E2-NVM: A Memory-Aware Write Scheme to Improve Energy Efficiency and Write Endurance of NVMs using Variational Autoencoders

Saeed Kargar
University of California, Santa Cruz
skargar@ucsc.edu

Binbin Gu
University of California, Irvine
binbing@uci.edu

Sangeetha Abdu Jyothi
University of California, Irvine and VMware Research
sangeetha.aj@uci.edu

Faisal Nawab
University of California, Irvine
nawabf@uci.edu

ABSTRACT

We introduce *E2-NVM*, a software-level memory-aware storage layer to improve the Energy efficiency and write Endurance (E2) of NVMs. E2-NVM employs a Variational Autoencoder (VAE) based design to direct the write operations judiciously to the memory segments that minimize bit flips. E2-NVM can be augmented with existing indexing solutions. E2-NVM can also be combined with prior hardware-based solutions to further improve efficiency. We performed real evaluations on an Optane memory device that show that E2-NVM can achieve up to 56% reduction in energy consumption.

1 INTRODUCTION

NVM technologies suffer from two main challenges that needs to be taken into consideration: (1) NVM write operations demand a significant amount of current and power. For flipping an individual bit in PCM, for instance, it requires around 50 pJ/b. This is significant when compared to writing a whole DRAM page which needs only 1 pJ/b [6]. (2) NVM has low write endurance (the number of writes that can be applied to a segment of storage media before it becomes unreliable.) NVM write endurance is on the order of 10^8–10^9 writes, which is significantly lower than DRAM write endurance which is on the order of 10^{15} writes [27, 29, 40].

To overcome the energy consumption and write endurance problems in NVM, two approaches were developed. The first approach develops hardware-based write optimization techniques [1, 10, 15, 23, 46] that are mostly based on a *Read-Before-Write* (RBW) pattern [52]. In RBW, a write operation w to a memory location x is always preceded with a read of x. The value to be written by w is compared with the old content of x, and only the bits that are different are written. This reduces the number of flipped bits, which reduces energy consumption and increases write endurance [52]. The second approach tackles the problem of energy consumption and write endurance by minimizing write amplification [4, 9, 25, 34, 45, 54]. However, these methods conflate the problem of energy efficiency and write endurance with the problem of write amplification. Although in many cases a technique that leads to reducing write amplification has the side-effect of increasing energy efficiency and write endurance, this is not always the case as shown by prior work [6, 26, 27] and our evaluations in this paper.

In this work, we identify a crucial opportunity to increase energy efficiency and write endurance that prior solutions overlooked—*memory-awareness*. Prior methods pick the memory location for a write operation arbitrarily (new data items select an arbitrary location in memory, and updates to data items overwrite the previously-chosen location.) This misses the opportunity to judiciously pick a memory location that is similar to the value to be written (in terms of their hamming distance), which can reduce the number of updated bits. Reducing the number of bit flips increases write endurance and reduces power consumption in many NVM technologies [6, 22, 26, 48, 53].

We present a software-level memory-aware solution, E2-NVM. E2-NVM is implemented in software and does not suffer from the compute and space constraints of solutions implemented in the memory controller. E2-NVM is implemented as a storage layer that maps free memory locations according to their hamming distance. Incoming write operations are then intercepted, and placed on a free memory location that is similar in terms of their hamming distance. To perform this mapping, E2-NVM trains a deep learning model using the free memory locations. The use of deep learning is possible because E2-NVM is implemented in software rather than in the memory controller. In the paper, we present and discuss the challenges we faced in applying VAE to this problem. This includes using an efficient model, which is a combination of a VAE and a clustering model, to overcome the limitations of traditional clustering methods when they have to deal with high dimensional data. We also tackle the problem of supporting memory segments of variable sizes. We propose a data padding strategy that allows using the same VAE model for memory segments with different sizes.

2 BACKGROUND

2.1 System Model

The system model consists of hardware and software components. E2-NVM does not require any special hardware. We consider a hybrid DRAM-NVM architecture, where both devices are placed on the memory bus. The NVM device, in addition to the memory segments, contains a *memory controller* that intercepts all operations to NVM. The memory controller may utilize a wear leveling solution that swaps memory segments periodically. The details of wear leveling methods are typically proprietary. However, prior work has indicated that wear leveling approaches perform a memory segment swap every ψ write operations. Typically, the value of ψ is in the order of 10s of writes [22]. E2-NVM is a storage layer that sits between software applications (such as data stores) and the hardware components.

2.2 Motivation: Software-Level Bit Flipping Reduction

With the advent of Compute Express Link (CXL), new capabilities such as memory pooling were defined that will broaden the deployment of production CXL solutions. CXL is an open-source multi-protocol method that specifies how to deliver high-performance interconnects between different CPUs, GPUs, TPUs

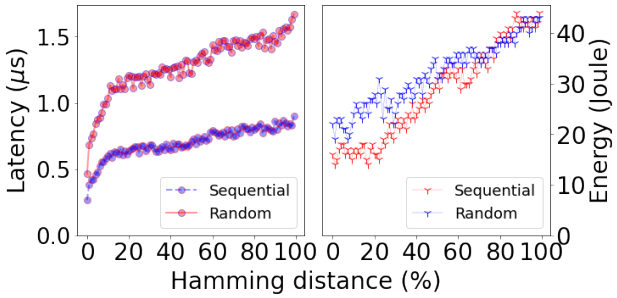


Figure 1: The latency and memory energy consumption on a real Intel Optane memory device [29].

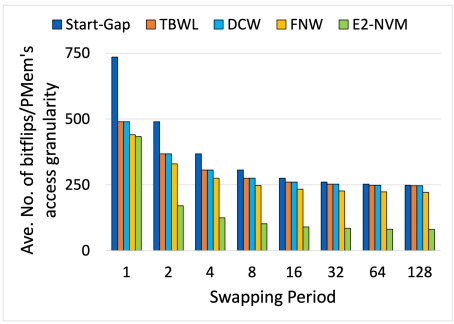


Figure 2: The average number of bit updates for different wear-leveling techniques when swapping period changes.

and memory device types. This means that CXL-enabled non-volatile memories will be able to work with CPUs other than Intel's, which will enable much bigger pools of this non-volatile memories compared to the largest DRAM cache in commercial storage, which is a couple of TB. CXL makes it especially possible for data centers, IoTs and mobile devices to deploy non-volatile memory in their systems [49].

To see how bit flip reduction affects the latency and energy consumption of a NVM device, we have conducted an experiment on a real Optane memory device, which is one type of PCM, using the Persistent Memory Development Kit (PMDK) [1], formerly known as NVML. In this test, first, we allocate a contiguous region of N Optane blocks of 256B. During each "round" of the experiment, we first initialize all the blocks with random data, and then update the blocks with new data with content that is x% different than the data that is already in the block (hamming distance). We use PMDK's transactions to persist writes. We measure the latency and energy consumption for each round. Figure 1 shows that by overwriting similar content, which needs less bit flipping, we can achieve an average energy savings of up to 56%. The experiments also show the potential of improving write latency which is important as it can offset some of the overhead that is incurred by software-level solutions that aim to reduce bit flips. This improvement in latency is due to the ability to write fewer cache lines when the cache line to be written is identical to the one in the memory segment. In this case, the memory controller avoids writing them, which reduces the average latency [26].

This potential of reducing bit flips using software-level solutions overcomes two challenges that faced hardware solutions: The first is that to be deployed on hardware, algorithms need to be small and simplistic—in terms of computation power and memory space—to fit in the memory controller. The second is that developing hardware-based methods is not accessible to researchers. This is evident by how most storage solutions for wear leveling and bit flip reduction are proprietary and requires manufacturing new hardware to implement a new solution. It is worth noting that although we provided our results on Optane, which is one type of PCM, E2-NVM is applicable to other phase change material-based technologies, such as phase-change random access memory (PRAM) and Resistive RAM (RRAM), which can benefit from bit flip reduction. Since E2-NVM's main focus is to improve energy consumption of the system, our proposed method can be especially attractive to the applications that use low power PCM devices due to relying on energy-harvesting systems or batteries [7], such as the Internet of Things (IoT) and mobile devices.

Figure 2 shows how E2-NVM can achieve its goals despite the interference and segment swapping from the underlying memory controller. We used Amazon Access Samples Data Set [41], which is described in the evaluation section. We also show how E2-NVM compare with prior hardware-based bit flip-reduction techniques that we describe in more detail in the evaluation section [10, 23, 26, 37, 52]. The figure shows the performance of E2-NVM while varying the frequency, ψ, of the underlying wear-leveling swapping of memory segments (this experiment utilizes an emulation of the memory controller as such parameters cannot be manipulated on typical real memory controllers.) When the frequency ψ is 1, then the swap is performed for every write operation, which means that E2-NVM judicious memory segment choice is swapped. This leads to not observing the benefits of the software-level approach. (A low ψ value is also not good for hardware-based methods because it means that more bit flips are incurred due to frequent swapping.) As we increase ψ to normal levels, E2-NVM shows that software-level approaches are capable of significant improvement.

2.3 Related Work

Integrating NVMs into existing computer systems requires to develop new NVM-friendly data structures [44] that focus on special properties such as reducing write amplification [25], being lock-free [42, 43], reducing bit flips [10, 15, 28], and so on. The research is broadly classified into three main classes:

Bit flip reduction. One of the most prevalent techniques in this category is called "Read-Before-Write" (RBW) [10, 15, 23]. For example, Flip-n-Write (FNW) [10] compares the current content of the memory location (the old data) with the content to-be-written (the new data). This enables FNW to decide whether to write the new data in its original format or to flip it before writing it if that leads to reducing bit flips. Data Content-aware (DATACON) [48], which is implemented inside the memory controller, reduces the latency and energy of PCM writes by redirecting the write requests to a new physical address within memory to overwrite memory locations containing all-zeros or all-ones depending on the content of the incoming writes. Hamming-Tree [28, 30] is another method that minimizes bit flips through organizing memory contents based on their hamming distance on a tree data structure. Predict and Write (PNW) [26] uses a simple clustering model to find memory segments that minimize bit flips.

Reducing write amplification. Techniques for write amplification reduction include delaying the consolidation of writes [25, 34], caching [4, 9, 45], and others [35, 54]. With the introduction of NVM to the memory hierarchy, it turns out that reducing write amplification can have the positive side-effect of improving energy efficiency and write endurance since less data is written. Nevertheless, reducing write amplification does not enable reaching the full potential of such improvements that can

[1]Persistent Memory Development Kit https://pmem.io/pmdk/.

be attained with bit flip reduction [6, 26]. This is because—unlike flash—NVM cells are written individually, which means that the number of flipped bits is more critical to optimize than the total number of written words [6].

Wear leveling. Wear leveling [10, 15, 22, 47, 48, 53] extends the lifetime of NVM devices by distributing the writes evenly across the memory blocks of NVM so that no *hot* area reaches its maximum lifespan [8]. This, however, means that wear leveling does not reduce the actual number of bit flips—it only distributes them across the device. The benefits of reducing bit flips in terms of energy efficiency would not be observed with wear leveling. In fact, wear leveling may introduce more bit flips—and thus more energy consumption—due to the swap operation.

3 E2-NVM DESIGN

3.1 Variational Autoencoder (VAE)

The representation ability of dimensionality reduction techniques like PCA is limited at scale. Many applications from power systems to health care to storage systems and database systems use deep learning as a feasible alternative that can provide low dimensional learned features with lower preprocessing and training delay while preserving intrinsic local structure in data [3, 17, 20, 51]. In this work, we choose Variational Autoencoders as our model of choice for several reasons: high representation ability, fast training, and the ability to jointly perform clustering and model training [20]. VAE can be considered as a generative variant of Auto Encoder (AE), as it enforces the latent code of AE to follow a predefined distribution [39].

Our VAE consists of an encoder, a decoder, and a loss function. The encoder part is a deep neural network with weights and biases θ, which takes a memory segment x as input and encodes it into a latent (hidden) representation space z, which has much less features/dimensions than x. The main responsibility of the encoder is to learn an efficient compression of the data x into this lower-dimensional space z. The decoder part is another deep neural network with weights and biases ϕ, which takes the latent representation z produced by the encoder and outputs the parameters to the probability distribution of the data. Finally, since the loss function of the VAE is the negative log-likelihood with a regulizer, the total loss is $\sum_{i=1}^{N} l_i$ for the total data points, where the loss function l_i for a single data point x_i is calculated as bellow:

$$l_i(\theta, \phi) = -\mathbb{E}_{z \sim q_\theta(z|x_i)}[\log p_\phi(x_i|z)] + \mathbb{KL}(q_\theta(z|x_i)||p(z))$$

where $q_\theta(z|x_i)$ and $p_\phi(x_i|z)$ denote the encoder's and decoder's distributions, respectively. The first term is the reconstruction loss (expected negative log-likelihood of the ith data point). The expectation is taken with respect to the encoder's distribution over the representations. This is the reason that makes the decoder learn to reconstruct the data. The second term is the Kullback-Leibler divergence between the encoder's distribution $q_\theta(z \mid x)q$ and $p(z)$, which measures how p is close to q. It is worth noting that, in the VAEs, p is specified as a standard normal distribution with mean 0 and variance 1 [2].

3.2 E2-NVM Design

At the core of E2-NVM is a VAE, an unsupervised ML model, which in combination with K-means clustering maps memory locations into clusters based on the similarity of their content. The VAE-based clustering model is trained/re-trained based on the bit-wise contents of the available memory locations/segments on PCM, and learns the existing data distribution in memory. E2-NVM integrates the VAE's reconstruction loss and the K-means clustering loss to jointly train cluster label assignment and learning of suitable features for clustering. In other words, E2-NVM jointly optimizes (a) reduction of high-dimensional input to low-dimensional latent space representation and (b) clustering in the low-dimensional latent space. E2-NVM can scale to much larger input sizes compared to prior work since clustering is performed on the latent space.

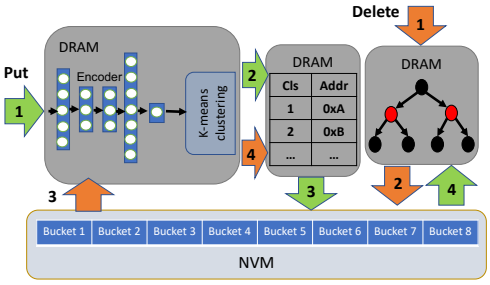


Figure 3: Memory layout of a E2-NVM-based key-value store.

Each memory location is encoded as a vector of bits, each of which is used as a feature/dimension. The entire data zone/memory pool can be encoded as a 2D tensor (that is, an array of vectors) of shape (n, m), where the first axis (n) represents the samples (old data) and the second axis (m) represents the features. The model takes the input of size m and downsizes it to a low-dimensional latent space (e.g., size 10) and feeds it to the K-means clustering model. E2-NVM also employs a padding strategy (§4) to accommodate data items of arbitrary size.

3.3 E2-NVM Integration and Operations

We present a persistent key/value store that is built on a hybrid DRAM-NVM memory using E2-NVM (Figure 3.)

3.3.1 System Model. The components of the system are:

E2-NVM. This component includes the VAE and k-means clustering models. This component is trained using the data on NVM to enable creating k clusters based on hamming distance similarity. After training, only the encoder part of the VAE and the K-means clustering models are needed. In operation, this component is used to predict the cluster of a given data object.

Cluster-to-memory dynamic address pool. This component is responsible for tracking the free memory segments that belong to each cluster. It is implemented as a simple mapping data structure where the key is the cluster id and the value is a list of all the free memory addresses that belong to the cluster. This map is initially populated in the initialization phase when E2-NVM is trained. The population is performed by running the prediction algorithm using E2-NVM on the free memory addresses. During normal operation, the mapping structure is mutated in response to operations. A PUT operation would result in removing the chosen address from the pool and the DELETE operation results in adding (recycling) the deleted memory address back to the pool. It is worth noting that, when a write request comes to the system and its corresponding cluster is found, E2-NVM returns the first available address in the cluster. Although searching within clusters can result in finding the perfect matches, we do not do that since all the similar addresses in terms of their hamming distance are grouped into one cluster. So, we just take the first available address in the cluster knowing that it will have a very similar content to the request. Our evaluation shows that this design decision allows good bit flip reduction even without having to search for the ideal address within a cluster.

The storage overhead of the dynamic address pool is proportional to the number of memory segments (or buckets) in the memory pool. Given a specific memory pool size,

for large memory segments, the size of the table does not grow significantly. For small values, however, the number of addresses that needs to be stored per memory segment can grow substantially. To limit the table size, we have two options: (1) Setting a fixed number of entries in the table, so the size of the table cannot not grow to more than a specific maximum threshold. (2) Depending on the size of the memory pool, choosing the size of the memory segments in a way that while limiting the size of the table within a specific threshold, we achieve the expected energy performance. To find the most efficient segment size for a specific size memory pool, we have conducted some tests, which are discussed in section 4.1.4.

Data index. This is the component that corresponds to the application. In this case, it is a key-value store. This indexing component maps each key to the memory location that contains its value in NVM.

NVM storage. This component represents the NVM persistent storage used to store data for the key-value store. The storage is divided into fixed-sized memory segments.

3.3.2 Data operations. Key-value operations can be divided into operations that write data (PUT and UPDATE), delete data (DELETE), and read data (GET and SCAN). Each one of these three types is implemented via an algorithm that would potentially mutate the state of the four components in Figure 3.

Write operation algorithm. Algorithm 1 illustrates the pseudo-code of the write operation under the E2-NVM scheme (shown by the green arrows in Figure 3). To locate the appropriate memory location for an incoming key/value write, i.e., new PUT or UPDATE operations, the input data is first processed by E2-NVM's encoder (step 1). In E2-NVM, the input data is transformed to the latent space representation using the VAE encoder. Then, using K-means clustering, the low-dimensional representation is mapped to the cluster that is closest to the value-to-be-written in terms of hamming distance (line 1 of the algorithm). Then, the chosen cluster is passed to the dynamic address pool (step 2 in the figure and line 2 in the algorithm). The dynamic address pool chooses a memory segment from the cluster and assigns it to the write operation. After removing the address from DAP (line 4), the write operation is applied to the chosen address in NVM (step 3 and lines 5 and 6). Finally, an index entry is added to the application's data index (step 4 and line 7).

Algorithm 1: Write operation

```
// D' and D: old and new (key,value)
// DAP: Cluster to Memory Dynamic
 Address Pool
Write (D: (key,value)){
 1: E = E2-NVM-model.predict(D); //predict
    the entry
 2: A = DAP.get(E);//get the address
 3: D'= Read(A); //old (key,value)
 4: DAP.remove(A) //remove the address from
    DAP
 5: for each bit in {D} and {D'}
 6:   if they differ, update memory bit
 7: RB-Tree.put(D, A) //update the index}
```

Algorithm 2: DELETE operation

```
// D': old key
// DAP: Dynamic Address Pool
// RB-Tree: Red-Black Tree
Delete (D': key){
 1: A = RB-Tree.get(D'); //get the address
 2: Reset-Flag-Bit(A);//delete
 3: E = E2-NVM-model.predict(Read(A));
    //predict the entry
 4: DAP.update(A:address, E:entry);//add
    the address back to DAP}
```

DELETE operation algorithm. Algorithm 2 illustrates the delete operation (shown by the red arrows in Figure 3). To perform a DELETE operation, the key is first sent to the tree to find the item's location in the key-value store in NVM (step 1, line1). The associated entry is then deleted from the key-value store by resetting the associated flag bit (step 2, line 2). We recycle the recently freed address back to the free list; the address (and its content) is sent through the encoder and then K-means clustering to find a suitable cluster (step 3, line 3). Then, the memory address is added to the corresponding cluster in the dynamic address pool so it can be used for future operations (step 4, line4).

Read operation algorithm. The GET operation is sent through the data indexing tree to find its location in NVM storage.

SCAN operation algorithm. Similar to GET operations, a SCAN operation is directed to the indexing data structure to find the range of key-value pairs to be read and returned to the user.

3.4 E2-NVM benefits

VAE-based clustering employed by E2-NVM offers several benefits over prior memory-aware techniques: lower training time, higher accuracy, and better scalability. To demonstrate the benefits of E2-NVM, in Figure 4, we compare E2-NVM with two methods: K-means clustering and K-means clustering in combination with PCA as employed by a recent memory-aware clustering method PNW [26]. In the comparison, we use the MNIST dataset. We measure two key metrics of performance, latency and number of bit flips.

We train the clustering models on NVIDIA Tesla K80 GPU. We group the incoming data of different sizes into 20 clusters. The size of the input data is 70, 000 and the number of features or the latent space representation is varied from 32 to 16384. Figure 4 shows that when the number of features—here the number of bits—increases beyond a couple of thousands, pre-processing latency of K-means clustering is extremely high. The results show that using K-means clustering alone (without PCA) is not a feasible choice for item sizes of kilobytes or more since the pre-processing time goes up exponentially with the increase in the number of features (the number of bits). For large data sizes, the second mode (PCA + K-means) is the only viable choice under PNW due to latency constraints. However, due to loss of information arising from dimensionality reduction with PCA, clustering efficiency is affected. Hence, the number of bit flips increases when moving from K-means to K-means+PCA.

Figure 4 also presents the results for the VAE-based clustering model of E2-NVM. This model needs significantly less time than PNW for training, which includes both VAE and K-means clustering training. This is because the VAE can decrease the dimensionality from tens of thousands to hundreds very fast while also minimizing data loss. E2-NVM minimizes both latency and the number of bit flips significantly. This enables E2-NVM to support memory-awareness for data with larger sizes in the order of kilo to megabytes.

4 THE PADDING STRATEGY

4.1 Padding Strategies

Overview. In machine learning methods, the input size needs to be defined when the model is created. To support inputs of

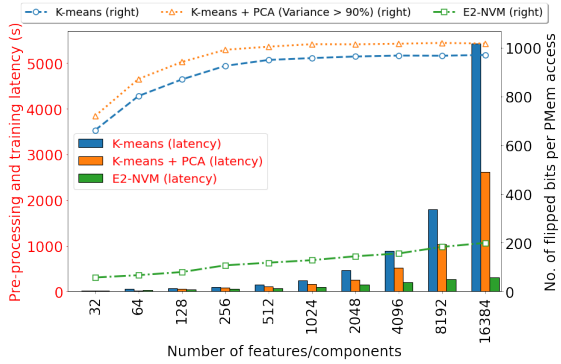


Figure 4: Comparison of E2-NVM with PNW (K-means alone and K-means+PCA) in terms of the number of bit flips and latency.

Table 1: An example of a PCM with 12 memory segments.

Cluster	Index	Content
0	0	[0, 0, 1, 1, 1, 1, 0, 1]
	1	[0, 0, 1, 0, 1, 1, 0, 0]
	2	[0, 0, 1, 1, 1, 1, 0, 0]
	3	[0, 0, 1, 1, 1, 0, 0, 0]
1	4	[1, 0, 0, 0, 1, 0, 1, 1]
	5	[0, 0, 0, 0, 1, 0, 1, 1]
	6	[0, 0, 0, 0, 1, 1, 1, 1]
	7	[0, 0, 0, 0, 1, 0, 1, 0]
2	8	[1, 0, 1, 1, 0, 0, 0, 0]
	9	[0, 1, 1, 1, 0, 0, 1, 0]
	10	[1, 1, 1, 1, 0, 0, 0, 0]
	11	[1, 1, 0, 1, 0, 0, 0, 0]

various sizes, we propose a padding strategy. In this strategy, first, our deep learning model is trained on fixed-sized features, for instance w, like the other deep learning models. When the model is ready, it can serve input data with size w. But, for input data with a different size, p, which is smaller than w, we need to transform it into an item with size w. To do that, in this paper, we use padding, which is adding q=w-p bits to the item to fit the input layer of our DL model. The ultimate goal is to pad the inputs of smaller sizes in such a way that it ends up in the cluster with the most similar items, which means minimizing the number of bit flips. It is worth noting that the padded part with size q is not a part of the main data and is added to the data just for clustering purposes. Only the actual data of size p is written to the target memory and the remaining q bits are ignored (not written to storage). Figure 5 shows an example of E2-NVM's different padding strategies on an input data d1:[0,0,0,1].

Challenges. The padding strategy decides (a) where the padded bits should be placed relative to the original input data, and (b) what bits to use in padding. The padding strategy influences the accuracy of detection and consequently finding a suitable memory segment. However, there is a trade-off between the complexity of the padding strategy and the effect on the accuracy of the deep learning model. In the remainder of this section, we discuss a number of strategies (see Figure 5) in the spectrum of this trade-off.

Padding location. The location of the padded bits can be one of the followings: (i) padding bits before the input data (beginning-padding), (ii) padding bits after the input data (end-padding), or (iii) padding bits are split, with one half before the input data and the other half after the input data (middle-padding).

Input	[0, 0, 0, 1]						
Padding Technique Loc	Beginning Padding: [?, ?, ?, ?, 0, 0, 0, 1]						
Padding Technique Type	Universal data-agnostic			Universal data-aware			Learned
	Zero	One	Rand	IB	DB	MB	
Output	[0, 0, 0, 0, 0, 0, 0, 1]	[1, 1, 1, 1, 0, 0, 0, 1]	[1, 1, 0, 1, 0, 0, 0, 1]	[0, 1, 0, 0, 0, 0, 0, 1]	[1, 1, 1, 1, 0, 0, 0, 1]	[0, 0, 1, 1, 0, 0, 0, 1]	[1, 1, 1, 1, 0, 0, 0, 1]
Pred Cls	[1]	[2]	[2]	[1]	[2]	[0]	[2]
Padding Technique Loc	Middle Padding: [0, 0, ?, ?, ?, ?, 0, 1]						
Padding Technique Type	Universal data-agnostic			Universal data-aware			Learned
	Zero	One	Rand	IB	DB	MB	
Output	[0, 0, 0, 0, 0, 0, 0, 1]	[0, 0, 1, 1, 1, 1, 0, 1]	[0, 0, 1, 0, 0, 1, 0, 1]	[0, 0, 1, 0, 0, 0, 0, 1]	[0, 0, 1, 1, 1, 1, 0, 1]	[0, 0, 0, 1, 1, 0, 0, 1]	[0, 0, 1, 0, 1, 1, 0, 1]
Pred Cls	[1]	[0]	[0]	[0]	[0]	[0]	[0]
Padding Technique Loc	End Padding: [0, 0, 0, 1, ?, ?, ?, ?]						
Padding Technique Type	Universal data-agnostic			Universal data-aware			Learned
	Zero	One	Rand	IB	DB	MB	
Output	[0, 0, 0, 1, 0, 0, 0, 0]	[0, 0, 0, 1, 1, 1, 1, 1]	[0, 0, 0, 1, 1, 1, 0, 1]	[0, 0, 0, 1, 1, 0, 0, 0]	[0, 0, 0, 1, 1, 1, 1, 1]	[0, 0, 0, 1, 1, 1, 0, 0]	[0, 0, 0, 1, 1, 0, 1, 1]
Pred Cls	[2]	[1]	[0]	[0]	[1]	[0]	[1]

Figure 5: An example of applying E2-NVM's different padding strategies on an input data d1:[0,0,0,1] based on the memory pool defined in Table 1.

4.1.1 Padding Type: Universal data-agnostic padding. The simplest padding strategy—in terms of complexity and overhead—is universal data-agnostic padding, where the padding bits are generated given a simple fixed rule independent of data. Specifically, there are three types that we experiment with: (i) zero padding where all the padded bits are 0 bits, (ii) one padding where all the padded bits are 1 bits, and (iii) random padding where the padded bits are chosen randomly.

To illustrate how our padding strategy works, consider a storage system that is using a PCM as its persistent memory with a capacity of 12 equal sized memory segments, managed by a free-list, which we refer to as the dynamic-address-pool (Table 1). In this example E2-NVM groups these memory locations into 3 different clusters. Now, suppose that we have a new data object d1: [0,0,0,1] that we want to insert into the PCM using universal data-agnostic with beginning-padding location ([?,?,?,?,0,0,0,1]). Using one padding, 8-4=4 bits are added before the input data (in the question marks) to get [1,1,1,1,0,0,0,1]. Then, when it is sent to the model, cluster 2 is predicted to be the best cluster and one memory location is selected within this cluster (Figure 5). Note that the leftmost 4 bits are not written to the memory since they were added to the data to be able to utilize the deep learning model. Although this padding type is very simple, our DL model might not be utilized to its full potential since the padded bits added to the input data might not reflect the existing distribution in the memory content.

4.1.2 Padding Type: Universal data-aware padding. In this scheme, we aim to find a padding strategy that chooses the padding bits based on the content of the data objects in NVM or the content of the input items. The intuition behind data-aware padding strategies is that if the padding pattern matches patterns in data, the VAE can perform more efficient clustering. For the universal data-aware padding strategy, we propose three different padding schemes: (i) input-based padding (IB), (ii) dataset-based padding (DB), and (iii) memory-based padding (MB).

In input-based padding, the content of the padded part for an incoming data item is determined based on the distribution of ones and zeros in the input item. For instance, for the system we described in this section, if it receives d1:[0,0,0,1], the padded part will contain 1s and 0s with probability of 0.25 and 0.75, respectively, which is the same as the probability of 1's and 0's in d1. So, for the middle padding, the output format would be

[0,0,1,0,0,0,0,1], which results in cluster dataset-based padding uses the distribution of 1's and 0's in all the items it has received so far. For memory-based padding, we choose the probabilities based on the content of the existing memory locations on NVM that are going to be replaced by the new incoming items.

Although universal data-aware padding strategies generally result in better clustering decisions compared to data-agnostic padding, there are still some items that are directed to the wrong clusters. This is because the distribution of 0's and 1's might not reflect the best padding to be performed on the input item. The next strategy aims to overcome this challenge.

4.1.3 Padding Type: Learned padding. The main shortcoming of both data-aware and data-agnostic padding techniques is that they do not always generate a padding that is tailored specifically to both the input item as well as prior data, some of which that are used in training the DL model. We overcome this by designing a learning-based strategy, where we train a model that enables us to predict the best padding strategy for a given input item which takes into account all the existing data. The learned padding model takes an incoming data item of arbitrary size as input and generates the padding bits as output. The intuition behind this padding strategy is that the padding model is trying to predict the best padding strategy that will place the incoming item in the right cluster.

To this aim, we utilize an Long Short-Term Memory (LSTM) model to generate more meaningful padded data, so E2-NVM can predict similar memory locations with higher accuracy, which in turn improves the energy consumption of the system. Figure 6 shows the architecture of our proposed LSTM whose underlying algorithm was developed by Hochreiter and Schmidhuber in 1997 [21]. This figure shows that the model has a hidden state where it represents the state of the current timestamp, which is known as short term memory. In addition to that LSTM also has a cell state represented by C_{t-1} and C_t for previous and current timestamp respectively (known as long term memory).

As it is shown in Figure 6, in this paper, E2-NVM utilizes an LSTM with a sliding window strategy which takes as input 64 bits and predicts 8 bits in a single step. The window is slid by 8 bits after each prediction to generate the required number of padded bits. To illustrate the goal of our learned padding approach, consider the same storage system in Table 1. For the sake of simplicity, suppose that, in this example, our LSTM model takes input size of 7 bits and predicts 1 bit at a time. Now, let's further assume that our system receives [1,0,1,1,0,0,0],[0,1,1,1,0,0,1], [1,1,1,1,0,0,0], [1,0,0,0,1,0,1], [0,0,0,0,1,0,1], and [0,0,0,0,1,1,1]. Since the memory contents are 8 bits and E2-NVM works on input sizes of 8 bits, we feed them to our following simple LSTM model to make them 8 bits:

```
# define model
model = Sequential()
model.add(LSTM(10, input_shape=(1,7)))
model.add(Dense(1, activation='linear'))
# compile model
model.compile(loss='mse', optimizer='adam')
model.fit(X, y, epochs=20, shuffle=False, verbose=0)
# make predictions
yhat = model.predict(X, verbose=0)
```

As it is clear in Table 1, the best case scenario happens when their eighths bits are predicted as 0, 0, 0, 1, 1, and 1, which is congruent with the results of the LSTM model: [0.006], [0.024], [-0.027], [1.056], [0.869], and [1.038]. As a result, all the mentioned items are assigned to their correct clusters, which in turn improves system's energy efficiency.

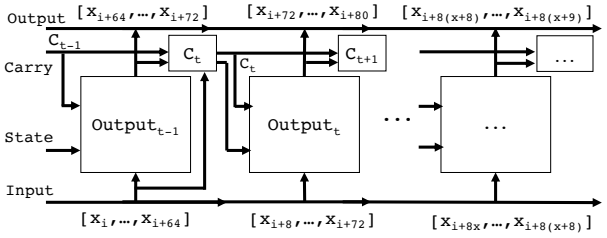


Figure 6: The anatomy of the LSTM model used in E2-NVM.

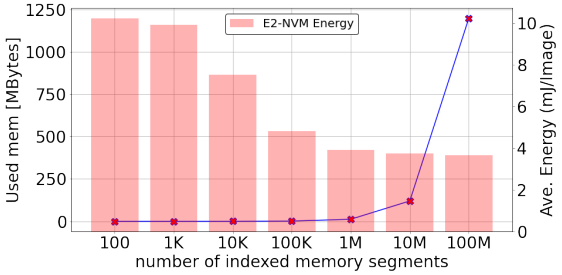


Figure 7: E2-NVM's memory and energy consumption for indexing different numbers of memory segments.

4.1.4 Additional design considerations. As we discussed before, E2-NVM is built on DRAM, and might need to index a large portion of NVM, which usually comes at much bigger sizes than DRAM. This means that if the memory segments that we use in the system are small (for example 1KB) and we want to index a NVM of size 1 TB, we will need to have E2-NVM index around 1 billion memory segments, which takes a lot of space in DRAM. To solve this problem, in E2-NVM, instead of indexing the whole NVM device at the beginning, a dynamic incremental approach can be adopted, which starts by indexing a portion of the memory, and as time progresses, more addresses that were not initially mapped can be added incrementally to DAP. Also, free memory locations are dynamically added back (recycled) to the free lists of DAP after DELETE operations.

Figure 7 illustrates the amount of memory that E2-NVM uses for indexing different number of memory segments for the PubMed data set [16]. The results show that although indexing a smaller number of memory segments takes less space, it also means that not only do we need to retrain the DL model more frequently, but we will also have fewer choices to find the most similar location for incoming writes, which results in increasing the number of bit flips and energy consumption. From the results presented in Figure 7, we observe that by having 100K to 1M memory segments, we can have the best of both worlds. While we do not see any tangible performance degradation in terms the energy consumption, indexing this number of memory segments will consume a couple of MBs in memory. Furthermore, by indexing more than 1M memory segments, we do not see any significant improvements in energy consumption.

To overcome the overhead incurred due to small key-value pairs, batching can be applied so that small writes are grouped together to form larger writes to memory segments. This way, E2-NVM needs to map the free memory locations based on the batch size rather than the key-value pair size, which leads to reducing E2-NVM footprint. It is worth noting that we do not make assumptions about word/byte-alignment. However, we want to note that padding bytes are not stored. So, it does not impact the storage of the data. Rather, padding is only performed

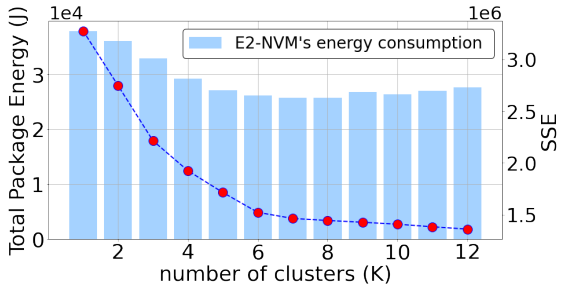
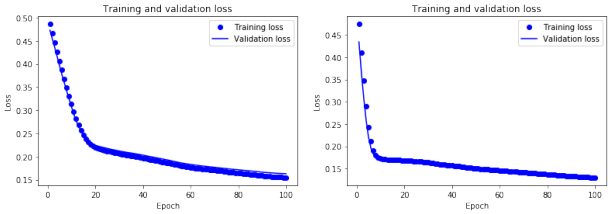


Figure 8: Sum of Square Error graph versus E2-NVM's energy consumption to find the optimal K.



(a) AMZ (b) CCTV

Figure 9: The training and validation loss for feature extraction during training for different datasets.

for the prediction part of E2-NVM. The reason for this is that we want to allow data with variable size to be applied to the model.

As a result, selecting very small memory segments (less than 1KB) when the memory pool size is very big (64 GB or more) is not a good design decision because although the NVM's write energy consumption is very low, the system suffers from scalability issues (dynamic address pool takes a big space in DRAM). Likewise, although selecting big memory segments takes up small space in DRAM, it is not energy efficient because we will have fewer choices to find the most similar locations for the incoming writes, which results in increasing the number of bit flips and energy consumption. Therefore, this test can help us find the most efficient memory segment size. This depends on (1) the size of the area in NVM that we want to index (memory pool size), (2) the size of DRAM that we use, and (3) the amount of energy consumption that the system can tolerate.

Another important design consideration is to guarantee that the dynamic address pool will never run out of free memory addresses and E2-NVM will always be able to serve the incoming requests. To this end, we set a minimum threshold to number of addresses in each cluster and will trigger the re-training process in the background when one of the clusters reaches to the threshold. After the new model is ready, we switch to the new model. It is worth noting that we do not need to train the model in E2-NVM as long as the performance is not affected substantially (please see Section 5.3 for more details).

Determining the Number of Clusters. A decision that needs to be made before E2-NVM starts training its DL model is to determine the number of clusters (K). There are a number of factors that need to be taken into consideration: (1) As the number of clusters increases, the memory contents inside clusters become more similar to each other, and it saves more bit flips, which means consuming less energy (see Section 5). (2) Although having more clusters saves more energy when writing on NVM, it also means that E2-NVM needs more time to finish training its DL model, which means increasing latency and energy consumption of the system.

Figure 8 shows the results of the test that we conducted to determine the optimal value for K. For energy-consumption, our experiments show a"valley" trend (the blue bar graph in the figure): the energy-consumption is relatively high at both the lowest and highest Ks, and is relatively low at intermediate Ks. This is because, in low Ks, the number of clusters is too small and the contents inside each cluster are not very similar to each other, which leads to high energy consumption in NVM. Conversely, increasing the number of clusters beyond a certain level increases the total energy consumption while providing only a limited return on bit flip reduction, also leading to energy inefficiency in DRAM and CPU. This "energy-valley" trend indicates that choosing the best value for K in an energy-aware fashion requires a good trade-off between the amount of energy that the system requires for writing on NVM and the energy that E2-NVM needs to train its DL model.

In this paper, we use the "elbow method" [26, 38], which is one of the most common techniques that is used to find the optimal value for K in K-means clustering. The elbow method is expressed as the following Sum of Squared Error (SSE) [50]:

$$SSE(X, \Pi) = \sum_{i=1}^{K} \sum_{x_j \in C_i} \|x_j - m_i\|_2^2 \quad (1)$$

where $\|.\|_2$ denotes the Euclidean (L2) norm, $m_i = \frac{1}{|C_i|} \sum_{x_j \in C_i} x_j$ is the centroid of cluster C_i where the cardinality is $|C_i|$, $\Pi = \{C_1, C_2, ..., C_K\}$, and $X = \{x_1, ..., x_i, ..., x_N\}$ (N is the feature vector).

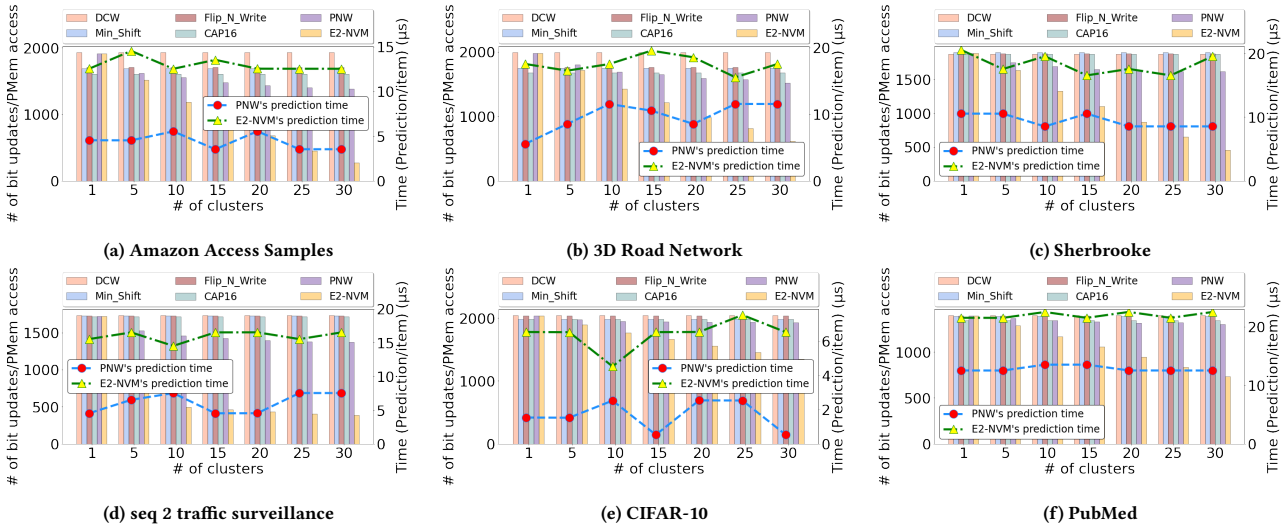
To determine the optimal number of clusters, we identify a sharp decrease known as the "elbow" or "knee", which suggests the optimal value for K [26, 38, 50]. Figure. 8 shows an example of choosing the optimal K by seeing the significant decrease in the SSE graph, which is in K = 6 (the data set is CIFAR-10). As we can see in this figure, this value is also a good estimation for the energy consumption of E2-NVM for different values of K.

5 EXPERIMENTS

5.1 Methodology

The experiments are executed on (1) an Intel Core i7 processor running at 4.7 GHz with 4 cores, each of which has 1MB L2 Cache and 12MB L3 Cache using 32GB of Intel® Optane™ Memory Series 3D Xpoint™ and 16 GB of DRAM, and (2) an Intel Xeon® @2.6GHz with 16 cores, an Nvidia Tesla K20m GPU with 5GB memory using 32 GB DDR4 main memory and a 256 GB SSD hard drive. The machine has 32GB DDR3 main memory, 128GB of Intel® Optane™ Persistent Memory 200 Series (PMEM Module), and a 256 GB SSD hard drive. We use the latter machine to get the results in Figures 7, 11, 16 and 18. Although the amount of energy might be different for different setups, both machines showed similar behavior in terms of the relationship between the number of flipped bits and energy consumption. We utilize thread-safe methods in E2-NVM. This is the case for the data structures that we utilize to maintain address pools and mapping (contention in other parts such as the VAE would not lead to concurrency anomalies since operations on them are read-only).

There are two methods to measure energy consumption: (1) Power Monitors, which use hardware tools to measure the actual power of the device. Despite being very precise, they are extremely difficult to set up. (2) Energy Profilers, which are vastly used by researchers, do not require any special hardware, or power sensors, and estimate the power cost of different hardware using estimation models [12]. In this paper, we use an energy profiler named Perf, which is a performance analysis tool and



(a) Amazon Access Samples (b) 3D Road Network (c) Sherbrooke

(d) seq 2 traffic surveillance (e) CIFAR-10 (f) PubMed

Figure 10: The average number of actual bit updates per PMem's cache line access granularity as well as the latency of prediction per item in E2-NVM for the real-world textual and multimedia datasets.

a part of the Intel's RAPL interface [18, 32]. we measured the energy and power consumption of the memory (both DRAM and PMEM), while running our tests using the perf [13] tool:

```
$ perf stat -a -r 5 -e power/energy-cores/,
↪  power/energy-ram/, power/energy-gpu/,
↪  power/energy-pkg/, power/energy-psys/ ./test
```

This tool provides the collection of energy measurements from various components of a computer system such as: cores, Intel's GPUs, package (all the core and un-core components), DRAM, total power consumption of a node, and so on. Also, the sampling rate in our tests is 1000 samples per second.

5.2 Overview and setup

In this section, we evaluate our proposed method in terms of bit flips, energy efficiency, and performance. We perform experiments on a real Intel Optane memory device to measure energy efficiency and performance overhead. Also, we perform experiments with emulated Optane memory to measure bit flip reduction (which cannot be measured using the real device.)

We compare E2-NVM with two main groups of solutions: 1) persistent K/V stores that use specialized data structures to deal with the limitations of NVMs [25, 26, 36, 45, 54]. These methods generally focus on reducing write amplification. 2) hardware-based bit flip optimization methods that use the RBW technique to alleviate the limitations of NVMs [10, 23, 37, 52]. Unlike the previous category, this group focuses directly on decreasing the number of bit flips.

5.2.1 Workloads. We have used various types of real-world and synthetic workloads in our evaluations.

Synthetic workloads. In the first synthetic workload, we run the YCSB benchmark [11] to evaluate E2-NVM. We load a 10-GB data set into the database as the "old data" in the load phase. Then, we run the workloads one by one, and compare the results. The six core workloads that we used in our tests have different read-write ratios and access patterns: Workload-A has 50% reads and 50% updates, Workload-B has 95% reads and 5% updates, and Workload-C has 100% reads; the keys are chosen from a Zipfian distribution, and the updates operate on already-existing keys. Workload-D involves 95% reads and 5% inserting new keys (temporally weighted distribution). Workload-E involves 95% range queries and 5% inserting new keys (Zipfian distribution), while Workload-F has 50% read-modify-writes and 50% reads.

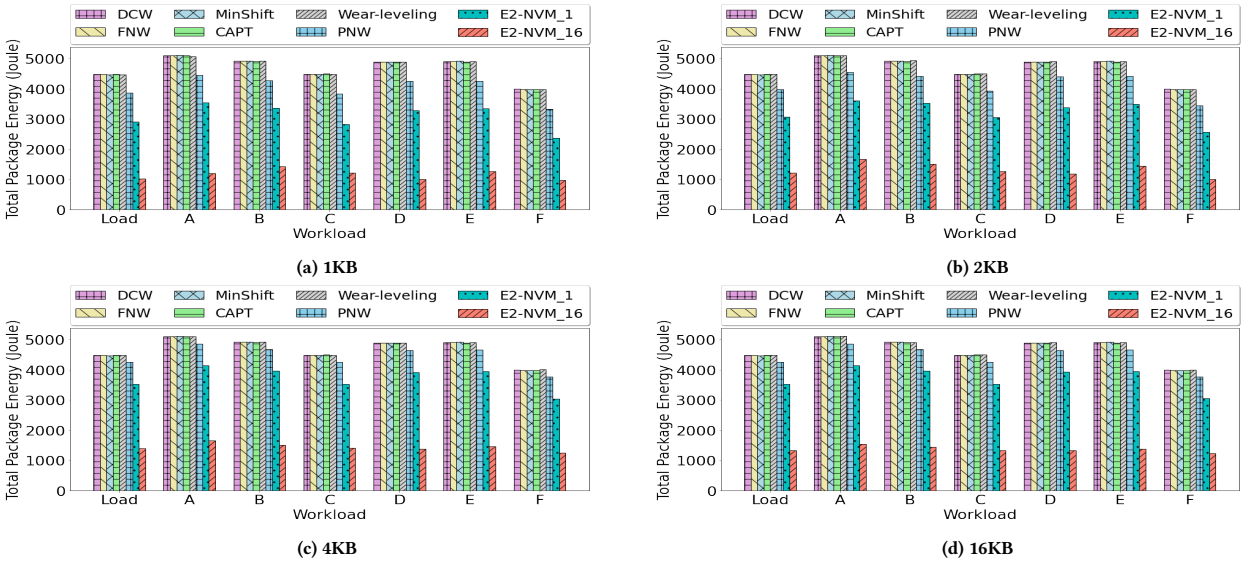
Real-world workloads (Numerical). We use Amazon Access Samples [16] that contain 30K access log entries. We also use the 3D Road Network Data Set [19, 31] that contains 434874 entries of road networks information of North Jutland, Denmark. Finally, we use the collections of the DocWord database named "PubMed", which consists of 730 million entries [16].

Real-world workloads (Images). We use two of the most widely used datasets for machine learning and computer vision research, MNIST and CIFAR-10 datasets [33]. The former is a dataset of 60,000 28x28 grayscale images of the 10 digits, along with a test set of 10,000 images. The latter dataset is a subset of the 80 million tiny images dataset and consists of 60,000 32x32 color images, which are grouped into 10 different classes.

Real-world workloads (Videos). In the last set of tests, we use two video datasets: 1) The Sherbrooke video dataset [24], which is more than two-minute-long video (with resolution 800x600), which was filmed at the Sherbrooke/Amherst intersection in Montreal by a camera located a couple of meters above the ground, and 2) Traffic Surveillance video [5], which is collected from seven intersections in the Danish cities of Aalborg and Viborg. In this test, we use two sequences of RGB cameras called CCTV1 and CCTV2. For the first dataset, we stored the first 30 seconds of this video as the old data and then we replaced it with the rest of the video as the new data. We did the same with the second datasets with one difference and that is storing first one minute of the video as the old data and using the rest of the video as the new data.

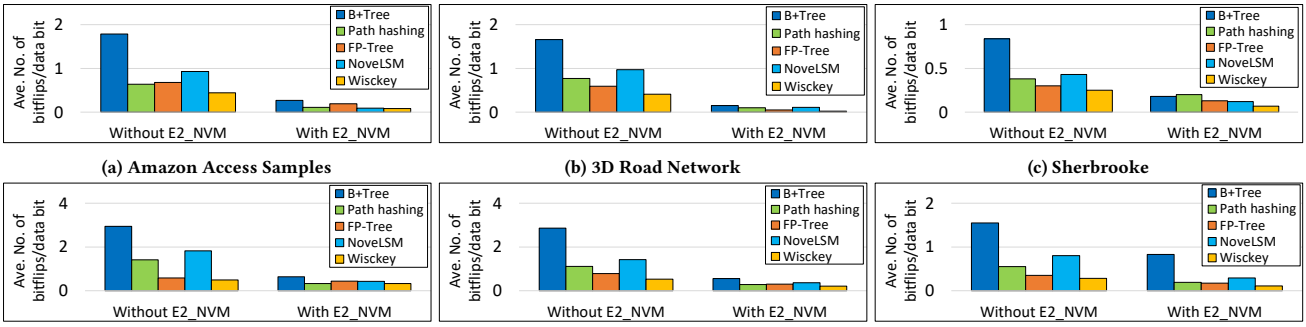
5.3 Evaluation Results

Deep learning model characteristics. In the first set of experiments, Figure 9 shows the E2-NVM's learning curves, which is a metric to show how well the DL model is "generalizing" the learned patterns. We have evaluated our DL model on the training dataset and on a hold-out validation dataset, which is completely isolated from the training dataset. In this figure, we have used the loss metric whereby smaller values indicate better learning and a value of 0 indicates that the training dataset was learned perfectly, and no mistakes were made. As we can see from the results, our deep learning model converges very quickly,



(a) 1KB (b) 2KB (c) 4KB (d) 16KB

Figure 11: The average amount of energy consumed per PMem's cache line access granularity when memory segment size changes for YCSB workloads.



(a) Amazon Access Samples (b) 3D Road Network (c) Sherbrooke

(d) Log-normal data distribution (e) Zipf data distribution (f) PubMed

Figure 12: The impact of augmenting E2-NVM to data stores in terms of the average number of bit updates per writing 1 data bit.

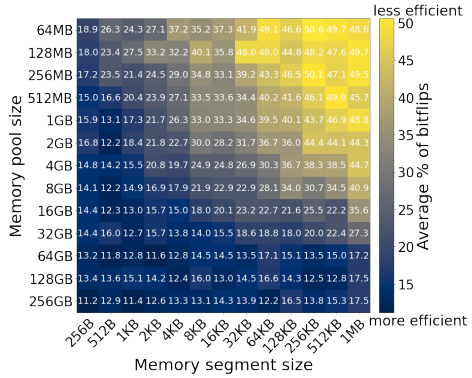
which shows the ability of E2-NVM to learn and generalize the existing patterns in memory segments.

Comparison with RBW and memory-aware methods. Figure 10 shows the results for comparing E2-NVM with the following RBW methods: DCW [52], MinShift [37], FNW (Flip 'n Write) [10], and Captopril [23]. We also compare with a clustering-based memory-aware solution, PNW [26]. We vary the number of clusters, k, ranging from k=1 to k=30 for different datasets (note that k=1 is a baseline that aims to show the performance without getting the benefits of the clustering-based methods. Also, the only methods that are impacted by increasing the number of clusters are the clustering-based methods, PNW and E2-NVM.). We have compared the performance of E2-NVM to others in terms of the number of bits updated/written per PMem access. In this figure, when we pick k=1, the result for E2-NVM, PNW, and DCW are the same since they all behave similarly with no clustering. Figure 10 shows that our method enhances the previous ML-based method (PNW [26]) by up to 3.2x and the baseline bit flip optimized methods by up to 4.23x.

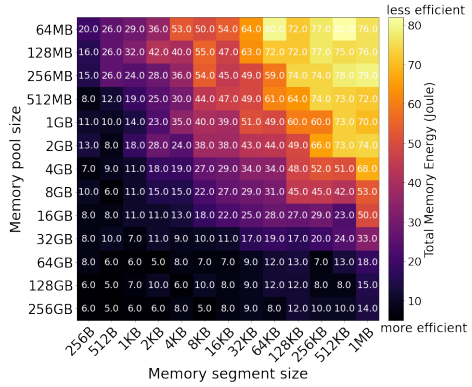
We have also calculated the model prediction latency for both PNW and E2-NVM. In our method, the delay is higher compared to PNW because E2-NVM performs two predictions, one by the DL model and the other by the clustering ML model. This highlights a performance-accuracy trade-off (where performance is the prediction overhead and accuracy is the accuracy of finding a similar memory segment) where E2-NVM favors accuracy but introduces more overhead compared to PNW.

In the next experiment, we calculate the average amount of energy that is consumed per PMem's cache line access granularity when the memory segment size changes for YCSB workloads on a real Optane memory device. Figure 11 shows that by choosing smaller segment size, our method can save more energy since the model can predict memory segments with higher accuracy, which leads to minimizing the number of bit flips. This figure also shows another factor that affects the energy consumption of our method—the number of clusters. When there are more clusters, the similarity among the items within a cluster increases, which leads to fewer number of bit flips and less energy consumption.

Augmenting E2-NVM to existing NVM data structures. In the next experiment, we tested the performance of the implemented methods, such as B+-Tree [9], Wisckey [35], Path Hashing [54], FP-Tree [45], and NoveLSM [25]), in terms of the number of updated bits in two different ways: before plugging to E2-NVM, and after plugging to E2-NVM. The results are shown in Figure 12. When not plugged to E2-NVM, B+-Tree has the worst performance because, in a regular B+-Tree [9], the items in leaf nodes need to be sorted, which increases the number of movements and bit flips. After we tested the performance of the



(a) Average updated bits ratio



(b) Total memory energy

Figure 13: E2-NVM's performance in terms of the average updated bits ratio and total memory energy efficiency for different memory segment and memory pool sizes.

methods in terms of bit flips, we plugged them to E2-NVM and repeated the same tests. After plugging each method to E2-NVM, their performance improves by up to 91% by preventing a lot of unnecessary bits from being flipped. These tests validate the ability to plug existing data indexing structures to E2-NVM to reduce bit flipping.

The effect of memory pool size and memory segment size on the total memory energy. Figure 13 illustrates the overall performance of E2-NVM in terms of energy efficiency and updated bits for different memory segment and pool size choices for the mixture of all the real workloads in this paper. For the overall energy efficiency shown in Figure 13b, we observe that the E2-NVM's power consumption increases as the ratio of the memory segment size to the memory pool size increases, which also aligns with results of the average bit flips ratio shown in Figure 13a. Based on this observation, we conclude that the smaller we choose the size of the memory segments compared to the size of the data zone (pool size), the more tree nodes are available for E2-NVM, which also results in fewer bit flips and less energy consumption.

Padding evaluations. In multimedia applications, such as CCTV and video storage, the size of data objects is typically fixed depending on the chosen resolution. This enables us to use E2-NVM directly as we can fix the input size of the model to match the data object size. However, to generalize our solution to handle data items of arbitrary sizes using the same model, we introduced padding methods. Figure 14 shows the performance of our proposed padding strategies. In this experiment, there are three padding positions. In the first one, the original data is at the beginning and the padding part is concatenated to the rightmost side of the data (padding at the end). In the second position, the data stays in the middle of the frame and padded bits are split into equal parts and appended to both sides of the data (padding in the edges). Finally, in the third position, the data is placed at the rightmost side of the frame and the padding part is attached to the left side of the data (padding in the beginning).

For each test, first, the DL model is trained on the training dataset (80% of the complete dataset). To generate the test set, we crop one-third of the data items in the positions that we mentioned and then pad it. So, for each dataset, we test 7 padding strategies and across 3 different padding positions.

In Figure 14, we make several observations about padding strategies, i.e., input-based (IB), dataset-based (DB), memory-based (MB), learned-based(LB), zero (0), and one (1) paddings. First, data-aware padding schemes outperforms the data-agnostic schemes in terms of reducing the number of bit flips. Second, learned padding scheme has the highest performance in terms of the average number of bit flips. Third, padding in the edges has higher variance compared to padding at the beginning or end. To summarize, the performance of padding improves with increasing complexity of models, from data-agnostic schemes to data-aware schemes and the learned padding.

The accuracy of mapping a data item to the most appropriate cluster drops when a large fraction of bits are padded bits. In Figure 15, we evaluate the impact of padding by measuring the number of bit flips under different percentages of data frames are being padded. In this experiment, we first trained the DL model on the CCTV's training dataset. Then, we cut off different percentages of the frames of the testing dataset and feed them to E2-NVM. Since the size of the input is smaller than what the model has been trained on, E2-NVM uses the learned padding strategy to fit the data items into the original frame sizes. In other words, in this experiment, we generate the missing parts of the frames in the testing dataset, so we can compare it with the baseline where the frames of the testing dataset were intact (the 0% padding line in the figure).

Figure 15 shows that when there is no padding (0% padding), the number of bit flips is minimum, the best performance. The reason is that when the input data frames are the same size as the training data frames, E2-NVM can find the best cluster for the incoming data items and minimize the number of bit flips. For padded data, we measure the number of bit flips per word based on the written bits only (padded bits are not written to NVM.) When the percentage of padding increases, it becomes more difficult for the system to predict the best padded part and find the most similar clusters. However, when the percentage of padding is low (10%), there is minimal loss in performance. This motivates combining the padding strategy with batching—as described in the padding section—to reduce the percentage of padding.

Memory overhead analysis of E2-NVM in terms of energy consumption. To see how training and re-training of our deep learning model affect the energy consumption of the system, we conducted an experiment to track the behavior of our method as time goes by. In this test, we first created a transactional object store with a total pool size of 8GB and element/segment size of 64KB in the Intel Optane DC PMM (in the App Direct Mode) and seeded it with data items from ImageNet [14], which contains over 14 million labeled images. Also, we re-sized the images to fit the size of the elements (64KB) in the pool. Figure 16 shows the result in terms of the amount of total package energy that is consumed after a specific time. In this test, we did the following steps, which are marked on Figure 16 from 1 to 4: (1) We trained our DL model on the memory contents of the pool before we

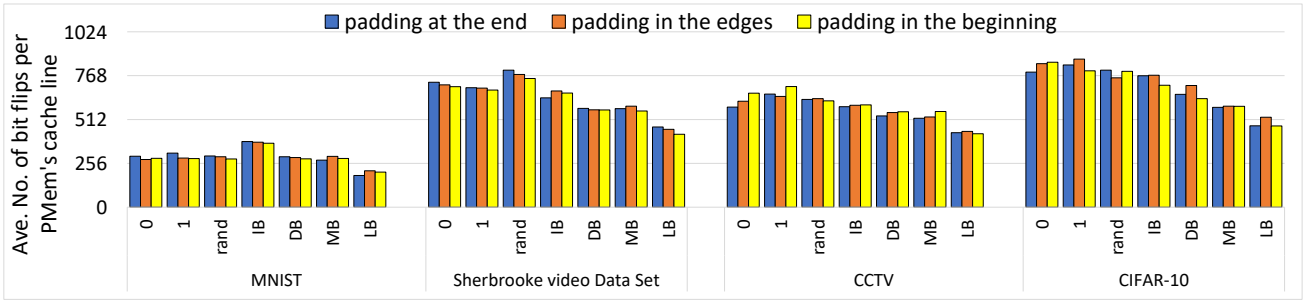


Figure 14: The average number of bit flips per word after applying different padding strategies.

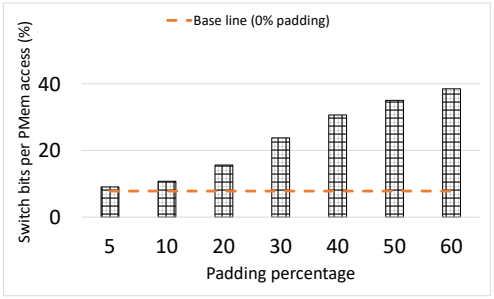


Figure 15: The number of bit flips when different percentages of the video frame size is padded by the learned padding scheme for the CCTV dataset.

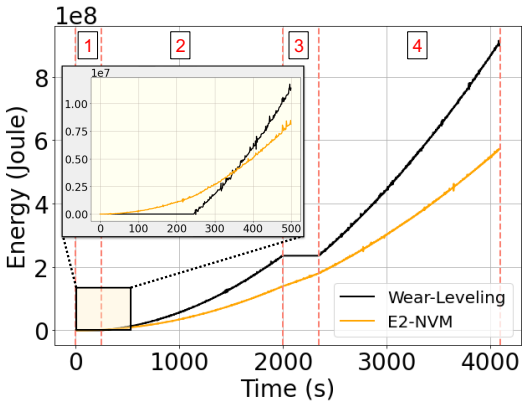


Figure 16: Tracking the package energy sampled every 1ms for E2-NVM when it goes through periodic training, re-training and writing phases compared to the wear-leveling technique over time.

start accepting the write requests (stage 1 in the figure). (2) After our DL model is trained, we started writing new data items on the existing memory contents of the pool from the ImageNet data set. In this step, we overwrote the pool with the items from the same data set (combination of both existing and new images) five times. (3) In step 3, we retrained the model to reflect the changes in the data distribution. It is worth noting that, for the sake of this experiment, we stopped writing on NVM until the retraining process finishes although, in E2-NVM, the writing process does not have to be stopped because the retraining is done in the background lazily. (4) In the last step, we resumed writing new data on NVM. We overwrote the pool with ImageNet data for four more times. As we saw this test, the total energy that E2-NVM saves up by writing similar contents will make up its energy overhead caused by training, retraining and prediction.

Another important observation that we can make from this test is that we can make a precise estimation of the cost of the re-training process in terms of energy consumption and latency by looking at the the cost of the training process of the same model at the initialization phase (stage 1 in Figure 16). The reason behind is that the training/re-training cost depends on the size of the data set (memory pool), dimensionality (memory segment), the complexity of the deep learning model that we use, and the hardware that we run our model on, which all are the same for both training and re-training processes.

Adaptability of E2-NVM to the dynamic changes. To analyze the behavior of E2-NVM in different scenarios and its adaptability to the system's changes, we conduct the last experiment in a dynamic environment when the content of the memory and the incoming workload changes over the course of time. You can see the results in Figure. 17. In this test, we use three image data sets from Tensorflow, i.e., MNIST, Fashion-MNIST, and CIFAR-10. For this test, we did the following scenarios:

Scenario 1: we seeded the data zone in NVM with a completely random content and then trained the E2-NVM model on this content. After training the model and creating the cluster-to-memory dynamic address pool, we started streaming 54K images (Figure. 17 part I) from MNIST as the new data into the system to overwrite the old data followed by deleting half of the items to make the system dynamic. As we see from the results (Fig 17 part I-a), because the content that the model is trained on and the incoming writes are different, the number of bit flips fluctuates a lot, but as time goes by, fluctuations become narrower toward the bottom (Figure. 17 part I-b) due to the fact that E2-NVM updates the cluster contents by recycling the deleted items and bringing them back to the cycle.

Scenario 2: we trained the model one more time with the current content and updated the dynamic address pool. Then, we continued streaming 27K images from the same data set (MNIST) as the new data into the system to overwrite the old data. Figure 17 shows that fluctuations and the average number of bit flips decrease. Even at the end of this stage, where the old data is almost completely replaced with the new one, we still do not see significant changes in performance (Fig 17 part II).

Scenario 3: Starting from this point, we send a mixture of 27K items from two different data sets, i.e., Fashion-MNIST and MNIST, at the ratio of 1 to 2. Figure. 17 part III shows that the performance is affected immediately (the number of updated bits increases) since two-third of the incoming data are entirely from the content that the model has never seen before and results in a larger hamming distance.

Scenario 4 (part IV in the figure): we sent 30K images from the third data set, i.e., CIFAR-10. The number of updated bits

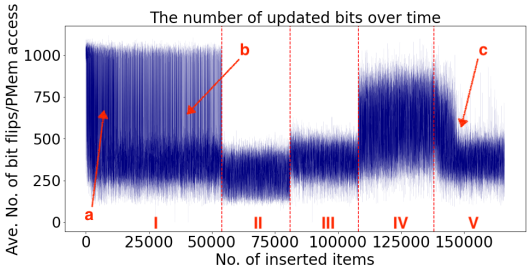


Figure 17: Tracking the performance of E2-NVM by changing the memory content and incoming writes over the course of time.

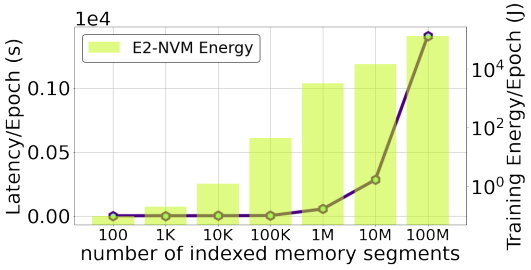


Figure 18: E2-NVM's training costs in terms of latency and energy consumption per epoch for indexing different number of memory segments.

fluctuated more since (1) the old data contains the items from completely different data sets and (2) the model has never seen (been trained) the incoming data.

Scenario 5: In this phase, we continued sending 28K images from CIFAR-10 with one difference: we re-trained our model on the existing content. Figure. 17 part V-c shows that the results improve very fast since the data set and the content that the model is trained on are from the same type. As a result, we have seen that, depending on the application and the workload, we do not always have to re-train the model rapidly, and we can use the same model for a certain amount of time before it needs to be re-trained. This allows us to do the re-training in the background lazily and update the model periodically while the current model is serving the requests. To this aim, E2-NVM needs to know when to start re-training the model before the old one becomes inefficient, i.e. the system's performance decreases in terms of energy consumption. This is of great importance because we might not want to give all the available resources to the model since the system needs to serve the requests without any problem while the new model is being re-trained. We performed additional tests to evaluate the costs for re-training a new model (Figure 18). This experiment is performed on ImageNet [14].

In Figure 18, we measured the re-training time of the model per epoch for a different number of memory segments. We run these tests on the second hardware setup. Based on the results, the model needs more time and energy to be re-trained as the number of memory segments increases. This gives us an idea of setting the load factor so that we have enough time to finish re-training the new model before the old model becomes inefficient. Training the model sooner than this threshold would not result in a noticeable performance improvement since the pattern of the bits in the content is almost the same as the training time. Also, by waiting too long before re-training the model, the system misses the opportunity to improve the performance considerably.

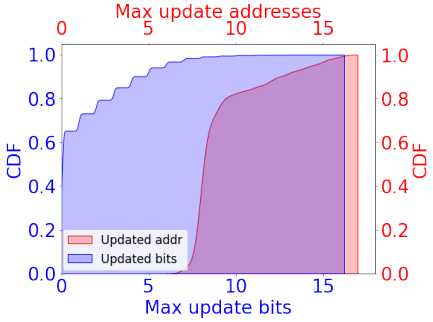


Figure 19: The maximum update addresses and wear-leveling as CDFs by applying E2-NVM with k=30 clusters.

Although decreasing the number of writes is important, wear-leveling is equally important to extend the lifetime of PCM. The reason is that some blocks of the memory device may receive a much higher number of writes than the other blocks, and as a result, wear out sooner[26]. Therefore, to observe the performance of E2-NVM in terms of the distribution of the maximum number of bit flips and the wear-leveling of PCM, we conduct two more tests. In these tests, we run E2-NVM when k = 30 clusters, on the mixture of MNIST and Fashion-MNIST data sets. For this test, we first warm up the data zone with 28K items from the combination of both data sets. Then, we stream 112K writes from the same data sets to the system. During the test, we also perform delete actions to make space for incoming writes (each word in the data zone is updated 4 times on average).

Figure 19 shows the maximum number of times the addresses in the data zone are written and the wear-leveling of memory bits as a cumulative distribution function (CDF). This figure illustrates two results: (1) the estimation of the likelihood to observe an address in the data zone of PCM that is written less than or equal to a specific number of times, and (2) the estimation of the likelihood to observe a memory bit in the data zone of PCM that is written less than or equal to a specific number of times. For example, as we can see in Figure. 19, the estimated likelihood to observe an address in the PCM data zone to be written less than or equal to 10 (P ($X \leq 10$)) is 81% (red color). Similarly, we observe that the estimated likelihood of a memory bit being written less than or equal to 5 times is 85%. This likelihood rises to 98% when a memory bit being written equals to 7 times (blue color). This important observation shows that (1) E2-NVM distributes write activities across the whole PCM chip, and (2) E2-NVM distributes bit flips evenly across the whole data zone of the PCM chip, and as a result, the lifetime of PCM is extended more.

6 CONCLUSION

In this paper, we have explored the use of software-level approaches to improve energy efficiency and write endurance of NVMs. Specifically, a deep learning model is used to map memory locations based on the hamming distance of their content. This mapping is used when new writes arrive to assign them to memory location with similar content. This reduces the number of bit flips, which leads to better write endurance and energy efficiency.

7 ACKNOWLEDGMENTS

We sincerely thank the anonymous reviewers for their insightful comments and feedback. This research is supported in part by the NSF under grant CNS-1815212.

REFERENCES

[1] A. Alameldeen and D. Wood. Frequent pattern compression: A significance-based compression scheme for l2 caches. Technical report, University of Wisconsin-Madison Department of Computer Sciences, 2004.

[2] J. Altosaar. *Tutorial - What is a Variational Autoencoder?*, Aug. 2016.

[3] M. Andalibi, M. Hajihosseini, S. Teymoori, M. Kargar, and M. Gheisarnejad. A time-varying deep reinforcement model predictive control for dc power converter systems. In *2021 IEEE 12th International Symposium on Power Electronics for Distributed Generation Systems (PEDG)*, pages 1–6. IEEE, 2021.

[4] J. Arulraj et al. Let's talk about storage & recovery methods for non-volatile memory database systems. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 707–722, 2015.

[5] C. H. Bahnsen and T. B. Moeslund. Rain removal in traffic surveillance: Does it matter? *IEEE Transactions on Intelligent Transportation Systems*, pages 1–18, 2018.

[6] D. Bittman et al. Optimizing systems for byte-addressable {NVM} by reducing bit flipping. In *17th {USENIX} Conference on File and Storage Technologies ({FAST} 19)*, pages 17–30, 2019.

[7] M. Capra, R. Peloso, G. Masera, M. Ruo Roch, and M. Martina. Edge computing: A survey on the hardware requirements in the internet of things world. *Future Internet*, 11(4):100, 2019.

[8] Y. Che, Y. Yang, A. Awad, and R. Wang. A lightweight memory access pattern obfuscation framework for nvm. *IEEE Computer Architecture Letters*, 19(2):163–166, 2020.

[9] S. Chen and Q. Jin. Persistent b+-trees in non-volatile main memory. *Proceedings of the VLDB Endowment*, 8(7):786–797, 2015.

[10] S. Cho and H. Lee. Flip-n-write: A simple deterministic technique to improve pram write performance, energy and endurance. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 347–357, 2009.

[11] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears. Benchmarking cloud serving systems with ycsb. In *Proceedings of the 1st ACM symposium on Cloud computing*, pages 143–154, 2010.

[12] L. Cruz. Tools to measure software energy consumption from your computer. http://luiscruz.github.io/2021/07/20/measuring-energy.html, 2021. Blog post.

[13] A. C. De Melo. The new linux'perf'tools. In *Slides from Linux Kongress*, volume 18, pages 1–42, 2010.

[14] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[15] D. B. Dgien, P. M. Palangappa, N. A. Hunter, J. Li, and K. Mohanram. Compression architecture for bit-write reduction in non-volatile memory technologies. In *2014 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*, pages 51–56. IEEE, 2014.

[16] D. Dua and C. Graff. UCI machine learning repository, 2017.

[17] B. Gu, S. Kargar, and F. Nawab. Efficient dynamic clustering: Capturing patterns fromhistorical cluster evolution. *arXiv preprint arXiv:2203.00812*, 2022.

[18] P. Guide. Intel® 64 and ia-32 architectures software developer's manual. *Volume 3B: System programming Guide, Part*, 2(11), 2011.

[19] C. Guo, Y. Ma, B. Yang, C. S. Jensen, and M. Kaul. Ecomark: evaluating models of vehicular environmental impact. In *Proceedings of the 20th International Conference on Advances in Geographic Information Systems*, pages 269–278, 2012.

[20] X. Guo, L. Gao, X. Liu, and J. Yin. Improved deep embedded clustering with local structure preservation. In *IJCAI*, pages 1753–1759, 2017.

[21] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[22] J. Huang, Y. Hua, P. Zuo, W. Zhou, and F. Huang. An efficient wear-level architecture using self-adaptive wear leveling. In *49th International Conference on Parallel Processing-ICPP*, pages 1–11, 2020.

[23] M. Jalili and H. Sarbazi-Azad. Captopril: Reducing the pressure of bit flips on hot locations in non-volatile main memories. In *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1116–1119. IEEE, 2016.

[24] J.-P. Jodoin, G.-A. Bilodeau, and N. Saunier. Urban tracker: Multiple object tracking in urban mixed traffic. In *IEEE Winter Conference on Applications of Computer Vision*, pages 885–892. IEEE, 2014.

[25] S. Kannan et al. Redesigning lsms for nonvolatile memory with novelsm. In *2018 {USENIX} Annual Technical Conference ({USENIX}{ATC} 18)*, pages 993–1005, 2018.

[26] S. Kargar, H. Litz, and F. Nawab. Predict and write: Using k-means clustering to extend the lifetime of nvm storage. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, pages 768–779. IEEE, 2021.

[27] S. Kargar and F. Nawab. Extending the lifetime of nvm: challenges and opportunities. *Proceedings of the VLDB Endowment*, 14(12):3194–3197, 2021.

[28] S. Kargar and F. Nawab. Hamming tree: The case for memory-aware bit flipping reduction for nvm indexing. In *CIDR*, 2021.

[29] S. Kargar and F. Nawab. Challenges and future directions for energy, latency, and lifetime improvements in nvms. *Distributed and Parallel Databases*, pages 1–27, 2022.

[30] S. Kargar and F. Nawab. Hamming tree: The case for energy-aware indexing for nvms. In *SIGMOD*, 2023.

[31] M. Kaul, B. Yang, and C. S. Jensen. Building accurate 3d spatial networks to enable next generation intelligent transportation systems. In *2013 IEEE 14th International Conference on Mobile Data Management*, volume 1, pages 137–146. IEEE, 2013.

[32] K. N. Khan, M. Hirki, T. Niemi, J. K. Nurminen, and Z. Ou. Rapl in action: Experiences in using rapl for power measurements. *ACM Transactions on Modeling and Performance Evaluation of Computing Systems (TOMPECS)*, 3(2):1–26, 2018.

[33] A. Krizhevsky, G. Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[34] W. Li et al. Hilsm: an lsm-based key-value store for hybrid nvm-ssd storage systems. In *Proceedings of the 17th ACM International Conference on Computing Frontiers*, pages 208–216, 2020.

[35] L. Lu, T. S. Pillai, H. Gopalakrishnan, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau. Wisckey: Separating keys from values in ssd-conscious storage. *ACM Transactions on Storage (TOS)*, 13(1):1–28, 2017.

[36] C. Luo and M. J. Carey. Lsm-based storage techniques: a survey. *The VLDB Journal*, 29(1):393–418, 2020.

[37] X. Luo, D. Liu, K. Zhong, D. Zhang, Y. Lin, J. Dai, and W. Liu. Enhancing lifetime of nvm-based main memory with bit shifting and flipping. In *2014 IEEE 20th International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 1–7. IEEE, 2014.

[38] T. S. Madhulatha. An overview on clustering methods. *arXiv preprint arXiv:1205.1117*, 2012.

[39] E. Min, X. Guo, Q. Liu, G. Zhang, J. Cui, and J. Long. A survey of clustering with deep learning: From the perspective of network architecture. *IEEE Access*, 6:39501–39514, 2018.

[40] S. Mittal and J. S. Vetter. A survey of software techniques for using non-volatile memories for storage and main memory systems. *IEEE Transactions on Parallel and Distributed Systems*, 27(5):1537–1550, 2015.

[41] K. Montanez. Amazon Access Samples. UCI Machine Learning Repository, 2011.

[42] F. Nawab, D. Chakrabarti, T. Kelly, and C. Morrey. Zero-overhead nvm crash resilience. In *Non-Volatile Memories Workshop*, 2015.

[43] F. Nawab, D. R. Chakrabarti, T. Kelly, and C. B. Morrey III. Procrastination beats prevention: Timely sufficient persistence for efficient crash resilience. In *EDBT*, pages 689–694, 2015.

[44] F. Nawab et al. Dalí: A periodically persistent hash map. In *31st International Symposium on Distributed Computing (DISC 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

[45] I. Oukid et al. Fptree: A hybrid scm-dram persistent and concurrent b-tree for storage class memory. In *Proceedings of the 2016 International Conference on Management of Data*, pages 371–386, 2016.

[46] P. M. Palangappa and K. Mohanram. Flip-mirror-rotate: An architecture for bit-write reduction and wear leveling in non-volatile memories. In *Proceedings of the 25th edition on Great Lakes Symposium on VLSI*, pages 221–224, 2015.

[47] M. K. Qureshi, J. Karidis, M. Franceschini, V. Srinivasan, L. Lastras, and B. Abali. Enhancing lifetime and security of pcm-based main memory with start-gap wear leveling. In *2009 42nd Annual IEEE/ACM international symposium on microarchitecture (MICRO)*, pages 14–23. IEEE, 2009.

[48] S. Song, A. Das, O. Mutlu, and N. Kandasamy. Improving phase change memory performance with data content aware access. In *Proceedings of the 2020 ACM SIGPLAN International Symposium on Memory Management*, pages 30–47, 2020.

[49] M. Staimer. How the cxl interconnect will affect enterprise storage. https://www.techtarget.com/searchstorage/tip/How-the-CXL-interconnect-will-affect-enterprise-storage. Accessed: 2021-08-05.

[50] M. Syakur, B. Khotimah, E. Rochman, and B. D. Satoto. Integration k-means clustering method and elbow method for identification of the best customer profile cluster. In *IOP conference series: materials science and engineering*, volume 336, page 012017. IOP Publishing, 2018.

[51] Y. Wang, H. Yao, and S. Zhao. Auto-encoder based dimensionality reduction. *Neurocomputing*, 184:232–242, 2016.

[52] B.-D. Yang, J.-E. Lee, J.-S. Kim, J. Cho, S.-Y. Lee, and B.-G. Yu. A low power phase-change random access memory using a data-comparison write scheme. In *2007 IEEE International Symposium on Circuits and Systems*, pages 3014–3017. IEEE, 2007.

[53] Q. Zeng and J.-K. Peir. Content-aware non-volatile cache replacement. In *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 92–101. IEEE, 2017.

[54] P. Zuo and Y. Hua. A write-friendly hashing scheme for non-volatile memory systems. In *Proc. MSST*, 2017.