Learned Performance Model for SSD

Hyeon Gyu Lee, Minwook Kim, Juwon Lee, Eunji Lee, Bryan S. Kim, Sungjin Lee, Yeseong Kim, Sang Lyul Min, and Jin-Soo Kim

Abstract—The advent of new SSDs with ultra-low latency makes the validation of their firmware critical in the development process. However, existing SSD simulators do not sufficiently achieve high accuracy in their performance estimations for their firmware. In this paper, we present an accurate and data-driven performance model that builds a cross-platform relationship between the simulator and target platform. We directly execute the firmware on both platforms, collect its related performance profiles, and construct a performance model that infers the firmware's performance on the target platform using performance events from the simulation. We explore both a linear regression model and a deep neural network model, and our cross-validation shows that our model achieves a percent error of 3.1%, significantly lower than 18.9% from a state-of-the-art simulator.

Index Terms—Cross-platform, performance prediction, solid state drives, simulation

1 Introduction

In the past, accessing the storage media (such as the NAND flash in SSD) has been the traditional bottleneck in storage I/O. However, with the emergence of microsecond-scale storage technology such as 3D XPoint [1] and ultra-low latency flash [2], the performance of the in-device firmware such as the flash translation layer (FTL) in SSDs plays a significant role in I/O processing [3], [4]. Fig. 1 illustrates that the time spent on the FTL increases significantly, from 9% to 42% on average, as the underlying NAND hardware shifts from conventional to ultra-low latency Z-NAND [2], [5]. Furthermore, even for the existing NAND flash, it is widely known that garbage collection, the space reclamation task performed by the FTL, is the source of long-tail latencies in SSD [6], [7]. Therefore, the FTL must be carefully designed and validated so that it does not become the bottleneck in processing I/O.

The design of an SSD is simulated before prototyping and production because it comprises both hardware and firmware components in an embedded environment. In general, the main purpose of a simulator is to *simulate* the behavior of the target device on a host machine. By testing and verifying an SSD design with a simulator, the development time and cost can be significantly reduced. Furthermore, a simulator enables repeated testing of SSDs without flash memory wear-out issues. For a prototype-based validation, the NAND flash wears out in the process of testing, making it difficult to reproduce the same results across runs. To faithfully fulfill the above objectives, the simulators should be able to accurately model critical components of the target device. This includes the NAND flash subsystem, firmware (i.e., FTL), and the execution environment (i.e., embedded CPU).

Unfortunately, most existing simulation infrastructures focus on the NAND subsystem behavior only, neither modeling nor considering the SSD's internal firmware and the embedded CPU: the time it takes for the FTL to execute is simply assumed to be instantaneous [8], [9], [10], [11], [12], [13], or the FTL's performance is that of host CPU where the simulation runs [14], [15]. Only Amber [16] models the FTL performance by statically analyzing the firmware instructions. However, it does not capture the runtime dynamics of the FTL and exhibits a significantly high percent error of 18.9% on average. Table 1 summarizes the behavioral description and the model accuracy of these prior works.

Motivated by the need for a highly accurate model for the FTL, we propose a *learned* prediction of FTL performance during SSD simulation by directly executing the FTL and calibrating its performance through a trained data-driven model. In particular, our model is trained with a sufficiently large set of performance results from both the host CPU for the simulation and the target CPU for the prototype platform. During direct execution, we capture the hardware performance events related to the performance of FTL and reflect its dynamics on the target device. We explore both a simple linear regression and a deep neural network (DNN) with hyperparameter search to find the most concise and accurate prediction model. Our evaluation shows that a linear regression model achieves an average percent error of 9.7% and our DNN model, a significantly lower percentage of 3.1%.

Our model provides a more convenient test environment for SSDs with improved accuracy, alleviating dependence on the target prototype board. In particular, our model reduces the required amount of execution of workloads on the target prototype board that has limited endurance. This allows the simulator to faithfully produce accurate results on a diverse set of workloads while circumventing the issues of limited hardware availability and wear out. In addition, we also enhance the speed for tests with long-running workloads without executing them on the target, as the execution environment on a host is generally faster than that of the device target platform. As such, our model makes it easy to rapidly develop a complex hardware-software design such as an SSD. To the best of our knowledge, this is the first work that applies a data-driven approach to the SSD performance prediction.

2 Related Work

Learned prediction with data-driven approach is widely used in performance and power modeling to reflect the performance of an application on a target platform. In particular, LACross [19] and P4 [20] use deep neural networks for cross-platform performance prediction by profiling the performance events. These works capture the non-linear differences between two systems (the host and the target), and predict the performance and power of a target system with limited availability using data from the host system. Meanwhile, Ithemal [21] targets to predict it from instructions with an LSTM. They have high accuracy in predicting the performance for general applications on a target platform.

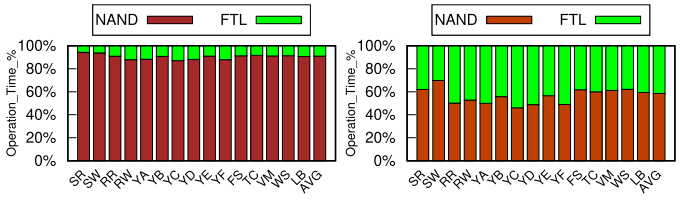
Inspired by these approaches, we apply a data-driven methodology to predict the SSD performance on the target platform with that on the host.

3 Learned Performance Model for SSD

Our approach for high accuracy in modeling SSD's performance bridges the gap between the simulator and the target device by

- *Hyeon Gyu Lee, Minwook Kim, Sang Lyul Min, and Jin-Soo Kim are with the Seoul National University, Seoul 03080, South Korea. E-mail: {flewin, ace, symin, jinsoo.kim}@snu.ac.kr.*
- *Juwon Lee is with FADU, Seoul 06145, South Korea. E-mail: jwlee@fadutec.com.*
- *Eunji Lee is with Soongsil University, Seoul 06978, South Korea. E-mail: ejlee@ssu.ac.kr.*
- *Bryan S. Kim is with Syracuse University, Syracuse, NY 13244 USA. E-mail: bkim01@syr.edu.*
- *Sungjin Lee and Yeseong Kim are with DGIST, Daegu 42988, South Korea. E-mail: {sungjin.lee, yeseongkim}@dgist.ac.kr.*

Manuscript received 8 Sept. 2021; accepted 3 Oct. 2021. Date of publication 19 Oct. 2021; date of current version 4 Nov. 2021.
This work was supported by the National Research Foundation (NRF) of Korea under Grants NRF-2018R1A5A1060031, NRF-2019R1A2C2089773, and NRF-2020R1A4A4079859.
(Corresponding author: Bryan S. Kim.)
Digital Object Identifier no. 10.1109/LCA.2021.3120728



(a) Conventional NAND (b) Z-NAND

Fig. 1. *Measured operation time for each part of SSD.* We assume four channels, four ways per channel, and two dies per way for the parallelism of the NAND subsystem. For Z-NAND, 12μs and 400μs for each 16KB flash page read and program are assumed respectively [5], while 150μs and 2600μs are assumed for them in the conventional NAND. More details on the workload in the x-axis are presented later in Table 4.

TABLE 1
Comparison Among Existing Works

Simulators	Behavior description		Accuracy on performance	
	NAND	FTL	NAND	FTL
SSDModel [17]	✓	✗	✗	✗
FlashSim [8]	✓	✓	✓	✗
SSDSim [9]	✓	✓	✓	✗
NANDFlashSim [11]	✓	✓	✓	✗
VSSIM [14]	✓	✓	✓	✗
WiscSim [12]	✗	✓	✗	✗
SimpleSSD [18]	✓	✓	✓	✗
MQSim [13]	✓	✓	✓	✗
FEMU [15]	✓	✓	✓	✗
Amber [16]	✓	✓	✓	▲

The accuracy on the performance of the NAND subsystem is checked if it is compared against a real hardware. Most simulators have behavioral models for both the NAND subsystem and the FTL, but they have non-sufficient accuracy on the performance of FTL.

directly executing the FTL on the host system and inferring its performance on the target device. More specifically, we collect the performance events from the host and the I/O processing time from both the host and the target while running the FTL. The performance event vectors (collected by perf [22]) and the processing time from the host are the input to the model, and the processing time on the target is the output from the model. This prediction model is trained offline with a sufficiently large number of workloads (as described in Section 4), and this model predicts the FTL performance on the target platform online when simulating the SSD on the host. We use both a linear regression model and a neural network to predict the SSD performance via hardware performance counters.

Dataset Collection. We measure the performance and profile its related event vectors while running a basic multi-threaded FTL: details for this FTL and event vectors are described in Section 4. We sequentially group consecutive 1000 I/O requests to represent a single data record so that the events and performance capture a steady state. We collect the dataset with a sufficient workloads that consist of 55 million requests in total, and the number of records for each workload is demonstrated in Table 4.

Linear Model. After this data collection process, we build a linear model that performs a simple linear regression to predict the I/O processing time. We validate this model with the "*k*-fold cross-validation" and determine the prediction accuracy. This validation process adopts a *leave-one-out* strategy, by training the model with the data from *k-1* workloads and then testing the model with the data from the remaining workload. We have the dataset from 15 workloads described in Table 4, so *k* becomes 15 for our case. We test all 15 models trained without the data from each workload.

Neural Network. We build a neural network model with the same dataset, but we additionally perform an exhaustive hyperparameter search to find the model parameters that yield the highest accuracy. Table 2 shows the configuration space we explored in this step. We empirically choose the values for them to find configurations that achieve high accuracy. We empirically test these parameters and validate each model with the cross-validation aforementioned. For each case of hyperparameter, we compute the average percentage error from 15 error values calculated by 15 tests. We perform this procedure for all cases of the hyperparameter of the model stated in Table 2, and we find the most accurate model with the lowest average percentage error value. Our final neural network model selected through this hyperparameter search has six hidden layers with 128 neurons per layer, with a learning rate value of 0.001.

TABLE 2
Hyperparameter Space for Determining Complexity of the DNN Model

Parameter	Range
# of hidden layers	{4, 5, 6, 7}
# of neurons per layer	{128, 256, 512}
Learning rate	{0.0001, 0.0002, 0.001, 0.002}
Batch size	32
Maximum # of iteration	20
Activation	Sigmoid
Loss function	Mean Square Error
Optimizer	Adam

4 Evaluation

In this section, we evaluate the accuracy of our linear regression and neural network model for modeling the SSD's firmware performance and compare them to Amber [16] that statically analyzes the FTL. We first describe the experimental setup and then present the evaluation results.

4.1 Experimental Setup

FTL. We implement a basic multi-threaded page-mapping FTL that supports high concurrency, including garbage collection. The garbage collector runs greedily, triggers when the free blocks fall below 10% of the total number of physical blocks, and reclaims space until the free blocks exceed 15%. The SSD's logical capacity is set to 1TiB with an over-provisioning factor of 25%.

Hardware. We have two sets of hardware: one for the simulation and another for the target prototype: accuracy is defined by how close the simulated performance is to the actual performance on the target prototype. Amber [16] requires hardware specifications for both the CPU and DRAM to model the firmware performance. Amber's CPU model is hardwired to ARMv8 cores: it is different from the ARM core on the target prototype, but the difference between ARMv8 and ARM Cortex-A53 is much smaller than that with the Intel Xeon processor. For DRAM, we use less than 512MB out of the 16GB available on the target prototype, including 256MB mapping table and 1MB each for user data read and write buffer. We modify the timing parameters for Amber's DRAM model to correctly reflect the target prototype's DDR4. Table 3 summarizes the hardware specification between two platforms for experiments.

Workloads. We measure the performance of the SSD (including both simulators and the real implementation on the target prototype) using a variety of workloads—both synthetic and real-world traces— as shown in Table 4. For synthetic workloads, we use four workloads: sequential read, sequential write, random read, and random write, with ten million I/O requests each. For real workloads, we use traces from Filebench, Linkbench, TPC-C, and YCSB. Each benchmark is executed on a real SSD, and we extract traces through `blktrace`. For YCSB, the workloads are run on top of RocksDB using an SSD as a backing storage device.

TABLE 3
Comparison Between Platforms

Host for simulator	Features	Target prototype
Intel Xeon E5-2650v4	**ISA**	ARM Cortex-A53
2.2GHz	**CPU frequency**	1.2GHz
32KB	**L1 cache size**	32KB
256KB	**L2 cache size**	256KB
2.5MB	**L3 cache size**	No L3
PCIe 3.0	**System bus**	AXI4
DDR4 2400MT/s	**DRAM speed**	DDR4 2400MT/s
32GB	**DRAM capacity**	16GB

Our target prototype consists of a ZCU102 evaluation board that has four 1.2GHz ARM Cortex-A53 SoC cores, commonly used for high-performance SSD firmware [23].

TABLE 4
Workload Characteristics

Label	Workload	# of Rec	W%	R%	FP(GiB)		DA(GiB)	
					W	R	W	R
SR	Sequential read	10000	0	100	0.0	152.6	0.0	152.6
SW	Sequential write	10000	100	0	152.6	0.0	152.6	0.0
RR	Random read	10000	0	100	0.0	152.6	0.0	152.6
RW	Random write	10000	100	0	152.6	0.0	152.6	0.0
YA	YCSB-A	1240	41	59	3.8	4.0	3.9	5.6
YB	YCSB-B	951	34	66	2.4	3.4	2.4	4.8
YC	YCSB-C	912	33	67	2.2	3.3	2.3	4.7
YD	YCSB-D	849	36	64	2.3	3.2	2.3	4.2
YE	YCSB-E	834	37	63	2.3	3.3	2.3	4.0
YF	YCSB-F	1245	41	59	3.8	4.0	3.9	5.6
FS	Fileserver	3258	100	0	21.7	0.0	518.8	0.0
TC	TPC-C	637	52	48	1.1	2.1	2.5	2.3
VM	Varmail	1305	100	0	4.2	0.0	16.0	0.0
WS	Webserver	1877	100	0	60.5	0.0	228.4	0.0
LB	Linkbench	2225	86	14	4.8	1.9	14.6	2.3

# of Rec (number of records) is the size of each dataset (grouped by 1000 requests). FP (access footprint) is the size of the logical address space range accessed, and DA (Data accessed) is the total amount of read or written data.

Tools. For building models, we use the LinearRegression model on Scikit-learn 0.20.4 and the neural network model on PyTorch 1.4.0, both included in Python 3.6.9. We run our hyperparameter search process with Ray 1.0.1.post1.

4.2 Accuracy on SSD Performance

Fig. 2a compares the prediction results for average runtime from three models: static analysis model from Amber, linear regression model, and our DNN model. We migrate the static model in Amber to our FTL with modified timing parameters for CPU and DRAM. The results from our DNN model are from the aforementioned k-fold cross-validation process for each workload. The measured SSD performance values from both platforms show significant differences as plotted by bars on the figure. The performance ratio between the platforms ranges from 1.16× to 1.74× for the SSD, and we check two metrics for comparison: absolute percentage error (right y axis) and absolute error (left y axis). We statically add the NAND I/O latency for each host request assuming Z-NAND[2], [5], and use the same NAND subsystem configuration described in Fig. 1.

Static Model by Amber. Fig. 2a shows that the performance estimation on Amber does not completely reflect the real performance on FTL because of its hard-coded static model. High error values come from several causes: First, the model simply counts the number of instructions statically, without reflection of dynamics on their execution. Second, for the DRAM accesses, it only counts accesses for the payload and the mapping, excluding other metadata. In addition, the model does not properly count the impact of

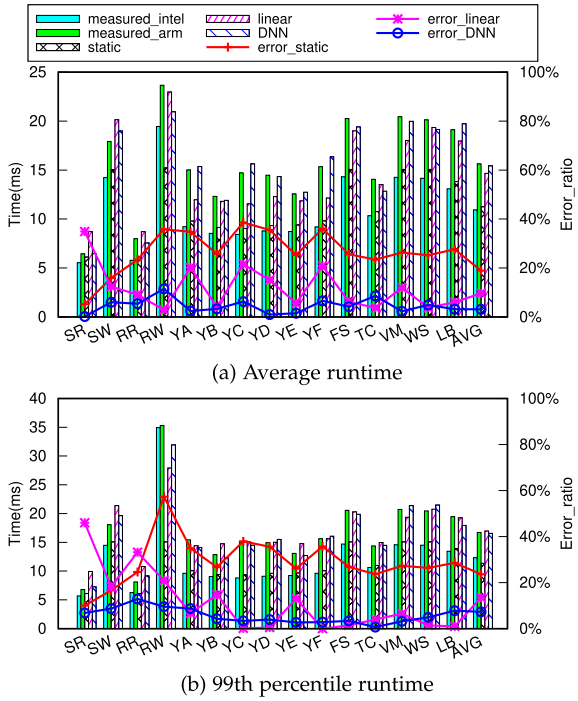


Fig. 2. *SSD performance prediction results.* The results are demonstrated for the runtime of a consecutive 1000 I/O requests.

other background tasks such as garbage collection. These problems on the static model by Amber results in high errors for most of the workloads, 18.9% on average.

Linear Regression Model. Fig. 2a demonstrates that prediction with linear regression shows relatively higher percent error values than that with the deep neural network. It shows 9.7% errors on average, slightly lower than those for the static model, but several cases show error values that exceed 30% of the measured values. Therefore, it does not provide reasonable accuracy on the prediction.

Our DNN Model. We can observe from Fig. 2a that our prediction model can achieve a substantially low percentage error, reaching under 3.1% on average. More importantly, the percent error values reach under 12% for all cases, indicating that the DNN model works well across a wide range of workloads. In three workloads, random writes, TPC-C, and Webserver, the linear model is marginally better than the DNN model.

In terms of the total absolute errors, the static model based on Amber is off by 211.1 seconds, when the total measured time on the target prototype is 827.7 seconds. The linear model performs better than the static model with a 84.1 seconds of total error. However, our DNN model achieves a very low absolute error of 53.1 seconds: 3.98× smaller than the estimation by Amber.

Fig. 2b compares the prediction results from three models for 99th percentile runtime. For our DNN, we do not change the prediction process except the criteria for choosing the best model for high accuracy on the 99th percentile runtime on the hyperparameter search. We can observe that our model also achieves a low percentage error for it, reaching under 7.3% error on average, with a maximum of 12.9%. The static and linear models show 23.6% and 13.5% errors on average, respectively. The accuracy improvement from the static model is slightly larger for 99th percentile runtime than for average runtime. This is because the static model is hard-coded and does not completely reflect the dynamics of the FTL execution.

In summary, with our building process, we find the high-accurate performance prediction model for our FTL. When the FTL code or workload set changes, the best-scenario parameters on our

TABLE 5
Hardware Performance Events Profiled on the Host

Event	Description	Coeff.
CPU_CYCLES	Clock cycles	0.25272
INSTR	Instructions	0.01207
RS_UOPS	Micro operations dispatched	0.03795
BR_INSTS	Branch instructions retired	-0.03385
BR_MISSES	Branch instructions missed	-0.09058
BUS_CYCLES	Bus cycles	0.17518
L1D_LOADS	L1 data cache loaded	0.00592
L1D_STORES	L1 data cache stored	0.34963
LLC_REFS	Last level cache referenced	0.21461
LLC_MISSES	Last level cache missed	0.35629
STALLED_CYCLES	Resource stalls	0.24247

model can change. However, with a sufficient workload set, we believe that we can find the best-accurate model again with the offline hyperparameter search process aforementioned.

5 Discussion

We verify the appropriateness of the performance events selected in [19], [20] by analyzing the correlation between the measured event values and the performance on the target platform. Table 5 enumerates our selected events related to the FTL performance with their respectively analyzed Pearson correlation coefficient values. We exclude the events that do not occur while executing the FTL, such as the number of floating-point operations. Overall, we observe that the trend of coefficient values for the FTL is different from that of general applications' performance events [24]. In particular, the coefficients related to memory and I/O are higher than those related to CPU. This implies that the FTL has different performance characteristics from other general applications: it is specialized to I/O with memory and storage access, so we cannot achieve high accuracy only with CPU-related performance features.

6 Conclusion

In this paper, we present the design of a data-driven performance model for SSD to bridge the gap in its performance between a platform for simulation and a target device. We explore both a simple linear model and a deep neural network to build a cross-platform relationship and find parameters for the most accurate model. This carefully built model forecasts the SSD performance with high accuracy when compared to a target device. In effect, our model achieves 3.1% error on average, significantly lower than 18.9% from a state-of-the-art simulator. Our work focuses on predicting the overall performance of the SSD, and has not been evaluated to predict the performance of each I/O request. We leave such fine-grained performance modeling for future work.

References

[1] F. T. Hady, A. Foong, B. Veal, and D. Williams, "Platform storage performance with 3D XPoint technology," *Proc. IEEE*, vol. 105, no. 9, pp. 1822–1833, Sep. 2017.

[2] Samsung, "Ultra-low latency with Samsung Z-NAND SSD," 2017. [Online]. Available: http://www.samsung.com/us/labs/pdfs/collateral/Samsung_Z-NAND_Technology_Brief_v5.pdf

[3] M. Jung, "OpenExpress: Fully hardware automated open research framework for future fast NVMe devices," in *Proc. USENIX Annu. Tech. Conf.*, 2020, pp. 649–656.

[4] J. Zhang, M. Kwon, M. Swift, and M. Jung, "Scalable parallel flash firmware for many-core architectures," in *Proc. USENIX Conf. File Storage Technol.*, 2020, pp. 121–136.

[5] W. Cheong *et al.*, "A flash memory controller for 15μs ultra-low-latency SSD using high-speed 3D NAND flash with 3μs read time," in *Proc. IEEE Int. Solid-State Circuits Conf.*, 2018, pp. 338–340.

[6] B. S. Kim, H. S. Yang, and S. L. Min, "AutoSSD: An autonomic SSD architecture," in *Proc. USENIX Annu. Tech. Conf.*, 2018, pp. 677–690.

[7] S. Yan *et al.*, "Tiny-tail flash: Near-perfect elimination of garbage collection tail latencies in NAND SSDs," in *Proc. USENIX Conf. File Storage Technol.*, 2017, pp. 15–28.

[8] Y. Kim, B. Tauras, A. Gupta, and B. Urgaonkar, "FlashSim: A simulator for NAND flash-based solid-state drives," in *Proc. Int. Conf. Advances Syst. Simul.*, 2009, pp. 125–131.

[9] Y. Hu, H. Jiang, D. Feng, L. Tian, H. Luo, and S. Zhang, "Performance impact and interplay of SSD parallelism through advanced commands, allocation strategy and data granularity," in *Proc. Int. Conf. Supercomput.*, 2011, pp. 96–107.

[10] M. Jung, E. H. Wilson, D. Donofrio, J. Shalf, and M. T. Kandemir, "NANDFlashSim: Intrinsic latency variation aware NAND flash memory system modeling and simulation at microarchitecture level," in *Proc. IEEE 28th Symp. Mass Storage Syst. Technol.*, 2012, pp. 1–12.

[11] M. Jung *et al.*, "NANDFlashSim: High-fidelity, microarchitecture-aware NAND flash memory simulation," *ACM Trans. Storage*, vol. 12, no. 2, pp. 1–32, 2016.

[12] J. He, S. Kannan, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau , "The unwritten contract of solid state drives," in *Proc. Eur. Conf. Comput. Syst.*, 2017, pp. 127–144.

[13] A. Tavakkol, J. Gómez-Luna, M. Sadrosadati, S. Ghose, and O. Mutlu, "MQSim: A framework for enabling realistic studies of modern multi-queue SSD devices," in *Proc. USENIX Conf. File Storage Technol.*, 2018, pp. 49–66.

[14] J. Yoo *et al.*, "VSSIM: Virtual machine based SSD simulator," in *Proc. IEEE Symp. Mass Storage Syst. Technol.*, 2013, pp. 1–14.

[15] H. Li, M. Hao, M. H. Tong, S. Sundararaman, M. Bjørling, and H. S. Gunawi, "The CASE of FEMU: Cheap, accurate, scalable and extensible flash emulator," in *Proc. USENIX Conf. File Storage Technol.*, 2018, pp. 83–90.

[16] D. Gouk *et al.*, "Amber*: Enabling precise full-system simulation with detailed modeling of all SSD resources," in *Proc. IEEE/ACM Int. Symp. Microarchit.*, 2018, pp. 469–481.

[17] N. Agrawal, V. Prabhakaran, T. Wobber, J. D. Davis, M. S. Manasse, and R. Panigrahy, "Design tradeoffs for SSD performance," in *Proc. USENIX Annu. Tech. Conf.*, 2008, pp. 57–70.

[18] M. Jung *et al.*, "SimpleSSD: Modeling solid state drives for holistic system simulation," *IEEE Comput. Archit. Lett.*, vol. 17, no. 1, pp. 37–41, Jan.–Jun. 2017.

[19] X. Zheng, L. K. John, and A. Gerstlauer, "LACross: Learning-based analytical cross-platform performance and power prediction," *Int. J. Parallel Program.*, vol. 45, no. 6, pp. 1488–1514, 2017.

[20] Y. Kim, P. Mercati, A. More, E. Shriver, and T. Rosing, "P4: Phase-based power/performance prediction of heterogeneous systems via neural networks," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des.*, 2017, pp. 683–690.

[21] C. Mendis, A. Renda, S. Amarasinghe, and M. Carbin, "Ithemal: Accurate, portable and fast basic block throughput estimation using deep neural networks," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 4505–4515.

[22] A. C. De Melo , "The new linux perf tools," in *Proc. Linux Kongress*, 2010, pp. 1–42.

[23] ARM, "ARM storage solution for SSD controllers," 2020. [Online]. Available: https://armkeil.blob.core.windows.net/developer/Files/pdf/arm-storage-solution-for-ssd-solutions-brief.pdf

[24] A. Iranfar, W. S. De Souza , M. Zapater, K. Olcoz, S. X. de Souza, and D. Atienza, "A machine learning-based framework for throughput estimation of time-varying applications in multi-core servers," in *Proc. IFIP/IEEE 27th Int. Conf. Very Large Scale Integration*, 2019, pp. 211–216.