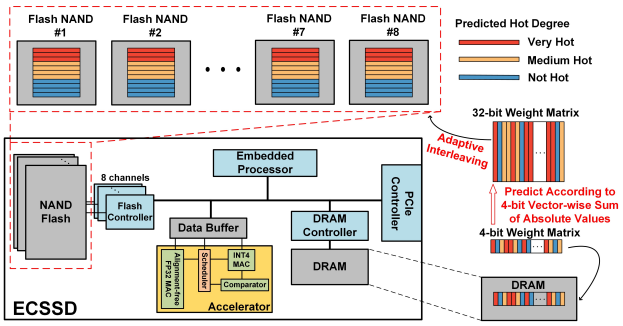

Figure 7: Learning-based interleaving method. For a nearly balanced channel access pattern, all the 32-bit weight vectors are interleaved according to the predicted hot degrees.

classification are implemented tile-by-tile. Each time, a batch of 4-bit input vectors is loaded into the data buffer from the host and computed with one tile of 4-bit weight data, which is fetched from DRAM together with the index and physical address information of corresponding 32-bit weight vectors. The low-precision results then are filtered according to the threshold set by the `Filter_threshold` API to generate the candidates for later full-precision classification. Then the physical addresses of candidate 32-bit weight vectors are passed to flash controllers from the data buffer. Corresponding candidate 32-bit weight vectors are fetched from flash channels and loaded into the data buffer. At the same time, the corresponding batch of pre-aligned 32-bit floating-point input features is loaded into the data buffer from host. Finally, the alignment-free FP MAC circuit in the accelerator finishes the candidate-only classification and sends the final results back to the host. For both 4-bit and 32-bit processes, the data buffer works in a ping-pong manner to overlap the buffer read and write in order to reduce the whole execution time. The processing of the INT4 MAC circuit and alignment-free FP32 MAC circuit can be overlapped, which means that when the FP32 MAC circuit is computing with the first weight tile, the INT4 MAC circuit is computing with the second weight tile. In this way, the whole dual-module processing is continuous, and the total processing time can be reduced significantly.

5 LEARNING-BASED ADAPTIVE INTERLEAVING FRAMEWORK

In this section, we first analyze the limitations of two data storing methods including sequential storing and uniform interleaving. Then, we present the learning-based adaptive interleaving framework for the imbalanced data access issue.

5.1 Sequential Storing

Naturally, it is the most straightforward way to sequentially divide and store the whole 32-bit weight matrix into all the flash channels. However, the whole computation is a tile-by-tile process. Mostly, the candidate 32-bit weight vectors in adjacent tiles are stored in the same flash channel. Thereby, only one flash channel is accessed each time and other channels are idle, wasting the channel level bandwidth resource.

5.2 Uniform Interleaving

To make all the 8 flash channels working in parallel and fully utilize the channel level bandwidth, uniform interleaving is a possible solution which is shown in Fig. 6. It means uniformly interleaving all the 32-bit weight vectors into 8 channels. For example, interleave No.1 ~ No.8 32-bit weight vectors into No.1 ~ No. 8 flash channels, and then interleave No.9 ~ No.16 32-bit weight vectors into No.1 ~ No.8 flash channels, and so on. Compared with sequential storing, now the 8 flash channels can work in parallel for data access and the total channel level bandwidth utilization is improved. However, the flash channel access pattern under the uniform interleaving method is not balanced as the results of candidate filtering are discrete. In most cases, there are some channels containing more candidate 32-bit weight vectors than others, causing the imbalanced data access workload. The access time of some channels is much longer than others, which is evaluated in Section 6.6. For one specific weight tile, the final data access time is decided by the busiest flash channel. Such an imbalanced data access pattern in uniform interleaving method results in that the total channel level bandwidth utilization is still far away from ideal.

5.3 Learning-based Adaptive Interleaving

As the final solution, we propose the learning-based adaptive interleaving framework that utilizes the information of projected 4-bit weight vectors to predict the hot degree of corresponding 32-bit weight vectors and balance the data access workload of each flash channel in ECSSD. The learning-based adaptive interleaving method is shown in Fig. 7. For a specific 32-bit weight vector, the hot degree reveals the possibility of being selected as a candidate for further full-precision computation. For example, very hot means that the specific 32-bit weight vector is very possible to be selected as a candidate. According to the sum of the absolute value of each element in each 4-bit weight vector, corresponding 32-bit weight vectors are divided into 3 grades: very hot, medium hot and not hot. The larger the sum of absolute value is, the more possible that the corresponding 32-bit vector will be selected as a candidate. After that, the hot degrees of weight vectors are further fine-tuned according to the frequency of being filtered as a candidate on the training dataset. With the goal of making the workload in each flash channel nearly the same, all the weight vectors are interleaved into

Table 2: ECSSD Configurations

ECSSD			
Flash Capacity	4TB	Flash Channels	8
DRAM Capacity	16GB	Page Size	4KB
Accelerator	1	Data buffer	4MB
Flash Protocol	NVDDR3	Interface	PCIe 3.0 x4
Accelerator			
Technology	28nm	Voltage Supply	0.9V
Frequency	400MHz	Index Buffer	4KB
Weight Buffer(INT4)	128KB	Output Buffer(INT4)	2KB
Input Buffer(INT4)	4KB	Input Buffer(FP32)	100KB
Weight Buffer(FP32)	400KB	Output Buffer(FP32)	1KB
FP32 MAC	64	INT4 MAC	256
Comparator	1	Scheduler	1

the 8 flash channels in the light of fine-tuned hot degrees. Originally, the firmware of the embedded processor allocates a specific range of logical addresses to each flash channel. The framework only needs to assign a logical address from the specified logical address range to the specific 32-bit weight vector. Then the FTL supported by the embedded processor can directly help to interleave these 32-bit vectors into designated channels as the framework indicates. Finally, the flash channel access pattern is almost balanced and the total data access time is reduced significantly.

6 EVALUATION

In this section, We first introduce our evaluation methodology. Then, we evaluate the area and power consumption of the ECSSD accelerator. After that, we show the end-to-end performance improvement. We further evaluate our three main contributions including alignment-free MAC circuit, heterogeneous data layout and learning-based adaptive interleaving. Finally, we compare our ECSSD with other state-of-the-art architectures on large-scale benchmarks.

6.1 Methodology

Software Implementation. We use PyTorch framework [36] to implement the approximate screening algorithm. According to the sensitivity study in [22], we set the projection scale of hidden dimension as 0.25 and the precision of the screener to be 4-bit integer to have a good-quality classification.

Hardware Implementation. We implement the inserted accelerator of ECSSD in RTL and synthesize with Design Compiler under 28nm technology to get the area and power evaluation results. We build a simulator that can interface with MQSim [46] to evaluate the performance of the ECSSD system. We also simulate a simple host to coordinate with ECSSD.

ECSSD Configurations. The configuration of ECSSD is shown as Table 2. Here we adopt a medium-end SSD configuration for our ECSSD. With 8 flash channels and a page size of 4KB, the total capacity of ECSSD is 4TB. The bandwidth of each flash channel is set as 1GB/s and the bandwidth of the DRAM is 12.8GB/s. The peak performance of the INT4 MAC circuit/alignment-free FP32 MAC circuit is 200GOPS/50GFLOPS respectively.

Benchmarks. We implement different models and datasets on different applications such as language processing [29], neural machine translation [45] and recommendation [27], similar to ENMC [22]. We also synthesize 3 larger datasets with 10M, 50M and 100M category sizes to evaluate the performance of ECSSD. The benchmark details are in Table 3. For LSTM-W33K, the original hidden size is 1500. For Transformer-W268K and XMLCNN-670, the original hidden size is 512. For all the other benchmarks, the original hidden size is 1024. Take the XMLCNN-S100 benchmark as an example, the hidden size of the 4/32-bit weight matrices are 256/1024 respectively. Thus, the sizes of its 4/32-bit weight matrices are 12.8GB/400GB respectively.

Table 3: Benchmark models and datasets.

Model	Dataset	Category Size	Abbr.
GNMT	WMT16	32,317	GNMT-E32K
LSTM	Wikitext-2	33,278	LSTM-W33K
Transformer	Wikitext-103	267,744	Transformer-W268K
XMLCNN	Amazon-670k	670,091	XMLCNN-A670K
XMLCNN	S10M	10,000,000	XMLCNN-S10M
XMLCNN	S50M	50,000,000	XMLCNN-S50M
XMLCNN	S100M	100,000,000	XMLCNN-S100M

6.2 Evaluation for Area and Power Consumption

Table 4 shows the breakdown area and power evaluation of the accelerator in ECSSD. The total area of inserted logic is 0.18mm^2, which satisfies the area budget limitation discussed in Section 3.3. The total power is 52.93mW, which is comparable to state-of-the-art in-storage-computing architectures such as GenStore [25]. Concretely, the alignment-free FP32 MAC circuit takes 75.7% of the total area and 63.9% of the total power. The INT4 MAC circuit and the threshold comparator take 24% of the total area and 36% of the total power. We also evaluate that, to achieve the same performance, the naive FP32 MAC circuit needs 0.24mm^2 area and 51.8mW power.

Table 4: Area and Power Estimation.

	Area (mm^2)	Power (mW)		Area (mm^2)	Power (mW)
FP32 MAC	0.139	33.87	**INT4 MAC**	0.044	19.04
Comparator	0.0004	0.016	**Scheduler**	0.0002	0.004
Total Area 0.1836mm^2; Total Power 52.93mW					

6.3 Evaluation for End-to-end Performance Improvement

In this part, we demonstrate the increment of each proposed technique step by step. The average results on all the benchmarks are shown in Fig. 8. The starting baseline, on the left side of Fig. 8, is equipped with naive FP MAC circuit, sequential storing method together with homogeneous data layout that both 4-bit and 32-bit weight data are stored in NAND flash. We can observe that, as only one channel is accessed each time, the channel bandwidth utilization for floating-point data transfer is lower than 10%. With the help of the uniform interleaving method, all the 8 channels can work in parallel, and the channel bandwidth utilization raises to 44.31%, leading to a 4.06x total speedup. Then with the help of the alignment-free FP Mac circuit technique, the floating-point computing time can be hidden by the memory access time. Furthermore, with the help of heterogeneous data layout design, the

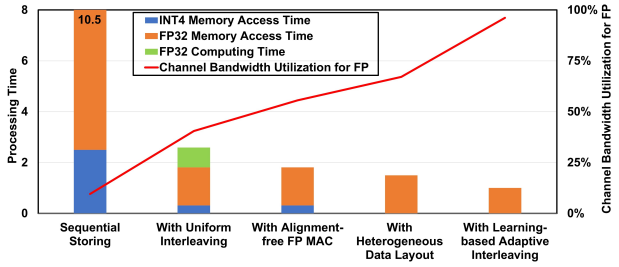

Figure 8: Breakdown analysis for each proposed technique in ECSSD.

4-bit data transfer and 32-bit data transfer are separated, which leads to the channel level bandwidth utilization raising to 67.6% for floating-point data transfer. Finally, with the help of the proposed learning-based adaptive interleaving technique, the channel level bandwidth utilization for floating-point data transfer raises to 94.7%. Compared with the starting baseline equipped with the sequential storing method, it finally achieves 10.5x speedup.

6.4 Evaluation for Alignment-free FP MAC Circuit

We implement the naive FP MAC circuit and our proposed alignment-free FP MAC circuit that are discussed in Section 4.2 with RTL code synthesized using Design Compiler. We also implement SK Hynix's optimized FP MAC circuit [18] for comparison. Their FP MAC circuit's key idea is that the mantissa parts of all the FP products are aligned after the FP multiplication, before FP addition. In this way, the number of area-consuming shifter components in FP adders can be reduced by half to save area.

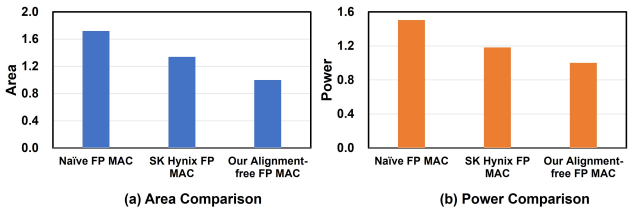

Figure 9: Area and power comparison of naive, SK Hynix's and our alignment-free floating-point MAC circuit.

The normalized area and power comparison results of these three FP MAC circuit techniques are shown in Fig. 9. To achieve the computational performance that can match SSD's high internal bandwidth, the naive FP MAC circuit and SK Hynix's FP MAC circuit respectively need 1.73x and 1.38x more area than our proposed alignment-free FP MAC circuit. Besides, the naive FP MAC circuit and SK Hynix's FP MAC circuit need 1.53x and 1.19x more power than ours. The obvious area and power benefits owe to our alignment-free FP MAC technique that can not only eliminate all the area-consuming alignment-related components such as mantissa shifters and exponent comparators in the FP adders but also simplify the FP multiplier into INT mantissa multiplier.

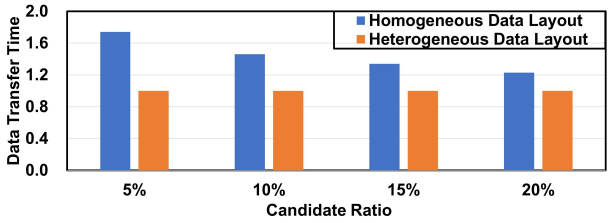

Figure 10: Data transfer time comparison of homogeneous data layout design and heterogeneous data layout design.

6.5 Evaluation for Heterogeneous Data Layout

The purpose of this experiment is to evaluate the performance improvement contributed by heterogeneous data layout design. For the baseline homogeneous data layout, both 32-bit weight data and 4-bit weight data are stored in NAND flash. We use the Transformer-W268K benchmark with different candidate ratio settings: 5%, 10%, 15% and 20% to implement the experiment. The speedup results are shown in Fig. 10. When the candidate ratio is 5%, the heterogeneous design can achieve 1.73x speedup over the homogeneous one. On average, our heterogeneous data layout design can achieve 1.43x speedup than the homogeneous baseline, under these 4 practical candidate ratio settings.

The speedup of heterogeneous design comes from the fact that in the heterogeneous scenario, the data transfer processes of 4-bit weight data and 32-bit weight data are completely separated. All the channel level bandwidth is utilized for the 32-bit weight data transfer and extra DRAM bandwidth is utilized to transfer 4-bit weight data. In this way, SSD's internal bandwidth is further utilized and the data transfer interference between 4-bit and 32-bit weight data is avoided.

6.6 Evaluation for Learning-based Adaptive Interleaving

Fig. 11 shows the data access pattern comparison between the uniform interleaving method and the learning-based adaptive interleaving method. Here we test these two methods on one specific 32-bit weight data tile in GNMT-E32K benchmark with 10% candidate ratio. The result shows that our learning-based adaptive interleaving method can achieve a more balanced data access pattern, which means more balanced workload in each NAND flash channel.

Fig. 12 shows the performance comparison of the three different storing strategies mentioned in Section 5 on four benchmarks. On average, our proposed learning-based adaptive interleaving method can achieve 1.43x performance improvement over the uniform interleaving method and 7.57x performance improvement over the sequential storing method. The significant improvement comes from the fact that our learning-based interleaving method can achieve better channel level parallelism than the sequential storing method, in which only one channel is accessed each time. Besides, the more balanced channel access pattern of learning-based interleaving efficiently improves the channel bandwidth utilization, leading to performance improvement over uniform interleaving.

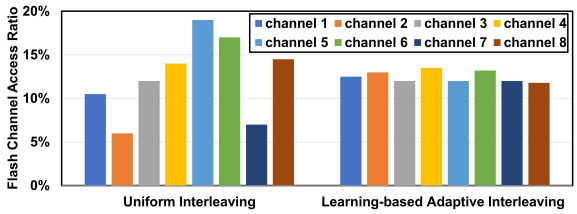

Figure 11: Flash channel access patterns of uniform interleaving method and learning-based interleaving method.

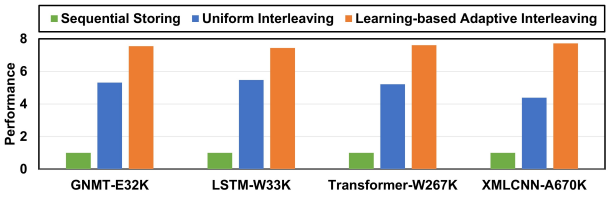

Figure 12: Performance comparison of sequential storing, uniform interleaving and learning-based adaptive interleaving method.

6.7 Comparison with Other Architectures

Baselines. GenStore [25] is a famous in-storage-computing architecture for private information retrieval. It leverages the accelerators inserted into SSD to accelerate query processing. Especially, there is a proprietary accelerator for each channel. Thus, for an 8-channel SSD, there are totally 8 channel-level accelerators and each of them works independently without inter-channel communication. SmartSSD platform [17, 38, 39] is a computational storage system with programmable acceleration ability that integrates Xilinx's FPGA near Samsung's SSD. The FPGA is connected to the SSD through a 3GB/s PCIe switch.

We comprehensively compare our ECSSD with 8 baselines including CPU-based baselines, GenStore-like baselines and SmartSSD-based baselines. Condisering Samsung and Xilinx would further equip their following SmartSSD products with higher bandwidth, here we also build simulation baselines SmartSSD-H-N and SmartSSD-H-AP with higher 6GB/s bandwidth between FPGA and SSD for bandwidth sensitivity study. For a fair comparison, we construct our ECSSD and the two Genstore-like baselines with almost the same area for computing logic under 28nm technology. The construction details of all the baselines are indicated below.

CPU-N & CPU-AP. The CPU baselines are based on the Intel Xeon Silver 4110 CPU. The CPU-N is a naive baseline without the approximate screening algorithm. The CPU-AP is equipped with the approximate screening algorithm.

GenStore-N & GenStore-AP. The GenStore-AP/GenStore-N is an in-storage-computing baseline with/without the approximate screening algorithm. For both GenStore-N and GenStore-AP, separated naive channel-level FP32 accelerators are equipped for each flash channel and all the data is stored in the 8 flash channels uniformly. GenStore-AP has an extra SSD-level INT4 accelerator.

SmartSSD-N & SmartSSD-H-N. The SmartSSD-N and SmartSSD-H-N are naive near-storage-computing baselines without the approximate screening algorithm. For both of them, the computational resource on the equipped FPGA is configured as a naive FP32 accelerator.

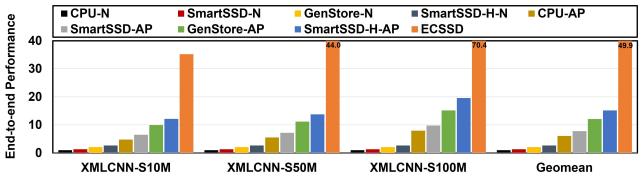

Figure 13: End-to-end performance comparison with other architectures.

SmartSSD-AP & SmartSSD-H-AP. For both SmartSSD-AP and SmartSSD-H-AP, the approximate screening algorithm is equipped and the computational resource on FPGA is configured as an INT4 accelerator and a naive FP32 accelerator.

Comparison Results and Analysis. As shown in Fig. 13, we run three large-scale extreme classification benchmarks for the nine architectures. On average, our ECSSD achieves 49.87x, 37.83x, 24.51x, 19.11x, 8.22x, 6.28x, 4.05x and 3.24x performance improvement over CPU-N, SmartSSD-N, GenStore-N, SmartSSD-H-N, CPU-AP, SmartSSD-AP, GenStore-AP and SmartSSD-H-AP respectively.

Compared with our in-storage-computing ECSSD, the performance of CPU-N and CPU-AP is severely bounded by the low SSD I/O bandwidth, suffering a huge amount of lengthy data movement from SSD storage to main memory and later to the caches and computing units in CPU cores.

Compared with SmartSSD-N, SmartSSD-H-N and GenStore-N, the benefits of our ECSSD mainly come from three points. First, the three baselines do not adopt the approximate screening algorithm to reduce the amount of floating-point computation. Second, the performance of the naive FP MAC circuit in GenStore-N is lower than the performance of the alignment-free MAC circuit in our ECSSD. Third, although the computational performance in SmartSSD-N is powerful enough, the bandwidth between its SSD and FPGA restricts the overall performance.

Compared with SmartSSD-AP, SmartSSD-H-AP and GenStore-AP, the benefits of our ECSSD are attributed to multiple facts. First, the uniform interleaving method of the GenStore-AP causes the imbalanced channel access pattern in SSD, which lowers the channel bandwidth utilization and slows down the total data transfer speed. Second, the random floating-point data access in SmartSSD-AP/SmartSSD-H-AP also slows down the overall performance. With higher bandwidth in SmartSSD-H-AP, the slow data transfer issue in SmartSSD-AP is alleviated, showing the powerful potential of future upgraded SmartSSD products. Third, the homogeneous data layout in the three baselines causes data transfer interference between 4-bit and 32-bit weight data, causing longer processing time.

7 DISCUSSION

7.1 Scalability

Scaling up. In this subparagraph, we mainly discuss the scalability of the DRAM capacity in a single ECSSD. To avoid transfer interference between 4-bit and 32-bit data, ECSSD's DRAM is utilized to hold the 4-bit weight matrix. We comprehensively study

the size of popular extreme classification layers [21, 37, 42] and design the DRAM with 16GB capacity so that the DRAM can hold the 12.8GB 4-bit weight matrix of the large 100-million-category scenario. Here we also analyze the scenarios with 8GB or 32GB DRAM capacity. For the 8GB DRAM capacity scenario, the maximum scale of the extreme classification that can be deployed in a single ECSSD would be severely bound to 50-million categories, limiting the usability of ECSSD for popular extreme classification problems. For the 32GB DRAM capacity scenario, the maximum scale of the deployed extreme classification in a single ECSSD could be 200-million categories. However, the larger DRAM would cause at least 40% increase in power consumption, which is not a trivial cost for ECSSD. Overall, 16GB DRAM capacity in ECSSD is a sweet-point design choice that balances both hardware cost and maximum classification scale supported.

Scaling out. As the scale of extreme classification problem keeps growing in the real world, we also propose a scale-out method that we would partition the larger classification layer into multiple ECSSDs and do the execution in parallel. For example, when the category size goes up to 500 million, the 4/32-bit weight matrix would be 64GB/2TB respectively, which far exceeds the capacity of the server main memory. With our scale-out ECSSD system, the huge classification layer will be partitioned into 5 ECSSDs for parallel execution so that in each ECSSD the DRAM is still enough to hold the 4-bit weight data. In this way, we only need to extend the number of cost-efficient ECSSDs in one server system instead of adding more expensive server systems.

7.2 Comparison with GPU

GPUs are widely used to accelerate compute-intensive NN tasks. However, for the data-intensive extreme classification problem, even if we use a high-end NVIDIA RTX 3090 GPU [35], its 24GB embedded memory is far from enough to hold all the parameters of the huge classification layer. Its performance has to suffer the lengthy data movement from the storage, similar to the CPU. Even one single RTX 3090 consumes 32x higher power than our ECSSD scheme. If we want to completely eliminate the slow data movement from storage, multiple GPUs with the model partition method are needed to hold all the model parameters and execute in parallel. However, even for the popular 100-million-category classification problem, at least 18 RTX 3090 GPUs are needed, causing at least 573x higher power consumption than our ECSSD scheme. Only if both the performance requirements and cost budget are extremely high, it is suitable to adopt the multiple-GPU scheme for extreme classification. Otherwise, our ECSSD scheme is a better choice with much higher power efficiency.

7.3 Comparison with Near-DRAM-Computing ENMC

ENMC [22] is a near-DRAM-computing accelerator that adopts approximate screening algorithm to implement extreme classification application. The ENMC accelerator circuit is placed at each rank of the DRAM to utilize the high rank-level bandwidth. Consisting of totally 64 DRAM ranks, the ENMC is a huge and expensive 512GB near-DRAM-computing system with 800 GFLOPS peak performance. Due to the intrinsic differences between flash media and DRAM media, ENMC can achieve higher peak performance than our single ECSSD. However, our ECSSD can achieve much better energy efficiency and cost efficiency than ENMC. Behind ENMC's high peak performance and high DRAM capacity, ENMC at least costs 154x larger 28nm fabricated chip area, 19.1x higher power and 10x more expensive memory infrastructure than our ECSSD[1, 30–32]. In detail, our ECSSD achieves 0.018GFLOPS/dollar and 4.55GFLOPS/W while ENMC only achieves 0.002GFLOPS/dollar and 3.805GFLOPS/W. With 8.87x higher cost efficiency and 1.19x higher energy efficiency, our ECSSD is more suitable for power-limited or money-limited scenarios. Besides, when the scale and footprint of the extreme classification keep growing, it is hard to continue extending ENMC's huge 512GB DRAM capacity to hold more parameters and its end-to-end performance would be severely degraded by the lengthy data movement from storage to ENMC. However, our ECSSD scheme can still be scaled out efficiently without performance degradation, which has been discussed in the scaling out part of Section 7.1.

8 RELATED WORK

In-Storage-Computing Systems. As the memory footprint of many data-intensive applications keeps growing, emerging in-storage computing becomes a suitable solution due to SSD's high capacity and high internal bandwidth.[19, 20, 25, 44, 52]. Our ECSSD is the first in-storage-computing system for the extreme classification problem which explores deeper on data access patterns, channel level bandwidth utilization, and how to improve the computational ability under limited internal overhead budget.

Flash-centric accelerator. Behemoth[12] is an NLP training accelerator that tightly couples its novel flash memory system with powerful computational cores. To achieve the 840TFLOPS high peak performance, Behemoth builds 524288 computational units with high hardware cost and modifies the original SSD architecture heavily for higher bandwidth, such as much more parallel flash channels than common SSD and simplified hardware-based memory controller instead of the original FTL in SSD. Compared with Behemoth, we adopt different design philosophies: we design our ECSSD based on the conventional SSD architecture with as lightweight modifications as possible. Essentially, Behemoth and ECSSD are suitable for different usage scenarios. ECSSD mainly targets cost-limited and power-limited scenarios. And Behemoth is good for scenarios with high cost budgets and extremely high performance requirements.

9 CONCLUSION

In this work, we employ a hardware/data layout co-design approach to propose the in-storage-computing architecture ECSSD that addresses the extreme classification problem. We believe that ECSSD can motivate more innovations and thinkings for improving the computational ability and internal bandwidth utilization of future in-storage-computing systems.

ACKNOWLEDGMENTS

We thank our shepherd and all the reviewers for the constructive comments. This work was supported in part by Samsung Semiconductor, Inc.

REFERENCES

[1] Amazon. 2022. Price of typical 512GB DDR4 DRAM memory. https://www.amazon.com/512GB-8x64GB-288-Pin-Reduced-Memory/dp/B07X22FZJJ

[2] ARM. 2011. ARM R5 Processor. https://developer.arm.com/Processors/Cortex-R5

[3] ARM. 2022. ARM SSD Controller Solutions. https://www.arm.com/solutions/storage

[4] Rahul Chauhan, Kamal Kumar Ghanshala, and R.C Joshi. 2018. Convolutional Neural Network (CNN) for Image Detection and Recognition. In *2018 First International Conference on Secure Cyber Computing and Communication (ICSCCC)*. 278–282. https://doi.org/10.1109/ICSCCC.2018.8703316

[5] Jack Choquette, Wishwesh Gandhi, Olivier Giroux, Nick Stam, and Ronny Krashinsky. 2021. NVIDIA A100 Tensor Core GPU: Performance and Innovation. *IEEE Micro* 41, 2 (2021), 29–35. https://doi.org/10.1109/MM.2021.3061394

[6] Alexis Conneau, Holger Schwenk, Loıc Barrault, and Yann Lecun. 2016. Very deep convolutional networks for natural language processing. *arXiv preprint arXiv:1606.01781* 2, 1 (2016).

[7] Michael Cornwell. 2012. Anatomy of a solid-state drive. *Commun. ACM* 55, 12 (2012), 59–63.

[8] Wu Hao, Rongfang Bie, Junqi Guo, Xin Meng, and Shenling Wang. 2018. Optimized CNN Based Image Recognition Through Target Region Selection. *Optik* 156 (2018), 772–777. https://doi.org/10.1016/j.ijleo.2017.11.153

[9] Ali Ibrahim, Mario Osta, Mohamad Alameh, Moustafa Saleh, Hussein Chible, and Maurizio Valle. 2018. Approximate computing methods for embedded machine learning. In *2018 25th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*. IEEE, 845–848.

[10] Mohsen Imani, Ricardo Garcia, Saransh Gupta, and Tajana Rosing. 2018. Rmac: Runtime configurable floating point multiplier for approximate computing. In *Proceedings of the International Symposium on Low Power Electronics and Design*. 1–6.

[11] Mingu Kang, Sujan K Gonugondla, and Naresh R Shanbhag. 2020. Deep in-memory architectures in SRAM: An analog approach to approximate computing. *Proc. IEEE* 108, 12 (2020), 2251–2275.

[12] Shine Kim, Yunho Jin, Gina Sohn, Jonghyun Bae, Tae Jun Ham, and Jae W. Lee. 2021. Behemoth: A Flash-centric Training Accelerator for Extreme-scale DNNs. In *19th USENIX Conference on File and Storage Technologies (FAST 21)*. USENIX Association, 371–385. https://www.usenix.org/conference/fast21/presentation/kim

[13] Akshay Kulkarni and Adarsha Shivananda. 2021. *Deep Learning for NLP*. Apress, Berkeley, CA, 213–262. https://doi.org/10.1007/978-1-4842-7351-7_6

[14] Maksim Lapin, Matthias Hein, and Bernt Schiele. 2017. Analysis and optimization of loss functions for multiclass, top-k, and multilabel classification. *IEEE transactions on pattern analysis and machine intelligence* 40, 7 (2017), 1533–1554.

[15] Hyeon Gyu Lee, Juwon Lee, Minwook Kim, Donghwa Shin, Sungjin Lee, Bryan S. Kim, Eunji Lee, and Sang Lyul Min. 2021. SpartanSSD: a Reliable SSD under Capacitance Constraints. In *2021 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*. 1–6. https://doi.org/10.1109/ISLPED52811.2021.9502476

[16] Joo Hwan Lee, Hui Zhang, Veronica Lagrange, Praveen Krishnamoorthy, Xiaodong Zhao, and Yang Seok Ki. 2020. SmartSSD: FPGA Accelerated Near-Storage Data Analytics on SSD. *IEEE Computer Architecture Letters* 19, 2 (2020), 110–113. https://doi.org/10.1109/LCA.2020.3009347

[17] Joo Hwan Lee, Hui Zhang, Veronica Lagrange, Praveen Krishnamoorthy, Xiaodong Zhao, and Yang Seok Ki. 2020. SmartSSD: FPGA Accelerated Near-Storage Data Analytics on SSD. *IEEE Computer Architecture Letters* 19, 2 (2020), 110–113. https://doi.org/10.1109/LCA.2020.3009347

[18] Seongju Lee, Kyuyoung Kim, Sanghoon Oh, Joonhong Park, Gimoon Hong, Dongyoon Ka, Kyudong Hwang, Jeongje Park, Kyeongpil Kang, Jungyeon Kim, Junyeol Jeon, Nahsung Kim, Yongkee Kwon, Kornijcuk Vladimir, Woojae Shin, Jongsoon Won, Minkyu Lee, Hyunha Joo, Haerang Choi, Jaewook Lee, Donguc Ko, Younggun Jun, Keewon Cho, Ilwoong Kim, Choungki Song, Chunseok Jeong, Daehan Kwon, Jieun Jang, Il Park, Junhyun Chun, and Joohwan Cho. 2022. A 1ynm 1.25V 8Gb, 16Gb/s/pin GDDR6-based Accelerator-in-Memory supporting 1TFLOPS MAC Operation and Various Activation Functions for Deep-Learning Applications. In *2022 IEEE International Solid- State Circuits Conference (ISSCC)*, Vol. 65. 1–3. https://doi.org/10.1109/ISSCC42614.2022.9731711

[19] Yunjae Lee, Jinha Chung, and Minsoo Rhu. 2022. SmartSAGE: Training Large-Scale Graph Neural Networks Using in-Storage Processing Architectures. In *Proceedings of the 49th Annual International Symposium on Computer Architecture* (New York, New York) *(ISCA '22)*. Association for Computing Machinery, New York, NY, USA, 932–945. https://doi.org/10.1145/3470496.3527391

[20] Jilan Lin, Ling Liang, Zheng Qu, Ishtiyaque Ahmad, Liu Liu, Fengbin Tu, Trinabh Gupta, Yufei Ding, and Yuan Xie. 2022. INSPIRE: In-Storage Private Information Retrieval via Protocol and Architecture Co-Design. In *Proceedings of the 49th Annual International Symposium on Computer Architecture* (New York, New York) *(ISCA '22)*. Association for Computing Machinery, New York, NY, USA, 102–115. https://doi.org/10.1145/3470496.3527433

[21] 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

[22] Liu Liu, Jilan Lin, Zheng Qu, Yufei Ding, and Yuan Xie. 2021. ENMC: Extreme Near-Memory Classification via Approximate Screening. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture* (Virtual Event, Greece) *(MICRO '21)*. Association for Computing Machinery, New York, NY, USA, 1309–1322. https://doi.org/10.1145/3466752.3480090

[23] Liu Liu, Zheng Qu, Lei Deng, Fengbin Tu, Shuangchen Li, Xing Hu, Zhenyu Gu, Yufei Ding, and Yuan Xie. 2020. DUET: Boosting Deep Neural Network Efficiency on Dual-Module Architecture. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 738–750. https://doi.org/10.1109/MICRO50266.2020.00066

[24] Sihang Liu, Aasheesh Kolli, Jinglei Ren, and Samira Khan. 2018. Crash Consistency in Encrypted Non-volatile Main Memory Systems. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 310–323. https://doi.org/10.1109/HPCA.2018.00035

[25] Nika Mansouri Ghiasi, Jisung Park, Harun Mustafa, Jeremie Kim, Ataberk Olgun, Arvid Gollwitzer, Damla Senol Cali, Can Firtina, Haiyu Mao, Nour Almadhoun Alserr, Rachata Ausavarungnirun, Nandita Vijaykumar, Mohammed Alser, and Onur Mutlu. 2022. GenStore: A High-Performance in-Storage Processing System for Genome Sequence Analysis. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems* (Lausanne, Switzerland) *(ASPLOS 2022)*. Association for Computing Machinery, New York, NY, USA, 635–654. https://doi.org/10.1145/3503222.3507702

[26] MARVELL. 2022. MARVELL SSD Controller Solutions. https://www.arm.com/solutions/storage

[27] 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.

[28] Tharun Kumar Reddy 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. *Advances in Neural Information Processing Systems* 32 (2019).

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

[30] MicroCenter. 2023. Inland Platinum 4TB SSD. https://www.microcenter.com/product/627020/inland-platinum-4tb-ssd-m2-2280-nvme-pcie-gen-30x4-3d-nand-internal-solid-state-drive,-pcie-express-31-and-nvme-13-compatible,-ultimate-gaming-solutio

[31] Micron. 2023. How Much Power Does Memory Use? https://www.crucial.com/support/articles-faq-memory/how-much-power-does-memory-use

[32] MUSE. 2023. 28nm chip fabrication cost. https://www.musesemi.com/shared-block-tapeout-pricing

[33] Kenji Nakashima, Joichiro Kon, and Saneyasu Yamaguchi. 2018. I/O Performance Improvement of Secure Big Data Analyses with Application Support on SSD Cache. In *Proceedings of the 12th International Conference on Ubiquitous Information Management and Communication* (Langkawi, Malaysia) *(IMCOM '18)*. Association for Computing Machinery, New York, NY, USA, Article 90, 7 pages. https://doi.org/10.1145/3164541.3164560

[34] Maxim Naumov, Dheevatsa Mudigere, Hao-Jun Michael Shi, Jianyu Huang, Narayanan Sundaraman, Jongsoo Park, Xiaodong Wang, Udit Gupta, Carole-Jean Wu, Alisson G Azzolini, Dmytro Dzhulgakov, Andrey Mallevich, Ilia Cherniavskii, Yinghai Lu, Raghuraman Krishnamoorthi, Ansha Yu, Volodymyr Kondratenko, Stephanie Pereira, Xianjie Chen, Wenlin Chen, Vijay Rao, Bill Jia, Liang Xiong, and Misha Smelyanskiy. 2019. Deep learning recommendation model for personalization and recommendation systems. *arXiv preprint arXiv:1906.00091* (2019).

[35] NVIDIA. 2022. GeForce RTX 3090 Family. https://www.nvidia.com/en-us/geforce/graphics-cards/30-series/rtx-3090-3090ti/

[36] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach 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. *Advances in neural information processing systems* 32 (2019).

[37] 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.

[38] Sahand Salamat, Armin Haj Aboutalebi, Behnam Khaleghi, Joo Hwan Lee, Yang Seok Ki, and Tajana Rosing. 2021. NASCENT: Near-Storage Acceleration of Database Sort on SmartSSD. In *The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays* (Virtual Event, USA) *(FPGA '21)*. Association for Computing Machinery, New York, NY, USA, 262–272. https://doi.org/10.1145/3431920.3439298

[39] Sahand Salamat, Hui Zhang, Yang Seok Ki, and Tajana Rosing. 2022. NASCENT2: Generic Near-Storage Sort Accelerator for Data Analytics on SmartSSD. *ACM Trans. Reconfigurable Technol. Syst.* 15, 2, Article 16 (jan 2022), 29 pages. https:

//doi.org/10.1145/3472769

[40] Sanchari Sen and Anand Raghunathan. 2018. Approximate Computing for Long Short Term Memory (LSTM) Neural Networks. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 37, 11 (2018), 2266–2276. https://doi.org/10.1109/TCAD.2018.2858362

[41] Lidan Shang, Qiushi Yang, Jianing Wang, Shubin Li, and Weimin Lei. 2018. Detection of rail surface defects based on CNN image recognition and classification. In *2018 20th International Conference on Advanced Communication Technology (ICACT)*. 45–51. https://doi.org/10.23919/ICACT.2018.8323642

[42] 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.

[43] Yong-Goo Shin, Yoon-Jae Yeo, Min-Cheol Sagong, Seo-Won Ji, and Sung-Jea Ko. 2019. Deep fashion recommendation system with style feature decomposition. In *2019 IEEE 9th International Conference on Consumer Electronics (ICCE-Berlin)*. IEEE, 301–305.

[44] Xuan Sun, Hu Wan, Qiao Li, Chia-Lin Yang, Tei-Wei Kuo, and Chun Jason Xue. 2022. RM-SSD: In-Storage Computing for Large-Scale Recommendation Inference. In *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. 1056–1070. https://doi.org/10.1109/HPCA53966.2022.00081

[45] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. *Advances in neural information processing systems* 27 (2014).

[46] Arash Tavakkol, Juan Gómez-Luna, Mohammad Sadrosadati, Saugata Ghose, and Onur Mutlu. 2018. MQSim: A Framework for Enabling Realistic Studies of Modern Multi-Queue SSD Devices. In *16th USENIX Conference on File and Storage Technologies (FAST 18)*. 49–66.

[47] Devesh Tiwari, Sudharshan S. Vazhkudai, Youngjae Kim, Xiaosong Ma, Simona Boboila, and Peter J. Desnoyers. 2012. Reducing Data Movement Costs Using Energy-Efficient, Active Computation on SSD. In *2012 Workshop on Power-Aware Computing and Systems (HotPower 12)*. USENIX Association, Hollywood, CA. https://www.usenix.org/conference/hotpower12/workshop-program/presentation/Tiwari

[48] Swagath Venkataramani, Xiao Sun, Naigang Wang, Chia-Yu Chen, Jungwook Choi, Mingu Kang, Ankur Agarwal, Jinwook Oh, Shubham Jain, Tina Babinsky, Nianzheng Cao, Thomas Fox, Bruce Fleischer, George Gristede, Michael Guillorn, Howard Haynie, Hiroshi Inoue, Kazuaki Ishizaki, Michael Klaiber, Shih-Hsien Lo, Gary Maier, Silvia Mueller, Michael Scheuermann, Eri Ogawa, Marcel Schaal, Mauricio Serrano, Joel Silberman, Christos Vezyrtzis, Wei Wang, Fanchieh Yee, Jintao Zhang, Matthew Ziegler, Ching Zhou, Moriyoshi Ohara, Pong-Fei Lu, Brian Curran, Sunil Shukla, Vijayalakshmi Srinivasan, Leland Chang, and Kailash Gopalakrishnan. 2020. Efficient AI System Design With Cross-Layer Approximate Computing. *Proc. IEEE* 108, 12 (2020), 2232–2250. https://doi.org/10.1109/JPROC.2020.3029453

[49] Dewei Wang, Chuan-Tung Lin, Gregory K. Chen, Phil Knag, Ram K. Krishnamurthy, and Mingoo Seok. 2022. DIMC: 2219TOPS/W 2569F2/b Digital In-Memory Computing Macro in 28nm Based on Approximate Arithmetic Hardware. In *2022 IEEE International Solid- State Circuits Conference (ISSCC)*, Vol. 65. 266–268. https://doi.org/10.1109/ISSCC42614.2022.9731659

[50] Jianguo Wang, Dongchul Park, Yang-Suk Kee, Yannis Papakonstantinou, and Steven Swanson. 2016. SSD In-Storage Computing for List Intersection. In *Proceedings of the 12th International Workshop on Data Management on New Hardware* (San Francisco, California) *(DaMoN '16)*. Association for Computing Machinery, New York, NY, USA, Article 4, 7 pages. https://doi.org/10.1145/2933349.2933353

[51] Peng Wen, Weihua Yuan, Qianqian Qin, Sheng Sang, and Zhijun Zhang. 2021. Neural attention model for recommendation based on factorization machines. *Applied Intelligence* 51, 4 (2021), 1829–1844.

[52] Mark Wilkening, Udit Gupta, Samuel Hsia, Caroline Trippel, Carole-Jean Wu, David Brooks, and Gu-Yeon Wei. 2021. RecSSD: Near Data Processing for Solid State Drive Based Recommendation Inference. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems* (Virtual, USA) *(ASPLOS 2021)*. Association for Computing Machinery, New York, NY, USA, 717–729. https://doi.org/10.1145/3445814.3446763

[53] Kan Wu, Andrea Arpaci-Dusseau, Remzi Arpaci-Dusseau, Rathijit Sen, and Kwanghyun Park. 2019. Exploiting Intel Optane SSD for Microsoft SQL Server. In *Proceedings of the 15th International Workshop on Data Management on New Hardware* (Amsterdam, Netherlands) *(DaMoN'19)*. Association for Computing Machinery, New York, NY, USA, Article 15, 3 pages. https://doi.org/10.1145/3329785.3329916

[54] 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.

[55] Hamoud Younes, Ali Ibrahim, Mostafa Rizk, and Maurizio Valle. 2019. Algorithmic level approximate computing for machine learning classifiers. In *2019 26th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*. IEEE, 113–114.

[56] Hang Zhuang, Chao Wang, Changlong Li, Qingfeng Wang, and Xuehai Zhou. 2017. Natural language processing service based on stroke-level convolutional networks for Chinese text classification. In *2017 IEEE international conference on web services (ICWS)*. IEEE, 404–411.