This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/LES.2020.3007361, IEEE Embedded Systems Letters

Predicting Failures in Embedded Systems using Long Short-Term Inference

Tianyi Zhang[1], Minjun Seo[2], Bryan Donyanavard[2], Nikil Dutt[2], and Fadi Kurdahi[2]

July 2, 2020

***Abstract*— Users of embedded and cyber-physical systems expect dependable operation for an increasingly diverse set of applications and environments. Reactive self-diagnosis techniques either use unnecessarily conservative guardbands, or do not prevent catastrophic failures. In this letter we utilize machine-learning techniques to design a prediction engine in order to predict failures on-device in embedded systems. We evaluate our prediction engine's effectiveness for predicting temperature behavior on a mobile system-on-chip, and propose a realizable hardware implementation for the use-case.**

I. Introduction

The complexity of embedded system platforms and the applications they support are continuously increasing: they run large and evolving applications on heterogeneous multi- or many-core processing platforms. Examples include automated and autonomous driving, smart buildings, industry 4.0, and personal medical devices. Such systems are required to provide dependable operation for the user while dealing with a large number of internal and external variabilities, threats, and uncertainties in their lifetimes.

To provide such dependable operation, self-diagnosis techniques are developed for early detection of degradation and imminent failures, in order to maximize system life-cycle. These techniques can be combined with unsupervised platform self-adaptation to meet performance and safety targets. Self-diagnosis techniques that are reactive may (a) not be sufficient to address catastrophic failures, or (b) take overly conservative approaches that hinder performance.

For example, consider thermal management of an embedded system-on-chip (SoC). One technique is to define a temperature threshold, and throttle performance (e.g., via dynamic voltage-frequency scaling (DVFS)) when the threshold is exceeded. This approach is reactive and must act conservatively to prevent overheating. The conservative frequency throttling may degrade performance, potentially unnecessarily. If the temperature behavior could be predicted, a proactive approach could manage the temperature without sacrificing performance excessively. However, system dynamics such as temperature can behave nonlinearly, and are hard to predict without workload knowledge.

Machine learning techniques such as neural networks are useful for identifying complex system dynamics. However, neural networks are complex and difficult to deploy on power-constrained embedded systems. In this paper, we propose a failure prediction technique for embedded systems using long short-term memory (LSTM), a type of recurrent neural network (RNN). We demonstrate the effectiveness of our predictor for predicting temperature behavior with respect to a threshold on an ODROID-XU3 [9] platform, making it a candidate for mitigating overheating failures and implementing efficient control policies. We specify an implementation that is realizible in hardware on low-power embedded systems. The specific contributions are as follows:

- We propose a method for hardware hazard prediction called Long Short-Term Prediction Model.
- We propose an architecture and hardware implementation of non-intrusive prediction engine based on Long Short-Term Prediction Model to predict temperature behavior in embedded systems.
- We evaluate the predictor using measured temperature data from an ODROID XU-3.

II. Background and Related Work

When modern systems-on-chip (SoCs) operate near peak performance for extended periods, power dissipation can increase the temperature to the point that it adversely impacts chip reliability. If we can provide proactive thermal management, we can avoid potentially dangerous execution scenarios. Proaction requires prediction. A number of strategies have been proposed for on-chip thermal prediction, and the methods can be classified into two categories.

The first prediction method builds models based on measured temperature and power consumption [21], [19], [14], [15], [17]. The second method builds the prediction model indirectly using equations, without thermal measurements [5], [4], [7]. However, there have been many successful applications of machine learning techniques employed in failure detection or prediction of large-scale systems. With sufficient sensor input, machine learning models can extract complex or subtle dynamics, potentially resulting in accurate predictions when applied to new execution scenarios. Failure prediction has been proposed using support vector machines (SVMs) [10], [3], convolutional neural networks (CNNs) [16], and a combination of techniques [8].

RNNs are naturally suited for learning temporal sequences and modeling time series behaviors. RNNs have been applied to predict various behavior in large-scale systems [20], [13], [6]. In [6], the authors compare an RNN solution with an LSTM solution, and observe that LSTMs significantly outperform RNNs in terms of accuracy.

In [18], [2], [11] LSTMs are used in other domains for time series predictions such as water quality estimation, stock transaction prediction, mechanical states, and more. The authors compare LSTM networks with alternatives such as back propagation neural networks, online sequential extreme learning machines, and support vector regression machines (SVRM), and demonstrate the superiority of LSTMs.

III. CONTRIBUTIONS

We propose a method for predicting runtime behavior in hardware: the Long Short-Term Prediction Engine. In this section, we describe how our predictor is composed by walking through our use-case: predicting runtime temperature behavior on an embedded system-on-chip. Our goal is to predict temperature behavior such that critical thermal scenarios can be detected in advance and avoided, with a solution that can feasibly be integrated in an embedded SoC. Our SoC consists of four ARM A15 cores, with shared L2 cache connected via bus. We measure total power and temperature of the entire core cluster, as well as per-core utilization. To generate workloads, we use a synthetic microbenchmark [12] that is configurable. The microbenchmark is able to stress the architecture in a wide range and we generated a "general-purpose" workload by executing the microbenchmark in phases that exercised different behavior in these various dimensions. We execute different sequences on multiple cores to emulate different applications to train the model and test its performance. The prediction engine consists of two parts: a short-term binary model and a long-term regression model. The short-term binary model makes precise predictions quickly, useful for subtle changes, i.e., anticipating violations of a temperature threshold. The long-term regression model can make a prediction further in advance, useful to predict general behavior in less-critical scenarios, i.e., predicting temperature trends in a safe state.

A. Short-Term Binary Model

The short-term binary model is used to predict unwanted behavior, i.e., constraint violation. In our case in which we have a temperature threshold we do not want to violate, the short-term binary model is utilized when the measured temperature is nearing the threshold. In this scenario, a slight rise in temperature will cause a failure (violation of constraint), thereby it is important to have a high recall rate. The recall rate must be tuned carefully to balance accuracy and overhead.

1) Model Defintion

Our initial short-term binary model is defined as follows:

[1]Harbin Institute of Technology, 1162620312 at stu.hit.edu.cn
[2]Center for Embedded and Cyber-physical Systems, UC Irvine, {minjun.seo,bdonyana,dutt,kurdahi} at uci.edu
*This work was partially supported by NSF Grant CCF-1704859

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/LES.2020.3007361, IEEE Embedded Systems Letters

- Input: temperature, core utilization, power
- Output: probability of failure (after boundary limitation, the model produces a binary result: '0' refers to normal and number '1' refers to failure)

2) Model Training

Figure 1 shows measured temperature data from the ODROID-XU3 [1]. We first isolate the data above the critical point (85 °C) to use as training data. Because the range of the data is reduced, we amplify the changes of data to increase its variation. When performing amplification at runtime, we must consider constraints such as the real-time hardware implementation and the short failure intervals. We create a method called Sliding Average Amplification to efficiently preprocess data in order to increase variation and applied it on the four features. The method takes local data (5 timesteps) and uses Min-Max Normalization to amplify the values. The following equations show the calculation of Sliding Average Amplification. D(t) refers to the feature value at t and i refers to the number of timesteps defined as local data.

$$average(t)=\frac{1}{n}\sum_{i=0}^{n}D(t-i) \quad (1)$$

$$max(t)=MAX\{D(t-i),D(t-i+1),...,D(t)\} \quad (2)$$

$$min(t)=MIN\{D(t-i),D(t-i+1),...,D(t)\} \quad (3)$$

$$amplified(t)=\frac{D(t)-average(t)}{max(t)-min(t)}\times 100 \quad (4)$$

Figure 2 shows the amplified data along with the original. The orange curve is the original data and the blue curve is the amplified data.

3) Improved Loss Function

Our initial binary model still has a significant issue: it is trained with imbalanced data. Normal samples (i.e., non-critical temperatures) account for nearly 99.5 % of the training data. Due to the low ratio of failure samples (i.e., critical temperatures), the model is highly confident in identifying critical samples, which is misleading. We augment the classic binary cross-entropy loss function with weights in order to increase model sensitivity to normal samples. y is the predicted value and $\hat{y}$ is the actual value. The weight factor α is determined empirically based on the rate of failure samples in the training data.

$$Loss=-(\alpha y\log\hat{y}+(1-\alpha)(1-y)\log(1-\hat{y})) \quad (5)$$

$$\alpha=0.992 \quad (6)$$

4) Model Structure

We propose the simplest structure of an RNN prediction model that provides the required accuracy in order to minimize the hardware overhead. The LSTM internal structure is defined in the following equations. x refers to input features, h is output result, W,b are weights and bias and c are intermediate variables.

$$i_t=\sigma(W_{xi}x_t+W_{hi}h_{t-1}+b_i) \quad (7)$$

$$f_t=\sigma(W_{xf}x_t+W_{hf}h_{t-1}+b_f) \quad (8)$$

$$o_t=\sigma(W_{x0}x_t+W_{h0}h_{t-1}+b_0) \quad (9)$$

$$\tilde{c}_t=tanh(W_{xc}x_t+W_{hc}h_{t-1}+b_c) \quad (10)$$

$$c_t=f_t\odot c_{t-1}+i_t\odot\tilde{c}_t \quad (11)$$

$$h_t=o\odot tanh(c_t) \quad (12)$$

Figure 3 (black and blue) illustrates the architecture of the proposed RNN/LSTM model which contains two RNN/LSTM layers (the RNN and LSTM structure provide comparable accuracy, shown in Section IV), one fully connected layer, and one binary classification layer based on sigmoid activation. The input features are time sequences of temperature, per-core utilization, and power. After calculation of time step t in the first layer, the result is conveyed to step $t+1$ the same layer and the step t in the second layer. At the same time, step $t+1$ data is added into the next step calculation. In each RNN/LSTM layer, there are 8 time steps and 64 hidden layers. In the last time step, the result is passed to a fully-connected layer and a sigmoid layer for classification. The output result is the failure probability. When the value is greater than 0.9, we define it as failure and output 1.

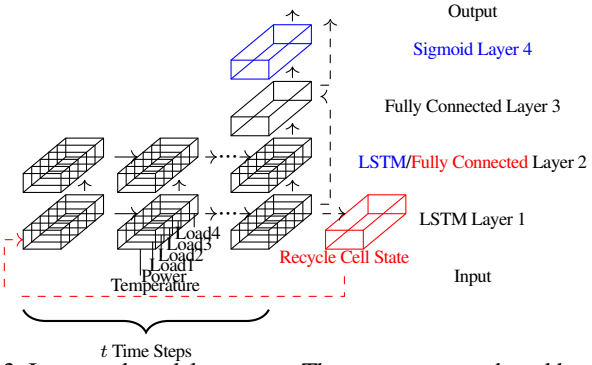


Fig. 3: Integrated model structure. The structures are shared between the short-term binary model and the long-term regression structure, depending on which is active. Functionality and structure specific to the short-term binary model is in blue, and specific to long-term regression model is red.

B. Long-Term Regression Model

The long-term regression model is used to predict behavior in the normal state. In this state, temperature varies in a large range depending on how the system is being exercised. Our goal is to predict the temperature sufficiently in advance to make runtime decisions in order to avoid critical states completely while also optimizing performance. In order to proactively avoid critical states without unnecessarily sacrificing performance, it is necessary to ensure that the prediction engine can be applied during normal execution. As the system state is non-critical, precision can be sacrificed for universality. To this end, we build a regression model for long-term prediction.

1) Model definition

- Inputs: temperature, power, per-core utilization
- Outputs: temperature

2) Model Training

In this case, we utilize a larger range of training data (60-85 °C). We observe temperature variation generally due to change in core utilization and operating frequency. We categorize training workloads as following: unicore, multicore, and shifting. We execute combinations of synthetic benchmarks to compose our workloads. The benchmarks

[1]Our use-case system, containing the described SoC.

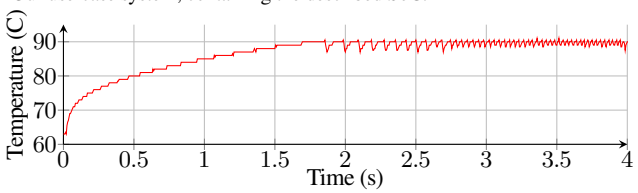


Fig. 1: Temperature data collected from the ODROID XU-3 executing a combination of synthetic microbenchmarks.

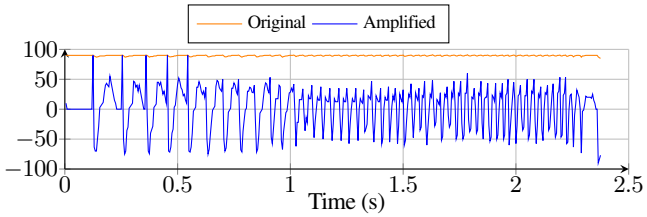


Fig. 2: Temperature data amplified using Sliding Average Amplification. We focus on data above 85 °C (critical temperature).

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/LES.2020.3007361, IEEE Embedded Systems Letters

vary in instructions-per-cycle (IPC), utilization, and cache miss rate, exercising the processor in a wide range.

For unicore workloads, we first run each benchmark on one core to emulate stable workload state. Then we combine multiple benchmarks and start them one by one to emulate changing workload state on one core. For multicore workloads, we assign different benchmarks on different cores and start them simultaneously. For shifting workloads, we assign the same benchmarks on different cores and start them at different times.

Raw data collected from the ODROID-XU3 does not initially appear stable, making filtering essential.[2] After trying several filters to smooth the raw data and considering the hardware feasibility, we conclude that the data preprocessed by recursion average filter produces the most accurate model. Filter sizes of each input are determined empirically.

3) Model Structure

LSTM has the nature of storing long-term memory, therefore, to deal with the long-term cases, we choose LSTM structure for our model. Compared to a short-term model, increased historical data is needed to ensure precision when predicting a large temperature range far in advance. This leads to increased model time step and execution time. Therefore, we apply stateful LSTM theory in the cell structure, fitting output cell state as the initial state. In this way, the structure can remember long-term memory and better adapt.

Figure 3 (black and red) illustrates the architecture of the proposed LSTM model. The input features are time sequences of temperature, per-core utilization, and power. After calculation of step t, the cell state is recycled to next term calculation. There are 8 time steps in the LSTM layer and 64 hidden layers in each cell. We need 16 previous steps for prediction, therefore the cell state will be passed for initialization every second iteration.

C. Hardware Implementation Framework

To integrate the short- and long-term models, we specify single a shared-hardware implementation that supports all of Figure 3. A judgement module receives temperature values from the sensor and decides which model to activate. If temperature is $\geq$85 °C, the short-term prediction model is activated and its weights are loaded into the model structure. If it is $<$85 °C, the long-term prediction model is activated.

To reduce structural overhead, the core LSTM and fully connected layers are partially shared, composed with the least common parameters (LSTM: 8 time steps, 64 hidden layers; fully connected: 4 hidden layers). The excess time steps can be stored in a state buffer and fed back (Figure 3).

Using the LSTM implementation of Chang et al. [1], we calculate 12960 FFs, 7201 LUTs, and 16 BRAM overhead. The LSTM hardware is 20 times faster than the Zync ZC7020 ARM-based hard-core processor (4.4 µs per inference) , 44 times more power-efficient than a software implementation with the Zync ZC7020 (performance-per-watt).

IV. Evaluation

We evaluate the effectiveness of both our Short-Term Binary Model and Long-Term Regression Model separately, using additional measured data from the ODROID-XU3. The measured data consists of the model input data measured at 5 ms intervals. We perform sensitivity analyses of LSTM/RNN models for different parameters and structures.

A. Short-Term Binary Model Evaluation

1) Evaluation Metrics

The output of the short-term binary model is a binary classification. We evaluate the model by average precision score (AP) and F1 score. The average precision score summarizes a precision-recall curve as the weighted mean of precision achieved at each recall threshold, with the increase in recall from the previous threshold used as the weight:

$$AP = \sum_n (R_n - R_{n-1}) P_n \tag{13}$$

[2]Data is stored in a userspace buffer, sampled from sensors via kernel drivers every 5ms.

where P_n and R_n are the precision and recall at the nth threshold. F1 score is a measure of a test's accuracy, and is defined as the weighted harmonic mean of the precision and recall of the test. F1 score conveys a balance between precision (P) and recall (R):

$$F1 = \frac{2 \times P \times R}{P + R} \tag{14}$$

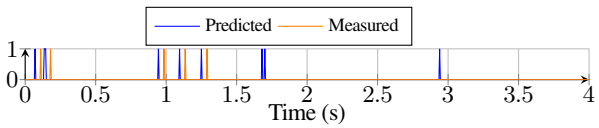


Fig. 4: Sample prediction of one workload. Binary events (i.e., experiencing critical temperature) are predicted and observed.

2) Evaluation Results

The model can predict up to 8 steps (40 ms) ahead. The F1 score is 0.43 and the AP score is 0.78. The latency of short-term binary model is 0.088 ms (Based on execution in python, no hardware acceleration). Figure 4 shows the prediction result of one dataset. The orange shows measured failures and the blue shows predicted failures. Observe that there are a number of mispredicted failures (false positives). This is preferable to false negatives (non-predicted failures), as we are trying to anticipate and potentially avoid undesirable system state. In fact, in the experiment shown in Figure 4, the recall value is 1, which means that all measured failures are predicted – i.e., we have no false negatives.

a) Model Structure Tradeoffs

To ensure practical utility of our hardware predictor in low-power embedded systems, it is important to balance precision and complexity. Considering the feasibility constraints, we explore the impact of several hyper-parameters and layer structures on the model performance. Parameters include RNN type, model structure, number of hidden neurons, decimal digits, and number of time steps. We evaluate the RNNs and LSTMs based on AP, F1, recall (performance), runtime, and degree of prediction. Figure 5 shows how different hyper-parameters effect the model performance. The left y-axes measure AP score, F1 score, and recall score. The right y-axes measure the time it takes to generate one prediction. The solid lines refer to the model with LSTM layers and the dotted lines refer to the model with RNN layers. Figure 5a shows how the number and type of layers effect performance. It indicates that LSTM has better accuracy. Prediction time increases with the number of layers. Therefore, it is best to apply 2-layer LSTM. Figure 5b shows how the number of previous timesteps effects performance. After five timesteps, the accuracy plateaus and prediction time increases, therefore using five timesteps is the best choice. Figure 5c shows how the number of neurons effects performance. Accuracy pleateus beyond 32 neurons, thus we choose 32 neurons in the network. Figure 5d shows how the decimal digit influences performance. Two digits is the minimum number to maintain accuracy. Figure 5e shows how accuracy degrades as the prediction moves further in advance.

B. Long-Term Regression Model

For the regression model, we use mean absolute error (MAE) to evaluate the accuracy, where y_i is the predicted temperature k steps in advance (P_i), and $\hat{y}_i$ is the measured temperature at step $i+k$ (M_{i+k}):

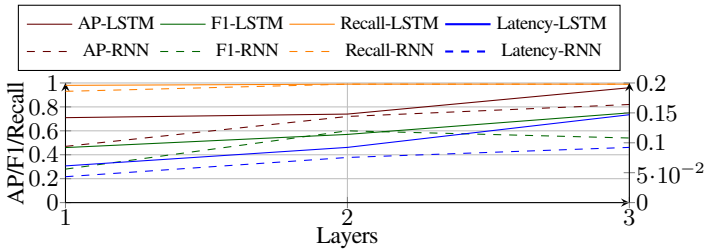
$$MAE = \frac{1}{n} \sum_{i=1}^{n} |P_i - M_{i+k}| \tag{15}$$

Figure 6 shows a sample time plot of one one experiment. The orange dashed line shows the measured temperature 64 steps (320 ms) in advance. The latency of the long-term regression model is 0.108 ms (no hardware acceleration). The blue is the predicted temperature in realtime. The MAE achieved by the predictor for 320 ms in advance is 0.018. The highest accuracy achieved by existing prediction methods is 0.024 MAE [17], and the longest prediction step is 500 ms [4], which we improve by 25% and 36% respectively.

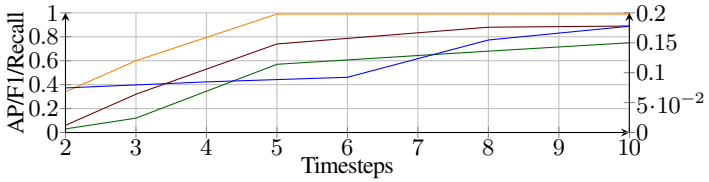
This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/LES.2020.3007361, IEEE Embedded Systems Letters

V. Conclusion

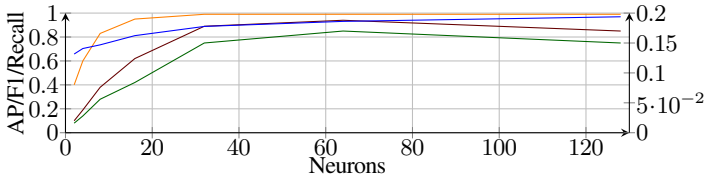
We propose a new LSTM-based method for hardware hazard prediction called Long Short Term Prediction Engine. The prediction engine uses two models to provide prediction of both urgent and normal conditions, which have different prediction requirements. The integrated model is trained and tested on data collected on the ODROID-XU3 platform. The short term model makes precise binary prediction near critical conditions 40 ms in advance, and reaches 0.78 average precision score. The long term model outputs temperature values up to 320 ms in advance with a MAE of 0.018. We simplify the structure of the network and hyper-parameters to find one suited for hardware realization, sharing parts of the network and automatically switching between the two models according to temperature.



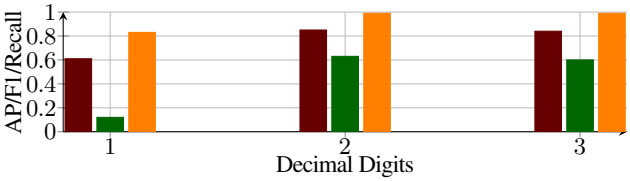
(a) Comparison for number of network layers.



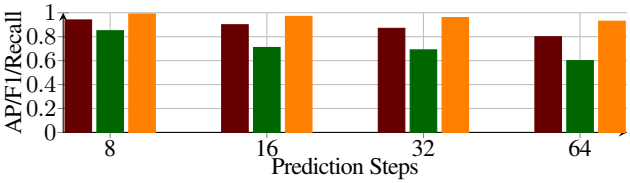
(b) Comparison for number of time steps considered in the network.



(c) Comparison for number of neurons.



(d) Comparison for number of decimal places (precision) used in the model.



(e) Comparison for various degrees of prediction (i.e., how many steps in advance. One time step in our case is 5 ms.

Fig. 5: Sensitivity analysis of model structure.

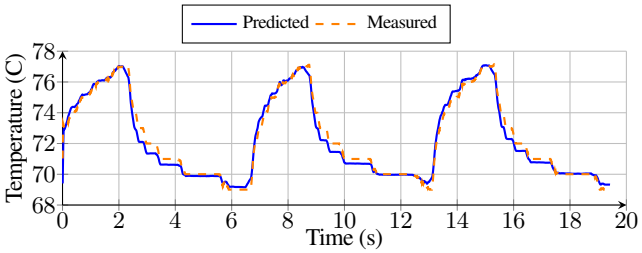


Fig. 6: LSTM prediction accuracy for 64-step (320 ms) prediction, compared to measured behavior.

[1] A. X. M. Chang, B. Martini, and E. Culurciello, "Recurrent neural networks hardware implementation on fpga," *arXiv preprint arXiv:1511.05552*, 2015.

[2] Z. Chen, Y. Liu, and S. Liu, "Mechanical state prediction based on lstm neural netwok," in *Chinese Control Conference*, 2017.

[3] A. Chigurupati, R. Thibaux, and N. Lassar, "Predicting hardware failure using machine learning," in *Reliability and Maintainability Symposium*, 2016.

[4] R. Cochran and S. Reda, "Consistent runtime thermal prediction and control through workload phase detection," in *ACM/IEEE Design Automation Conference*, 2010.

[5] A. K. Coskun, T. S. Rosing, and K. C. Gross, "Utilizing predictors for efficient thermal management in multiprocessor socs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2009.

[6] F. D. d. S. Lima, G. M. R. Amaral, L. G. d. M. Leite, J. P. P. Gomes, and J. d. C. Machado, "Predicting failures in hard drives with lstm networks," in *Brazilian Conference on Intelligent Systems*, 2017.

[7] Y. Ge, Q. Qiu, and Q. Wu, "A multi-agent framework for thermal aware task migration in many-core systems," *IEEE Transactions on Very Large Scale Integration Systems*, 2012.

[8] I. Giurgiu, J. Szabo, D. Wiesmann, and J. Bird, "Predicting dram reliability in the field with machine learning," in *ACM/IFIP/USENIX Middleware Conference: Industrial Track*, 2017.

[9] Hardkernel, "ODROID-XU," Tech. Rep. [Online]. Available: http://www.hardkernel.com/main/main.php

[10] R. Kumar, S. Vijayakumar, and S. A. Ahamed, "A pragmatic approach to predict hardware failures in storage systems using mpp database and big data technologies," in *IEEE International Advance Computing Conference*, 2014.

[11] S. Liu, G. Liao, and Y. Ding, "Stock transaction prediction modeling and analysis based on lstm," in *IEEE Conference on Industrial Electronics and Applications*, 2018.

[12] T. Mück, S. Sarma, and N. Dutt, "Run-dmc: Runtime dynamic heterogeneous multicore performance and power estimation for energy efficiency," in *International Conference on Hardware/Software Codesign and System Synthesis*, 2015.

[13] S. Huang, C. Fung, K. Wang, P. Pei, Z. Luan, and D. Qian, "Using recurrent neural networks toward black-box system anomaly prediction," in *IEEE/ACM International Symposium on Quality of Service*, 2016.

[14] S. Sharifi, D. Krishnaswamy, and T. S. Rosing, "Prometheus: A proactive method for thermal management of heterogeneous mpsocs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2013.

[15] G. Singla, G. Kaur, A. K. Unver, and U. Y. Ogras, "Predictive dynamic thermal and power management for heterogeneous mobile platforms," in *Design, Automation Test in Europe Conference Exhibition*, 2015.

[16] X. Sun, K. Chakrabarty, R. Huang, Y. Chen, B. Zhao, H. Cao, Y. Han, X. Liang, and L. Jiang, "System-level hardware failure prediction using deep learning," in *ACM/IEEE Design Automation Conference*, 2019.

[17] E. W. Wächter, C. de Bellefroid, K. R. Basireddy, A. K. Singh, B. M. Al-Hashimi, and G. Merrett, "Predictive thermal management for energy-efficient execution of concurrent applications on heterogeneous multi-cores," *IEEE Transactions on Very Large Scale Integration Systems*, 2019.

[18] Y. Wang, J. Zhou, K. Chen, Y. Wang, and L. Liu, "Water quality prediction method based on lstm neural network," in *International Conference on Intelligent Systems and Knowledge Engineering*, 2017.

[19] B. Wojciechowski and J. Biernat, "Temperature prediction for multi-core microprocessors with application to dynamic thermal management," in *International Workshop on THERMal INvestigation of ICs and Systems*, 2012.

[20] C. Xu, G. Wang, X. Liu, D. Guo, and T. Liu, "Health status assessment and failure prediction for hard drives with recurrent neural networks," *IEEE Transactions on Computers*, 2016.

[21] I. Yeo, C. C. Liu, and E. J. Kim, "Predictive dynamic thermal management for multicore systems," in *ACM/IEEE Design Automation Conference*, 2008.