# Machine Learning Based Design Space Exploration for Hybrid Main-Memory Design

Satyabrata Sen
Computational Sciences and Engineering Division
Oak Ridge National Laboratory
Oak Ridge, TN, USA
sens@ornl.gov

Neena Imam
Computational Research and Development Programs
Oak Ridge National Laboratory
Oak Ridge, TN, USA
imamn@ornl.gov

## ABSTRACT

We develop a machine learning (ML) based design space exploration (DSE) method that builds predictive models for various responses of a hybrid main-memory system. To overcome the challenges associated with latency, capacity, and power of memory systems in future extreme-scale machines, the hybrid memory architectures are being considered in which novel non-volatile memory (NVM) systems augment the traditional DRAM. However, way before their actual design and implementation, these emerging hybrid memory systems need to be simulated and analyzed to fully understand their capabilities and limitations. As the conventional architectural-level memory simulators require significant amounts of computational costs and time, we propose to utilize ML techniques for developing various memory-response models that can instantly provide a predicted response corresponding to any new memory configuration. Specifically, in this work, we apply four supervised ML techniques to build regression models for memory latency, bandwidth, power, and total read/write responses. The training and validation data for the ML methods are generated using NVMain memory simulator for DRAM, NVM, and their hybrid combinations. We demonstrate the results of the ML based memory-DSE method in terms of the learning curve characteristics for hyperparameter tuning and the statistical error analyses of the designed predictive models.

## CCS CONCEPTS

• **Hardware → Memory and dense storage**; *Emerging architectures*; • **Computing methodologies → Supervised learning by regression**; *Neural networks*; *Support vector machines*; *Classification and regression trees.*

## KEYWORDS

Hybrid main memory, nonvolatile memory, design space exploration, machine learning, learning curves, predictive models

**ACM Reference Format:**
Satyabrata Sen and Neena Imam. 2019. Machine Learning Based Design Space Exploration for Hybrid Main-Memory Design. In *Proc. Intl Symp. on Memory Systems (MEMSYS '19), Sep. 30-Oct. 3, 2019, Washington, DC, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3357526.3357544


*MEMSYS '19, September 30-October 3, 2019, Washington, DC, USA*

## 1 INTRODUCTION

The evolution of high-performance computing (HPC) systems has always been driven exclusively by their processors' performance. However, on the path to Exascale computing, it has been realized over the past decade that the primary bottleneck of system performance is shifting from the processor to the memory performance. The memory-wall problem [51], which refers to the increasing gap between the frequencies of CPUs and the latencies of memory system, has been a dominant bottleneck to the memory designers over the last couple of decades. In addition to that, today's multicore HPC systems face the less-memory-capacity-per-core problem because the current memory systems cannot scale at the same rate as the number of cores. To alleviate this less capacity issue, if systems rely more on the hard-drive type storages then the memory latency problem further worsens. Also, the power-hungry nature of current memory system is a huge concern when the upcoming Exascale machines are being designed with a certain cap on the acceptable power budget [1, Sec. 5.2]. Therefore, combined with the age-old latency problem, the capacity and power issues has made the memory system one of the challenging systems to design for future HPC machines.

To overcome these challenges, several novel non-volatile memory (NVM) concepts have been proposed in recent years, which include new switching mechanisms (e.g., phase transition, ionic reactions, quantum mechanical phenomena, etc.) and materials (e.g., ferromagnetic metals, chalcogenides, carbon materials, etc.) [6], [7]. This has led to development of various NVM devices with different degrees of maturity. They include widely deployed NAND flash as well as newer emerging technologies, such as the phase change random-access-memory (PCRAM), metal-oxide resistive RAM (RRAM), spin torque transfer RAM (STTRAM), ferro-electric RAM (FeRAM), magnetic RAM (MRAM), and more exotic technologies (e.g., the memristor and carbon nanotube based memories) [28]. Although some of these memory prototypes are being considered as emerging technologies as they still require significant R&D activities, in general with the promise of high performance, good scalability, and new functionalities, NVM technologies are becoming strong contender for the future memory systems.

With the introduction of NVMs in HPC design, several new questions arise whose answers are not yet clear. For example, which layer is the best place in the memory hierarchy for NVMs, is it cache, main memory, or storage? What issues do we need to address when NVMs are used as main memory? What would be the data read,

write, and migration policies if NVMs are used with DRAMs in a hybrid main-memory architecture? To answer these questions, we need to fully understand the overall performance and limitations of NVMs way before they are designed and incorporated in the HPC systems. This entails to performing a rigorous design space exploration (DSE) of state-of-the-art memory systems such that designers can choose appropriate memory parameters/configurations for satisfying performance, power, reliability, and other metrics.

Traditionally, architectural-level simulators and emulators are used on representative benchmarks to explore the memory design space; examples of which include NVDIMMSim [44], NVMain [39, 40], Messier [2], Siena [37], Quartz [48], and HME [13]. Although these simulators/emulators provide valuable insights into the memory performance, this simulator-based memory-design approach is often inefficient due to the significant computational costs of current simulator infrastructure. The problem further intensifies for state-of-the-art memory systems whose increasing complexity, along with multifactored interactions among memory types, memory controllers, and application softwares, result into an exponential growth of the design space and consequently of the number of potential simulation points. On the other hand, the ever-decreasing time-to-market requirement expects quicker simulation results. Therefore, memory system designers often try to circumvent the challenge of long simulation times by constraining the design space (based on their experiences/intuitions) and by reducing the size of simulator inputs. However, constraining the design space introduces risks of design bias in the inferred conclusions and lacks generalizability of the results to the broader parameter space.

Therefore, in this work, we develop a machine-learning (ML) based memory-DSE method that builds prediction models for various memory system responses. Nowadays, ML algorithms are being successfully used in different research areas, such as robotics, image processing, finance, and others. For the HPC systems too, ML techniques have been used for the microarchitectural DSE [17, 21, 29, 32, 33], architecture/compiler co-design [14], multicore resource management [34], manycore system optimization [27], GPU DSE [25], and hybrid processor-cache DSE [18] problems. Related to the memory-system design problems, there have been a few ML-based work, such as the memory controller optimization [22], memory hierarchy DSE [3], memory prefetcher design [19], and hybrid-memory page scheduler design [11]. In this context, our work proposes and evaluates ML based DSE approach for the hybrid main-memory design problem. The key idea of our approach is to build predictive models of various memory parameters by utilizing a small set of simulated memory configurations in the training and validation phase. These predictive models in essence approximate the memory-simulator function that characterizes the relationship between the memory configuration parameters (e.g., number of channels, ranks, banks, controller frequency, timing parameters) and memory responses (e.g., latency, bandwidth, power, total read/write). Once a predictive model is fully trained, it can be used to obtain thousands of responses of new memory design configurations within a few seconds. Since simulations are only required in the training phase, the ML based memory-DSE approaches are significantly efficient than other traditional approaches when they can accurately predict the memory performances while employing a small labeled training set.

The organization of this paper is as follows. In Section 2, we begin our discussion with an overview of state-of-the-art NVM and hybrid memory systems and their relevance to the upcoming HPC machines. Then, in Section 3, we provide brief overviews of the supervised learning techniques that we use to build regression models for the memory responses. Specifically, in this work, we apply two fixed-structure supervised models, artificial neural network (ANN) and support vector machine (SVM), and two varying-structure supervised models, random forest (RF) and gradient boosting (GB). To train and validate these ML models, we generate training data set using NVMain memory simulator for both DRAM and NVM, and also for their hybrid designs. The input memory-trace files for NVMain tool are simulated using the Gem5 simulator for two benchmarks: (i) STREAM and (ii) high performance conjugate gradient (HPCG). The details of these benchmarks and simulators, and how we do use them are described in Section 4, along with the particulars of ML training and validation stages. The results of the ML design are demonstrated in Section 5 in terms of both the learning curve characteristics for hyperparameter tuning and the error analysis of the designed models. In general, we found that SVM and RF methods yield better prediction accuracies respectively in the category of fixed and varying structure models. Lastly, the concluding remarks and some possible future directions are presented in Section 6.

## 2 BACKGROUND AND MOTIVATION

Over the past few decades, the primary memory component of computer architectures has been the DRAM. DRAMs are ubiquitous, they are present in mobile hand-held devices as well as in the most powerful supercomputers. As the HPC community races towards the holy grail of sustained exa-flop operations, increasing attention is now focused on emerging memory technologies other than DRAM. Conventional computer memory technologies, such as DRAM, will not be able to meet the challenges of future extreme-scale systems. These challenges will be on several fronts that include energy efficiency, system reliability, and application performance. For energy efficiency, the target goal for Exascale operation is 20 MW of system-wide power consumption [1]. Memory components consume a large fraction of the total power budget of a computing system. DRAM main memory may contribute to as much as 30-50% of a compute node's total power consumption [45], [47]. The poor energy efficiency of DRAMs is due the fact that they consume power even when they are in static mode. Another challenge is the plateauing of DRAM device scaling, contributing to lower memory capacity per node for future architectures. DRAM components include capacitors which are difficult to fabricate as device sizes shrink. It is expected that improvements in DRAM power and performance will flatten over the next few years. On-node main memory size limitations have serious negative implications in terms of application performance and efficiency. If applications are forced to engage in more internode communications due to small on-node main memory capacity, application performance and system reliability/resilience are sure to degrade.

Therefore, it is not surprising that emerging memory technologies with different attributes are being explored as alternates to DRAM technologies. NVM devices are attractive candidates due to certain physical properties. One of the main benefits of a NVM

device is that it does not need energy in static mode to keep its memory cells refreshed. Unlike DRAM, NVMs retain data without the need for standby power. They also have smaller feature sizes and hence can provide higher memory density. Another advantage of NVM devices is their persistence. Since NVMs can retain charge for a long time, applications may store checkpoints and other databases locally on NVMs [36]. However, NVMs have some disadvantages as well. One of the main disadvantages is lack of endurance. NVM devices, such as NAND flash memory, has low endurance and can degrade over time with repeated write operations. Not only is flash memory's write endurance several orders of magnitude lower than conventional DRAM's, the write operations for NVMs also have high latency. To offset some of these limitations, researchers are developing software and optimization techniques to efficiently integrate NVMs in future memory systems [36].

The International Technology Roadmap for Semiconductors (ITRS) has been carefully monitoring the roadmap for NVM technologies. In addition to the well-known NAND flash, NVM devices include different emerging technologies, such as PCRAM, RRAM, STTRAM, and FeRAM [28]. These non-conventional emerging technologies combine the non-volatility of conventional disks and the performance of DRAMs. In particular, the NAND flash memories are already in wide deployment. In 2012, sales of flash memory exceeded DRAM sales [47]. The cost per bit of flash memory is also lower than DRAM. NAND flash memories are also being deployed on state-of-the-art supercomputers. U.S. Department of Energy's Summit Supercomputer (currently ranked 1 on the TOP500 list [46]) has 1600 GB of NVM in each of its 4608 nodes and these can be configured either as burst buffers or as extended memory [15]. Less mature NVM devices, such as PCRAM, STTRAM, RRAM, are being aggressively investigated. Available data suggests that these emerging NVM devices can provide better endurance than NAND flash memory.

As we gain experience with these alternative memory technologies, we need to develop innovative integration strategies to gain benefits for future extreme scale architectures and applications. NVMs can be integrated in multiple ways in memory hierarchy. NVMs can be added to augment on-node DRAM [26, 30, 50]. They can also be configured as global storage devices [8, 9], as file cache [31, 41], or as CPU cache [43]. Functional integration strategies of NVMs may include acting as I/O burst buffers. NVMs can also function as persistent data structures and provide in-situ visualization and post-processing capabilities [49]. Thus, the optimum integration of NVMs in a computer memory architecture is an active and open area of research and also application dependent. Also, the impacts of the asymmetric latency and energies for read and write operations (characteristics of NVMs) on applications and programming systems need to be studied carefully.

## 3 OVERVIEW OF MACHINE LEARNING TECHNIQUES

In this section, we provide a brief overview of the ML techniques that we use to design predictive models for a hybrid memory architecture. Essentially, ML algorithms try to extract information from the available data automatically, and then use that information to achieve certain goals, for example, to find inherent structure within the data, or to learn how a function/system works producing some specific output values corresponding to certain input data. Mathematically, let us define $\boldsymbol{x} = [x_1, x_2, \ldots, x_P]^T$ as the $P$-dimensional input data vector, which is also sometimes referred to as the features; and let $y$ be the output or response value, denoted as $y = f(\boldsymbol{x})$, where $f(\cdot)$ is some fixed but unknown function representing systematic information that $\boldsymbol{x}$ provides about $y$. In our hybrid-memory design context, $\boldsymbol{x}$ represents various memory configuration parameters and $y$ is the corresponding memory responses. In this work, we apply four supervised learning methods that utilize the available training data $\{\boldsymbol{x}, y\}$ to form the regression fitting functions $\hat{f}(\cdot)$ that approximately represents the true relationship $f(\cdot)$. In the following, we present brief overviews of these four supervised ML algorithms.

**Artificial neural network** (ANN) [also known as the multilayer perceptron (MLP)] is a standard technique to model the nonlinear relationship between $\boldsymbol{x}$ and $y$. MLPs use a feed-forward multilayered network of neurons, connected by weighted edges, to map the input data to the output values [23]. Representing one of the hidden layers, say the $l$th layer, as

$$\boldsymbol{a}_l = g(\boldsymbol{W}_l \boldsymbol{a}_{l-1} + \boldsymbol{b}_l)\,,$$

where $g(\cdot)$ is an activation function; $\boldsymbol{a}_l, \boldsymbol{W}_l$, and $\boldsymbol{b}_l$ are the activation values, weights, and bias terms of the $l$th layer respectively; the output of an $L$-layered MLP is modeled as
$f(\boldsymbol{x}; \boldsymbol{W}_1, \ldots, \boldsymbol{W}_L, \boldsymbol{b}_1, \ldots, \boldsymbol{b}_L) =$

$$g\left(\boldsymbol{W}_L\, g(\boldsymbol{W}_{L-1}\, g(\cdots g(\boldsymbol{W}_2\, g(\boldsymbol{W}_1\, \boldsymbol{x} + \boldsymbol{b}_1) + \boldsymbol{b}_2)\cdots) + \boldsymbol{b}_{L-1}) + \boldsymbol{b}_L\right)\,.$$

The training phase of the MLP consists of finding the weights $\{\boldsymbol{W}_1, \boldsymbol{W}_2, \ldots, \boldsymbol{W}_L\}$ and bias terms $\{\boldsymbol{b}_1, \boldsymbol{b}_2, \ldots, \boldsymbol{b}_L\}$ in order to minimize the residual sum of squares (RSS) as
$\{\hat{\boldsymbol{W}}_1, \ldots, \hat{\boldsymbol{W}}_L, \hat{\boldsymbol{b}}_1, \ldots, \hat{\boldsymbol{b}}_L\} =$

$$\arg \min_{\substack{\{\boldsymbol{W}_1, \ldots, \boldsymbol{W}_L\} \\ \boldsymbol{b}_1, \ldots, \boldsymbol{b}_L}} \sum_{n=1}^{N} (y_n - f(\boldsymbol{x}_n; \boldsymbol{W}_1, \ldots, \boldsymbol{W}_L, \boldsymbol{b}_1, \ldots, \boldsymbol{b}_L))^2\,.$$

Typically, weights are found via an iterative method, commonly known as the back-propagation algorithm.

**Support vector machine** (SVM) models the regression function in such a way that it allows the predictions to have at most $\epsilon$ deviation from the actually measured $y$ without accounting for any error [12, 42]. Thus, by introducing a free parameter $\epsilon$, SVM allows the regression models to have small prediction errors, but does not penalize them for it. Specifically, using a mapping function $\phi(\cdot)$ on the input data $\boldsymbol{x}$, an $\epsilon$-SVM regression function is expressed as

$$f(\boldsymbol{x}; \boldsymbol{w}, b, \epsilon) = \boldsymbol{w}^T \phi(\boldsymbol{x}) + b\,,$$

where $\boldsymbol{w}$ and $b$ are the model parameters. Utilizing the training data $\{\boldsymbol{x}_n, y_n\}$, these model parameters are evaluated by solving a convex optimization problem:

$$\min_{\boldsymbol{w}, b, \zeta, \zeta^*} \frac{1}{2}\boldsymbol{w}^T \boldsymbol{w} + C \sum_{n=1}^{N} \left(\zeta_n + \zeta_n^*\right)$$

$$\text{subject to } \begin{cases} y_n - \boldsymbol{w}^T \phi(\boldsymbol{x}_n) - b \;\le\; \epsilon + \zeta_n \\ \boldsymbol{w}^T \phi(\boldsymbol{x}_n) + b - y_n \;\le\; \epsilon + \zeta_n^* \\ \zeta_n, \zeta_n^* \;\ge\; 0 \;\text{ for }\; n = 1, 2, \ldots N \end{cases}\,.$$

Here, the constant $C > 0$ is a regularization parameter, and $\zeta, \zeta^*$ are two slack parameters to ensure constraint feasibility. In general, this optimization problem is solved in its dual form that involves a positive semidefinite matrix $\boldsymbol{Q}$ having elements of the form $[\boldsymbol{Q}]_{n,n'} = K(\boldsymbol{x}_n, \boldsymbol{x}_{n'}) = \phi(\boldsymbol{x}_n)^T \phi(\boldsymbol{x}_{n'})$. This function $K(\boldsymbol{x}_n, \boldsymbol{x}_{n'})$ is commonly known as the SVM kernel, examples of which include the linear, polynomial, sigmoid, and radial basis functions.

**Random forest** (RF) is one of the *bagging* methods, which belong to a broader class of *ensemble learning* technique. In ensemble learning methods, multiple learners are used to obtain better predictive performance than could be obtained from any one of the constituent learner alone [38]. The motivation behind ensemble learning stems from the fact that every learner has an imperfect prediction capability, but a-priori it is not possible to know which one performs the best. Therefore, multiple learners are combined to create an ensemble model which reduces the effect of choosing a learner with a poor performance.

Specifically, in the RF technique, $B$ different regression-tree based base learners, $f_b(\cdot)$, are trained using $B$ bootstrapped data set $\{\{\boldsymbol{x}, y\}_b, b = 1, 2, \ldots, B\}$, which are obtained from the original data $\{\boldsymbol{x}, y\}$ via sampling with replacement procedure [5], [24, Ch. 8]. Each individual tree is grown very deep to overfit each bootstrapped data, i.e., to increase the variance while keeping the bias low. While building the regression trees, whenever a split in a tree is considered, a random sample of $M$ predictors are chosen as split candidates from the full set of $P$ predictors. The split is allowed to use only one of those $M$ predictors. Therefore, at each split in the tree, the base learning algorithms are allowed to consider only a random subset of all predictors, and thus the base regression trees decorrelate among themselves. Then, an average of all the trees are taken to formulate the bagging regression model as

$$f(\boldsymbol{x}) = \frac{1}{B} \sum_{b=1}^{B} f_b(\boldsymbol{x}_b).$$

The process of averaging these $B$ trees reduces the variance. It is to be noted here that the main difference between a general bagging approach and a random forest technique is the choice of the predictor subset size $M$, which is typically chosen as $M = \sqrt{P}$. If a random forest is built using $M = P$, then it simply becomes a general bagging approach.

**Gradient boosting** (GB) method builds the base regression-tree based learners sequentially using the residual information [16], [24, Ch. 8]. The motivation behind such approach is that each tree when fitted to the residuals can improve the overall predictions in areas where the previously grown trees did not perform well. Mathematically, the boosting regression model with $B$ regression trees is written as

$$f(\boldsymbol{x}) = \sum_{b=1}^{B} \lambda f_b(\boldsymbol{x}; r_b),$$

where $\lambda$ is the shrinkage parameter that controls the learning rate, and $r_b$ is the residual values applied to the $b$th tree. Initialized with the output values, $r_b$ is sequentially evaluated as $r_b = r_{b-1} - \lambda f_{b-1}(\boldsymbol{x}; r_{b-1})$. It is to be noted that, as boosting is a sequential approach, the construction of each tree depends strongly on the trees that have already been grown.

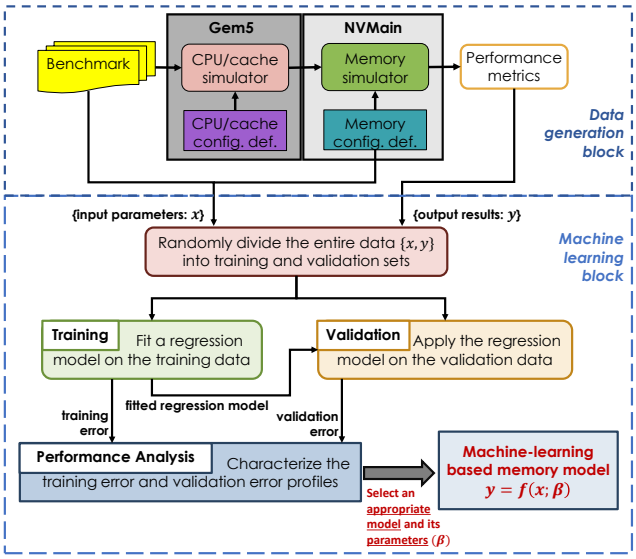



**Figure 1: Operational flowchart of the proposed machine learning based memory design space exploration method.**

## 4 ML BASED DSE METHOD

In this section, we describe in detail the procedures that we follow to perform a ML based DSE for hybrid main-memory design. Fig. 1 schematically shows our approach; the top portion depicts the block diagram of the data generation process, whereas the bottom portion shows the details of the ML procedure. As we apply the supervised learning techniques, the generation of training and validation data, that adequately represent the behaviors of hybrid main-memory systems, is as important as the application and design of the ML algorithms.

### 4.1 Data Generation

To generate the testing and validation data for the ML methods, we use a memory simulator tool, NVMain [39, 40], along with the Gem5 simulator [4] and two benchmark programs, the details of which are presented below.

*4.1.1 Benchmarks.* We use two benchmarks to simulate the performances of a hybrid main-memory using the Gem5 and NVMain simulators.

- *STREAM* [35] benchmark is the industry standard for measuring the memory bandwidth of a system and the rate of computation for simple vector kernels. It is specifically designed to work with data sets much larger than the available cache on any given system, so that the results are (presumably) more indicative of the main-memory performance of very large, vector style applications. There are four separate kernels, copy, scale, sum, and triad, performed by STREAM. Among them, triad, sum, and scale kernels perform arithmetic operations, and the copy kernel counts the read and written bytes. Each of the four kernels adds independent information to the results.

- *HPCG* [20] benchmark is designed to exercise computational and data access patterns that more closely match the commonly used scientific applications, in particular those with low computation-to-data-access ratios. As many real world problems are modeled with the partial differential equations (PDEs), HPCG benchmark generates a regular sparse linear system to model a finite difference discretization of a three-dimensional PDE (e.g., heat diffusion equation) on a semi-regular grid. Then, the resulting sparse linear system is solved using domain decomposition with a conjugate gradient method that uses a simple, symmetric Gauss-Seidel preconditioner. The basic computational blocks of HPCG benchmark are available in four routines: (i) SpMV (sparse matrix-vector multiplication), (ii) SYMGS (symmetric Gauss-Seidel smoother), (iii) WAXPBY (vector update) and (iv) DDOT (global dot product).

*4.1.2 Gem5 Simulator.* To simulate the memory responses via NVMain tool, we need to first provide NVMain simulator with the memory-trace files of the benchmarks, and we use the Gem5 simulator [4] to generate those memory-trace files. The Gem5 is a computer-system simulator platform with various modular components that simulate the passing of time as a series of discrete events. It can be used in two operating modes: (i) *syscall emulation* (SE) mode in which the system services are provided directly by the simulator, and (ii) *full system* (FS) mode in which a complete system with devices and an operating system can be simulated. Various CPU models, such as Alpha, ARM, Power, and 64 bit x86, and flexible memory systems are supported by Gem5 at varying degrees.

We use the Gem5 simulator in the SE mode with a basic configuration of one CPU and main-memory architecture. As our primary focus is to build ML based predictive models for memory responses, we did not consider any memory hierarchy with caches. The memory traces corresponding to the running of each benchmarks are obtained via trace-based debugging option of Gem5. In general, with the debugging option kept on, Gem5 prints out traces of potentially interesting events associated with its components, such as bus, cache, disk, etc. We specifically record those traces for the main memory.

*4.1.3 NVMain Simulator.* NVMain is an architectural-level simulator to model energy and cycle-accurate operations of DRAM, NVM, and their hybrid designs [39, 40]. It has several key parameters that can be modified to simulate different memory types and configurations.

- *Memory parameters* specify the architectural construction of a memory system. For example, CHANNELS, RANKS, BANKS, ROWS, COLS parameters respectively specify the number of channels, number of ranks on each channel, number of banks, number of wordlines in each memory bank, and number of logical columns in each memory bank. Also, the frequency of the interconnect to the memory (CLK) and frequency at which the memory controller runs (CPUFreq) can also be specified.
- *Timing parameters* are specified in the units of memory clock cycles, and they represent various memory functionalities. For example, column read time (tBURST), data restoration time (tRAS), row activation time (tRCD), precharge time (tRP), write-to-precharge time (tWR), and many more.
- *MLC parameters* specify various multi-level cell (MLC) configurations, for example, MLCLevels, SET/RESET pulse times, number of program pulses to write 00, 01, 10, and 11, and others.
- *Control parameters* are related with the operations of the tool, and they include PrintGraphs, PrintPreTrace, EchoPreTrace, TraceReader, InitPD, and others.

In this work, we simulate both DRAM and NVM separately and their hybrid combinations. Specifically, NVMain simulates DDR3 type of DRAM and implements the hybrid-memory system as a heterogeneous main-memory architecture with both on- and off-package memories along with a hardware-based page migrator [10, 40]. For each of the three memory configurations, we vary the values of CLK, CPUFreq, CHANNELS, tRAS, tRCD, while keeping the other parameters fixed. As NVM does not require any row restoration process, tRAS is kept at zero for the NVM configurations. Changes in the CLK and tRCD values also affect the values of tRCD, tWP, tCWD, tWTR, tXP, tWRPPDEN, and tWRAPDEN. In the hybrid configurations, we apply various combinations of these individual DRAM and NVM memory specifications to simulate the hybrid-memory architecture. At the simulation output, NVMain tool produces a detailed list of performance metrics, including average queue and response latencies, bandwidth, amount of data read and written, total power, total energy, number of simulation cycles, and others. These metrics are made available at the individual channel, rank, bank, and subarray levels of the simulated memory.

## 4.2 Machine Learning

As shown in the bottom portion of Fig. 1, we consider the benchmark and memory specifications as the input parameters $\boldsymbol{x}$ and the NVMain's output metrics as the measurements $y$ in the ML algorithms. Following the standard procedure, we randomly divide the entire data $\{\boldsymbol{x}, y\}$ into the training and validation sets respectively at 80%-20% splits. Then, using the training data, we train a learning model $f(\cdot;\beta)$ to figure out the values of model parameter $\beta$ that minimizes the training-error measure, which is in general expressed as the mean-squared error (MSE). Next, the designed model $f(\cdot;\hat{\beta})$ is applied on the validation data to evaluate the validation MSE for understanding the generalization performance of the designed model. Finally, the learning curves comprising of the training and validation error performances are analyzed to select an appropriate model and its parameters.

*4.2.1 Hyperparameter Tuning.* In general, the performances of the training and validation MSEs are strongly related with the flexibility or complexity of the chosen model, or more formally, with the degrees of freedom of the model. A less flexible/complex model has fewer degrees of freedom than a more flexible/complex model. As the flexibility of the learning model increase, it can better fit the training data, and as a result the training MSE monotonically decreases. On the other hand, the validation MSE initially decreases as the model flexibility increases. However, as the model flexibility continues to increase, the validation MSE levels off after some points, and then starts to increase again. This results in a U-shaped

behavior of the validation MSE. Particularly, by designing an overly flexible (or complex) model, we can achieve an extremely small training MSE, but we would encounter a large validation MSE. This effect is known as the *overfitting problem*, and it happens because an overly complex model would try too hard to learn patterns in the training data, and as a result may pick up some patterns that are caused by noise rather than the training data. Therefore, when such an overfitted model is applied to the validation data, the validation MSE would become large because the model looks for certain learned patterns in the data which simply do not exist. To avoid the overfitting problem, we need to select a learning model that is as flexible as to yield small values of both the training and validation errors. The process by which such an optimal complexity of the model (expressed via hyperparameters) is found is known as the hyperparameter tuning. Thus, the choice of hyperparameters defines and controls the structure of the learning model, whereas the model parameters $\beta$ learns patterns from the data.

## 5 RESULTS AND DISCUSSIONS

To illustrate the effectiveness of ML based DSE technique for hybrid main-memory design, in this section, we describe the design characteristics of the ML models and their error analyses. As mentioned earlier, we use four ML methods, MLP, SVM, RF, and GB, to build learning models for several memory responses, such as latency, bandwidth, power, and data read/written. However, before discussing various ML results, we provide a description of the data simulation and ML modeling setup.

### 5.1 Data Simulation Setup

The test and validation data for the ML models are simulated using NVMain memory simulator and Gem5 simulator for the STREAM and HPCG benchmarks. We use the Gem5 simulator in the default SE mode only to obtain the memory trace files while running each benchmarks. The configuration files of NVMain are changed to simulate different DRAM, NVM, and their hybrid architectures. Specifically, we use the following parameters:

- *DRAM*: controller frequency = 400, 666, 1250, 1600 MHz; number of channels = 2, 4; latency = 9 clock cycles; data restoration time = 24 clock cycles.
- *NVM*: controller frequency = 400, 666, 1250, 1600 MHz; number of channels = 2, 4; latency = 50, 75, 100, 125, 150, 200 clock cycles; data restoration time = 0 clock cycles.
- *Hybrid*: combinations of DRAM and NVM parameters.

Besides, we keep fixed various other parameters, such as, ranks = 1, banks = 8, rows = 65536, cols = 32, column read time = 4 clock cycles, etc. In addition to these simulator parameters, we keep two more parameters as inputs to the ML algorithms: (i) benchmark parameter, specifying either STREAM or HPCG, and (ii) hybrid memory parameter, specifying the ratio of the number of NVM channels to that of DRAM channels.

### 5.2 ML Modeling Setup

While building the ML models for each memory responses, their performances are analyzed in terms of training and validation RMSEs with respect to the modeling complexity. We discuss below which hyperparameters we tuned to vary the complexity of the models.

- *MLP*: For an MLP model, there are many hyperparameters that we can set to design the modeling topology and convergence properties. Examples of hyperparameter of an MLP model include the number of hidden layers, number of hidden units per layer, choice of activation function, learning rate (gradient descent step size), momentum term (used to avoid inferior local minima), distributions of initial weights, etc. In this work, we use an MLP model with two hidden layers having $L$ and $L/2$ units respectively in the first and second layers. We vary the values of $L$ to tune the complexity of the model. All other parameters of MLP are kept at their defaults values, for example, activation function is ReLU, learning rate = 0.001, momentum term = 0.9, etc.
- *SVM*: In an SVM model, examples of hyperparameters include the SVM kernels and their parameters, no-penalty parameter $\epsilon$, penalty parameter of the error term $C$, tolerance for stopping criterion, and others. In this work, we vary the penalty parameter $C$ to control the regularization effect, while keeping the other parameters at their default values, e.g., radial basis function as the SVM kernel and $\epsilon = 0.1$.
- *RF*: The main hyperparameter in an RF model is the number of trees, which we tune in our work to vary the complexity of the model. Other than the number of trees, there are several other parameters in an RF model, such as the maximum depth of tree, minimum number of samples required to split an internal branch and to be at a leaf node, threshold for early stopping in tree growth, etc., which we keep fixed at their default values.
- *GB*: Similar to RF model, the hyperparameters of a GB model include the number of trees, maximum depth of individual trees, minimum number of samples required to split an internal branch and to be at a leaf node, shrinkage parameter (also called learning rate), etc. While keeping all the other parameters fixed at their default values, we vary the complexity of the GB model by changing the number of trees.

In addition to these four ML models, we also train a linear regression model and consider its performance as a baseline while analyzing the performances of other ML models.

After we tune the hyperparameters of each model, we evaluate the prediction performances of the models with respect to the validation data by evaluating three different statistical measures:

(1) Mean absolute error: $\text{MAE}(y, \hat{y}) = \frac{1}{N_\text{v}} \sum_{n=1}^{N_\text{v}} |y_n - \hat{y}_n|$

(2) Root mean squared error: $\text{RMSE}(y, \hat{y}) = \sqrt{\frac{1}{N_\text{v}} \sum_{n=1}^{N_\text{v}} (y_n - \hat{y}_n)^2}$

(3) R2 score: $R^2(y, \hat{y}) = 1 - \dfrac{\sum_{n=1}^{N_\text{v}} (y_n - \hat{y}_n)^2}{\sum_{n=1}^{N_\text{v}} (y_n - \bar{y})^2}$

Here, $y$ is the original output values obtained from the simulator and $\hat{y}$ is the predicted values from the ML models. Also, $\bar{y} = (1/N_\text{v}) \sum_{n=1}^{N_\text{v}} y_n$ denotes the sample mean of the original data. Smaller the values of MAE and RMSE metrics and closer the R2-score value to 1.0, better is the designed ML model.

In the following subsections, we present results of the learning curve analyses and the regression error characterizations of the fitted models for each memory responses.

## 5.3 Latency Models

The behaviors of the learning curves while building the ML models for latency are shown in Fig. 2 for MLP, SVM, RF, and GB methods. As evident from Fig. 2(a), both the training and validation RMSEs decrease as we increase the size of the first hidden layer ($L$) in the MLP model. Then, as $L$ is further increased, the validation RMSE approximately plateaus down, but starts to increase again at higher $L$ values. On the other hand, the training RMSE monotonically decreases or approximately stays the same as $L$ is increased. This characteristic of the learning curves enables us to select a value of $L$ at which both the training and validation RMSE values are small. Once such an $L$ value is chosen, we build an MLP model with two hidden layers having $L$ and $L/2$ units respectively in the first and second layers. We repeat the same procedure for characterizing the learning curves of other ML models: SVM, RF, and GB, and Figs. 2(b)-2(d) show the behaviors of the training and validation RMSEs with respect to the chosen hyperparameter values. From these learning curves, we select the appropriate values of the hyperparameters to design the predictive models for latency.

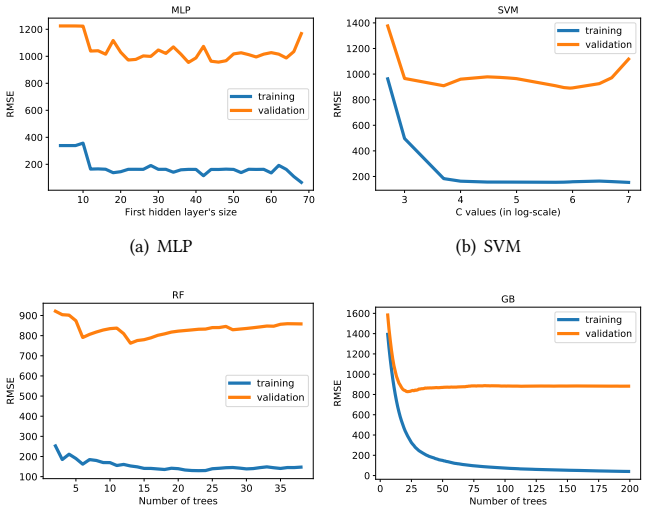
The prediction performances of the designed ML models for latency are shown in Fig. 3 and the corresponding statistical measures for these predictions are listed in Table 1. In Fig. 3, we plot the original simulated data for latency along with the corresponding predicted latency values by the designed ML models. We notice that all the four ML models fit the latency data well when the latency values are relatively small or large, but the fitting is not so good for the medium latency values of the order of 4000-5000 cycles. From these plots and the associated statistical metrics, it is evident that the RF method outperforms the other techniques in designing an ML model for predicting the latency values. It is also clear from Table 1 that all four models significantly outperform the linear regression model.

## 5.4 Bandwidth Models

We apply a similar approach as with the latency models to design ML models for bandwidth. We first characterize the learning curves for each model to figure out appropriate values of the hyperparameters, and then use those hyperparameters to design the ML models and subsequently assess their performances in predicting the bandwidth values. Fig. 4 shows the performance of each designed ML models in terms of their predicted bandwidth values in comparison to the original simulated bandwidth values. The statistical measures of these fits are tabulated in Table 1, and based on all the three statistical metrics, it is clear that the MLP model outperforms the other models by a considerable margin.

## 5.5 Power Models

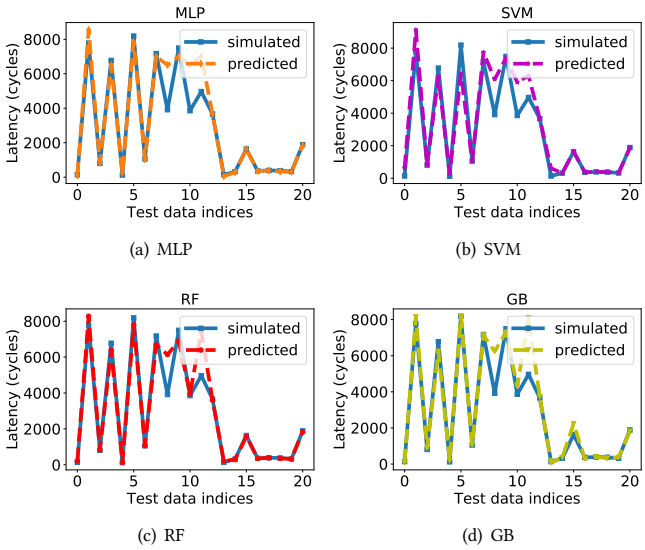
Fig. 5 shows performances of the designed ML models in predicting the memory power values. As before, we plot both the simulated and predicted power values in the same figure to show the accuracies of the ML based predictions. Also, before assessing the




(a) MLP (b) SVM

(c) RF (d) GB

**Figure 2: Learning curves of the ML models for latency.**




(a) MLP (b) SVM

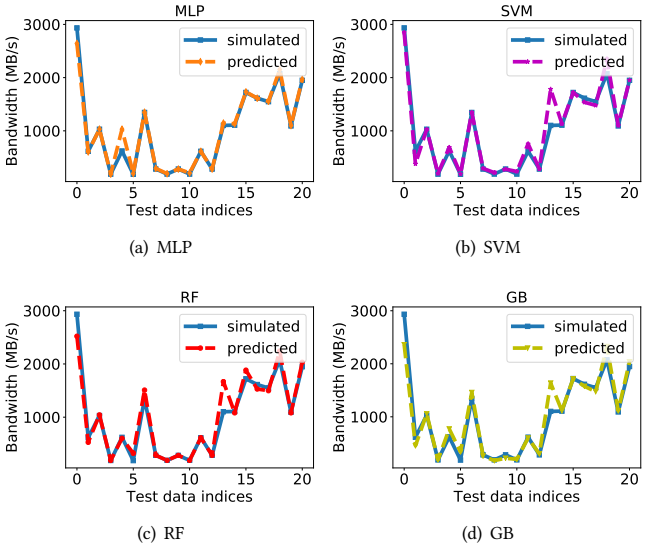
(c) RF (d) GB

**Figure 3: Prediction performances of the ML models for latency.**

prediction performances, the hyperparameters of all four ML models are chosen by following the learning curve analyses method. In particular, for the SVM model, the default value of parameter $\epsilon$ is changed from 0.1 to $0.1\times10^{-6}$ to match the small power values. The statistical metrics associated with the predictions of power values by the designed ML models are also listed in Table 1. Compared to other models, MLP clearly fails to provide a reasonable fit to the simulated power data. The other three methods, SVM, RF, and GB, have almost similar prediction performances, with SVM showing the best prediction capability among them.

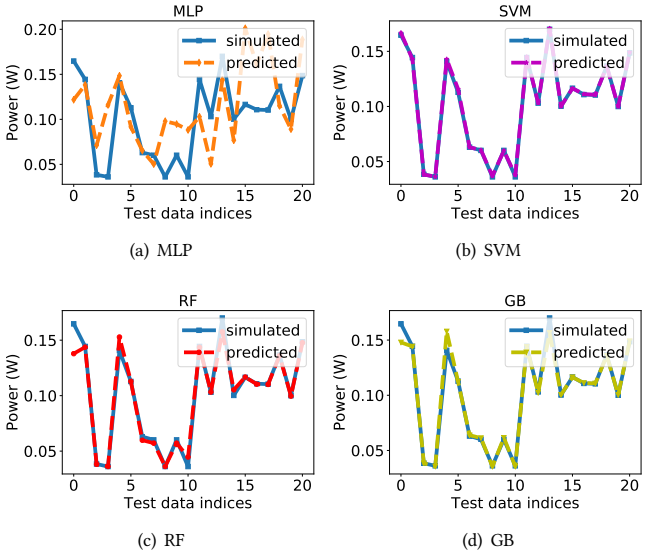
**Table 1: Prediction performances of the designed ML models for various memory responses**

| Memory parameters | Error metrics | Linear | MLP | SVM | RF | GB |
|---|---|---|---|---|---|---|
| Latency (cycles) | MAE | 1209.13 | 468.60 | 516.45 | 361.43 | 569.21 |
| | RMSE | 1398.71 | 954.37 | 890.32 | 762.44 | 827.58 |
| | R2 score | 0.77 | 0.89 | 0.91 | 0.93 | 0.92 |
| Bandwidth (MB/s) | MAE | 267.27 | 45.99 | 85.25 | 94.56 | 111.20 |
| | RMSE | 332.59 | 111.94 | 175.27 | 171.30 | 192.94 |
| | R2 score | 0.80 | 0.98 | 0.95 | 0.95 | 0.93 |
| Power (W) | MAE | $10.3 \times 10^{-3}$ | $37.3 \times 10^{-3}$ | $0.8 \times 10^{-3}$ | $3.7 \times 10^{-3}$ | $2.7 \times 10^{-3}$ |
| | RMSE | $15.5 \times 10^{-3}$ | $44.9 \times 10^{-3}$ | $1.1 \times 10^{-3}$ | $7.4 \times 10^{-3}$ | $6.0 \times 10^{-3}$ |
| | R2 score | 0.87 | 0.36 | 0.99 | 0.97 | 0.98 |
| Memory read | MAE | 39.88 | $0.6 \times 10^{6}$ | $1.9 \times 10^{6}$ | 5.76 | 11.57 |
| | RMSE | 48.32 | $1.5 \times 10^{6}$ | $3.9 \times 10^{6}$ | 15.11 | 25.34 |
| | R2 score | 1.00 | 0.99 | 0.99 | 1.00 | 1.00 |
| Memory write | MAE | 39.86 | $0.4 \times 10^{5}$ | $2.0 \times 10^{5}$ | 4.52 | 11.32 |
| | RMSE | 48.31 | $0.7 \times 10^{5}$ | $4.2 \times 10^{5}$ | 14.04 | 25.07 |
| | R2 score | 1.00 | 0.99 | 0.99 | 1.00 | 1.00 |




(a) MLP (b) SVM (c) RF (d) GB

**Figure 4: Prediction performances of the ML models for bandwidth.**



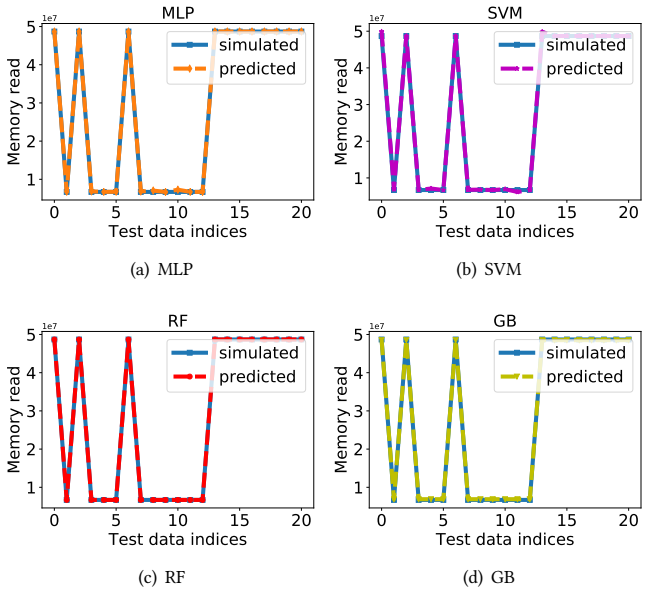

(a) MLP (b) SVM (c) RF (d) GB

**Figure 5: Prediction performances of the ML models for power.**

## 5.6 Memory Read/Write Models

The simulated values of amounts of data read from memory show a simple pattern: for a particular benchmark, any pure DRAM and NVM memories yield the same amounts of data read values, whereas a hybrid DRAM/NVM memory produce a little higher data read values. This pattern in the simulated values is well understood even by a linear regressor, as can be seen in Table 1. The four designed ML models, MLP, SVM, RF, and GB, are also able to obtain excellent fits to the values of data read from memory, and their prediction performances are depicted in Fig. 6. The statistical metrics corresponding to these predictions are tabulated in Table 1, which shows that RF and GB models yield the best predictions of data read values while considering all the three metrics, and all four models perform equally well when considering only the R2-score metric.

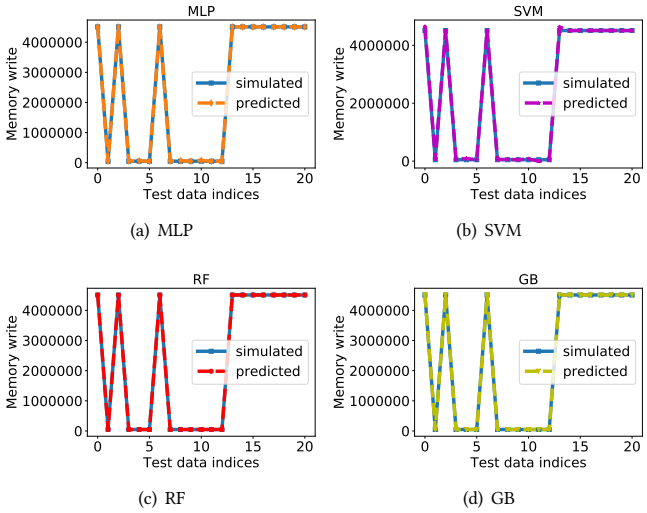
For the simulated values of data written to memory, we observe very similar characteristics from the designed ML models, whose prediction performances are shown in Fig. 7 and in the last row of Table 1.




(a) MLP (b) SVM

(c) RF (d) GB

**Figure 6: Prediction performances of the ML models for amounts of data read from memory.**




(a) MLP (b) SVM

(c) RF (d) GB

**Figure 7: Prediction performances of the ML models for amounts of data written to memory.**

## 6 CONCLUSIONS

In this paper, we developed a machine learning (ML) based design space exploration (DSE) method to build predictive models for various responses of a hybrid main-memory system. Hybrid memory architectures are now being considered for the future extreme-scale machines as the conventional DRAM systems will not be able to meet several challenges associated with latency, capacity, energy efficiency, and application performance. Therefore, various novel non-volatile memory (NVM) systems have been proposed in recent years, and their usability is being assessed along with conventional DRAMs in the hybrid memory architectures. In this regard, to quickly analyze and predict performances of the hybrid main-memory systems, we utilized the ML techniques to build predictive models for different memory responses. This approach overcomes the problem of enormous computational costs and time associated with the traditional simulator based memory-DSE method.

Specifically, we applied two fixed-structure supervised models, artificial neural network (ANN) and support vector machine (SVM), and two varying-structure supervised models, random forest (RF) and gradient boosting (GB), to design the predictive models for memory responses. The training and validation data for these ML models were generated using NVMain memory simulator for both DRAM and NVM, and also for their hybrid combinations. The input memory-trace files for NVMain tool were simulated using the Gem5 simulator for two benchmarks: STREAM and HPCG. Each of the four ML techniques were used to build learning models for memory latency, bandwidth, power, and total read/write responses. Also, the learning curves of each model were analyzed to tune the hyperparameters, and thus to avoid the overfitting problem. Overall, the SVM and RF methods were found to be yielding better prediction accuracies respectively in the fixed and varying structure models.

In our future work, we plan to train and validate our ML models with several other benchmarks selected from memory read-heavy, memory write-heavy, and compute-heavy categories. Incorporation of more benchmarks will enable us to build the ML based memory-response models having more generalized prediction capabilities. In addition, we will utilize more advanced ML methods, such as the transfer learning and semi-supervised learning, to move beyond the supervised learning domain which strongly depends on the labeled simulated data for training and validation purposes. In particular, the transfer learning approach will be useful in those memory systems for which we would not have sufficient number of simulated data, but the knowledge gained from one (or many) similar/related system(s) will be applied to build the predictive models. On the other hand, the semi-supervised learning method will be effective in exploiting the unlabeled data (i.e., memory configurations without any simulated responses) to improve the quality of the predictive models. Further, we plan to validate the performance of our ML based memory-response models with real data.

## ACKNOWLEDGMENTS

This work is supported by the United States Department of Defense (DoD) and used resources of the Computational Research and Development Programs at the Oak Ridge National Laboratory, managed by UT-Battelle, LLC, for the U.S. Department of Energy, under contract DE-AC05-00OR22725. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a nonexclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this manuscript, or allow others to do so, for United States Government purposes. The Department of Energy will provide public access to these results of federally

sponsored research in accordance with the DOE Public Access Plan (http://energy.gov/downloads/doe-public-access-plan)..

## REFERENCES


[1] S. Ashby, P. Beckman, J. Chen, P. Colella, B. Collins, D. Crawford, J. Dongarra, D. Kothe, R. Lusk, P. Messina, et al. Fall 2010. *The Opportunities and Challenges of Exascale Computing*. Advanced Scientific Computing Advisory Committee (ASCAC) Subcommittee Rep.

[2] A. Awad, G. B. Voskuilen, A. F. Rodrigues, S. D. Hammond, R. J. Hoekstra, and C. Hughes. 2017. *Messier: A Detailed NVM-Based DIMM Model for the SST Simulation Framework*. Technical Report. Sandia National Laboratories (SNL-NM), Albuquerque, NM.

[3] S. Baek, D. Son, D. Kang, J. Choi, and S. Cho. 2014. Design space exploration of an NVM-based memory hierarchy. In *IEEE 32nd Intl Conf. Computer Design (ICCD)*. Seoul, South Korea, 224–229.

[4] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, et al. 2011. The gem5 simulator. *ACM SIGARCH Computer Architecture News* 39, 2 (May 2011), 1–7.

[5] L. Breiman. 2001. Random forests. *Machine Learning* 45, 1 (Oct. 2001), 5–32.

[6] A. Chen. 2014. Emerging research device roadmap and perspectives. In *IEEE Intl Conf. on IC Design Tech.* Austin, TX, 1–4.

[7] A. Chen. 2016. A review of emerging non-volatile memory (NVM) technologies and applications. *Solid-State Electronics* 125 (Nov. 2016), 25 – 38.

[8] J. Coburn, A. M. Caulfield, A. Akel, L. M. Grupp, R. K. Gupta, R. Jhala, and S. Swanson. 2011. NV-Heaps: Making persistent objects fast and safe with next-generation, non-volatile memories. In *Proc. 16th Intl Conf. Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. Newport Beach, CA, 105–118.

[9] J. Condit, E. B. Nightingale, C. Frost, E. Ipek, B. Lee, D. Burger, and D. Coetzee. 2009. Better I/O through byte-addressable, persistent memory. In *Proc. ACM SIGOPS 22nd Symp. Operating Systems Principles (SOSP)*. Big Sky, MT, 133–146.

[10] X. Dong, Y. Xie, N. Muralimanohar, and N. P . Jouppi. 2010. Simple but Effective Heterogeneous Main Memory with On-Chip Memory Controller Support. In *Proc. 2010 ACM/IEEE Intl Conf. for High Performance Computing, Networking, Storage and Analysis (SC '10)*. New Orleans, LA, 1–11.

[11] T. D. Doudali, S. Blagodurov, A. Vishnu, S. Gurumurthi, and A. Gavrilovska. 2019. Kleio: A Hybrid Memory Page Scheduler with Machine Intelligence. In *Proc. 28th Intl Symp. High-Performance Parallel and Distributed Computing (HPDC)*. Phoenix, AZ, 37–48.

[12] H. Drucker, C. J. C. Burges, L. Kaufman, A. Smola, and V. Vapnik. 1996. Support vector regression machines. In *Proc. 9th Intl Conf. Neural Information Processing Systems (NIPS)*. Denver, CO, 155–161.

[13] Z. Duan, H. Liu, X. Liao, and H. Jin. 2018. HME: A lightweight emulator for hybrid memory. In *Design, Automation and Test in Europe Conf. Exhibition (DATE)*. Dresden, Germany, 1375–1380.

[14] C. Dubach. 2009. *Using machine-learning to efficiently explore the architecture/compiler co-design space*. Ph.D. Dissertation. University of Edinburgh, UK.

[15] Summit Oak Ridge Leadership Computing Facility. 2019. https://www.olcf.ornl.gov/summit/

[16] J. H. Friedman. 2001. Greedy function approximation: A gradient boosting machine. *Annals of statistics* 29, 5 (Oct. 2001), 1189–1232.

[17] Q. Guo, T. Chen, Y. Chen, Z.-H. Zhou, W. Hu, and Z. Xu. 2011. Effective and efficient microprocessor design space exploration using unlabeled design configurations. In *Proc. 22nd Intl Joint Conf. Artificial Intelligence (IJCAI)*. Barcelona, Spain, 1671–1677.

[18] M. S. Haque, A. Li, A. Kumar, and Q. Wei. 2015. Accelerating non-volatile/hybrid processor cache design space exploration for application specific embedded systems. In *20th Asia and South Pacific Design Automation Conf.* Chiba, Japan, 435–440.

[19] M. Hashemi, K. Swersky, J. Smith, G. Ayers, H. Litz, J. Chang, C. Kozyrakis, and P. Ranganathan. 2018. Learning Memory Access Patterns. In *Proc. 35th Intl Conf. Machine Learning (ICML)*. Stockholm, Sweden, 1919–1928.

[20] HPCG. 2015. http://www.hpcg-benchmark.org/

[21] E. Ipek, S. A. McKee, R. Caruana, B. R. de Supinski, and M. Schulz. 2006. Efficiently exploring architectural design spaces via predictive modeling. In *Proc. 12th Intl Conf. Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. San Jose, CA, 195–206.

[22] E. Ipek, O. Mutlu, J. F. MartÃŋnez, and R. Caruana. 2008. Self-optimizing memory controllers: A reinforcement learning approach. In *Intl Symp. Computer Architecture*. Beijing, China, 39–50.

[23] A. K. Jain, J. Mao, and KM Mohiuddin. 1996. Artificial neural networks: A tutorial. *Computer* 3 (March 1996), 31–44.

[24] G. James, D. Witten, T. Hastie, and R. Tibshirani. 2013. *An Introduction to Statistical Learning with Applications in R*. Springer, New York.

[25] W. Jia, K. A. Shaw, and M. Martonosi. 2012. Stargazer: Automated regression-based GPU design space exploration. In *IEEE Intl Symp. Performance Analysis of Systems Software*. New Brunswick, NJ, 2–13.

[26] D. Kim, S. Lee, J. Chung, D.H. Kim, D.H. Woo, S. Yoo, and S. Lee. 2012. Hybrid DRAM/PRAM-based main memory for single-chip CPU/GPU. In *Proc. 49th Annual Design Automation Conf. (DAC)*. San Francisco, CA, 888–896.

[27] R. G. Kim, J. R. Doppa, and P. P. Pande. 2018. Machine learning for design space exploration and optimization of manycore systems. In *Proc. Intl Conf. Computer-Aided Design (ICCAD)*. San Diego, CA, 48:1–48:6.

[28] M.H. Kryder and K. Chang Soo. 2009. After hard drives: What comes next? *IEEE Transactions on Magnetics* 45, 10 (Oct. 2009), 3406–3413.

[29] B. C. Lee and D. M. Brooks. 2006. Accurate and efficient regression modeling for microarchitectural performance and power prediction. In *Proc. 12th Intl Conf. Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. San Jose, CA, 185–194.

[30] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger. 2009. Architecting phase change memory as a scalable DRAM Alternative. In *Proc. 36th Annual Intl Symp. Computer Architecture (ISCA)*. Austin, TX, 2–13.

[31] E. Lee, H. Bahn, and S. H. Noh. 2013. Unioning of the buffer cache and journaling layers with non-volatile memory. In *Proc. 11th USENIX Conf. File and Storage Technologies (FAST)*. 73–80.

[32] D. Li, S. Wang, S. Yao, Y. Liu, Y. Cheng, and X. Sun. 2016. Efficient design space exploration by knowledge transfer. In *Intl. Conf. Hardware/Software Codesign and System Synthesis (CODES+ISSS)*. Pittsburgh, PA, 1–10.

[33] D. Li, S. Yao, Y. Liu, S. Wang, and X. Sun. 2016. Efficient design space exploration via statistical sampling and AdaBoost learning. In *53rd ACM/EDAC/IEEE Design Automation Conf. (DAC)*. Austin, TX, 1–6.

[34] J. F. Martinez and E. Ipek. 2009. Dynamic multicore resource management: A machine learning approach. *IEEE Micro* 29, 5 (Sept. 2009), 8–17.

[35] J. D. Mccalpin. 1995. Sustainable Memory Bandwidth in Current High Performance Computers. http://www.cs.virginia.edu/~mccalpin/papers/bandwidth/bandwidth.html

[36] S. Mittal and J.S. Vetter. 2014. *A Survey of Software Techniques for Using Non-Volatile Memories for Storage and Main Memory Systems*. Technical Rep. ORNL/TM-2014/633. Oak Ridge National Laboratory.

[37] I. B. Peng and J. S. Vetter. 2018. Siena: Exploring the design space of heterogeneous memory systems. In *Intl Conf. for High Performance Computing, Networking, Storage and Analysis (SC18)*. Dallas, TX, 427–440.

[38] R. Polikar. 1995. Ensemble Learning. In *Ensemble Machine Learning Methods and Applications*, Cha Zhang and Yunqian Ma (Eds.). Springer, New York, Chapter 1, 1–34.

[39] M. Poremba and Y. Xie. 2012. NVMain: An architectural-level main memory simulator for emerging non-volatile memories. In *IEEE Computer Society Annual Symp. on VLSI*. Amherst, MA, 392–397.

[40] M. Poremba, T. Zhang, and Y. Xie. 2015. NVMain 2.0: A user-friendly memory simulator to model (non-)volatile memory systems. *IEEE Computer Architecture Letters* 14, 2 (July 2015), 140–143.

[41] L. Shi, J. Li, C. Jason Xue, and X. Zhou. 2013. Hybrid nonvolatile disk cache for energy-efficient and high-performance systems. *ACM Trans. Des. Autom. Electron. Syst.* 18, 1 (Jan. 2013), 8:1–8:23.

[42] A. J. Smola and B. Schölkopf. 2004. A tutorial on support vector regression. *Statistics and Computing* 14, 3 (Aug. 2004), 199–222.

[43] C. W. Smullen, V. Mohan, A. Nigam, S. Gurumurthi, and M. R. Stan. 2011. Relaxing non-volatility for fast and energy-efficient STT-RAM caches. In *IEEE 17th Intl Symp. High Performance Computer Architecture*. San Antonio, TX, 50–61.

[44] J. Stevens, P. Tschirhart, Mu-Tien Chang, I. Bhati, P. Enns, J. Greensky, Z. Chisti, Shih-Lien Lu, and B. Jacob. 2013. An integrated simulation infrastructure for the entire memory hierarchy: Cache, DRAM, nonvolatile memory, and disk. *Intel Technology Journal* 17, 1 (2013), 184–200.

[45] M.E. Tolentino, J. Turner, and K.W. Cameron. 2009. Memory MISER: Improving main memory energy efficiency in servers. *IEEE Trans. Comput.* 58, 3 (March 2009), 336–350.

[46] TOP500 Supercomputer Sites. 2018. https://www.top500.org/

[47] J. Vetter and S. Mittal. 2015. Opportunities for nonvolatile memory systems in extreme-scale high performance computing. *Computing in Science and Engineering* 17, 2 (March 2015), 73–82.

[48] H. Volos, G. Magalhaes, L. Cherkasova, and J. Li. 2015. Quartz: A lightweight performance emulator for persistent memory software. In *Proc. 16th Annual Middleware Conf.* Vancouver, Canada, 37–49.

[49] H. Volos, A.J. Tack, and M.M. Swift. 2011. Mnemosyne: Lightweight persistent memory. *ACM SIGPLAN Notices* 46, 3 (March 2011), 91–104.

[50] G. Wu, H. Zhang, Y. Dong, and J. Hu. 2012. CAR: Securing PCM main memory system with cache address remapping. In *IEEE 18th Intl Conf. Parallel and Distributed Sys.* Singapore, Singapore, 628–635.

[51] Wm. A. Wulf and S. A. Mckee. 1995. Hitting the memory wall: Implications of the obvious. *Computer Architecture News* 23 (March 1995), 20–24.