TM-Training: An Energy-Efficient Tiered Memory System for Deep Learning Training in NPUs

JAEYONG PARK, School of Electrical Engineering, Korea University, Seongbuk-gu, Korea (the Republic of)
SANGUN CHOI, School of Electrical Engineering, Korea University, Seoul, Korea (the Republic of)
JONGMIN KIM, School of Electrical Engineering, Korea University, Seoul, Korea (the Republic of)
GUNJAE KOO, Computer Science and Engineering, Korea University, Seongbuk-gu, Korea (the Republic of)
MYUNG KUK YOON, Department of Computer Science and Engineering, Ewha Womans University, Seoul, Korea (the Republic of)
YUNHO OH, School of Electrical Engineering, Korea University, Seoul, Korea (the Republic of)

DRAM accounts for a large fraction of the total cost of ownership of memory systems in deep learning acceleration systems. To achieve sustainable scalability, tiered memory systems with denser technologies become critical. Prior work has proposed various tiered memory systems, but no matter how a system is designed, data movements between tiers consume substantial energy. In particular, as model sizes and memory capacity demands grow, the data movements between memory tiers become more frequent, posing a challenge that tiered memory systems may reduce deployment costs but suffer from low energy efficiency. If a memory system proactively places data into a tier and timely fetches it from the tier, then the excessive data movement between tiers can be mitigated. We find that a system can statically anticipate such behaviors for all pages and localities. With this insight, we propose a new DNN acceleration system called TM-Training using flash memory. TM-Training capitalizes on the repetitive nature of the same computational patterns during execution, enabling the static establishment of optimal data placement for subsequent operations. Moreover, TM-Training employs a new data-splitting scheme to enable precise memory management. Our evaluation demonstrates that TM-Training reduces inter-tier data traffic by 64% and achieves a 55% higher throughput per watt in training than prior work.

CCS Concepts: • **Computer systems organization** → **Neural networks**; • **Software and its engineering** → **Virtual memory**; **Main memory;**

Additional Key Words and Phrases: Tiered memory system, flash-based memory system, DNN accelerator

This work is supported by Samsung Electronics Co., Ltd (IO230419-05997-01).
Authors' Contact Information: Jaeyong Park, School of Electrical Engineering, Korea University, Seongbuk-gu, Seoul, Korea (the Republic of); e-mail: jypark9818@korea.ac.kr; Sangun Choi, School of Electrical Engineering, Korea University, Seoul, Seoul, Korea (the Republic of); e-mail: sangun_choi@korea.ac.kr; Jongmin Kim, School of Electrical Engineering, Korea University, Seoul, Seoul, Korea (the Republic of); e-mail: jmkim99@korea.ac.kr; GunJae Koo, Computer Science and Engineering, Korea University, Seongbuk-gu, Seoul, Korea (the Republic of); e-mail: gunjaekoo@korea.ac.kr; Myung Kuk Yoon (Co-corresponding author), Department of Computer Science and Engineering, Ewha Womans University, Seoul, Seoul, Korea (the Republic of); e-mail: myungkuk.yoon@ewha.ac.kr; Yunho Oh (Co-corresponding author), School of Electrical Engineering, Korea University, Seoul, Seoul, Korea (the Republic of); e-mail: yunho_oh@korea.ac.kr.

ACM 1553-3077/2025/11-ART32
https://doi.org/10.1145/3721484

ACM Reference Format:
Jaeyong Park, Sangun Choi, Jongmin Kim, GunJae Koo, Myung Kuk Yoon, and Yunho Oh. 2025. TM-Training: An Energy-Efficient Tiered Memory System for Deep Learning Training in NPUs. *ACM Trans. Storage* 21, 4, Article 32 (November 2025), 26 pages. https://doi.org/10.1145/3721484

1 Introduction

The dramatic expansion in the size of **Deep Neural Network (DNN)** models has led to a substantial increase in data requirements for training. This tendency significantly elevates the **total cost of ownership (TCO)** for DNN acceleration systems. Addressing this escalating demand necessitates the development of scalable memory systems tailored for DNN training workloads. To tackle these challenges, prior work has proposed tiered memory system architectures for DNN training [16, 19, 37, 43]. The previously proposed memory systems comprise multiple tiers with various memory technologies. Typically, a higher tier equips a memory technology that provides higher performance, but a more expensive one. In contrast, a lower tier equips a technology that exhibits lower performance but is denser and more cost-effective. Those systems classify hot data and cold data and place them into different tiers. Such data management can be done with either data migration (also called promotion and demotion) or initial data placement. Prior work typically comprises two tiers, and it utilizes **high bandwidth memory (HBM)** for Tier 1 memory and various non-volatile memory types for Tier 2 memory [19, 43]. The TCO and performance of a tiered memory system highly depend on the size of Tier 1. A critical mission is to achieve the performance per watt with a minimal Tier 1 size.

A suboptimal data placement policy for a two-tier memory system significantly increases data migration between memory tiers, constituting a substantial portion of the energy consumption in the entire memory system [50]. This tendency becomes more pronounced as the Tier 1 size diminishes. In our initial analysis, given a fixed total memory size, a two-tier memory system may increase the energy consumption caused by the data movement between tiers by up to 50% compared to that employing an ideal data placement. As such, designing a data placement technique is the key to achieving sustainable scalability of tiered memory systems.

DNN training workloads exhibit the characteristic of repeatedly performing the same operation. As such, corresponding memory access behaviors also exhibit similar patterns across iterations. If a tiered memory figures out the hotness of data and performance characteristics of each tier, then it can place data in a proper tier and proactively fetch the demanded data. Such a scheme would reduce the number of pages moving between tiers, eventually achieving better energy efficiency.

We propose a new DNN acceleration system called **Tiered Memory-based DNN Training (TM-Training)** using flash memory. In this article, we consider a two-tier memory system with HBM DRAM as Tier 1 and NAND Flash as Tier 2. The key idea is to design a new data placement strategy that leverages the unique characteristics of the DNN training workloads. TM-Training splits and stores a tensor, both Tier 1 and Tier 2, as needed. This approach leverages the bandwidth of Tier 1 and Tier 2 for prefetching the data to on-chip SRAM buffers. TM-Training proactively promotes only hot data or data that cannot be prefetched from Tier 2 to on-chip SRAM within computation time to Tier 1. Based on the policy, TM-Training focuses on minimizing data migration between memory tiers. The proposed data-splitting technique automatically recognizes the size of DNN models and hot data based on our theoretical performance modeling with data access patterns. To enable precise management, TM-Training splits the data and stores only the necessary amount of the data in Tier 1. Through these design strategies, TM-Training not only reduces deployment costs but also significantly improves energy efficiency for recently adopted DNN training workloads compared to HBM-only systems and prior work.

TM-Training integrates a request timing monitor and a data placement manager, ensuring efficient data placement without imposing significant hardware or software overhead. The request timing monitor measures three timing values: the earliest time at which on-chip SRAM buffers can store the data, the time at which the **Matrix Multiplication Unit (MXU)** of a DNN accelerator utilizes the data for **General Matrix-Matrix Multiplication (GEMM)** or **General Matrix-Vector Multiplication (GEMV)** operations, and the time at which the operations using the data finish. The data placement manager utilizes these timing values and other information, such as memory bandwidth and data size. To efficiently determine the optimal placement, the data placement manager obtains an approximated value using a relatively simple algorithm and then refines it through detailed adjustments.

In our evaluation, TM-Training achieves a 55% better throughput per watt than prior work [19, 43]. With Tier 1 accounting for 12.5% of the total memory capacity, TM-Training achieves the same throughput per watt as the HBM-only memory system while reducing cost. Since HBM is over 100× more expensive than SSD for the same capacity, this configuration lowers the memory system cost by 87% [45–47]. In contrast, prior work reduces cost but delivers lower throughput per watt than the HBM-only memory system.

In this article, we make the following contributions:

— We demonstrate that integrating tiered memory into a DNN accelerator results in significant energy overhead, primarily attributed to extensive data migration between memory tiers.
— We show that DNN training workloads often exhibit repetitive behaviors, wherein tensors with identical dimensions are repeatedly fetched during execution. So, memory access behaviors of a DNN training workload are highly static and predictable.
— We propose TM-Training, a DNN acceleration system based on a flash-based tiered memory system. At run time, TM-Training determines an optimal data placement through a new data-splitting scheme.
— On average, TM-Training reduces data migration by 64% and 40% compared to the two prior work used for evaluation. As such, TM-Training achieves 18% and up to 55% better energy efficiency than HBM-only baseline and prior work, respectively.

The rest of this article consists of the following sections: Section 2 explains why tiered memory is promising for DNN acceleration systems and the challenges of tiered memory systems. Section 3 introduces the key concepts and workflow of TM-Training architecture. Section 4 explains the architectural details of TM-Training. Section 5 explains the experimental results. Section 7 explains related work. Section 8 concludes this article.

2 Why TM-Training?

In this section, we explain the background knowledge of TM-Training, which is the concept of tiered memory systems. Also, we explain the critical problem that we address.

2.1 Flash-based Tiered Memory Systems

In recent years, the growing size of DNN models has increased data requirements for training [4]. The two-year-old GPT-3 contains 175 billion parameters, and its size is over 570 GB [5, 10]. Such a trend still holds. The newer GPT-4 contains 1,000× more parameters than GPT-3. As the size of models used in DNN training increases, the required memory capacity also grows due to higher model complexity and larger batch sizes. However, traditional DRAM-based memory systems face limitations in improving both cost-efficiency and memory density [15]. To resolve these challenges, prior work has explored NAND Flash and various emerging memory technologies (e.g., STT-MRAM, ReRAM) as alternatives [22, 24, 31]. While those technologies are promising,

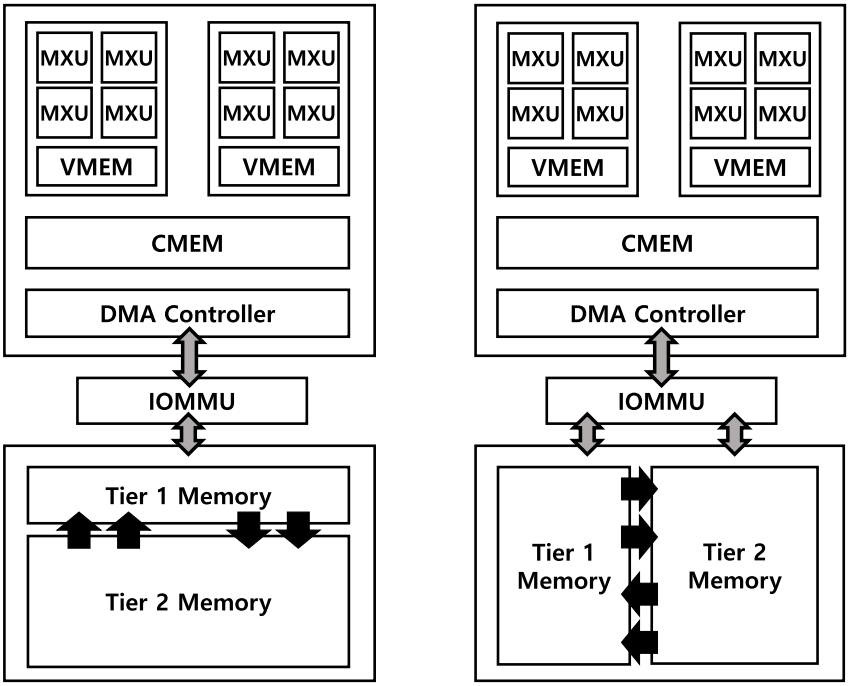


Fig. 1. Example of vertically-tiered and horizontally-tiered memory systems with a single TPU chip. In the former case, only Tier 2 has its own address space, while Tier 1 operates as a hardware cache. In the latter case, all tier memories have their own address space, and memory requests need address translation by IOMMU. Gray arrows represent data traffic between off-chip memory and on-chip SRAM, which is also generated in DRAM-only memory systems. The black arrow indicates data traffic unique to the tiered memory system.

they suffer from lower bandwidth and longer latency than DRAM, thus incurring a slowdown of systems. To overcome this limitation, tiered memory systems have been proposed [35]. In such systems, a higher tier equips a more expensive and high-performance technology, and a lower tier employs a more cost-efficient and dense but slower technology. For example, a two-tier memory system may employ HBM DRAM as Tier 1 and NAND flash memory as Tier 2 [1, 15].

Prior work has proposed the memory systems that employ byte-addressable NAND flash as a tier of memory systems [1, 3]. To use flash-based devices (e.g., SSD) as a memory tier, a system requires a **memory-mapped I/O (MMIO)** interface. Also, a memory system should leverage an in-SSD DRAM buffer to make itself byte-addressable while processing the memory requests [1, 3]. If a read request arrives at an SSD, then the SSD controller serves the request by searching the DRAM in the SSD with the given address. If the DRAM hits, then it issues a PCIe MMIO response to the host CPU. If a miss occurs, the controller then reads the page from NAND Flash with the address. The PCIe standard allows the SSD to use **Base Address Registers (BARs)** to inform the host CPU about the memory mappable region. We apply the details of flash-based tiered memory systems based on the prior work.

Figure 1 shows DNN accelerators based on two types of tiered memory systems. In vertically-tiered memory systems, only Tier 2 memory is mapped to address space, while Tier 1 works as a hardware cache [2, 36, 40, 55]. Tier 1 in the vertically-tiered memory system can be fully utilized if hot data generates numerous memory accesses. We refer to the cache configuration from a page-based DRAM cache called Unison Cache [23]. In horizontally-tiered memory systems, Tier 1 works

as a buffer, and it flushes all the data in use and fetches new data if a miss occurs [8, 11, 34, 35]. Gray arrows represent data traffic between off-chip memory and on-chip SRAM, which is common to both DRAM-only memory systems and tiered memory systems. The black arrow indicates data traffic unique to tiered memory systems. As the Tier 1 size decreases, the proportion of data traffic between memory tiers increases.

Typically, tiered memory systems employ hardware- or software-based data promotion or replacement policies [2, 11, 34–36, 55]. In particular, prior work targeting various DNN workloads has employed timely data placement based on data access patterns such as hotness and lifetime [12, 13, 16, 19, 27, 29, 43, 56]. However, those existing solutions have been proposed for GPU-based systems, which have operational characteristics different from those of DNN accelerators, and it is challenging to find prior work that specifically proposes solutions tailored for DNN accelerators [37]. Unlike GPUs, DNN accelerators utilize on-chip SRAM as scratchpad memory, resulting in high bursts of memory requests within a short period. Also, DNN accelerators perform GEMM and GEMV operations using systolic array-based cores, resulting in differences in the timing of output data generation and the order in which individual output values are produced.

Therefore, the existing solutions for GPU-based systems or general workloads often make suboptimal decisions regarding data migration between memory tiers. While those techniques are effective, it is inevitable to perform a large number of data migrations between memory tiers. The data migrations between the tiers that are apart physically incur substantial energy consumption. In particular, non-volatile memory, such as NAND Flash, tends to exhibit lower static power but higher read/write energy than HBM. Therefore, Tier 2 access can emerge as a dominant factor in energy consumption if redundant access increases.

To analyze the challenge of tiered memory systems, we conduct a simulation study. We design four types of flash-based tiered memory systems based on two criteria: how to place tiers and data placement policy. We classify the data placement policy into offline optimal and online policies. The offline optimal policy is a theoretical, idealized approach assuming complete knowledge of future workload characteristics and data access patterns. The online policy operates in real-time without prior knowledge of future workload or data access patterns. The policy makes decisions dynamically as workloads progress. The four types are Hor_Off, Hor_On, Ver_Off, and Ver_On. “Hor” or “Ver” indicates whether the hierarchy between tier memories is horizontal or vertical. “Off” or “On” means whether the data placement policy operates as offline optimal or online. Hor_Off assumes that the system already knows the entire workload progress and the data used, allowing the system to proactively promote frequently reused hot data to Tier 1. However, the system avoids promoting data if prefetching the data directly from Tier 2 to the on-chip SRAM of the accelerator does not cause memory stalls. For data placement policy, the system applies Belady’s optimal algorithm, which represents an ideal approach. Also, the system proactively fetches the data that the system will access in the next load/store.

Hor_On, similar to Hor_Off, places hot data in Tier 1. However, unlike Hor_Off, Hor_On promotes as much of the remaining data as possible without verifying whether skipping the promotion would still prevent additional memory stalls. For Tier 1 eviction, the system employs the **Least Recently Used (LRU)** algorithm as a replacement policy. We assume that Hor_On does not need to check whether additional stalls caused by memory operations occur even if the data is not promoted. Initially, the system stores all data in Tier 2 before executing the workload. If Tier 2 lacks sufficient space, then the system places the remaining data in Tier 1.

Ver_Off and Ver_On initially store all the data in Tier 2 before workload execution, since Tier 1 functions as a hardware cache. Ver_Off applies Belady’s optimal algorithm as a Tier 1 replacement policy. Also, Ver_Off proactively fetches the necessary data similar to Hor_Off. However, Ver_On applies the LRU algorithm as a replacement policy if a page fault occurs in Tier 1.

Table 1. Baseline Architecture Configuration

Precision	BFloat16
Dataflow	Weight Stationary
Number of Chips/Cores per Chip/MXU per Core	1/2/4
MXU Size	128×128
VMEM per Core/Shared CMEM (MB)	16/128
Clock Frequency (MHz)	1,050
IOMMU	2,048 entries, 4 KB page

Prior work has proposed various page replacement policies applicable to our experiments [28, 30]. HeteroOS tracks data hotness by monitoring the lifetime of individual data, only focusing on long-lived data [28]. Also, HeteroOS aggressively evicts data based on the LRU algorithm before reaching the Tier 1 capacity limit. LPR leverages machine learning techniques to improve page replacement decisions [30]. By learning from past replacement behavior, LPR adjusts its policy through a reward-penalty system using both LRU and **Most Recently Used (MRU)** algorithms during training. These advancements in page replacement policies aim to reduce the implementation costs of LRU-based policy by leveraging workload-specific characteristics or use learning-based approaches to approximate optimal decisions. However, the energy efficiency and performance achieved by applying these policies remain within the range of results produced by our designed online and offline-optimal approaches. Therefore, we conduct experiments using the two policies, namely, online- and offline-optimal.

We analyze the behaviors of four types of tiered memory systems with a DNN accelerator, designed with reference to the Google TPUv4 [25, 26]. **Common Memory (CMEM)** and **Vector Memory (VMEM)**, described in Figure 1, are on-chip SRAM buffers used as scratchpad memory. An accelerator chip consists of two cores, each containing a VMEM and four MXUs, with both cores sharing the CMEM. We model all the configurations and the accelerator with SCALE-Sim v2 [44]. The detailed simulation configurations are described in Table 1. For a fair comparison, all the cases contain the same 32 GB of total address space per accelerator chip, which is the same as the baseline. For the horizontally-tiered memory system, we set the total size of Tier 1 and 2 to 32 GB. As Tier 1 size varies, Tier 2 memory size becomes (32 GB - Tier 1 size). In the vertically-tiered memory system, the size of Tier 2 is always fixed at 32 GB, while we change the size of Tier 1. Also, all cases assume support for 4KB page granularity.

In real systems, the total memory capacity is typically configured based on the model size, which is a dominant factor in determining the total size of data used. Therefore, to clarify that the Tier 1 size variations applied in our experiments reflect diverse design choices applicable to typical training servers, we determine the total Tier 1 size used based on the ratio of the Tier 1 size to the model parameter size. Considering the BFloat16 numeric format mentioned in Table 1 and the accelerator chips used for each workload listed in Table 3, a ratio of 5:1 corresponds to allocating 50% of the total memory capacity to Tier 1. As the ratio doubles from 5:1 to 10:1, the proportion of Tier 1 within the total memory capacity also doubles.

We employ HBM as Tier 1 and flash memory as Tier 2. HBM is chosen over the typical DDR DRAM for its large bandwidth, essential for DNN accelerators to prefetch data quickly, prioritizing performance impact over latency. Flash memory serves as Tier 2 due to its high energy efficiency and low static power, aligning with the goal of creating an energy-efficient system. Table 2 describes the performance parameters of HBM and NAND Flash used in our

Table 2. Comparison of Memory Technologies

Memory Technology	HBM	NAND Flash (SSD)
Bandwidth	1,200 GB/s	Seq. Read: 15 GB/s, Seq. Write: 13.8 GB/s
Dynamic Energy Consumption	3.97 pJ/bit	75 pJ/bit
Static Power Consumption	684 mW	1.6 mW

Table 3. Target Workloads and System Configurations

DNN Model	BERT-Large	T5	Chinchilla	GPT-3	PaLM
Parameter Size	340M	11B	70B	175B	540B
Sequence Length (token)	512	512	2,048	2,048	2,048
Batch Size	256	128	2,048	2,048	2,048
TPU Pod (1xn)	1	1	2	2	2
Model Parallelism	1	1	2	4	12
Data Parallelism	4	16	256	256	256
Total TPU Chips	4	16	1,024	2,048	6,144
Data Size per Iteration	86 GB	316 GB	1.5 TB	3.3 TB	3.1 TB

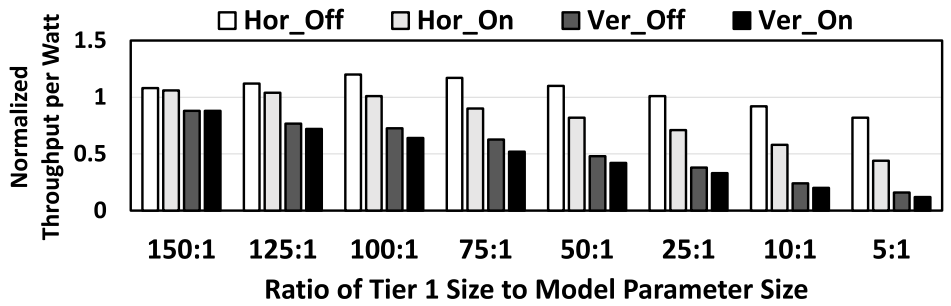


Fig. 2. Energy efficiency breakdown depending on the ratio of Tier 1 size to model size. All results are normalized to a 32 GB HBM-only system. If the total Tier 1 size is more than 25 times the model size, then Hor_Off exhibits higher energy efficiency than the HBM-only system, while other cases show lower efficiency. The energy efficiency decreases significantly in Ver_Off and Ver_On, reaching 10%–11% if the ratio is 5:1.

simulation experiments [14, 25, 39]. The "Seq. Read" and "Seq. Write" refer to sequential read and sequential write, respectively.

We list the DNN training benchmark workloads and system configurations in Table 3. We focus on transformer-based models that are popularly employed in various DNN applications [5, 10, 20, 41]. For the PaLM model, we apply the same configuration from the prior work [9]. For the other workloads, we scale down the number of TPU chips, considering the model size.

Figure 2 depicts the energy efficiency of all cases. Regardless of the ratio, the horizontally-tiered memory systems outperform the vertically-tiered ones. For all cases except Hor_Off, energy efficiency decreases as the ratio decreases. This tendency arises from both the increased energy consumption in the memory system and reduced throughput as Tier 1 size decreases. A smaller Tier 1 size leads to more frequent data migrations between memory tiers, where the low bandwidth

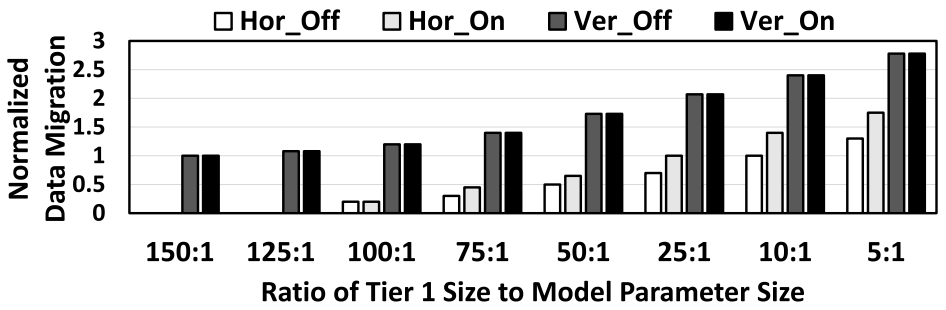


Fig. 3. The size of migrated data depending on the ratio of Tier 1 size to model size. All results are normalized to the size of the total data used for running the workload.

of NAND Flash causes additional memory stalls. Also, frequent data migration leads to higher energy consumption due to the characteristics of NAND Flash, which consumes less static power but more dynamic energy than HBM. Hor_On exhibits up to 46% lower energy efficiency than Hor_Off, indicating that a general policy relying on tracking data hotness and an LRU-based algorithm fails to achieve sufficient energy efficiency. As shown by Ver_Off, using a vertically-tiered memory system inevitably suffers from low energy efficiency, regardless of the Tier 1 size and data placement policy. We find the main cause of such tendency through the results shown in Figure 3.

Figure 3 depicts the amount of data migration between Tier 1 and Tier 2 varying the ratio of the Tier 1 size to the model size. Applying vertically-tiered memory systems generates more data traffic than applying horizontally-tiered memory systems, regardless of the data placement policy. In the vertically-tiered memory systems, memory requests cannot directly access Tier 2, so all the data should be prefetched into Tier 1. These characteristics significantly increase data migration even with the large Tier 1. In our experiments, the data size created or used in one layer per chip is about 9 GB for forward pass and 22.5 GB for backward pass. If Tier 1 is smaller than 24 GB, then the 22.5 GB of data does not fit in Tier 1 memory. Thus, Tier 1 memory evicts some data, incurring extra data migration between memory tiers. Ver_Off, the upper bound of applying the vertically-tiered memory system, exhibits more data migration than both horizontally-tiered cases, suggesting the memory system is unsuitable for DNN training workloads. For example, if the ratio of the Tier 1 size to the model size is 5:1, then Hor_Off and Hor_On have 1.5× and 2.1× data migration, respectively, while Ver_Off and Ver_On have 3.6× and 3.2× data migration, respectively.

We also analyze the correlation between data migration in tiered memory and the energy efficiency of the overall system. We break down the energy consumption in the tiered memory system across all cases and measure the proportion of each element. We categorize the types of energy consumption as follows: The first type is the energy consumed by promotion and demotion between tiers. The second type is the energy consumed by prefetching data into on-chip SRAM buffers, triggered by a memory request from DMA controller. The third type is the energy consumed by the static power of the memory itself regardless of data read and write.

Figure 4 depicts the breakdown of energy consumption. As the Tier 1 size decreases, the proportion of energy dedicated to data migration between memory tiers becomes more pronounced. This trend is particularly evident in Ver_Off and Ver_On. Although the energy expended for data migration between tiered memory and on-chip SRAM buffers is smaller than that of Hor_Off and Hor_On, the substantial energy required for data migration between memory tiers outweighs this reduction. For instance, if the ratio is 5:1, then Ver_Off consumes 0.45% less energy than Hor_Off

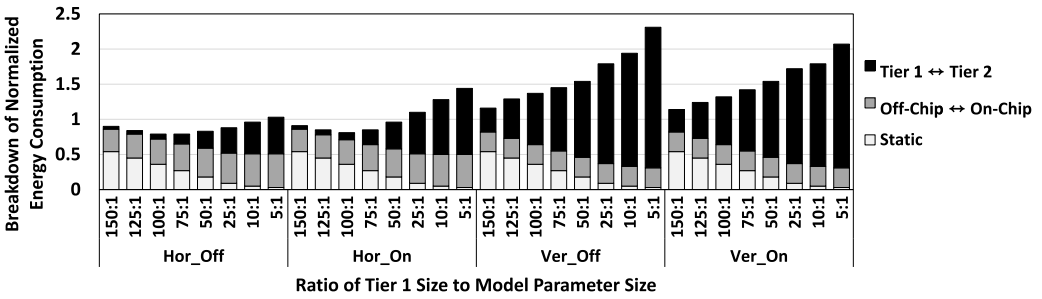


Fig. 4. Breakdown results of total energy consumption in the off-chip memory system. All results are normalized to a 32 GB HBM-only system. As the size of Tier 1 decreases, data traffic between tiers becomes a dominant factor for energy consumption.

in the data migration between off-chip memory and on-chip SRAM. However, Ver_Off consumes 4.09× more energy than Hor_Off in the data migration between memory tiers. As a result, Ver_Off consumes 2.24× more energy than Hor_Off for the total energy consumption in the memory system.

The significant difference in energy consumption by the off-chip memory system, depending on the Tier 1 size and the data placement policy, has a substantial impact on the energy consumption of the overall system. Based on the number of MAC operations and SRAM accesses obtained from SCALEsim and the energy parameters derived from Accelergy, we measure the energy consumption by the accelerator [44, 53]. We find that the off-chip memory in the HBM-only baseline accounts for an average of 27.6% of the total energy consumption . In other words, the accelerator consumes 2.62× more energy than the off-chip memory system. However, if the Tier 1 size is constrained and a suboptimal policy is applied, then the memory system consumes more than twice the energy compared to the baseline, as shown in Figure 4. In this case, the off-chip memory system accounts for over 43.3% of the total energy consumption.

We observe that data traffic between tiers significantly impacts overall energy consumption. Our analysis reveals that data traffic increases in both vertically-tiered and horizontally-tiered memory systems, even with a simplistic data placement policy. As we analyze, Tier 2 exhibits a substantial surge in data traffic. Given the limited bandwidth of Tier 2, the resulting inefficiency in data traffic has a noteworthy impact on the overall energy consumption. In addition, we confirm that the policy for classifying hot data and cold data is suboptimal for DNN training workloads. For Hor_On, as the size of Tier 1 decreases, data migration between memory tiers increases by up to 1.86× compared to Hor_Off.

3 TM-Training

To address the challenges discussed in the previous section, we propose a novel DNN acceleration system based on a flash-based tiered memory system called TM-Training. TM-Training aims to minimize data migration between memory tiers using the horizontally-tiered memory system. The key concept behind TM-Training is the static prediction of data placement that approximates optimality, motivated by heuristic analysis of memory access patterns in DNN training workloads.

3.1 DNN Training Workload Characteristics

To minimize data migration between memory tiers, it is crucial to consider the characteristics of DNN training workloads. Transformer-based models contain multiple layers, each containing multi-head attention and feed-forward networks. Since all layers share the same dimension, there

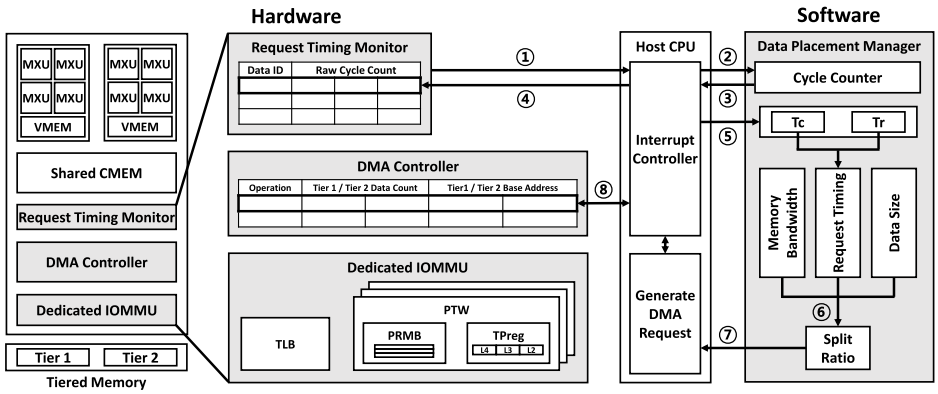


Fig. 5. Overview of TM-Training. Request timing monitor records timing information as data moves through CMEM, VMEM, and back to off-chip memory. The data placement manager derives energy-efficient data placement (as a split ratio for the data). If the system performs an operation that has been performed before, then it applies the already derived split ratio for storage in Tier 1 and Tier 2. The DMA controller receives DMA descriptors from the host CPU. The dedicated IOMMU performs address translation using 128 PTWs.

is no need to differentiate between the layers. Similarly, all attention heads in a layer share the same data dimension. Therefore, we can expect that the characteristics of the workload generate memory requests with the same pattern repeatedly.

We measure the earliest time the system can store each data in CMEM. We assume that memory requests for the data are generated at that time. While all the workloads that we use in this article often exhibit repetitive computation patterns, a system generates different patterns if a cold miss occurs in CMEM at the start of execution. In the case of BERT-Large, each attention head on a single chip needs about 26.7 MB of data to operate. Since the CMEM capacity for each core is smaller than that, the system prefetches the data required for two attention heads at once at the beginning of execution. Therefore, repetition of the same memory request pattern is generated from subsequent operations. For GPT-3, the data size for one attention head is 120 MB, which is larger than the CMEM capacity. It leads to a repetition of almost the same pattern from the start of execution. For both BERT-Large and GPT-3, while the system repeats the memory access pattern, there is a small, negligible error.

3.2 TM-Training Overview and Workflow

Figure 5 shows an overview of TM-Training, highlighting its key components: the request timing monitor and the data placement manager. The request timing monitor, a hardware component, records raw cycle counts for each data, capturing the moments when data used for operations transits from tiered memory to on-chip SRAM buffers in an accelerator (e.g., CMEM and VMEM in Google TPUv4). If the request timing monitor records three values, then the accelerator also generates an interrupt to calculate T_c, T_r values. T_c and T_r correspond to (T_2 - T_1) and (T_3 - T_1), respectively.

The data placement manager, which is an OS-level software algorithm, leverages the recorded timing values from the request timing monitor and additional information such as memory bandwidth and data size. If data moves between CMEM, VMEM, and MXU within the accelerator, then the data placement manager generates an interrupt to the host CPU interrupt controller. Once an interrupt signal arrives, the data placement manager transmits the current count value to the

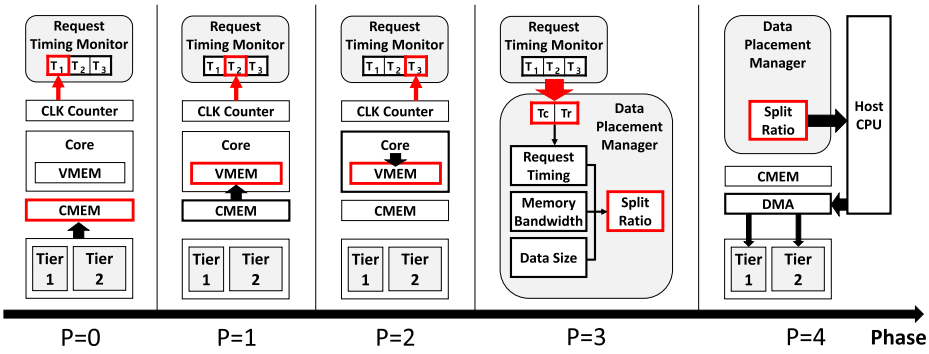


Fig. 6. Workflow of TM-Training. TM-Training determines data placement to minimize redundant Tier 1 access by considering the time each data is used.

request timing monitor. Each stored raw cycle count is a value corresponding to the earliest time the data can be prefetched in CMEM (T_1), the time it initiates waiting for the operation in VMEM (T_2), and the time the operation using the data in MXU ends (T_3). The data placement manager minimizes data migration between memory tiers by selectively promoting only essential data to Tier 1. During this process, the data placement manager employs a data-splitting technique to manage large-sized data precisely. With this scheme, the data placement manager actively engages in memory management tasks, including promotion and demotion.

The numbers attached to the arrows in Figure 5 represent the overall control flow among key components of TM-Training. After the data placement manager records three raw cycle counts in the request timing monitor for each data, the data placement manager derives the corresponding split ratio. Using this ratio, the host CPU generates memory requests for the data and sends descriptors to the DMA controller.

To manage data eviction in Tier 1, TM-Training applies the LRU algorithm as its eviction policy. However, data promoted from Tier 2 but not yet used in GEMM operations is excluded from eviction consideration to prevent premature demotion before being accessed. Also, the system proactively performs eviction and replacement operations before reaching the Tier 1 capacity limit, minimizing potential memory stalls.

TM-Training adopts only a horizontally-tiered memory system considering its energy efficiency-focused objectives. In the experiments discussed in Section 2, we observe that vertically-tiered memory systems suffer from significant data migrations between memory tiers, leading to unavoidable reductions in energy efficiency. As particularly shown in Figure 2, reducing the Tier 1 size in the vertically-tiered memory system drastically lowers energy efficiency, even with an ideal replacement policy.

Figure 6 depicts how TM-Training measures memory request timing and splits data to minimize data migration between memory tiers using its key components. We assume that the system stores the data required for the first self-attention in Tier 1 at the onset of workload execution. Also, the address mapping information for the data already resides in the TLB. During execution, TM-Training measures the current cycle at the software level. We assume the DNN accelerator has the same feature as Google TPUv4 [25]. The workflow of TM-Training encompasses the following stages, where P signifies a phase during which each characteristic operation occurs. During the process, the system executes each phase by moving data from tiered memory to on-chip SRAM.

CMEM Prefetching (P=0): Upon the initiation of prefetching data to CMEM by the DMA controller, the request timing monitor logs the cycle at which prefetching commenced. The compiler schedules the data as swiftly as possible, provided there is available space in CMEM. With this scheme, the request timing monitor stores the current cycle value as T_1.

VMEM Prefetching (P=1): Following the prefetching of data to CMEM, the data needed for the first GEMM operation starts moving from CMEM to VMEM. At this juncture, the request timing monitor records the current cycle value as T_2 value. If the data is not present in CMEM at T_2, then it cannot be immediately stored in VMEM. This delay directly elevates the likelihood of a memory stall in VMEM later. In the subsequent phase, the data placement manager ensures the completion of prefetching all data before T_2.

After Completing GEMM Operation (P=2): Once MXUs generate all the output data, the request timing monitor records T_3 value. (T_3 - T_1) refers to the time required for the data to be transferred and used in computation.

Determining Data Split Ratio (P=3): After measuring the request timing for each data, the data placement manager calculates T_c and T_r values and determines the ratio of data to be stored in Tier 1 and 2, considering memory request timing, memory bandwidth, run time, and the data size. Using this information, the data placement manager obtains the most energy-efficient data placement between memory tiers with simple algorithms (details in the next section). If the Tier 1 size decreases, then the system is more likely to demote data used in operations and stored in Tier 1. Consequently, redundant Tier 1 access leads to additional energy consumption due to data migration between memory tiers without any performance benefits, thus reducing energy efficiency.

Data-splitting: As the size of DNN models increases, the size of each data used in specific operations also grows. Therefore, managing each data as a unit is a coarse-grained policy in that we cannot determine data placement in units of size smaller than the data. Storing all the data with large dimensions in Tier 1 significantly increases data migration between memory tiers. However, storing all data in Tier 2 causes performance degradation, which is also a major factor in lowering energy efficiency. Therefore, we design fine-grained data placement by splitting the large data and storing each separated data in Tier 1 and 2, respectively. Prior work has proposed a similar technique to manage GPU memory more precisely [38]. In TM-Training, we define the split ratio as the size of the portion stored in Tier 1 divided by the total data size.

Storing Data (P=4): Once the host CPU sends a request to the DMA controller and the data placement manager has determined a split ratio, the data is divided according to the ratio and placed in both Tier 1 and Tier 2. If the split ratio has not been obtained yet, then TM-Training stores the data in Tier 1. Transformer-based models contain multi-head attention structures, which execute the same self-attention algorithm multiple times [5, 9, 10, 49].

4 TM-Training Implementation

4.1 Request Timing Monitor

The request timing monitor stores the IDs of data used in the training workload. Transformer-based models commonly perform a self-attention algorithm. The algorithm generates **Query (Q)**, **Key (K)**, and **Value (V)** matrices and a self-attention feature map through GEMM operations. After performing the algorithm on multiple attention heads, also known as multi-head attention, each head generates output feature maps. The system then concatenates these maps and uses them in the feed-forward network. Given these characteristics, we classify all data into 14 data types. Table 4 shows the 4-bit labels of all 14 data types. Each data type has a set of dedicated raw cycle count information (T_1, T_2, and T_3). The request timing monitor uses 64 bits to store the data type and corresponding raw cycle count values. Based on a total of 64 bits, 4 bits from the **most significant**

Table 4. Data Labels in Request Timing Monitor

Sublayer	Operation	Input Data	Weight
Self-Attention	Get Q	0000	0001
	Get K		0010
	Get V		0011
	Q × KT	0100	0101
	Q × KT × V	0110	0111
Concat Linear	Concat Linear	1000	1001
Feed Forward	Feed Forward 1	1010	1011
	Feed Forward 2	1110	1101

ALGORITHM 1: Split Ratio Approximation

```
1:  function SplitRatio(T_c, D, BW_1, BW_2)
2:      T_req = T_c                                  ▷ D: Data size.
3:      T_min = D/(BW_1 + BW_2)                      ▷ T_req: Memory requests interval.
4:      if T_req ≥ T_min then                        ▷ T_min: Minimum memory period.
5:          r' = 1 − T_req × BW_2/D                  ▷ BW_1, BW_2: Tier 1, Tier 2 bandwidth.
6:          (T_req = (1 − r') × D/BW_2)
7:      else
8:          r' = BW_1/(BW_1 + BW_2)
9:          (r' × (BW_2) = (1 − r') × (BW_1))
10:     return r'
```

bit (MSB) denote data ID, and the lower 60 bits denote T_1, T_2, and T_3 values. So, there are 42 64-bit SRAM storages in the request timing monitor.

If a DMA controller generates a memory request, then the system checks whether the request timing monitor has already measured the T_c value for the corresponding ID of the requested data. If the request timing monitor has already measured the T_c value, then it does not store the current cycle value in T_c again. If not, then the request timing monitor stores the current cycle counter value at that point in T_c. If the system fetches data from CMEM to VMEM to compute them in the MXU, then the request timing monitor checks whether the system has ever measured the T_v value of the corresponding ID. The request timing monitor stores the current cycle value at that point if the T_v value remains unmeasured. After completing the GEMM operation with the data and discharging all partial sums from the MXU, the request timing monitor records the current cycle value at that point into T_r.

4.2 Data Placement Manager

Algorithm Overview: The data placement manager employs three algorithms to determine the optimal data placement ratio. First, it calculates an approximated value to measure the split ratio efficiently. Second, it measures the run time and energy consumption for the corresponding operation in the HBM-only system. Since the workload initially runs with all necessary data in Tier 1, the data placement manager executes this algorithm without experiencing performance degradation from Tier 2 memory. As a result, despite the tiered memory environment, it measures the run time of the HBM-only system at first measurement. Third, the algorithm finds the most reasonable value in terms of energy efficiency by adjusting the approximated split ratio.

Split Ratio Approximation: Algorithm 1 describes logic to find the approximated split ratio. The approximated split ratio refers to the smallest split ratio (r) value that prevents Tier 2 from becoming a critical path of performance. As the r value becomes smaller, the system stores less amount of data in Tier 1. Placing more data in Tier 2 reduces data traffic between Tiers 1 and

ALGORITHM 2: Baseline Runtime and Energy Consumption

```
function OffChipAccessPerData(D, SRAMSize)
    N_MXUFetch = ⌈Col_Weight / Col_MXUs⌉        ▷ Col_Weight: Column size of weight operand matrix.
    if (type(Data) == Weight) then              ▷ Col_MXUs: Total column size of 4 MXUs.
        N = 1
    else
        if (D < SRAMSize) then                  ▷ SRAMSize: On-chip SRAM capacity for each core.
            N = 1
        else
            N = N_MXUFetch
    return N
function OnChipAccessPerData(D, P_energy, E_CMEM, E_VMEM)
    N_MXUFetch = ⌈Col_Weight / Col_MXUs⌉
    if (type(Data) == Weight) then
        N_CMEM, N_VMEM = 2
    else if (D < VMEMSize) then
        N_CMEM = 2
        N_VMEM = 1 + N_MXUFetch
    else if (D < SRAMSize) then
        N_CMEM = 1 + N_MXUFetch
        N_VMEM = 2 × N_MXUFetch
    else
        N_CMEM, N_VMEM = 2 × N_MXUFetch
    E_c = E_CMEM × n_CMEM, E_v = E_VMEM × n_VMEM
    return E_c + E_v
function OffChipEnergy(D, T_r, P_energy)
    HBMDynamicEnergy, HBMStaticPower ← P_energy   ▷ P_energy: A class that has all parameters related to energy.
    n ← OffChipAccessPerData(D, SRAMSize)
    HBMAccess = D × n
    E_off.d = HBMAccess × HBMDynamicEnergy
    E_off.s = HBMStaticPower × T_r              ▷ E_off.s, E_off.d: Static and dynamic energy of off-chip memory.
    return E_off.d + E_off.s
function OnChipEnergy(D, T_r, P_energy)
    E_p ← OnChipAccessPerData(D, P_energy, E_CMEM, E_VMEM)
    E_on.d = D × E_p, E_on.s = SRAMStaticPower × T_r   ▷ E_on.s, E_on.d: Static and dynamic energy of on-chip SRAM.
    return E_on.d + E_on.s
function MXUEnergy(#ofMacOp, P_energy)
    E_mac = #OfMacOp × MacEnergy
    return E_mac
function Total(D, T_c, T_r, P_energy, #ofMacOp)
    E_off ← OffChipEnergy(D, T_r, P_energy)
    E_on ← OnChipEnergy(D, T_r, P_energy)
    E_mac ← MXUEnergy(#ofMacOp, P_energy)
    return E_off + E_on + E_mac
```

2. Consequently, this reduction results in lower energy consumption. If there is no performance degradation even if r becomes smaller, then energy efficiency increases, because energy consumption decreases while maintaining performance. For this reason, we set the approximated ratio to the smallest value that does not increase run time. Using Algorithm 1, the data placement manager derives the optimal r value without calculating all possible real values. For approximation, the data placement manager first measures the memory requests interval (T_{req}) and the minimum time required for prefetching the data using both Tier 1 and Tier 2 (T_{min}). If T_{req} is larger than T_{min}, then it means that the system is memory-bound. Thus, the algorithm does not allocate more data to Tier 2 to prevent extra performance degradation. Otherwise, the algorithm finds the split ratio if the time required for prefetching from Tier 2 is equal to T_{req}.

Baseline Run Time and Energy Consumption: In Algorithm 2, we first measure the energy efficiency of the baseline before applying a split ratio. During the measurement, Algorithm 2 uses parameters such as *HBMDynamicEnergy* and *HBMStaticPower*. Energy and power consumption parameters for memory accesses and MAC operations in PEs are obtained through profiling before workload execution. Therefore, the data placement manager operates according to Algorithm 2

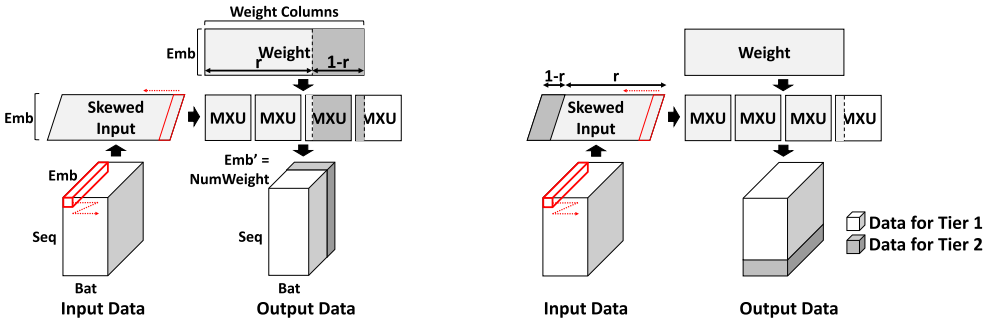
ALGORITHM 3: Split Ratio Adjustment

```
1:  function RuntimeOverhead(T_c, D, r, BW_2)
2:      T_req = T_c
3:      T_o = D × (1 − r)/BW_2 − T_req
4:      return T_o                                   ▷ T_o: Runtime overhead.
5:  function EnergyOverhead(D, T_r, E_off, r)
6:      E_base = E_off
7:      E_r = 2 × r × E_off.d + (1 − r) × D × T2ReadEnergy + Tier2Memsize × E_off.s + T2Static × T_r
8:      return E_r − E_base                          ▷ E_r − E_base: Energy consumption overhead.
9:  function Efficiency(r)
10:     E_r = E+ EnergyOverhead(D, T_c, T_r, E_off, r)
11:     T_r = T+ RuntimeOverhead(T_c, T_v, D, r, BW_2)
12:     return E_r/T_r                               ▷ E_r/T_r: Energy efficiency of the overall system.
13: function RatioAdjustment(r′, seq)
14:     r1 = r′, r2 = r′ − 1/seq                     ▷ r′: An approximated value from Algorithm 1.
15:     while True do                                ▷ seq: Sequence length of the target workload.
16:         EE_r1 = Efficiency(r1)
17:         EE_r2 = Efficiency(r2)
18:         if EE_r1 ≥ EE_r2 then
19:             break
20:         else
21:             r1 = r2
22:             r2 = r2 − 1/seq
23:     return r1
```

with knowledge of these parameter values. We estimate the run time for an operation using T_r. Given that the operation order and the data size they require are fixed, we also approximate energy consumption. The total energy is the sum of the energy generated from tiered memory (E_{off}), on-chip SRAM buffers (E_{on}), and MXU (E_{mac}). Since the MXU operates in a weight-stationary dataflow, we estimate the number of accesses in the on-chip SRAM buffers based on whether the data serves as a weight and whether it can be stored in the VMEM at once. Through a motivation study, we observe that considering the workload progress enables predicting whether the MXU frequently reuses data in off-chip memory. During self-attention operations, on-chip SRAM capacity is larger than the size of most of the data used. In this case, MXU operations such as GEMM reuse much of the data stored in VMEM or CMEM, while accessing off-chip memory only once per data. However, during feed-forward network operations, particularly for large models, the system cannot store the data entirely in on-chip SRAM in a single instance. Therefore, multiple accesses to off-chip memory are required. In this case, by considering the dimension of the systolic array, weight-stationary dataflow, and on-chip SRAM capacity, we estimate the number of accesses to data in off-chip memory.

Considering these characteristics, we estimate the energy consumption in the baseline. In function CheckDataType, the algorithm divides cases based on data size and data type. If the data type is weight, then each weight data enters the MXU only once. In contrast, if it is input data, then it enters the MXU multiple times, proportional to the number of weight data relative to the systolic array size. Lines 21–27 reflect the dataflow of weight stationary.

Split Ratio Adjustment: Algorithm 3 finds the ratio with the highest energy efficiency by changing the approximated ratio. Also, similar to the parameters used in Algorithm 2, parameters such as *T2ReadEnergy*, *Tier2Memsize*, and *T2Static* are obtained through profiling before workload execution. First, it calculates the run-time overhead and energy overhead with functions RuntimeOverhead and EnergyOverhead. Then, it adds each overhead to the baseline run-time and energy consumption, respectively, to determine the energy efficiency value by using the function Efficiency. Since the system splits the data according to the sequence length, the algorithm adjusts the ratio by the inverse of the data sequence length. This three-step algorithm efficiently derives the most energy-efficient split ratio. The function RatioAdjustment shows this algorithm.



(a) Data-splitting using embedding dimension (b) Data-splitting using sequence length

Fig. 7. Two methods for splitting output data during its storage in off-chip memory. Emb, Seq, and Bat mean embedding dimension, sequence length, and batch size, respectively. The column dimension of the weight matrix is one of the attention key, value dimensions, or the hidden size of the feed-forward network.

Since TM-Training aims to maximize energy efficiency, it accepts the split ratio even if it introduces performance overhead, provided that it improves energy efficiency. If there are also performance requirements in the real system, then we can modify the algorithm to reflect the requirements. If modifications are necessary, then TM-Training reflects such requirements between lines 19 and 20.

4.3 Data-splitting

The input data used in transformer training workload execution has three dimensions: Sequence length, embedding dimension, and batch size. There are three ways to divide the data, considering the split ratio while saving output data from CMEM to tiered memory. Depending on the dimension of the data, splitting data may require additional hardware or software-level support. We split the data based on sequence length, which requires the least overhead.

Figure 7 depicts the dataflow in GEMM operation according to the splitting method. First, Figure 7(a) illustrates how we split the output data in the embedding dimension. In weight stationary dataflow, the values in the embedding dimension of the data result from different weights. Since the MXU keeps weights and performs operations in parallel, MXU simultaneously generates output data to be stored in Tier 1 or 2. Second, as shown in Figure 7(b), the MXU generates all the data for Tier 1 and then the data for Tier 2 if we split the output data in the sequence length. We do not split the data with the batch size, because the batch size per core becomes small as we use data parallelism across multiple chips. Therefore, it is difficult to split the data close to the split ratio.

If on-chip SRAM buffers have enough free space and the next operation requires reusing the output data, then we store all the data in the CMEM. In that case, additional hardware/software support is unnecessary regardless of how MXU generates data for memory tiers. If on-chip SRAM buffers do not have free space or the next operation does not reuse the output data, then the DMA controller immediately transfers the data to tiered memory. In this situation, if we apply the method in Figure 7(a), then an accelerator requires additional hardware/software support that determines where to store each data. If we apply the method in Figure 7(b), then the DMA controller transfers the earlier generated data in Tier 1 and then stores the remaining data in Tier 2. As the host CPU informs the DMA controller of the data size to transfer in advance, the DMA controller transfers the output data to each tier memory sequentially without additional hardware/software support.

4.4 Implementation Overhead

On the hardware side, we design the request timing monitor and add additional hardware components in the dedicated IOMMU. According to prior work that has proposed the IOMMU, it requires an additional 34 KB of SRAM storage and 13.65 mW of leakage power per chip [21]. The request timing monitor consumes about 2.7 KB SRAM storage, since it has 42 64-bit fields. Considering the energy consumption shown in Table 2, hardware-side overhead is not critical for the overall system, since we save more than hundreds of mW by reducing the HBM size. On the software side, the most important factor is address translation or page mapping modification due to promotion and demotion. However, experiments in Section 2 show that data migration between memory tiers is more dominant than others. Since the data placement manager uses simple logic to calculate T_c, and T_r values and obtain the data-splitting ratio, the resulting overhead is also negligible.

5 Evaluation

5.1 Methodology

We model TM-Training and other architectures for comparison with SCALE-Sim v2 [44]. We model the software implementation and operations of the overall tiered memory system using SCALE-Sim simulation results such as run time and number of memory accesses. In particular, our modeling is based on trace files and analytical modeling in SCALE-Sim. Our implementation accurately performs cycle-level simulation for essential information for software implementation, including memory request and computation timing for each data. Also, we implement timing models of off-chip memory systems based on the memory parameters described in Table 2, which are used in the simulation, and prior work [14, 25, 38]. Our modeling measures memory stalls and the number of page promotions/demotions between tiers. With this implementation, we evaluate the behavior of TM-training and tiered memory systems. We get power and energy consumption parameters of all the on-chip hardware resources such as VMEM, CMEM, and MXU with Accelergy [53].

We use five transformer-based workloads that are used in Section 2. Typical training requires running multiple epochs, and each epoch requires numerous iterations. As all the iterations exhibit the same behavior, we run a single iteration for evaluation. Since TM-Training aims to achieve maximum energy efficiency by minimizing redundant data migration, we measure the size of data migration and energy efficiency by varying Tier 1 size. We set the default setup for the ratio of the Tier 1 size to the model size to 25:1, which corresponds to allocating 12.5% of the total memory capacity to Tier 1. Then, we vary the ratio from 5:1 to 150:1. Next, we measure throughput to show that TM-Training is effective without critical performance degradation. Also, to evaluate the scalability of TM-Training in handling larger datasets, we double the batch size of the workloads listed in Table 2 and measure energy efficiency. To support the increased batch size, we expand only the Tier 2 capacity while fixing the Tier 1 capacity. Last, we conduct sensitivity studies by applying a different page replacement policy and low SSD bandwidth to TM-Training.

The ideal case applies the offline optimal policy named Hor_Off in Section 2 as the page replacement policy. Also, during the storage of output data from the accelerator into Tier 1 and Tier 2, the ideal case uses the data split technique similar to TM-Training. While TM-Training determines the split ratio by adjusting values in units of $\frac{1}{\text{sequence length}}$ as described in Algorithm 3, the ideal case fine-tunes the split ratio precisely to achieve the most optimal configuration. The baseline refers to a memory system that uses only HBM DRAM. The detailed configurations are described in Table 1. We compare TM-Training with prior work named Sentinel, AutoTM, and DeepTM, which use tiered memory to run DNN training workloads in GPU [19, 43, 56]. Sentinel only accesses Tier 1 during data prefetching to on-chip SRAM buffers [43]. Tier 2 proactively promotes data required for future operations to Tier 1 while it also demotes data already used in processing. In

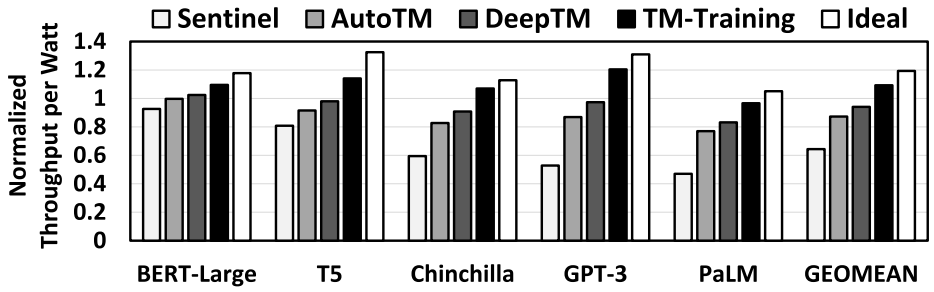


Fig. 8. Energy efficiency of the overall system. All results are normalized to the HBM-only baseline. We set the Tier 1 size to 12.5% of the total off-chip memory size, resulting in a ratio of the Tier 1 size to the model size of 25:1. In all results, TM-Training achieves better energy efficiency compared to prior work.

this context, Sentinel exhibits similar operating characteristics with the vertically-tiered memory system defined in Section 2. We implement the policy in our experimental environment by incorporating details such as migration intervals between memory tiers.

AutoTM uses **Integer Linear Programming (ILP)** to decide tensor movement and placement [19]. For each operation, it analyzes four cases, as there are four combinations where input and output data are stored across Tiers 1 and 2. For these cases, AutoTM predicts performance for applying each case using ILP. Then, it applies the most efficient case to the execution time. To reasonably model the policy in our simulation-based experimental environment, we derive all results of four cases for each operation. Then, we choose the best case among the results.

DeepTM highlights the risk of OOM errors from the ILP approach, causing performance overhead due to page fault handling [56]. Initially, DeepTM uses the ILP approach for data placement based on lifetime, size, access frequency, and recency, switching to a **Deep Reinforcement Learning (DRL)** approach during execution to prevent OOM errors. To design a simulation similar to the DRL approach, we apply the following policies: During the first layer execution, we apply the ILP approach as in AutoTM. From the second layer, we assume Tier 1 writes proceed without OOM overhead. If an OOM error occurs, then the system also performs Tier 1 eviction without run-time overhead. This design allows measurements comparable to cases where using the DRL approach.

5.2 Experimental Results

Energy Efficiency: Figure 8 depicts the energy efficiency of executing various training workloads. We set the Tier 1 size to 12.5% of the total tiered memory size, following a configuration commonly used in prior work. While running GPT-3, the variation in energy efficiency according to the applied policy is the greatest. On average, TM-Training achieves 1.71×, 1.26×, and 1.16× better energy than Sentinel, AutoTM, and DeepTM, respectively. Also, TM-Training shows 1.09× better energy efficiency than the baseline.

Although all cases reduce cost in the memory system by decreasing HBM capacity, only TM-Training achieves better energy efficiency than the baseline, while prior work architectures show lower energy efficiency. As shown in Table 2, NAND Flash consumes less static power than HBM but incurs higher dynamic energy consumption. Therefore, frequent data migration between memory tiers in prior work reduces the energy efficiency of the overall system due to increased Tier 2 access. These results highlight the importance of minimizing data migration between memory tiers to achieve better energy efficiency than the baseline.

There is some variation in the degree of improvement for each workload. For BERT-Large, the energy efficiency improvement is relatively small compared to prior work. BERT-Large requires

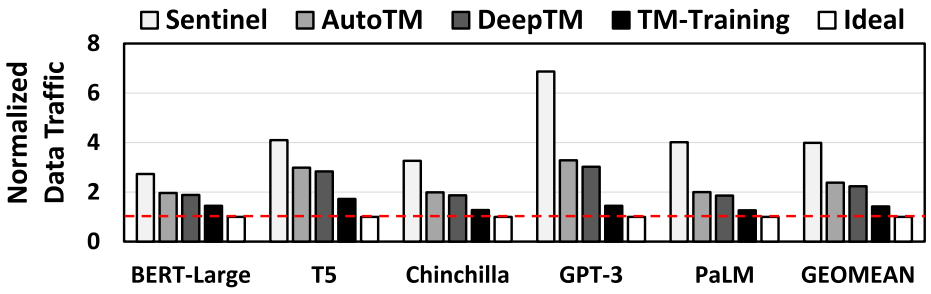


Fig. 9. Data traffic between tiers for cases running each workload. We set Tier 1 size to 12.5% of the total off-chip memory size. All results are normalized to the ideal case. In all results, TM-Training generates less data traffic than prior work.

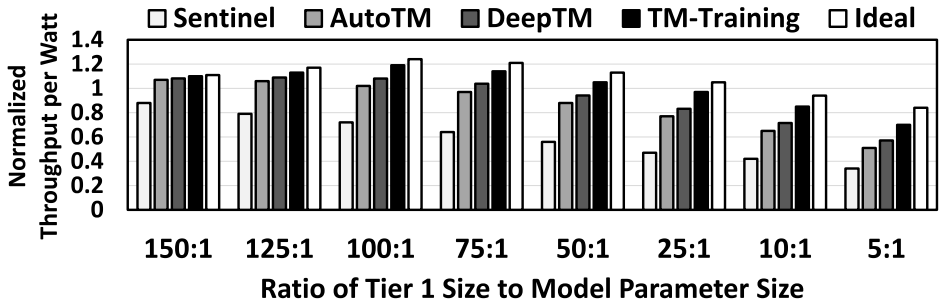


Fig. 10. Energy efficiency for running PaLM depending on the ratio of Tier 1 size to model parameter size. If the ratio is 100:1, then 50% of the total memory capacity is allocated to Tier 1. All results are normalized to the HBM-only system. For all configurations, TM-Training outperforms prior work.

smaller data than other workloads. Since the need for data-splitting arises from the challenge of managing data placement with large data, the benefits gained from data-splitting diminish as the data size becomes smaller.

Data Migration between Memory Tiers: We analyze the amount of data migrated between memory tiers. Figure 9 shows the normalized size of migrated data for all benchmarks. Unlike the aforementioned experiment measuring energy efficiency, we normalize all the results to the ideal case. It is because the HBM-only system does not have two memory tiers, and as a result, there is no data migration between memory tiers. With these experiments, we evaluate how closely each architecture approaches the ideal case in reducing data traffic. TM-Training consistently demonstrates an efficient reduction in data migration compared to prior work. On average, TM-Training exhibits 64%, 40%, and 36% less data migration compared to Sentinel, AutoTM, and DeepTM, respectively. In the case of BERT-Large, with its relatively small model size, the differences in data migration among policies are marginal. Even with this smaller model, TM-Training achieves a 47%, 27%, and 23% reduction in data migration compared to Sentinel, AutoTM, and DeepTM, respectively.

Effect of Varying Tier 1 Size: Figure 10 depicts the energy efficiency of the overall system for PaLM. Across all results, TM-Training achieves higher energy efficiency than the prior work. If the ratio is 100:1, then TM-Training achieves the highest energy efficiency. In that case, TM-Training gets 1.65×, 1.17×, and 1.1× higher energy efficiency than Sentinel, AutoTM, and DeepTM.

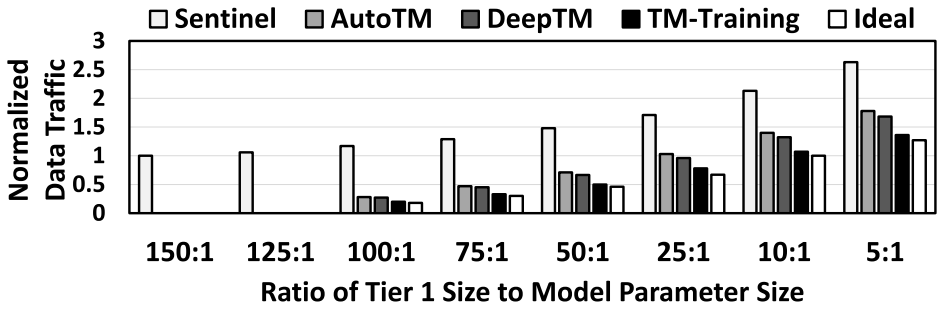


Fig. 11. Data traffic between memory tiers comparison for running PaLM. All results are normalized to the size of the total data used for running the workload. Sentinel, AutoTM, and DeepTM generate more data traffic than TM-Training.

As the Tier 1 size decreases, the efficiency gap becomes even larger. If the ratio is 5:1, then TM-Training improves energy efficiency by 2.06×, 1.37×, and 1.23× compared to Sentinel, AutoTM, and DeepTM.

Figure 11 shows the amount of data migration. For all Tier 1 memory size configurations, TM-Training offers a very similar amount of data migration to the ideal case. TM-Training successfully reduces the data migration by statically setting data placement for subsequent operations of the same type using the information obtained during the initial operation. If Tier 1 memory size is less than 32 GB, then Sentinel suffers from excessive data migration between memory tiers. This excessive data migration is because Sentinel operates as a vertically-tiered memory system. As a result, Sentinel generates almost 2× more data migration than TM-Training, regardless of the Tier 1 size.

AutoTM and DeepTM show a similar amount of data migration if the ratio is greater than 100:1. However, as the Tier 1 size becomes smaller, the two prior works generate a larger amount of data migration than TM-Training. For example, if the ratio of the Tier 1 size to the model size is 5:1, then AutoTM and DeepTM show 1.31× and 1.24× higher data migration than TM-Training. Since those prior work does not apply data-splitting, the data migration and resulting energy consumption increase significantly compared to TM-Training as the Tier 1 size gets smaller. Also, since the memory access patterns and the size of data being read or written in DNN accelerator-based systems are generally predictable, the advantages of deep reinforcement learning in DeepTM are less pronounced than the data-splitting strategy in TM-Training.

Running PaLM needs about 9 GB of memory space for the forward pass and about 22.5 GB for the backward pass. Thus, if Tier 1 is larger than 22.5 GB, then the system can run the workload with all data promoted to Tier 1. If it becomes smaller than 22.5 GB, then a smaller Tier 1 might not immediately significantly affect the overall system, depending on the policy. However, as the size of Tier 1 decreases, the gap between policies becomes wider.

Throughput: Although TM-Training focuses on improving energy efficiency, we also need to consider the performance of the overall system. Figure 12 depicts the geometric mean throughput results for all workloads. If the ratio is greater than 100:1, then it leads to negligible performance degradation compared to the HBM-only baseline. If the ratio becomes smaller, then TM-Training achieves similar or higher throughput than prior work. For example, TM-Training achieves 1.04× to 1.19× higher throughput compared to prior work if the ratio is 25:1, indicating that the Tier 1 size accounts for 12.5% of the total memory capacity.

Effect of Varying Input Data Size: Figure 13 shows the energy efficiency of the overall system while doubling the batch size. In this experiment, we set the Tier 1 size to 12.5% of the total memory

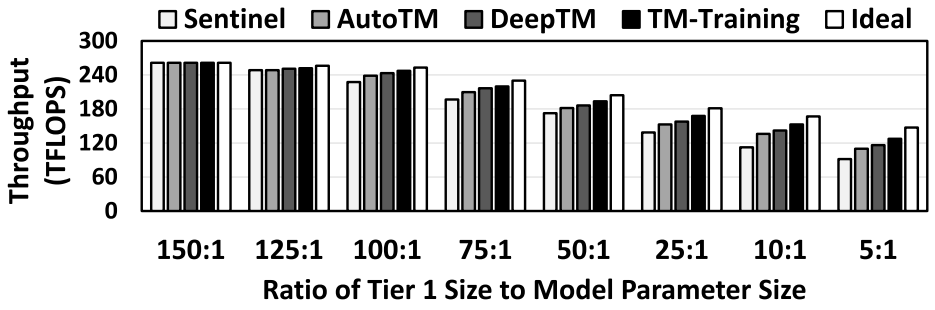


Fig. 12. Performance of the overall system depending on the ratio of Tier 1 size to model parameter size. We calculate the geometric means of the results for all workloads in Table 3. If the ratio is 100:1, then 50% of the total memory capacity is allocated to Tier 1. For all cases, TM-Training shows a smaller performance degradation compared to prior work, maintaining closer alignment with ideal throughput.

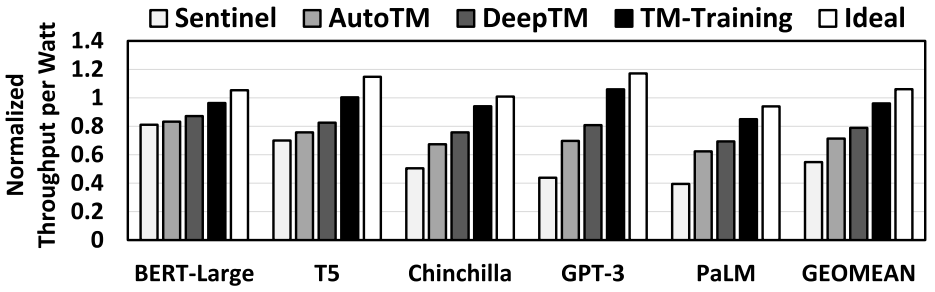


Fig. 13. Energy efficiency comparison as we double the batch size and memory capacity per chip. We set the Tier 1 size to 12.5% of the total off-chip memory size, resulting in a ratio of the Tier 1 size to the model size of 25:1. All results are normalized to the HBM-only baseline. TM-Training achieves better energy efficiency than prior work.

size, consistent with the experiment shown in Figure 8. With the increased data size to process while keeping the Tier 1 size fixed, we observe an increasing disparity in energy efficiency across architectures. On average, TM-Training achieves 1.75×, 1.35×, and 1.22× better energy efficiency than Sentinel, AutoTM, and DeepTM, respectively. Moreover, while other prior work exhibits a 21% to 45% lower energy efficiency than the HBM-only baseline, TM-Training achieves energy efficiency nearly equivalent to the baseline.

Effect of Varying Page Replacement Policy: Figure 14 shows the energy efficiency of the overall system while varying the page replacement policy. In this experiment, we measure the energy efficiency of TM-Training by applying the offline optimal policy used in Section 2. TM-Training_Off refers to TM-Training with the offline optimal policy applied instead of the LRU-based replacement policy. On average, TM-Training_Off achieves only 1.05× higher energy efficiency than TM-Training. Also, TM-Training shows 4% lower energy efficiency than the ideal case. These results suggest that TM-Training improves energy efficiency orthogonal to the page replacement policy. Also, the negligible difference between the ideal case and TM-Training_Off indicates that adjusting the data split ratio in units of $\frac{1}{\text{sequence length}}$ in TM-Training is sufficiently effective.

Effect of Varying SSD Bandwidth: Figure 15 depicts the energy efficiency results with Tier 2 bandwidth reduced by half. On average, TM-Training achieves 1.91×, 1.34×, and 1.24× higher

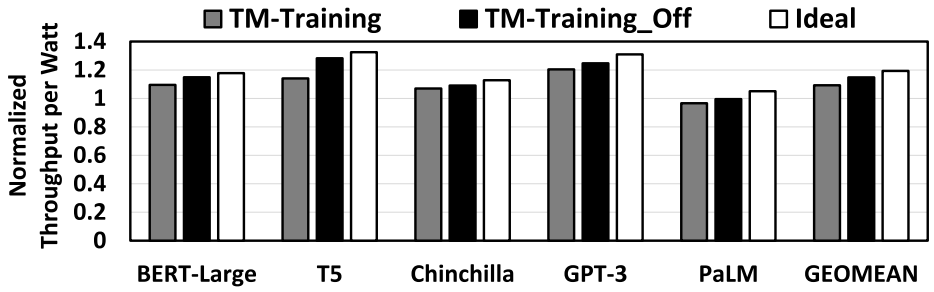


Fig. 14. Energy efficiency of the overall system. All results are normalized to the HBM-only baseline. We set the Tier 1 size to 12.5% of the total off-chip memory size, resulting in a ratio of the Tier 1 size to the model size of 25:1. In all results, TM-Training shows relatively small differences in energy efficiency depending on replacement policies.

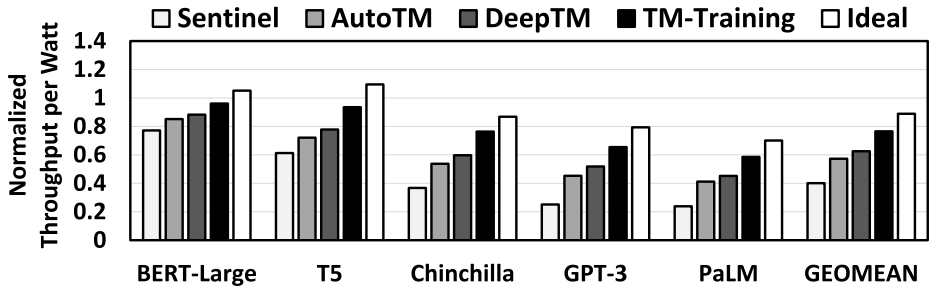


Fig. 15. Energy efficiency of the overall system with Tier 2 bandwidth reduced by half. All results are normalized to the HBM-only baseline. We set the Tier 1 size to 12.5% of the total off-chip memory size. We set the Tier 1 size to 12.5% of the total off-chip memory size, resulting in a ratio of the Tier 1 size to the model size of 25:1. In all results, TM-Training achieves better energy efficiency compared to prior work.

energy efficiency than Sentinel, AutoTM, and DeepTM, respectively, while it shows 23% less energy efficiency than the baseline. As Tier 2 bandwidth decreases, the throughput of the overall system declines. Also, the amount of data that can be prefetched directly from Tier 2 to the accelerator without additional memory stalls decreases. This limitation increases the frequency of promotions to Tier 1, leading to higher energy consumption. However, TM-Training mitigates performance degradation and energy consumption growth more effectively due to its data-splitting technique. This tendency becomes more pronounced with larger model sizes, widening the energy efficiency gap across architectures. For example, in the case of PaLM, TM-Training achieves 2.45×, 1.42×, and 1.3× higher energy efficiency than Sentinel, AutoTM, and DeepTM, respectively.

6 Discussion

In this section, we discuss the applicability of TM-Training to other types of DNN workloads such as CNN-based models. We also explore its compatibility with accelerators featuring non-systolic array-based cores.

Applicability to Non-transformer-based DNN Workloads: As shown in Figures 8 and 13, the differences in energy efficiency and throughput between TM-Training and prior work become relatively smaller for small models such as BERT-Large. If we use CNN-based models that have

much smaller parameter sizes, then the advantage of TM-Training may decrease due to significantly reduced data migration between memory tiers and notable differences in model structure. However, even for CNN-based models, the approaches of TM-Training can still be applied if the target workload involves performing repeated operations with the same dimensions across multiple layers, since TM-Training leverages the repeated computations both across layers and within a layer. For example, ResNet repeatedly applies identical convolutional operations within residual blocks [18]. Similar to ResNet, Inception Networks process feature maps through parallel branches with similar convolutional structures across multiple layers [48].

Compatibility with Non-systolic Array-based Accelerators: We adopted TPU v4 as the baseline architecture because it is well-suited for running large DNN models like LLMs and has a publicly available architecture and configuration. However, prior work also has proposed various DNN accelerator architectures with designs different from TPU v4. For example, prior work has explored DNN accelerators built on non-systolic array-based cores [32, 33]. Although computation time for each operation may vary depending on the architecture of the processing unit, DNN accelerators commonly feature scratchpad-based on-chip SRAM and compute units tailored for GEMM or GEMV operations. Despite differences in internal architectures, all the accelerators have deterministic operational characteristics. Therefore, the mechanism of TM-Training, which statically predicts near-optimal data placement by leveraging repetitive operational characteristics of target workloads, still applies to other architectures.

7 Related Work

Tiered Memory for DNN Training Workloads: Prior work has proposed techniques for using tiered memory while performing DNN training workloads [16, 19, 37, 43]. In terms of target workloads, those techniques have focused on CNN-based models such as ResNet and VGG. Although Sentinel and Han et al. have used BERT-Large and Transformer, respectively, these models are still much smaller than recently proposed DNN models [5, 9, 16, 43]. In terms of evaluation metrics, except for Mukherjee et al., all prior work mainly focuses on price and execution time [37]. Mukherjee et al. proposed a policy of storing only weight data in emerging memory, and all input data in DRAM assume that only inference workloads are in use. Unlike the prior work, TM-Training improves energy efficiency even with a smaller Tier 1 size.

Horizontally-tiered Memory System: There have been previous studies that propose various horizontally-tiered memory systems [1, 42]. FlatFlash proposed a lightweight, adaptive page promotion mechanism between SSD and DRAM to efficiently use byte-addressable SSD as part of the main memory. Ramos et al. applied **Phase Change Memory (PCM)** as a Tier 2. They have designed a policy that stores only performance-critical pages and frequently written pages in DRAM. Unlike prior work, TM-Training focuses on minimizing both data migration between memory tiers and the data migration between off-chip memory and processor.

Customized IOMMU for DNN Accelerator: Prior work has proposed IOMMU considering the characteristics of DNN accelerator [17, 21]. NeuMMU proposed a dedicated IOMMU to support CPU memory access, just as GPU can access CPU memory using unified memory [21]. NeuMMU prioritizes improving translation throughput by increasing the number of PTWs and by merging memory requests accessing the same physical page. Hao et al. proposed a 2-level IOMMU that includes L1 TLB and a shared L2 TLB to manage address translation requests from multiple accelerators [17]. Since translation techniques are orthogonal to our work, TM-Training employs the NeuMMU.

Exploiting Patterned Workload Characteristics: One of the main contributions of TM-Training is to statically determine data placement if an operation with the same characteristics has been performed previously. Prior work has exploited such characteristics in computer system

simulation designs. SimFlex has utilized statistical sampling to address the computational challenges in multiprocessor simulations, reducing simulation time by roughly a factor of 10,000 [52]. BarrierPoint predicts the overall execution time by dividing it into sections with repeating patterns and sections without repeating patterns based on **Instructions per Cycle (IPC)** [7]. There are other prior works that efficiently estimate system behaviors during the entire execution time, considering the repetitive nature of workloads [6, 51, 54].

8 Conclusion

Integration of a flash-based tiered memory into DNN accelerators introduces significant energy overhead due to the extensive data migration between memory tiers. To address this challenge, we propose a solution called TM-Training leveraging tensor splitting, which allows storing only essential components in Tier 1 while placing the remaining data in Tier 2. TM-Training exploits the behavioral characteristics of DNN training workloads that repeatedly execute the same phases. By strategically utilizing information from one segment of the total workload execution, TM-Training dynamically establishes data placement for the entire process. In our evaluation, TM-Training achieves an improvement in energy efficiency up to 55% compared to prior work.

References

[1] Ahmed Abulila, Vikram Sharma Mailthody, Zaid Qureshi, Jian Huang, Nam Sung Kim, Jinjun Xiong, and Wen-mei Hwu. 2019. FlatFlash: Exploiting the byte-accessibility of SSDs within a unified memory-storage hierarchy. In *24th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'19)*. 971–985.

[2] Anirudh Badam and Vivek S. Pai. 2011. SSDAlloc: Hybrid SSD/RAM memory management made easy. In *8th USENIX Symposium on Networked Systems Design and Implementation (NSDI'11)*.

[3] Duck-Ho Bae, Insoon Jo, Youra Adel Choi, Joo-Young Hwang, Sangyeun Cho, Dong-Gi Lee, and Jaeheon Jeong. 2018. 2B-SSD: The case for dual, byte-and block-addressable solid-state drives. In *ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA'18)*. IEEE, 425–438.

[4] Emily M. Bender, Timnit Gebru, Angelina McMillan-Major, and Shmargaret Shmitchell. 2021. On the dangers of stochastic parrots: Can language models be too big? In *ACM Conference on Fairness, Accountability, and Transparency (FAT*'21)*. 610–623.

[5] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D. Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell et al. 2020. Language models are few-shot learners. *Advan. Neural Inf. Process. Syst.* 33 (2020), 1877–1901.

[6] Trevor E. Carlson, Wim Heirman, and Lieven Eeckhout. 2013. Sampled simulation of multi-threaded applications. In *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS'13)*. IEEE, 2–12.

[7] Trevor E. Carlson, Wim Heirman, Kenzo Van Craeynest, and Lieven Eeckhout. 2014. BarrierPoint: Sampled simulation of multi-threaded applications. In *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS'14)*. IEEE, 2–12.

[8] Chiachen Chou, Aamer Jaleel, and Moinuddin Qureshi. 2017. BATMAN: Techniques for maximizing system bandwidth of memory systems with stacked-DRAM. In *International Symposium on Memory Systems*. 268–280.

[9] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann et al. 2022. PaLM: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311* (2022).

[10] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

[11] Thaleia Dimitra Doudali and Ada Gavrilovska. 2019. Mnemo: Boosting memory cost efficiency in hybrid memory systems. In *IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW'19)*. IEEE, 412–421.

[12] Assaf Eisenman, Darryl Gardner, Islam AbdelRahman, Jens Axboe, Siying Dong, Kim Hazelwood, Chris Petersen, Asaf Cidon, and Sachin Katti. 2018. Reducing DRAM footprint with NVM in Facebook. In *13th EuroSys Conference*. 1–13.

[13] Assaf Eisenman, Maxim Naumov, Darryl Gardner, Misha Smelyanskiy, Sergey Pupyrev, Kim Hazelwood, Asaf Cidon, and Sachin Katti. 2019. Bandana: Using non-volatile memory for storing deep learning models. *Proc. Mach. Learn. Syst.* 1 (2019), 40–52.

[14] Samsung Electronics. 2022. Samsung V-NAND SSD 990 PRO. Retrieved from https://download.semiconductor.samsung.com/resources/data-sheet/Samsung_NVMe_SSD_990_PRO_Datasheet_Rev.1.0.pdf

[15] Siddharth Gupta, Yunho Oh, Lei Yan, Mark Sutherland, Abhishek Bhattacharjee, Babak Falsafi, and Peter Hsu. 2023. AstriFlash: A flash-based system for online services. In *IEEE International Symposium on High-Performance Computer Architecture (HPCA'23)*. IEEE, 81–93.

[16] Myeonggyun Han, Jihoon Hyun, Seongbeom Park, and Woongki Baek. 2019. Hotness-and lifetime-aware data placement and migration for high-performance deep learning on heterogeneous memory systems. *IEEE Trans. Comput.* 69, 3 (2019), 377–391.

[17] Yuchen Hao, Zhenman Fang, Glenn Reinman, and Jason Cong. 2017. Supporting address translation for accelerator-centric architectures. In *IEEE International Symposium on High Performance Computer Architecture (HPCA'17)*. IEEE, 37–48.

[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*. 770–778.

[19] Mark Hildebrand, Jawad Khan, Sanjeev Trika, Jason Lowe-Power, and Venkatesh Akella. 2020. AutoTM: Automatic tensor movement in heterogeneous memory systems using integer linear programming. In *25th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'20)*. 875–890.

[20] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark et al. 2022. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556* (2022).

[21] Bongjoon Hyun, Youngeun Kwon, Yujeong Choi, John Kim, and Minsoo Rhu. 2020. NeuMMU: Architectural support for efficient address translations in neural processing units. In *25th International Conference on Architectural Support for Programming Languages and Operating Systems*. 1109–1124.

[22] Meenatchi Jagasivamani, Candace Walden, Devesh Singh, Luyi Kang, Shang Li, Mehdi Asnaashari, Sylvain Dubois, Donald Yeung, and Bruce Jacob. 2019. Design for ReRAM-based main-memory architectures. In *International Symposium on Memory Systems*. 342–350.

[23] Djordje Jevdjic, Gabriel H. Loh, Cansu Kaynak, and Babak Falsafi. 2014. Unison cache: A scalable and effective die-stacked DRAM cache. In *47th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE, 25–37.

[24] Youngbin Jin, Mustafa Shihab, and Myoungsoo Jung. 2014. Area, power, and latency considerations of STT-MRAM to substitute for main memory. In *International Symposium on Computer Architecture (ISCA'14)*. 1–4.

[25] Norm Jouppi, George Kurian, Sheng Li, Peter Ma, Rahul Nagarajan, Lifeng Nai, Nishant Patil, Suvinay Subramanian, Andy Swing, Brian Towles et al. 2023. TPU v4: An optically reconfigurable supercomputer for machine learning with hardware support for embeddings. In *50th Annual International Symposium on Computer Architecture*. 1–14.

[26] Norman P. Jouppi, Doe Hyun Yoon, Matthew Ashcraft, Mark Gottscho, Thomas B. Jablin, George Kurian, James Laudon, Sheng Li, Peter Ma, Xiaoyu Ma et al. 2021. Ten lessons from three generations shaped Google's TPUv4i: Industrial product. In *ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA'21)*. IEEE, 1–14.

[27] Hongju Kal, Seokmin Lee, Gun Ko, and Won Woo Ro. 2021. SPACE: Locality-aware processing in heterogeneous memory for personalized recommendations. In *ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA'21)*. IEEE, 679–691.

[28] Sudarsun Kannan, Ada Gavrilovska, Vishal Gupta, and Karsten Schwan. 2017. HeteroOS: OS design for heterogeneous memory management in datacenter. In *44th Annual International Symposium on Computer Architecture*. 521–534.

[29] Hiwot Tadese Kassa, Paul Johnson, Jason Akers, Mrinmoy Ghosh, Andrew Tulloch, Dheevatsa Mudigere, Jongsoo Park, Xing Liu, Ronald Dreslinski, and Ehsan K. Ardestani. 2023. MTrainS: Improving DLRM training efficiency using heterogeneous memories. *arXiv preprint arXiv:2305.01515* (2023).

[30] Hwajung Kim and Heon Y. Yeom. 2022. LPR: Learning-based page replacement scheme for scientific applications. In *23rd International Middleware Conference Industrial Track*. 36–42.

[31] Emre Kültürsay, Mahmut Kandemir, Anand Sivasubramaniam, and Onur Mutlu. 2013. Evaluating STT-RAM as an energy-efficient main memory alternative. In *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS'13)*. IEEE, 256–267.

[32] Hyoukjun Kwon, Prasanth Chatarasi, Vivek Sarkar, Tushar Krishna, Michael Pellauer, and Angshuman Parashar. 2020. Maestro: A data-centric approach to understand reuse, performance, and hardware cost of DNN mappings. *IEEE Micro* 40, 3 (2020), 20–29.

[33] Hyoukjun Kwon, Ananda Samajdar, and Tushar Krishna. 2018. MAERI: Enabling flexible dataflow mapping over DNN accelerators via reconfigurable interconnects. *ACM SIGPLAN Not.* 53, 2 (2018), 461–475.

[34] Haikun Liu, Renshan Liu, Xiaofei Liao, Hai Jin, Bingsheng He, and Yu Zhang. 2020. Object-level memory allocation and migration in hybrid memory systems. *IEEE Trans. Comput.* 69, 9 (2020), 1401–1413.

[35] Lei Liu, Shengjie Yang, Lu Peng, and Xinyu Li. 2019. Hierarchical hybrid memory management in OS for tiered memory systems. *IEEE Trans. Parallel Distrib. Syst.* 30, 10 (2019), 2223–2236.

[36] Radoslav Mladenov. 2012. An efficient non-volatile main memory using phase change memory. In *13th International Conference on Computer Systems and Technologies*. 45–51.
[37] Avilash Mukherjee, Kumar Saurav, Prashant Nair, Sudip Shekhar, and Mieszko Lis. 2021. A case for emerging memories in DNN accelerators. In *Design, Automation & Test in Europe Conference & Exhibition (DATE'21)*. IEEE, 938–941.
[38] Xiaonan Nie, Xupeng Miao, Zhi Yang, and Bin Cui. 2022. TSPLIT: Fine-grained GPU memory management for efficient DNN training via tensor splitting. In *IEEE 38th International Conference on Data Engineering (ICDE'22)*. IEEE, 2615–2628.
[39] Mike O'Connor, Niladrish Chatterjee, Donghyuk Lee, John Wilson, Aditya Agrawal, Stephen W. Keckler, and William J. Dally. 2017. Fine-grained DRAM: Energy-efficient DRAM for extreme bandwidth systems. In *50th Annual IEEE/ACM International Symposium on Microarchitecture*. 41–54.
[40] Moinuddin K. Qureshi, Vijayalakshmi Srinivasan, and Jude A. Rivers. 2009. Scalable high performance main memory system using phase-change memory technology. In *36th Annual International Symposium on Computer Architecture (ISCA'09)*. 24–33.
[41] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.* 21, 1 (2020), 5485–5551.
[42] Luiz E. Ramos, Eugene Gorbatov, and Ricardo Bianchini. 2011. Page placement in hybrid memory systems. In *International Conference on Supercomputing*. 85–95.
[43] Jie Ren, Jiaolin Luo, Kai Wu, Minjia Zhang, Hyeran Jeon, and Dong Li. 2021. Sentinel: Efficient tensor migration and allocation on heterogeneous memory systems for deep learning. In *IEEE International Symposium on High-Performance Computer Architecture (HPCA'21)*. IEEE, 598–611.
[44] Ananda Samajdar, Jan Moritz Joseph, Yuhao Zhu, Paul Whatmough, Matthew Mattina, and Tushar Krishna. 2020. A systematic methodology for characterizing scalability of DNN accelerators using SCALE-SIM. In *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS'20)*. IEEE, 58–68.
[45] Samsung. 2024. 990 PRO PCIe 4.0 NVMe SSD 1TB. Retrieved from https://www.samsung.com/us/computing/memory-storage/solid-state-drives/990-pro-pcie--4-0-nvme--ssd-1tb-mz-v9p1t0b-am/
[46] Anton Shilov. 2023. Report: DDR5 RDIMM production impacted by PMIC compatibility issues. Retrieved from https://www.anandtech.com/show/18830/report-ddr5-rdimm-production-impacted-by-pmic-compatibility-issues
[47] Anton Shilov. 2024. Explosive HBM demand fueling an expected 20% increase in DDR5 memory pricing - Demand for AI GPUs drives production cuts for standard PC memory. Retrieved from https://www.tomshardware.com/pc-components/gpus/explosive-hbm-demand-fueling-an-expected-20-increase-in-ddr5-memory-pricing-demand-for-ai-gpus-drives-production-cuts-for-standard-pc-memory
[48] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. Going deeper with convolutions. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR'15)*. 1–9.
[49] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advan. Neural Inf. Process. Syst.* 30 (2017), 6000–6010.
[50] Junyu Wei, Guangyan Zhang, Junchao Chen, Yang Wang, Weimin Zheng, Tingtao Sun, Jiesheng Wu, and Jiangwei Jiang. 2024. Exploiting data-pattern-aware vertical partitioning to achieve fast and low-cost cloud log storage. *ACM Trans. Stor.* 20, 2, Article 12 (Feb. 2024), 35 pages. DOI: https://doi.org/10.1145/3643641
[51] Thomas F. Wenisch, Roland E. Wunderlich, Babak Falsafi, and James C. Hoe. 2006. Simulation sampling with livepoints. In *IEEE International Symposium on Performance Analysis of Systems and Software*. IEEE, 2–12.
[52] Thomas F. Wenisch, Roland E. Wunderlich, Michael Ferdman, Anastassia Ailamaki, Babak Falsafi, and James C. Hoe. 2006. SimFlex: Statistical sampling of computer system simulation. *IEEE Micro* 26, 4 (2006), 18–31.
[53] Yannan Nellie Wu, Joel S. Emer, and Vivienne Sze. 2019. Accelergy: An architecture-level energy estimation methodology for accelerator designs. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD'19)*. IEEE, 1–8.
[54] Roland E. Wunderlich, Thomas F. Wenisch, Babak Falsafi, and James C. Hoe. 2003. SMARTS: Accelerating microarchitecture simulation via rigorous statistical sampling. In *30th Annual International Symposium on Computer Architecture (ISCA'03)*. 84–97.
[55] Vinson Young, Chiachen Chou, Aamer Jaleel, and Moinuddin Qureshi. 2018. Accord: Enabling associativity for gigascale dram caches by coordinating way-install and way-prediction. In *ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA'18)*. IEEE, 328–339.
[56] Haoran Zhou, Wei Rang, Hongyang Chen, Xiaobo Zhou, and Dazhao Cheng. 2024. DeepTM: Efficient tensor management in heterogeneous memory for DNN training. *IEEE Trans. Parallel Distributed Syst.* 35, 11 (2024), 1920–1935.

Received 1 July 2024; revised 15 December 2024; accepted 31 January 2025