Dynamic Voltage and Frequency Scaling in NoCs with Supervised and Reinforcement Learning Techniques

Quintin Fettes, *Student Member, IEEE*, Mark Clark, *Student Member, IEEE*, Razvan Bunescu, *Member, IEEE*, Avinash Karanth, *Senior Member, IEEE*, and Ahmed Louri, *Fellow, IEEE*

Abstract—Network-on-Chips (NoCs) are the de facto choice for designing the interconnect fabric in multicore chips due to their regularity, efficiency, simplicity, and scalability. However, NoC suffers from excessive static power and dynamic energy due to transistor leakage current and data movement between the cores and caches. Power consumption issues are only exacerbated by ever decreasing technology sizes. Dynamic Voltage and Frequency Scaling (DVFS) is one technique that seeks to reduce dynamic energy; however this often occurs at the expense of performance. In this paper, we propose *LEAD Learning-enabled Energy-Aware Dynamic voltage/frequency* scaling for multicore architectures using both supervised learning and reinforcement learning approaches. LEAD groups the router and its outgoing links into the same V/F domain and implements proactive DVFS mode management strategies that rely on offline trained machine learning models in order to provide optimal V/F mode selection between different voltage/frequency pairs. We present three supervised learning versions of LEAD that are based on buffer utilization, change in buffer utilization and change in energy/throughput, which allow proactive mode selection based on accurate prediction of future network parameters. We then describe a reinforcement learning approach to LEAD that optimizes the DVFS mode selection directly, obviating the need for label and threshold engineering. Simulation results using PARSEC and Splash-2 benchmarks on a 4×4 concentrated mesh architecture show that by using supervised learning LEAD can achieve an average dynamic energy savings of 15.4 percent for a loss in throughput of 0.8 percent with no significant impact on latency. When reinforcement learning is used, LEAD increases average dynamic energy savings to 20.3 percent at the cost of a 1.5 percent decrease in throughput and a 1.7 percent increase in latency. Overall, the more flexible reinforcement learning approach enables learning an optimal behavior for a wider range of load environments under any desired energy versus throughput tradeoff.

Index Terms—Dynamic voltage and frequency scaling (DVFS), machine learning (ML), ridge regression, reinforcement learning

1 Introduction and Motivation

The immense number of transistors that are packed onto multicore chips as transistor feature size continues to shrink into the sub-nanometer region has caused many new and unique power problems to emerge. With the increase in the number of cores, the underlying communication fabric called the Network-on-Chip (NoC) has become critical for data communication between the cores and the memory hierarchy. Two critical power problems facing Network-on-Chips are high static power and dynamic energy. Excess dynamic energy is the result of storing and switching data within routers and links. NoC size increases proportionally with core count to accommodate the increased data communication demands, only further exacerbating the high dynamic energy cost of the NoC. Dynamic Voltage and Frequency Scaling (DVFS) is the focus of much prior research and may be used to reduce dynamic energy. Excessive static power is a result of transistor leakage current and will only increase as greater numbers of transistors are packed onto chips. Power-gating is one useful and well researched method to reduce static power, however new challenges arise when applying this method to the NoC such as high wakeup delay, long break even times, and router blocking [16], [17], [18], [19], [20], [21], [22].

A vast amount of research exists on DVFS with the main goal of reducing dynamic energy at runtime while meeting strict performance criteria [1], [2], [3], [4], [5], [6], [26], [27], [28], [29], [34], [35], [36], [37]. Although static power continues to rise as transistor size shrinks, it is often not considered for these designs because static power is not affected by changes in clock frequency. However, it should be noted that in multi-supply voltage designs it is a change in supply voltage that leads to a subsequent change in clock frequency; therefore static power can be impacted by an increase/decrease in clock frequency. An optimal DVFS algorithm should operate at the lowest supply voltage allowable without causing significant performance degradation. This may

- *Q. Fettes, M. Clark, R. Bunescu, and A. Karanth are with the Department of Electrical Engineering and Computer Science, Ohio University, Athens, OH 45701. E-mail: {qf731413, mc591611, bunescu, kodi}@ohio.edu.*
- *A. Louri is with the Department of Electrical and Computer Engineering, George Washington University, Washington, DC 20052. E-mail: louri@gwu.edu.*

Manuscript received 21 June 2018; revised 18 Sept. 2018; accepted 20 Sept. 2018. Date of publication 10 Oct. 2018; date of current version 19 Feb. 2019. (Corresponding author: Avinash Karanth.)
Recommended for acceptance by F. Lamberti.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TC.2018.2875476

be accomplished by measuring various network metrics and determining which metric is the best at capturing network traffic. These metrics vary widely, some have used Voltage Frequency Island (VFI) utilization [5] and buffer utilization [2], others have used round-trip time (RTT) [4] or newer metrics such as network slack [6].

Accurately capturing network traffic is undoubtedly an important part of any DVFS scheme; similarly, it is hard to dispute the impact of the mode selection logic. V/F mode selection logic determines when to increase/decrease the supply voltage and is a critical design parameter. Increasing the supply voltage during times of low network utilization will consume excess power whereas decreasing the supply voltage during times of high network utilization will affect the throughput of the network. Decisions that affect how to tune the voltage/frequency can significantly affect the performance of the NoC as well as applications running on multicores. While older DVFS schemes relied on data-driven reactive mode selection, newer designs have begun to incorporate proactive techniques based on machine learning. Reactive techniques change the frequency or voltage after observing a relevant event, such as a change in buffer utilization. In contrast, proactive techniques based on machine learning determine the appropriate mode by predicting the future buffer utilization (supervised learning) or by maximizing an expected long term reward that trades off energy and throughput (reinforcement learning).

In this paper, we present **L**earning-enabled **E**nergy-**A**ware **D**ynamic voltage/frequency scaling (LEAD), a collection of DVFS techniques based on linear regression and reinforcement learning (RL) models that are trained offline. The proactive DVFS mode management strategies based on linear regression were originally introduced in our work from [45]. In that supervised learning setting, a linear regression model uses the buffer utilization, the change in buffer utilization and the change in energy/throughput in the current time window to estimate the value of network parameters for the next time window. A DVFS mode is then selected to be used in the next time window by comparing the predicted parameter against a set of tuned thresholds. In this work, we propose reinforcement learning with Deep Q Networks to directly optimize the desired energy/performance trade-off. RL algorithms are designed for optimizing control tasks; as such they are a natural fit for DVFS, which is by definition a control problem.

When using supervised learning, we had to identify or engineer a network parameter to use as a prediction target, such as buffer utilization in LEAD-τ; DVFS decisions were then made solely based on the predicted parameter value. However, this is likely to be sub-optimal, as making the best DVFS decision depends on more than just buffer utilization, which is empirically confirmed by the superior results achieved by the RL version of LEAD. In addition, the supervised LEAD models require setting multiple thresholds to achieve a desired energy/throughput tradeoff. These thresholds were tuned via a brute force grid search procedure, which is time consuming and not scalable. In contrast, the RL approach can more easily combine metrics and control the tradeoff of throughput versus energy via a single hyper-parameter in the reward function. Therefore, with an effective reward and an appropriate RL model, the effort required from a system designer is significantly reduced.

The main contributions of this work are as follows:

1) *Supervised Learning:* We trained linear regression models to predict future network parameters. V/F modes for the next time window are then chosen proactively based on the predicted network parameters. LEAD-τ predicts future input buffer utilization and selects future modes accordingly. LEAD-Δ predicts future change in input buffer utilization and increases/decreases voltage level accordingly. LEAD-G incorporates energy and throughput directly into the algorithm in the attempt to use explorative logic to find the mode that minimizes energy consumption.
2) *Reinforcement Learning:* LEAD-RL uses recent state-of-the-art RL techniques such as Deep Q Networks (DQNs) and variants of DQNs to select the V/F mode that maximizes an expected long term reward capturing a desired energy versus throughput tradeoff. This creates a model that prioritizes actions leading to maximal energy savings with minimal throughput loss by considering all possible voltage level selections for the next time window.
3) *Performance Evaluation:* For a 4×4 concentrated mesh architecture, our simulation results show that the supervised LEAD-τ achieves an average dynamic energy savings of 15.4 percent for a loss in throughput of only 0.8 percent with no significant impact on latency. In the reinforcement learning setting, LEAD-RL can achieve 20.3 percent dynamic energy savings at the cost of 1.5 percent less throughput and 1.7 percent increase in latency. We further perform sensitivity studies on the main hyper-parameters used by the RL model to evaluate their impact on the energy-efficiency and performance of NoCs.

2 Related Work

Prior research has focused on applying DVFS to individual on-chip components such as the processor, caches, memory or to the links and routers of the NoC. Voltage scaling has also been applied at various granularities: a coarse grained scheme might scale all routers at the same time while a fine grained scheme would scale each router individually. The trade-offs between various levels of granularity come in terms of design complexity, area overhead, and maximal energy savings [6]. Fine-grained DVFS schemes have greater potential for energy savings when applied to multi-core processors, however there is concern that the overhead associated with providing separate voltage domains for each router/link would offset any potential savings. This is because the power delivery network must be split N ways corresponding to the number of voltage domains which results in N times higher resistance. However this extra overhead is largely dependent on the voltage drop of inefficient on-chip voltage regulators. Newer voltage regulator frameworks have alleviated this concern using a hierarchy of on-chip and off-chip voltage regulators and many modern approaches can achieve 90 percent energy efficiency or greater [53].

Accurately capturing network bandwidth requirement is another key aspect to any DVFS design. Prior research has used many different performance metrics, such as VFI utilization [5], buffer utilization [2], round-trip-time [4], cache

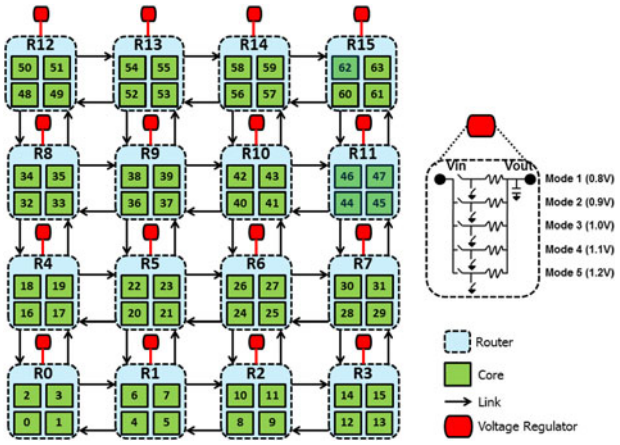


Fig. 1. *Network Topology:* LEAD is built on a concentrated mesh with 16 routers, 48 unidirectional links, and 64 cores. On-chip voltage regulators allow router level DVFS granularity with five separate V/F pairs ranging between 0.8 and 1.2 V. A simple VR schematic is shown in the legend.

coherence properties [26], and network slack [6]. There is also research that focuses solely on the impact of the mode selection logic. This research compares the energy/performance trade-offs associated with using various logical models or algorithms to determine when to increase/decrease supply voltage [5]. One technique to ensure a improved energy/performance trade-off when applying DVFS to the NoC is to scale the router and its' links together. Links that operate at higher voltage levels than their associated routers may consume excess energy in an idle network. On the other hand, links that operate at lower voltage levels may not meet network bandwidth requirements, thus leading to performance degradation.

RL has been used in [54] and [9] for the optimization of the energy versus throughput trade-off. In [54], RL was applied to NoC, LLC, and other uncore components, whereas our work only looks at energy savings in the NoC. In [9], RL was applied to multi-tasking systems in order to achieve better voltage and frequency settings for DVFS of the core. In section 8 we present a detailed comparison with the results from these papers. RL has also been applied to improve performance metrics that are different than those used in this paper. We have seen low-overhead RL applied to multi-processor systems were temperature, performance, and power were balanced to save energy while meeting stringent performance requirements [12]. Other research with many-core processors seeks to reduce high power density with RL based task allocation using core and router temperature predictions [52]. Some research has even focused on applying reinforcement learning to the voltage regulator hierarchy to enable more energy efficient voltage switching [53]. There has also been considerable work with reinforcement learning that does not pertain to energy management or DVFS. One such work seeks to reduce packet latency by using RL based NoC arbitration of shared resources such as cores, caches, and memory instead of traditional round robin approaches [50]. Another work uses online reinforcement learning to enable Q-routing, which can lower packet delivery times compared to traditional non-adaptive shortest path routing algorithms [51].

Previous work has shown that linear regression and reinforcement learning can be applied to the CPU or other on-chip components. Our design will apply linear regression and RL specifically to the NoC in order to achieve optimal dynamic energy savings while meeting strict performance requirements. The proposed LEAD models are trained offline, greatly reducing the overhead traditionally associated with online algorithms, especially in the RL setting. LEAD also applies DVFS to both the router and its outgoing links ensuring that we meet bandwidth requirements at times of high network traffic while still allowing dynamic energy savings at times of low network traffic.

3 LEAD Architecture

This section will discuss our proposed LEAD router microarchitecture, the network topology, linear regression and RL DVFS implementation.

3.1 Router Microarchitecture

LEAD is built upon a concentrated mesh topology that uses on-chip voltage regulators. This network has a concentration factor of 4 and consists of 16 routers, 64 cores, and 48 unidirectional links. Each router and its' outgoing links are scaled together allowing energy efficient per router DVFS. The network topology is shown in Fig. 1. Because LEAD is built upon a concentrated mesh, routers consist of 8 input ports and 8 output ports with 4 virtual channels per port. Cores have individual L1 caches while the L2 cache is shared among concentrated cores. After a packet is generated it sits in the input buffer where it waits for the route to be computed using XY dimension order routing (DOR) during the router computation (RC) stage of the router pipeline. After the route has been computed, a virtual channel is allocated and the head flit competes for the output channel in the switch allocation (SA) stage. Once the head flit has been awarded an output channel, it moves across the crossbar to the destination port where it waits for the rest of the body flits and the tail flit during switch traversal (ST). The router microarchitectures vary slightly between the linear regression models and the RL models in that they require the addition of several similar units. The router microarchitecture to enable linear regression based models LEAD-τ, LEAD-Δ, and LEAD-G are shown in Fig. 2. While the router microarchitecture for LEAD-RL is the same, it requires one less component and the Label component behaves slightly different. This will be explained in greater detail in the following section.

3.2 DVFS Implementation

All LEAD models are built on a simple voltage regulator scheme that allows per router DVFS and the selection of five different voltage levels. LEAD-τ, LEAD-Δ, and LEAD-G routers contain the addition of four new units that enable linear regression based proactive mode selection. The first unit is called Feature Extract and its function is to gather local router information and supply it to the next unit. The second unit, Non-ML Model can select modes directly using gathered information such as buffer utilization or link utilization and is only necessary for gathering training data. This unit is not used during testing. The third unit is Label and it multiplies each gathered feature by a pre-trained weight and sums the results to generate a label. The final

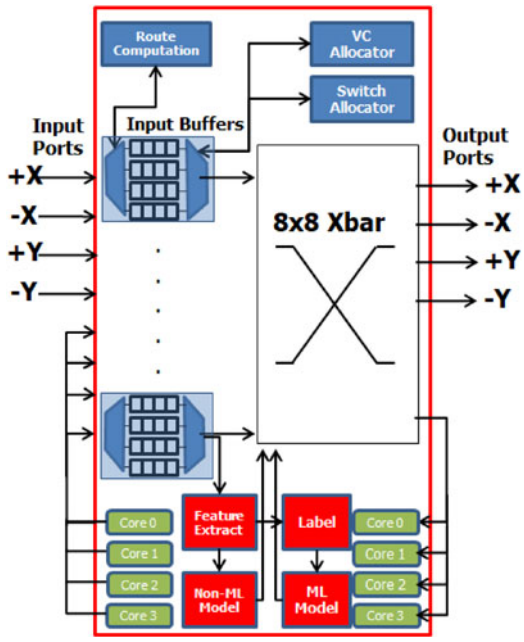


Fig. 2. *Router Microarchitecture:* The router architecture as well as the four additional units required to enable linear regression or RL based model selection; Feature Extract, Label, Non-ML Model, and ML Model. For Linear Regression, all four units are required but for RL only Feature Extract, Label, and ML Model are necessary and Label behaves slightly differently.

unit is ML Model. This unit selects an appropriate voltage level based on the value of the generated label.

LEAD-RL routers use many of the same components with slightly different functionality. LEAD-RL routers contain the addition of three units enabling RL based proactive mode selection. The first unit is the same as linear regression's first unit, Feature Extract. This unit gathers local router features and supplies them to the next unit. The second unit, Label, now implements five weighted sums of the gathered features through a series of parallel additions and multiplications, one sum per each Label value. Then a comparator is used to select the label value corresponding to the largest weighted sum. The third and final unit, ML Model, uses the highest scoring label to select the appropriate voltage level for the router and its outgoing links.

3.2.1 Operating Modes

We assume a simple voltage regulator design that allows for five voltage/frequency pairs (modes) similar to those proposed in our own prior work as well as other research [26]. The supply voltage changes in small 100 mV steps with proportional changes in clock frequency. These five V/F pairs include {0.8 V/1, 0.9 V/1.5, 1.0 V/1.8, 1.1 V/2 and 1.2 V/2.25 GHz}. The voltage regulator setup is shown in Fig. 1 where each voltage regulator can be switched into one of the five modes of operation on a per router granularity. We could have increased the number of modes, but this leads to increased design complexity and overhead with no guarantee of increased energy savings.

3.2.2 DVFS Models

This work focuses on measuring the impact that RL-based mode selection can bring on dynamic energy and performance in NoC for multi-core designs. We propose LEAD-RL (Fig. 4), a reinforcement learning model, and compare it against three linear regression based models described in prior work: LEAD-τ, LEAD-Δ, and LEAD-G (Fig. 3).

Baseline. The baseline model does not apply DVFS to the network; instead it operates all routers in the highest mode of and acts an upper bound on performance and dynamic energy consumed by the NoC.

LEAD-τ. LEAD-τ starts by initializing all routers to operate at the lowest mode of operation. Feature values are gathered every epoch and a label is generated. The label (predicted future input buffer utilization) is used to proactively select modes by comparing against a theoretical maximum. If the predicted future input buffer utilization is less than 5 percent of the theoretical maximum, then mode 1 is selected; between 5 and 10 percent, then mode 2 is selected; between 10 and 20 percent, then mode 3 is selected; between 20 and 25 percent, then mode 4 is selected; greater than 25 percent, then mode 5 is selected. This model emphasizes the importance of optimal mode selection and allowing the model to transition from any mode directly into the most optimal mode, which other designs do not allow, with the goal of maintaining as much performance as possible in relation to the baseline while still enabling dynamic energy savings.

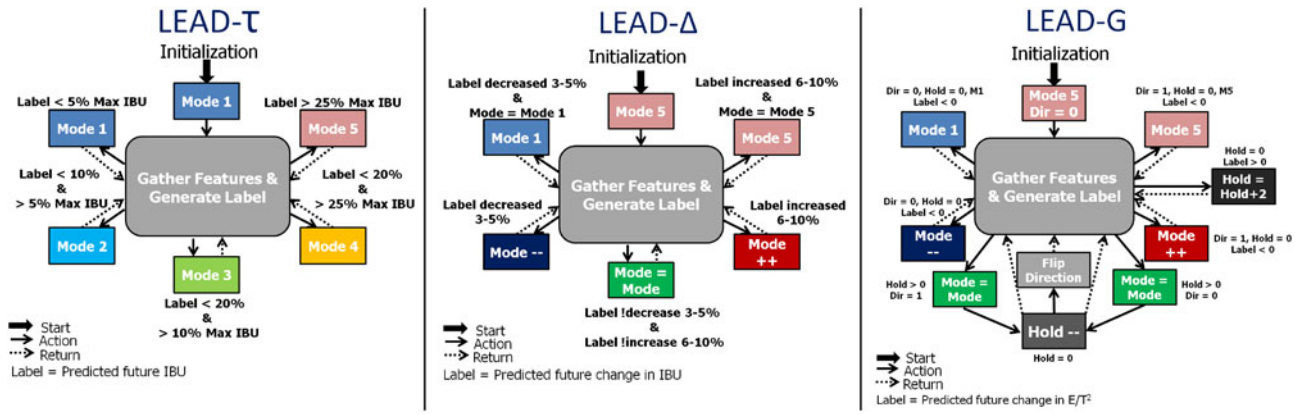


Fig. 3. *DVFS Models:* LEAD-τ predicts future input buffer utilization as a label and compares it to a theoretical maximum to select the optimal V/F mode. LEAD-Δ predicts the change in input buffer utilization between the current and future time window, whereas LEAD-G predicts the change in $\frac{energy}{throughput^2}$.

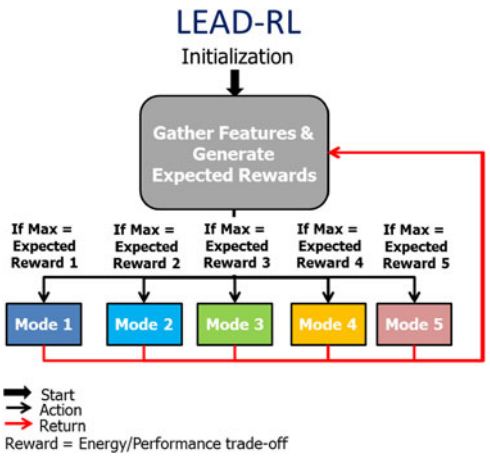


Fig. 4. *RL Model:* LEAD-RL generates long term expected rewards for each mode and selects the mode that is predicted to lead to the greatest long term reward over the next time window.

LEAD-Δ. LEAD-Δ starts by initializing all routers to operate at the highest mode of operation. At each epoch the router transitions one voltage level up or down based on the label (predicted future change in buffer utilization). If the buffers are predicted to decrease by at least 3-5 percent, then the router decreases voltage level. If the buffers are predicted to increase by at least 6-10 percent, then the router increases voltage level. We ensure that the model puts greater emphasis on energy savings than performance by making the transition thresholds to move up greater than the threshold to move down. This model is designed to exploit gradually changing traffic patterns where adjacent mode transitions are optimal.

LEAD-G. LEAD-G [5], [28] seeks to find the mode that minimizes predicted future $\frac{energy}{throughput^2}$. This model adds explorative logic and directly introduces dynamic energy and throughput into the label. LEAD-G starts by initializing all routers to the highest mode of operation with downwards exploration. Next the label (predicted change in $\frac{energy}{throughput^2}$) is calculated; if the label is negative, then the router transitions one more adjacent mode in the current explorative direction as this is predicted to result in greater energy savings for the next epoch. If the label is positive the router is put into a hold phase as the router has moved in a direction with less energy savings than the previous epoch. During the hold phase the router may not change voltage levels, we keep the hold phase at 2 epochs as was proposed in [5]. Once the hold phase expires, the explorative direction is flipped and the model explores in the opposite direction. This model is used strictly for comparative purposes and highlights a model that prioritizes energy savings above performance loss. Because this model may only transition into adjacent modes, it rarely operates in the mode with the optimal $\frac{energy}{throughput^2}$ value.

LEAD-RL: LEAD-RL utilizes Deep Reinforcement Learning techniques to determine an optimal mode selection action at every time window. During training, the agent observes transitions, the current state, the action taken, the reward received, and the resulting state. The goal of the agent is to maximize a long term expected reward that trades off dynamic energy versus throughput. By measuring the difference between the reward observed at every state and the reward actually received, the agent trains a neural network to closely approximate an action-value function defined as the long term expected reward for taking each action in a given state. LEAD-RL utilizes a set of features to represent the NoC state, and the trained action-value function to select the action that maximizes the long term expected reward at every time window. The logic behind LEAD-RL is further explained in Fig. 4.

4 Linear Regression Models

In this section, we briefly present linear regression and explain how it is used to train the supervised learning LEAD models. We also describe the feature set and discuss the labels predicted by each LEAD model.

Ridge regression is used to train each of the three supervised LEAD models, which refers to a version of linear regression that uses L2 regularization during training. The same feature set is used in all LEAD models, with only the predicted label being different. A training example is represented as a feature vector $\mathbf{x}_n$ and target label t_n. The algorithm learns a vector of weights $\mathbf{w} = [w_1, w_2, \ldots, w_K]$ that when multiplied with the features in $\mathbf{x}_n$ results in a system prediction $y(\mathbf{x}_n, \mathbf{w}) = \mathbf{w}^T\mathbf{x}_n$ that should be close to the target label t_n. The weights are scalar values that represent the significance of each feature with respect to the label. If a weight is zero, then the corresponding feature has no impact on calculating the label and that feature may be removed. The ridge regression equation used by our linear regression is shown below:

$$E(w) = \frac{1}{2}\sum_{n=1}^{N}\{y(\mathbf{x}_n, \mathbf{w}) - t_n\}^2 + \frac{\lambda}{2}\sum_{j=1}^{M} w_j^2. \quad (1)$$

In the above equation, the sum of squared errors between the predicted label $y(\mathbf{x}_n, \mathbf{w})$ and the actual label t_n is minimized. We train the model weight vector $\mathbf{w}$ offline using the vectors $x_n, n = 1, N$ of gathered features and their labels t_n. We apply L2 regularization to the trained model in order to minimize model complexity and alleviate over-fitting. We tune the regularization hyper-parameter λ by trying multiple values until the best fitting solution is found on the validation part of the data. After the model has been trained, the weights are exported and used by the network simulator during testing. All LEAD models are trained on 6 different traces, tuned on 3, and tested on the remaining 5.

An example $\mathbf{x}_n$ is represented as a vector of $K = 39$ features that capture relevant network parameters such as current input buffer utilization, link utilization, and number of requests sent/received. Features are extracted every time window using reactive versions of each model that rely on the current network parameter values to select V/F modes. The size of the feature set is kept small to avoid unnecessary computational overhead. All features are local router parameters and do not need global coordination. These features are further described in [45]. LEAD trains/validates various models using supervised learning. This means that the feature set as well as a corresponding target label are supplied during training and tuning. The label varies based on the LEAD model. For LEAD-τ, the label is the future input buffer utilization of the router for the next time window.

For LEAD-Δ the label is the difference between the routers' current and future input buffer utilization. For LEAD-G the label is the difference between the routers' current and future $\frac{energy}{throughput^2}$.

5 Reinforcement Learning Background

In this section, we introduce reinforcement learning and a number of modern techniques that we used in our LEAD-RL models to optimize their performance.

RL is a subset of Machine Learning (ML) that is concerned with problems in which no explicit label is provided during training. This is different from supervised learning, the method by which other LEAD models are trained, where labels are supplied during training. In LEAD-RL, the agent instead aims to maximize a cumulative reward in order to find an optimal policy for selecting actions, where the reward is a scalar value describing progress toward a goal after each action. The prevailing setting for RL is the Markov Decision Process (MDP); an MDP can be represented as a tuple, (S, A, P, γ). S is the set of all states s, A is the set of all possible actions a the agent can take, R is the reward function, $P(s', r|s, a)$ is the probabilistic transition function/dynamics model of the environment describing the probability of transitioning into a new state s' and observing a reward r when action a is taken from state s, and γ is the discount factor. The agent aims to learn an optimal policy $\pi^* : S \rightarrow A$ that maps states to actions such that the long term expected reward is maximized. We can model this problem as an agent which learns the optimal action-value function $Q^*(s, a)$ defined as

$$Q^*(s,a) = \max_{\pi} \mathbb{E}_P\big[r_t + \gamma r_{t_1} + \gamma^2 r_{t+2} + \cdots | s, a \sim \pi\big], \quad (2)$$

which is the expected sum of rewards r_t, discounted at each timestep t by a positive discounting factor $\gamma \leq 1$, when actions are taken according to policy π, maximized over all possible policies [49]. It can be shown that the optimal policy is $\pi^* = \arg\max_a Q^*(s, a)$, where the optimal Q^* satisfies the Bellman optimality equation [49]

$$Q^*(s,a) = \sum_{s',r} P(s', r|s, a)\Big(r + \max_{a'} Q_*(s', a')\Big). \quad (3)$$

When the state space is discrete, the tabular Q-learning algorithm and its numerous variants can be used to find the optimal Q-value function $Q^*(s, a)$ using a state-action table for storing the Q values. At each timestep, the Q-learning algorithm chooses actions based on the current Q, and updates Q using the observed reward according to a temporal difference rule [49]. The actions need to be chosen such that, over many timesteps, all actions are taken in all states. This poses a central conflict between exploration and exploitation. At any timestep, there is at least one action whose estimated value is the greatest; we call the greedy selection of this action exploiting; if instead one of the non-greedy actions is selected, we say the agent is exploring [38]. Exploitation will allow the agent to maximize its return in the short term; however, more exploration may allow the agent to better estimate the true value function and lead to a greater long term return[38]. The most common approaches to addressing the tradeoff between exploration and exploitation are ϵ-greedy methods. At every timestep t, the agent selects the greedy action $a_t = \arg\max_a Q(s, a)$ with probability $1 - \epsilon$. With probability ϵ the agent selects a random action instead.

5.1 Deep Q Networks with Experience Replay

In many cases the state space, the action space, or both, are too large to create a state-action Q table. One solution is to represent the states as a vector of possibly continuous features and use function approximation techniques to estimate the value of each state. However RL algorithms are known to be unstable when the action-value function $Q(s, a; \theta)$ is computed by some parameterized function approximator, such as a Neural Network [39]. The primary causes of this instability were found to be the sample correlation of sequential observations, and the correlation between the current approximate action-value $Q(s, a; \theta)$ and the target action value $r_t + Q(s', a'; \theta)$. Deep Q Networks (DQN) address these issues via *target networks* and *experience replay* [39].

Target networks attempt to alleviate the correlation between the approximate action value under the current policy and the target action value by calculating the target according to a different set of network parameters θ^-, which are updated only periodically. By doing this, the network training slows down somewhat, and training instability due to bootstrapping is alleviated [39].

Experience replay is a mechanism by which state transitions $e_t = (s_t, a_t, r_t, s_{t+1})$ are stored in a replay buffer, $D_t = (e_1, e_2, \ldots, e_t)$, of finite size. During training, gradient descent updates are performed on minibatches of samples $(s_t, a_t, r_t, s_{t+1}) \sim U(D)$, which are drawn uniformly at random from the replay buffer. Gradient descent updates with a learning rate α can then be performed on the loss function $L(\theta)$, as shown below:

$$y_j = r_j + \gamma \max_{a'} \hat{Q}(s_{j+1}, a'; \theta^-) \quad (4)$$

$$L(\theta) = \frac{1}{2}\big(y_j - Q(s_j, a_j; \theta)\big)^2 \quad (5)$$

$$\theta = \theta - \alpha \nabla_\theta L(\theta) \quad (6)$$

$$= \theta + \alpha\big(y_j - Q(s_j, a_j; \theta)\big)\nabla_\theta Q(s_j, a_j; \theta), \quad (7)$$

where r_j is the reward observed at the current timestep j, θ is the current set of network parameters, and θ^- are the saved copy of target parameters which are updated only periodically. By randomly sampling from the replay buffer, the correlation between sequential samples is removed and the changing distribution introduced by policy changes is smoothed over. Ultimately, the data looks more stationary, leading to an increase in performance [39].

5.2 Prioritized Replay

Sample correlation violates the identically and independently distributed assumption of stochastic gradient descent algorithms. While experience replay was introduced to eliminate sample correlation, sampling from the replay buffer uniformly at random is not the most efficient way to sample [43]. Ideally, we would want to sample transitions which maximally reduce the global loss function;

however, a more practical approach is to give transitions with higher Temporal Difference (TD) error more priority [43], where TD error is given by

$$\begin{aligned}\delta_j &= |y_j - Q(s_j, a_j; \theta)| \\ &= |r_j + \max_{a'} Q(s_{j+1}, a'; \theta^-) - Q(s_j, a_j; \theta)|.\end{aligned} \tag{8}$$

Also, the samples are sampled stochastically according to their priority, $\rho_j = \delta_j$, in order to prevent transitions with initially low TD error from being forgotten[43]. The probability of a transition being sampled is

$$P(j) = \frac{\rho_j^\alpha}{\sum_k \rho_k^\alpha}, \tag{9}$$

where α is a hyper-parameter which controls how much priority is given to each transition.

Estimating the action value function depends on updates corresponding to the same distribution as the expectation. However, prioritized replay introduces bias because it changes that distribution in an uncontrolled manner [43]. This is fixed by using importance sampling weights

$$w_j = (N \cdot P(j))^{-\beta}, \tag{10}$$

where β is annealed linearly from β_{start} to 1 by the end of learning, β_{start} is a hyper-parameter, and N is the size of the replay buffer. The required steps for Prioritized Replay can be seen in lines 16 and 22-23 of Algorithm 1.

5.3 Multi-Step Learning

While in one-step Q-Learning the agent observes a single reward then uses a greedy action selection at the next state to approximate the return thereafter, agents using multi-step returns instead observe multiple rewards before computing the approximation [46], as follows:

$$R_j^{(n)} = \sum_{k=0}^{n-1} \gamma^k r_{j+k} \tag{11}$$

$$y_j = R_j^{(n)} + \gamma^n \max_{a'} Q(s_{j+n}, a'; \theta^-). \tag{12}$$

Multi-step target returns with a well-tuned value for n will often lead to faster learning [38]. The required steps for multi-step learning can be seen in lines 11-15 and 18-19 of Algorithm 1.

5.4 Double Deep Q Learning

Q-learning involves maximization in the approximation of the return, as shown in Equation (4), copied below:

$$y_j = r_j + \gamma \max_{a'} \hat{Q}(s_{j+1}, a'; \theta^-). \tag{13}$$

Due to the maximization step, state-action values suffer from a maximization, or overestimation, bias. In many cases, overestimating the value of states can have a significant negative impact on performance [40].

In order to alleviate maximization bias, the tabular Double Q Learning algorithm instead uses two independent action-value functions that are updated with equal probability [40]. The maximizing action is selected according to the action-value function which is not being updated. In order to combine Double Q-Learning with DQNs, rather than declare two networks and two target networks, Double Deep Q Learning uses only the online network for Q and the target network for $\hat{Q}$ already present in Deep Q Learning and constructs the target y_j according to Equation (14) below:

$$y_j = r_j + \gamma \hat{Q}(s_{j+1}, \arg\max_a Q(s_{j+1}, a; \theta); \theta^-). \tag{14}$$

While using the target network as the second Q function does allow for some correlation between the two Q networks, in practice it is sufficient to alleviate maximization bias [41]. The required steps for Double Deep Q Learning can be seen in lines 18-19 of Algorithm 1.

5.5 Noisy Networks for Exploration

With engineered annealing schedules, ϵ-greedy methods can be very effective; however, these totally random, local changes to the policy are unlikely to lead to large-scale behavioral patterns needed for exploration in most environments[42].

Consider a normal linear layer in a neural network, $Y = \theta X + b$. An alternative approach to exploration when using neural networks to approximate the action-value function is Noisy Networks for Exploration [42] that replace the linear layer with a noisy linear layer

$$Y \stackrel{\text{def}}{=} (\mu^w + \sigma^w \odot \epsilon^w) X + (\mu^b + \sigma^b \odot \epsilon^b), \tag{15}$$

where $\epsilon = (\epsilon^w, \epsilon^b)$ are are randomly sampled, zero mean noise matrices with fixed statistics, and $\mu = (\mu^w, \mu^b)$, and $\sigma = (\sigma^w, \sigma^b)$ are matrices of learnable parameters [42]. The set of parameters is now $\zeta \stackrel{\text{def}}{=} \{\mu\sigma\}$.

Instead of selecting actions according to an ϵ-greedy policy, the agent can now act greedily according to a network using noisy linear layers in place of all linear layers. Ultimately, this allows the agent to learn exploration strategies unique to the task at hand, while automatically annealing or increasing the magnitude of noise parameters to sufficiently explore complex state spaces. Most of the necessary changes are to the architecture of the Neural Network used to approximate the action-value function. However, initialization is handled differently, seen in line 3 of Algorithm 1, noise must be sampled, seen in lines 8, 17, and 20, and the actions are selected greedily with respect to the noisy network, seen in line 9 of Algorithm 1.

6 The LEAD-RL Architecture

We used a deep Q network architecture that contained 2 hidden layers and an output layer with one node for each V/F mode; all layers are fully connected. The first hidden layer consisted of 16 ReLU units, and the second hidden layer contains 16 ReLU units. The output layer contains 5 linear units, one corresponding to each action.

6.1 Training and Testing the LEAD-RL Agent

Each router is allocated a different RL agent during training, leading to 16 unique policies. This means that 16 neural networks are trained offline with 16 replay buffers. The reasoning was that the optimal policy for a corner router may be vastly different from the optimal policy of a router in the

TABLE 1
Reduced Feature Set Used for LEAD-RL

Reinforcement Learning Feature Set
Feature 1: Current Input Buffer Utilization
Feature 2: Outgoing Link Utilization All Directions
Feature 3: Incoming Link Utilization All Directions
Feature 4: Requests Sent by All Cores
Feature 5: Requests Received by All Cores

The size of the feature set as well as computational overhead is drastically reduced with no significant impact on agent performance/energy savings.

center of the mesh; empirically, this has been confirmed as 16 independent agents significantly outperformed a single agent making decisions for each router. This design choice adds no computational overhead at test time; the parameters of each neural network can be stored and used to evaluate each action-value at the beginning of each new time window. The Q network for each router uses as input a vector of features that are collected for that router. Table 1 shows the features that are collected every each time window, scaled to be between 0 and 1.

Algorithm 1. Multi-Step Double Deep Q Learning with Prioritized Experience Replay and Noisy Networks for Exploration

Initialize Prioritized Replay Memory D to capacity N
Initialize n-Step Buffer M to capacity n
Initialize online Q parameters $\zeta = (\mu, \sigma)$, μ at random and $\sigma_{i,j} = .047$
Initialize target $\hat{Q}$ parameter $\zeta^- = \zeta$
for each episode **do**
  Observe and scale state vector s_1
  for t = 1,T **do**
    Sample zero mean noise $\mathcal{E}$
    Select action a_t using a greedy policy on $Q(s_t, \cdot; \zeta, \mathcal{E})$
    Execute a_t then observe reward r_t and next state s_{t+1}
    Store (s_t, a_t, r_t) at the end of M
    Pop $(s_{t-n+1}, a_{t-n+1}, r_{t-n+1})$ from the front of M
    Calculate $R^{(n)}_{t-n+1}$
    Store transition $(s_{t-n+1}, a_{t-n+1}, R^{(n)}_{t-n+1}, s_{t+1})$ in D
    Sample a prioritized minibatch of transitions $(s_j, a_j, R^{(n)}_j, s_{j+n})$ from D
    Calculate the IS weights w_j for all sampled transitions j
    Sample zero mean noise $\mathcal{E}'$
    Select each a'_j such that $a'_j = \arg\max_a Q(s_{j+n}, a; \zeta, \mathcal{E}')$

$$y_j = \begin{cases} R^{(n)}_j & \text{if terminal} \\ R^{(n)}_j + \gamma^n \hat{Q}(s_{j+n}, a'_j; \zeta^-, \mathcal{E}') & \text{else} \end{cases}$$

    Sample zero mean noise $\mathcal{E}''$
    Calculate the TD error $\delta_j = (y_j - Q(s_j, a_j; \zeta, \mathcal{E}''))^2$
    Update the priorities of each sampled transition j using δ_j
    Perform a gradient descent step on $(w \odot \delta)$ with respect to the network parameters ζ
    Every C steps set $\zeta^- = \zeta$
  end for
end for

The LEAD-RL agents are trained offline using Algorithm 1, and tested using Algorithm 2 which stops learning in order to eliminate the significant overhead that learning entails; as a result, the overhead added at test time is only the overhead introduced to evaluate the neural network.

Algorithm 2. Using the Trained Policy at Test Time

Load weights $\zeta = (\mu, \sigma)$ for action-value function Q
Set all $\sigma_{i,j} = 0$
Observe and scale state vector s_1
for t = 1,T **do**
  Select action a_t using a greedy policy on $Q(s_t, \cdot; \zeta)$
  Execute a_t then observe next state s_{t+1}
end for

While multi-step learning, double learning, and prioritized replay used in Algorithm 1 were found to increase convergence speed, thus speeding up the training process, only noisy networks were found to have a significant impact on the final performance of LEAD-RL. As higher training speeds can enable tuning over a larger space of architectures, all these techniques were used in our experiments. In addition, all of the improvements to DQN add no additional overhead to the model at test time, thus the increased training speed came at no additional cost.

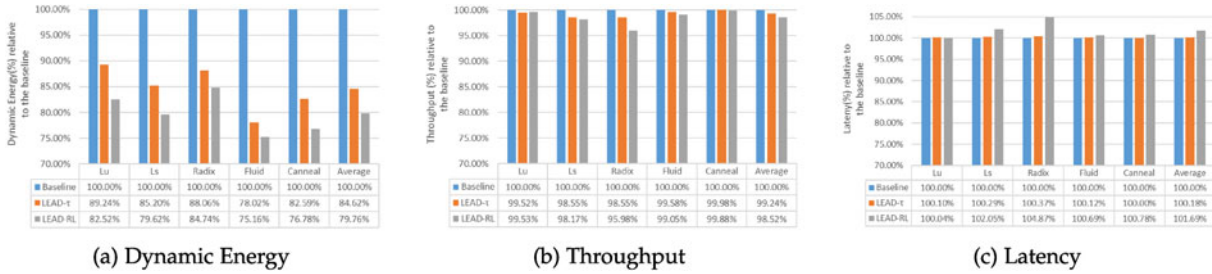
6.2 Reward Function Engineering

Unlike benchmark environments in RL such as Atari games or walking challenges, there is no intrinsic reward function in this application, so it had to be engineered from scratch. Ideally, we would use optimization target values calculated at the end of each trace file: throughput per cycle, latency per packet, and total energy. However, one reward only at the end of each trace would make the rewards very sparse, which introduces numerous challenges in the RL setting. Instead, it was necessary to engineer a reward which could provide a non-zero value at most timesteps. A simple linear combination of the number of packets sent in the previous RW window, p_t, and the dynamic energy consumed in the previous RW, e_t seems to be an obvious solution

$$r_t = p_t - \lambda e_t, \tag{16}$$

where λ is a tunable hyper-parameter to control the tradeoff between throughput and energy consumption.

However, there appears to be a flaw in the reward function; the data being run through the simulator was generated in advance of the simulation. This means that the total number of packets in each trace file is constant with respect to the simulation. Unlike the throughput per cycle metric that needs to be optimized, the number of packets p_t used in the reward is not divided by the total number of cycles in the simulation, because this number is not known at timestep t during the simulation. This could be an issue in an undiscounted reward setting, where the packet term in the cumulative reward would be the same, no matter what policy is used. However we are operating in the discounted setting in which the value of the next state and action $Q(s', a')$ is multiplied by a discount factor, $\gamma \in [0, 1]$. Therefore, if $\gamma < 1$ the agent perceives sending packets sooner as more rewarding. Nevertheless, this leads to a relatively



(a) Dynamic Energy (b) Throughput (c) Latency

Fig. 5. Performance of LEAD-RL and LEAD-τ as a percentage of the baseline model.

complicated relationship between the discount factor, γ, and λ, which makes tuning difficult. Instead, we chose to formulate the reward as

$$r_t = -(b_t + \lambda e_t), \tag{17}$$

where b_t is the buffer utilization of the previous timestep. While we are no longer directly optimizing throughput per cycle, including $-b_t$ in lieu of p_t in the reward makes both terms dependent on the actions selected by the agents. When trained with Algorithm 1, the agent using the reward described by Equation (17) was able to outperform agents using reward described by Equation (16). All results in this paper used the reward formulation from Equation (17).

7 Experimental Evaluation

Multi2sim [30], a cycle accurate full system simulator, is used to generate trace files using industry standard benchmarks such as blackscholes and FFT. A total of 14 trace files are generated using both PARSEC 2.1 [31] and SPLASH2 [32] benchmarks. Our in house network simulator uses the generated traces as input for real traffic patterns. It is crucial that LEAD be trained and tested using real traffic patterns because they are more indicative of real world performance than synthetic workloads. LEAD uses six benchmarks for training, three for validation, and five for testing. All LEAD models and the Baseline use the same five test traces, to allow for a fair experimental comparison. The main results in terms of dynamic energy, throughput, and latency are shown in Fig. 5 and summarized below:

1) LEAD-τ achieves an average dynamic energy savings of 15.4 percent for a loss in throughput of 0.8 percent with no significant impact on latency, relative to the baseline.
2) LEAD-RL saves 20.3 percent dynamic energy relative to the baseline model, at the cost of a 1.5 percent decrease throughput and a 1.7 percent increase in latency.

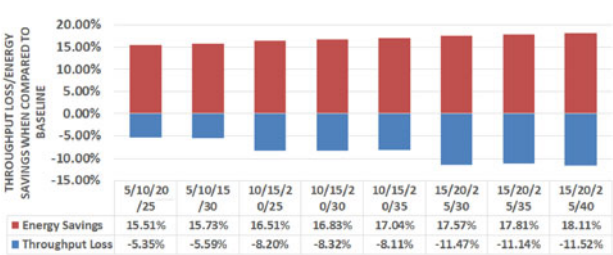


Fig. 6. Throughput loss/dynamic energy savings across multiple threshold selections for the lu trace with a window size of 100.

a) Relative to LEAD-τ, RL enabled the saving of an additional 4.9 percent energy at the cost of an additional 0.7 percent decrease in throughput and an additional 1.5 percent increase in latency.

Note that because all features are local to the router, if more routers/cores were added to the network all LEAD models could be easily scaled to the new architecture with no change in the algorithms.

7.1 Linear Regression Training and Evaluation

All three linear regression based LEAD models are trained separately on the same extracted features but with different labels. After the regression models have been trained and validated, they are exported back into our network simulator where they are used to predict labels. These labels are then used to select the appropriate modes, based on a set of tuned thresholds. In order to determine the optimal buffer utilization thresholds for the LEAD-τ model, we performed an exhaustive threshold study.

Fig. 6 shows the results of tuning the threshold on the barnes benchmark. The x-axis has 4 values which correspond to the four thresholds used to determine what mode a router should operate in for the next epoch. For example, 5/10/20/25 implies that when the buffers are predicted to be less than 5 percent full for the next epoch, the router should operate in mode 1 for the next epoch. When the routers are predicted to be between 5 and 10 percent full, the routers should operate in mode 2. When the routers are predicted to be between 10 and 20 percent full it should operate in mode 3. When the router is predicted to be between 20 and 25 percent full it should operate in mode 4. Finally, when a router is predicted to be more than 25 percent full it should operate at the highest mode for the next epoch. The tuning results show that the best threshold combination was 5/10/20/25, yielding 15.51 percent energy savings for a 5.35 percent throughput loss on the barnes benchmark. Therefore we use this threshold combination for our LEAD-τ model on the 5 test traces.

A breakdown of the time spent in each mode for all LEAD models is shown in Fig. 8. This breakdown is measured as a percentage of the total cycles all routers in the network operated at each of the five different voltage levels compared to the total number of simulation cycles. A comparison of the throughput, dynamic energy, and energy delay product (EDP) is shown in Fig. 7 where the data is normalized against a baseline that does not apply DVFS. We used the best performing thresholds 5/10/20/25 from Fig. 6 and due to space constraints only show various time window sizes for LEAD-τ. LEAD-τ performed very similar for both 500 and 1,000 cycles, resulting in 16-17 percent

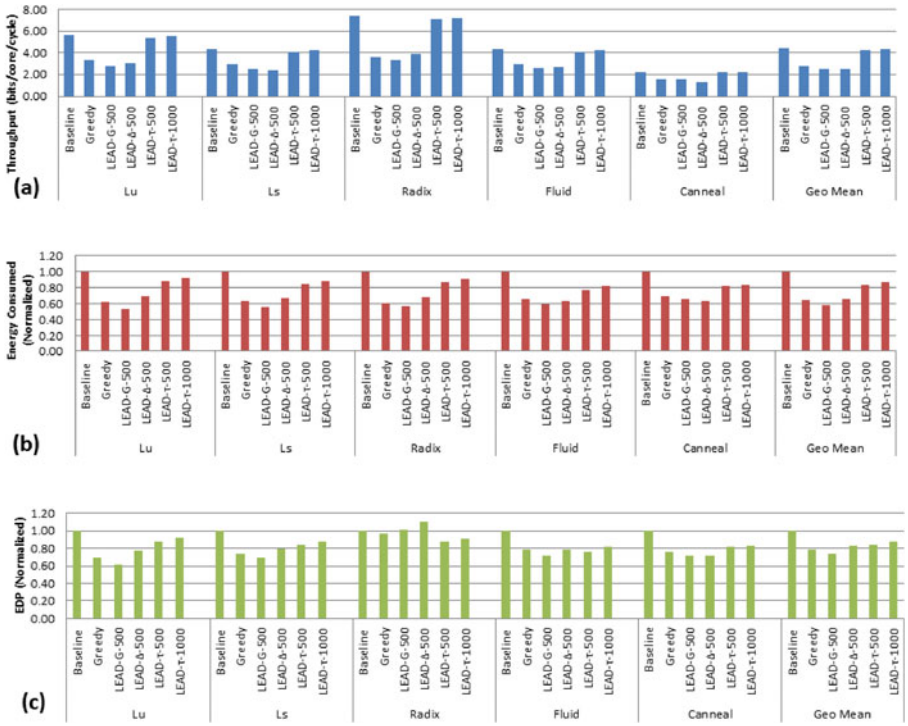


Fig. 7. (a) The throughput for linear regression based LEAD models compared against baseline and greedy. (b) The normalized total dynamic energy. For LEAD-τ, we also show how the model performed at two different window sizes of 500 and 1,000 cycles. (c) The normalized energy delay product with LEAD-τ evaluated for window sizes of 500 and 1,000 cycles.

energy savings at the cost of 3-4 percent less throughput. LEAD-Δ saved 34-35 percent dynamic energy, far more that LEAD-τ, however at a cost of 40-43 percent loss in throughput. LEAD-G is similar to LEAD-Δ in that it achieved even greater energy savings than LEAD-Δ of 42 percent for an almost equal loss in throughput of 42 percent. Fig. 7c shows that the EDP for all models follows the same trend as total dynamic energy. A more detailed discussion of the supervised LEAD models is presented in [45].

7.1.1 Overhead for Linear Regression

Because training is performed offline, the overhead incurred by the linear regression based LEAD models reduces to the runtime energy/area/timing cost to generate predictions and select corresponding V/F levels on a per router basis. Runtime overhead can be simplified to the cost of making predictions, which requires gathering local router features and multiplying them by their trained weights. Then these values are all summed to calculate the prediction. Other work [29] has already calculated the energy and area overhead to perform many different types of operations. A single 16 bit floating point addition consumes 0.4 pJ with an area cost of 1360 um^2. A single 16 bit multiply consumes 1.1 pJ with an area cost of 1640 um^2. We use 39 features for label generation for all linear regression based LEAD models which leads to 39 multiplies and 38 additions. The total energy overhead is 58.1 pJ with a total area overhead of 0.12 mm^2. Using Synopsys we also estimate the timing overhead to be 3-4 cycles. Predictions are made on a per router basis but only need to be generated once per time window, thus we can simply increase the epoch size to decrease the energy overhead. Units can be shared to reduce area overhead at the expense of increased timing costs.

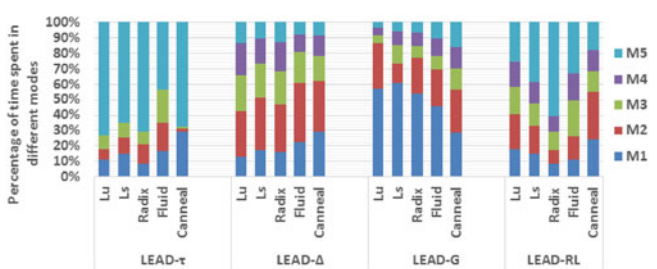


Fig. 8. Breakdown of time spent in each mode by routers during testing for various LEAD models.

7.2 RL Agent Training

The agents are trained using the Barnes, Bodytrack, Dedup, Ferret, Ocean, and Swaptions benchmarks, which were sampled randomly from the 9 training and validation traces. The remaining 3 benchmarks, FFT, Raytrace, and Black, were used for tuning the hyper-parameters. Because learning is offline, the policy is static at test time. To ensure the agent was converging to the optimal solution for any given trace file, the replay buffer had to be sufficiently large to store traces from every training benchmark simultaneously. Therefore, the agent was trained for at least 100 K timesteps, which typically amounts to approximately 2-3 passes over the training traces. To ensure that a sufficient number of samples are in the replay buffer to avoid decorrelation at the beginning of training, no gradient updates are performed until the replay buffer is filled with at least Train Start Size = 10 K samples.

The actual stopping condition also depends on the parameters of the noisy networks. For these experiments, training was considered complete when the noise parameters σ

TABLE 2
Hyperparameters

Hyperparameter	Value
Replay Buffer Size	100 K
Train Start Size	10 K
Minibatch Size	64
Learning Rate	0.0001
Target Net Update Freq.	128
Priority, α	0.6
β_{start}	0.4
σ_{init}	0.047
n steps	3
λ	0.1
γ	0.9

stopped changing to a significant degree. The σ variance parameters can be interpreted as the agent's uncertainty with respect to the true state of the NoC environment and the optimal action to take in that state. Empirically, we have observed that rewards that are highly non-stationary are correlated with high values of σ. With this in mind, the behavior of the σ parameters was also used to tune hyper-parameters such that the reward function shows a more stationary view of the expected return. Table 2 shows the values of all the hyper-parameters used during the training of the LEAD-RL model. Because the space of possible hyperparameters is very large, hyperparameters were selected by performing a greedy, hyperparameter-wise sequential search: starting from default values, the hyperparameters were each tuned one-by-one until values which appeared to perform best were found. The significance of λ, γ, and σ on final performance is discussed in depth in Section 8.

7.3 RL Agent Evaluation

The RL agents are trained offline, which means that at test time learning is stopped, no experience is appended to the Replay Buffer, and no gradient descent steps are performed. The policies for all of the agents remain static and deterministic and proceed according to Algorithm 2. This only requires querying the trained neural network to select actions. Online training instead would use lines 8-24 of Algorithm 1 at test time, which would require maintaining a prioritized replay buffer at each router, keeping noisy parameters (more than doubling the number of parameters), estimating gradients with respect to all of these parameters for each update, and hardware to enable these functions. Correspondingly, the computational cost at test time would be orders of magnitude higher than for offline training. However, by stopping training at test time, the agents lose some flexibility in adapting to non-stationary environments. This challenge was largely overcome by tuning the minibatch size, the replay buffer size, introducing Prioritized Experience Replay, and using a diverse set of benchmarks for training.

7.3.1 Architecture and LEAD-RL Overhead

As with the other LEAD models, the additional overhead incurred by LEAD-RL can be reduced to the runtime energy/area/timing cost to compute the state-action value for each mode and switch to the corresponding V/F levels on a per router basis. This overhead must be considered in terms of the number of agents which is heavily dependent on network topology. LEAD-RL is applied to a mesh with a concentration factor of four in order to reduce the number of agents and create a smaller neural network so that computational overhead is minimized. If LEAD-RL were applied to a mesh with concentration factor of 1, the number of agents would equal the network size. This would result in a larger neural network with the potential for increased power savings at the cost of increased computational overhead. LEAD-RL uses a neural network with two hidden layers with 16 neurons each and an output layer with 5 nodes. The three layers must be computed in sequential order, e.g., layer one must gather features and compute its values before the next layer can be computed. We could flatten the network but area overhead would increase. With our current design it is possible to parallelize all units and operations within each layer. For layer one we have a total of 80 multiplies, 64 additions, and 16 comparisons. For layer two we have a total of 256 multiplies, 240 additions, and 16 comparisons. For the third layer we have a total of 80 multiplies, 75 additions, and 5 comparisons. This equates to a total of 416 multiplies, 379 additions, and 37 comparisons to gather the features, compute state-action values, and to select the V/F mode with the largest action value. The total energy cost is a result of the total number of operations performed and does not change with parallel components; however we can reuse components between layers to reduce area overhead. The total energy overhead is 609.2 pJ and the total area overhead is 0.746 mm^2, which can be further reduced to 94.5 pJ and 0.081 mm^2, respectively, by reducing the precision from 16 bit to 8 bit. The total timing overhead is 11-12 cycles. State-action values are calculated on a per router basis and only need to be generated once per time window. Once again, we can reduce the energy overhead simply by increasing the time window. We could reduce area overhead by sharing more units, however timing overhead would increase. Additionally, we can make algorithmic improvements to reduce the number of parameters in the neural network, thus reducing the number of necessary operations. Recent work [47] shows that significant redundancy exists in neural networks and weak connections can be pruned reducing the number of parameters by as much as a factor of 10. Additionally, model compression using distilled knowledge from a larger neural network can be used to train a smaller neural network [48]. In other words, we can use the trained neural network from this paper to train a smaller equivalent network with less energy and area overhead.

8 Discussion of LEAD-RL

The RL architecture presented here for DVFS led to a better energy versus throughput tradeoff compared to linear regression based LEAD, in addition to providing significant design benefits such as scalability. On the other hand, the architecture needed at test time is more complex. In this section, we discuss several key insights regarding the RL approach to DVFS and compare with results from related work.

Model Behavior of RL versus Supervised Models. From Fig. 8 we can see that LEAD-τ routers spend the majority of their time in the highest mode of operation in order to meet strict performance demands. However, we see that this model can still effectively transition into lower modes in order to

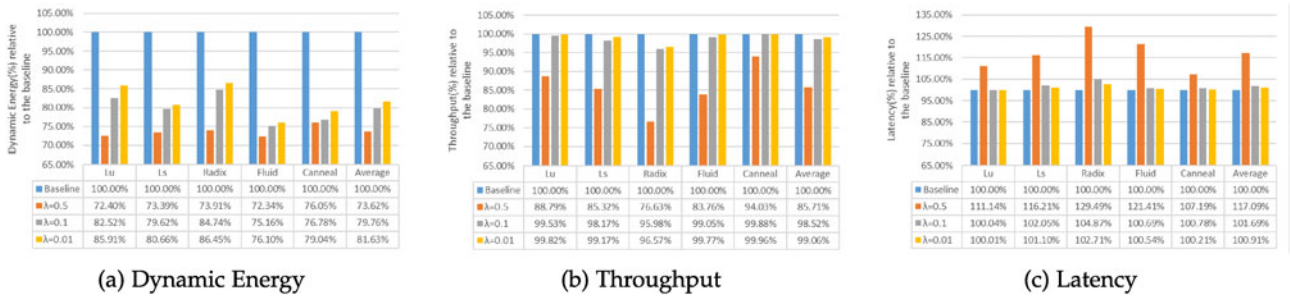


Fig. 9. Performance of LEAD-RL as a percentage of the baseline model when varying the value of λ.

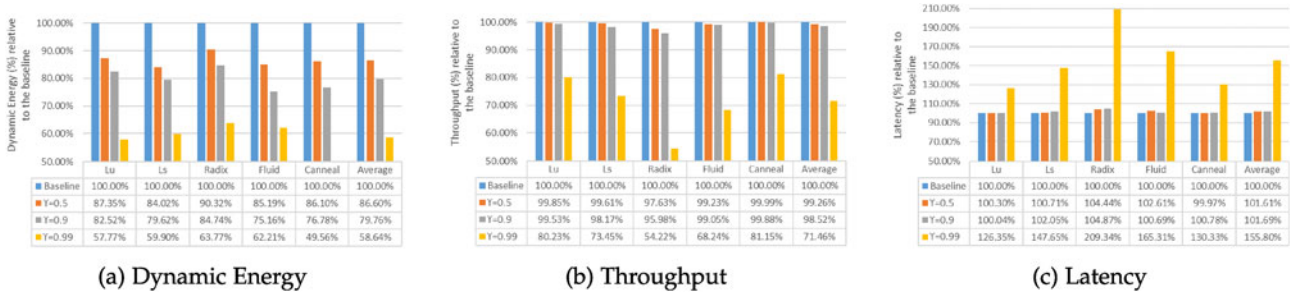


Fig. 10. Performance of LEAD-RL as a percentage of the baseline model when varying the value of γ.

exploit low network traffic and save energy. LEAD-Δ has the most equal mode distribution due to its' gradually changing nature. However, this does not equate to the best energy savings and performance trade-off as it is never allowed to transition directly into the optimal mode, only into the direction of the optimal mode. We see that LEAD-G spends the majority of its' time in the lowest mode of operation in order to maximize energy savings, but this also leads to massive performance degradation. In contrast, LEAD-RL routers spend the majority of their time in lower modes of operation, with a much more even distribution of time spent in higher modes when network-loads shift. This more gradual change in voltage mode as load shifts can be thought of as a blend of LEAD-τ and LEAD-Δ and showcases the improved energy savings of LEAD-RL. Because LEAD-τ routers must meet strict performance requirements, this model is best in a high-load environment. LEAD-Δ routers react slowly to changing traffic patterns, therefore this model would be best suited to low-medium load environments with lax performance requirements. LEAD-G routers focus solely on energy savings; therefore, this model would be best suited to low-load environment with no performance requirements. The flexibility of the RL approach enables the LEAD-RL agents to effectively learn an optimal behavior for any load environment, under arbitrary performance requirements. Thus, LEAD-RL routers can maintain strict performance requirements while still gradually changing voltage levels as load shifts.

Advantages over Supervised Learning. LEAD-RL brings several benefits over previous LEAD models; the first being improved overall energy versus throughput tradeoff. Controlling the energy versus throughput tradeoff can be done through a single hyper-parameter λ in the reward function, as shown in Fig. 9. Larger values result in more energy saving at the cost of throughput, while smaller values result in less throughput loss at the cost of more energy. This allows easy application specific tuning by a designer. Furthermore, generalizing the agent to use more than 5 DVFS modes is simple, as it requires only adding more output nodes in the router's deep Q network. In contrast, increasing the number of modes in the supervised LEAD models requires increasing the number of corresponding thresholds, which in turn increases the time complexity of the grid search procedure for threshold tuning.

RL versus Supervised Learning Based Policies. As seen in Fig. 8, the policies learned by LEAD-RL agents are much more diverse than LEAD-τ, while reducing dynamic energy and maintaining high throughput, unlike LEAD-Δ where added diversity does not translate into a good energy versus throughput tradeoff. Whereas LEAD-τ looks almost bi-modal, spending greater than 90 percent of cycles in modes 1 and 5 in 4 out of 5 test traces, LEAD-RL was able utilize modes 2, 3, and 4 effectively to reduce the energy cost of each workload.

Per Router Learning and Exploration. The noisy network σ parameters are expected to converge to a value representing the stochasticity of the action-value function estimation. Fig. 11 shows the behavior of the σ parameters during training for a subset of the routers. It can be seen that each agent develops its own exploration strategy in order to find its optimal policy: router 12, for instance, converges much faster and requires less exploration than the other routers. This lends credence to the idea that the problem, or at least the representation of the problem, for each agent differs to some degree. In addition, the different final values for the average σ parameter magnitude indicates that an optimal solution requires a separate policy for each router.

Non-Stationarity and Static Policy. Due to the highly non-stationary nature of the observations, which arise from differences in workloads across both sections of traces and different traces, in combination with static policies being implemented at runtime, learning a general policy for the problem proved challenging. To overcome this, the size of the replay buffer needed to be sufficiently large to hold transitions from multiple trace files, otherwise the value

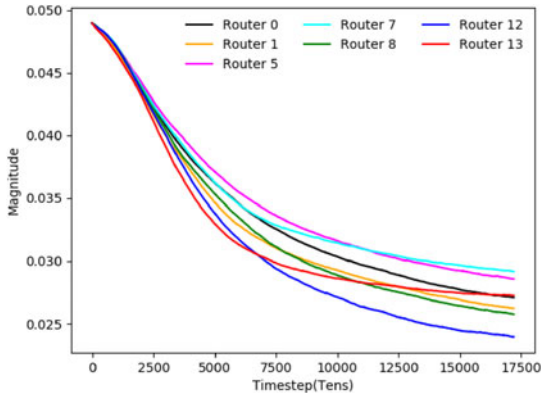


Fig. 11. *Average σ parameter magnitude versus time:* To reduce clutter, the figure shows only 8 randomly selected routers.

function would continue to vary significantly. Additionally, the size of minibatch sampled from the replay buffer had to be large enough to capture a good estimate of the true gradient; using minibatches that were too small also resulted in a less stable policy. Minibatches of size 64 were sufficient when prioritized replay was used, though without prioritized replay minibatches needed to be significantly larger. Finally, as more data was added to the training set, performance improved, hence the importance of using many training traces. In practice, data could potentially be accumulated and used to continuously train better models.

Noisy Network Initialization. The initialization of the noisy parameters σ was found to be of significant importance in this problem. When the initial values were too small, the agent tended to converge to sub-optimal policies. When the initial values were too large, the noisy parameters converged to larger values, which in turn caused the mean parameters μ to also converge to larger values. The resulting more complex model subsequently had poor generalization performance, i.e., a lower average reward (energy versus throughput tradeoff) at test time.

Sensitivity to Discounting Factor. When tuning γ, values which were too small were found to cause convergence to policies which are only locally optimal. In other words, the agent prioritizes actions which maximize reward over intervals which are too short. Conversely, values of gamma that were too large resulted in the agent considering future states which were not significantly affected by the current action. As a result, the expectation of the return is more non-stationary leading to slow or non-convergence of the value function estimation. Results for 3 values of γ are shown in Fig. 10.

Better Reward Function. Due to simulator limitations, we could not get a true measure of throughput at every time window. While a significant improvement in performance was achieved using Algorithm 1 with the reward in Equation (17), replacing $-b_t$ with a true measure of throughput could lead to further improved results. In our experiments the agent was accurate estimating value functions, evidenced by an average loss per transition at every update of $5 * 10^{-5}$, and a flattening of the average σ parameter magnitudes, seen in Fig. 11, indicating that the state-action space has been sufficiently explored. If a true measure of throughput were used in place of $-b_t$, it may be possible to more closely optimize for the end goal of maximum energy savings and minimum throughput loss.

Comparison with Related Work. In [54], RL achieved 33 percent dynamic energy savings when DVFS was applied to NoC, LLC, and other uncore components, at a cost of 2.5 percent decrease in throughput. In contrast, in our work DVFS is applied only to the NoC. We expect that using LEAD-RL with these additional chip components would further improve energy savings. In [9], RL was applied in a multi-tasking environment to do DVFS of the core. However, to achieve the 20.3 percent energy savings reported in our work, their approach would result in a greater loss in throughput, of 4.5 to 13.5 percent. In [26], cache coherence properties were used to optimize voltage and frequency settings, leading to energy savings of 40 percent at a cost of 3 percent in throughput. However, the RL agents in our work do not have access to cache coherence information, as such the results are not comparable.

9 Conclusion

We presented *LEAD Learning-enabled Energy-Aware Dynamic voltage/frequency* scaling for multicore architectures, a collection of machine learning approaches that are trained to proactively switch among a predefined set of DVFS modes in order to reduce energy consumption in NoCs, with minimal impact on throughput and latency. In the new LEAD-RL approach, a DVFS selection model is trained for each router in a mesh architecture, using modern reinforcement learning techniques such as deep Q-networks, noisy networks, replay buffers, and prioritized replay. The new RL approach is more scalable than the previously introduced supervised learning models, does not require expensive threshold tuning, and allows for an easier adjustment of the dynamic energy versus throughput tradeoff. All models are trained offline in order to minimize energy and area footprint at runtime. Simulations on a 4×4 concentrated mesh architecture using PARSEC and Splash-2 benchmarks show that the supervised learning LEAD models can achieve an average dynamic energy savings of 15.4 percent for a loss in throughput of 0.8 percent with no significant impact on latency. When reinforcement learning is used, LEAD increases average dynamic energy savings to 20.3 percent at the cost of a 1.5 percent decrease in throughput and a 1.7 percent increase in latency. Ultimately, LEAD-RL serves as a model which can be easily tailored to the needs of the workload, is trained automatically without much human engineering, and is scalable to large networks with many cores/routers and an arbitrary number of DVFS modes.

Acknowledgments

This research was partially supported by US National Science Foundation grants CCF-1054339 (CAREER), CCF-1420718, CCF-1318981, CCF-1513606, CCF-1703013, CCF-1547034, CCF-1547035, CCF-1540736, and CCF-1702980. We thank the anonymous reviewers for their excellent feedback.

References

[1] A. K. Mishra, R. Das, S. Eachempati, R. Iyer, N. Vijaykrishnan, and C. R. Das, "A case for dynamic frequency tuning in on-chip networks," in *Proc. Annu. IEEE/ACM Int. Symp. Microarchit.*, 2009, pp. 392–303.

[2] R. David, P. Bogdan, and R. Marculescu, "Dynamic power management for multicores: Case study using the intel SCC," in *Proc. Int. Conf. VLSI Syst.-on-Chip*, Oct. 2012, pp. 147–152.

[3] P. Bogdan, R. Marculescu, S. Jain, and R. Gavila, "An optimal control approach to power management for multi-voltage and frequency islands multiprocessor platforms under highly variable workloads," in *Proc. Int. Symp. Netw. Chip*, May 2012, pp. 35–42.

[4] L. Shang, L. Peh, and N. K. Jha, "Power-efficient interconnection networks: Dynamic voltage scaling with links," *Comput. Archit. Lett.*, vol. 1, no. 1, pp. 6–6, Jan. 2002.

[5] S. Herbert and D. Marculescu, "Analysis of dynamic voltage/frequency scaling in chip-multiprocessors," in *Proc. Int. Symp. Low Power Electron. Des.*, Aug. 2007, pp. 38–43.

[6] S. Eyerman and L. Eeckhout, "Fine-grained DVFS using on-chip regulators," *ACM Trans. Archit. Code Optimization*, vol. 8, Apr. 2011, Art. no. 1.

[7] S. Yeng, R. A. Shafik, G. V. Merrett, E. Stott, J. M. Levine, J. Davis, and B. M. Al-Hash, "Adaptive energy minimization of embedded heterogeneous systems using regression-based learning," in *Proc. 25th Int. Workshop Power Timing Model. Optimization Simul.*, Sep. 2015.

[8] R. Jain, P. R. Panda, and S. Subramoney, "Machine learned machines: Adaptive co-optimization of caches, cores, and on-chip network," in *Proc. Des. Autom. Test Eur. Conf. Exhib.*, Apr. 2016, pp. 253–256.

[9] G. Dhiman and T. S. Rosing, "Dynamic voltage frequency scaling for multi-tasking systems using online learning," in *Proc. Int. Symp. Low Power Electron. Des.*, Aug. 2007, pp. 207–212.

[10] H. Richard, "Machine learning based DVFS for energy efficient execution of multithreaded workloads," Dissertations and Theses Technical Reports-Computer Science, Texas State University, Nov. 2014, https://digital.library.txstate.edu/handle/10877/5363

[11] X. Chen, Z. Xu, H. Kim, P. V. Gratz, J. Hu, M. Kishinevsky, U. Ogras, and R. Ayoub, "Dynamic voltage and frequency scaling for shared resources in multicore processor designs," in *Proc. ACM/IEEE Des. Autom. Conf.*, Jul. 2013, Art. no. 114.

[12] H. Shen, J. Lu, and Q. Qiu, "Learning based DVFS for simultaneous temperature, performance and energy management," in *Proc. 13th Int. Symp. Quality Electron. Des.*, Mar. 2012, pp. 747–754.

[13] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, pp. 1929–1958, Jun. 2014.

[14] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 770–778.

[15] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 448–456.

[16] L. Chen, D. Zhu, M. Pedram, and T. Pinkston, "Power punch: Towards non-blocking power-gating of NoC routers," in *Proc. Int. Symp. High-Perform. Comput. Archit.*, Jul. 2015, pp. 378–389.

[17] H. Bokhari, H. Javaid, M. Shafique, J. Henkel, and S. Parameswaran, "darkNoC: Designing energy-efficient Network-on-Chip with Mult-Vt Cells for dark silicon," in *Proc. ACM/IEEE Des. Autom. Conf.*, Jun. 2014, pp. 1–6.

[18] L. Chen, L. Zhao, R. Wang, and T. Pinkston, "MP3: Minimizing performance penalty for power-gating of Clos Network-on-Chip," in *Proc. IEEE 20th Int. Symp. High Perform. Comput. Archit.*, Feb. 2014, pp. 296–307.

[19] L. Chen and T. Pinkston, "NoRD: Node-router decoupling for effective power-gating of On-Chip routers," in *Proc. 45th Annu. IEEE/ACM Int. Symp. Microarchit.*, Dec. 2012, pp. 270–281.

[20] A. Samih, R. Wang, A. Krishna, C. Maciocco, C. Tai, and Y. Solihin, "Energy-efficient interconnnect via router parking," in *Proc. IEEE 19th Int. Symp. High Perform. Comput. Archit.*, Feb. 2013, pp. 508–519.

[21] R. Das, S. Narayanasamy, S. Satpathy, and R. Dreslinski, "Catnap: Energy proportional multiple Network-on-Chip," in *Proc. 40th Annu. Int. Symp. Comput. Archit.*, Jun. 2013, pp. 320–331.

[22] R. Parikh, R. Das, and V. Bertacco, "Power-aware NoCs through routing and topology reconfiguration," in *Proc. 51st ACM/EDAC/IEEE Des. Autom. Conf.*, Jun. 2014, pp. 1–6.

[23] I. Vaisband and E. Friedman, "Dynamic power management with power Network-on-Chip," in *Proc. IEEE 12th Int. New Circuits Syst. Conf.*, Oct. 2014, pp. 225–228.

[24] M. Manda, S. Pakala, and P. Furth, "A multi-loop low-dropout FVF voltage regulator with enhanced load Regulation," in *Proc. IEEE 60th Int. Midwest Symp. Circuits Syst.*, Aug. 2017, pp. 9–12.

[25] T. Bai, V. Lee, and E. Ipek, "Voltage regulator efficiency aware power management," in *Proc. 22nd Int. Conf. Archit. Support Program. Languages Operating Syst.*, Apr. 2017, pp. 825–838.

[26] R. Hesse and N. Jerger, "Improving DVFS in NoCs with coherence prediction," in *Proc. ACM/IEEE Int. Symp. Netw.-on-Chip*, Sep. 2015, Art. no. 24.

[27] S. Son, K. Malkowski, G. Chen, M. Kandemir, and P. Raghavan, "Integrated link/CPU voltage scaling for reducing energy consumption of parallel sparse matrix applications," in *Proc. 20th IEEE Int. Parallel Distrib. Process. Symp.*, Apr. 2006, pp. 297–297.

[28] G. Magklis, P. Chaparro, J. Gonzalez, and A. Gonzalez, "Independent front-end and back-end dynamic voltage scaling for a GALS microarchitecture," in *Proc. Int. Symp. Low Power Electron. Des.*, 2006, pp. 49–54.

[29] M. Horowitz, "1.1 computing's energy problem (and what we can do about it)," in *Proc. IEEE Int. Solid-State Circuits Conf. Dig. Tech. Papers*, Feb. 2014, pp. 10–14.

[30] R. Ubal, B. Jang, P. Mistry, D. Schaa, and D. Kaeli, "Multi2Sim: A simulation framework for CPU-GPU computing," in *Proc. Int. Conf. Parallel Archit. Compilation*, 2012, pp. 335–344.

[31] C. Bienia and K. Li, "PARSEC 2.0: A new benchmark suite for chip-multiprocessors," in *Proc. 5th Annu. Workshop Model. Benchmarking Simul.*, Jun. 2009.

[32] S. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, "The SPLASH-2 programs: Characterization and methodological considerations," in *Proc. ACM/IEEE Int. Symp. Comput. Archit.*, Jun. 1995, pp. 24–36.

[33] C. Sun, C.-H. Chen, G. Kurian, L. Wei, J. Miller, A. Agarwal, L.-S. Peh, and V. Stojanovic, "DSENT—A tool connecting emerging photonics with electronics for opto-electronic networks-on-chip modeling," in *Proc. IEEE/ACM 6th Int. Symp. Netw. Chip*, 2012, pp. 201–210.

[34] A. Bianco, P. Giaccone, and N. Li, "Exploiting dynamic voltage and frequency scaling in networks on chip," in *Proc. IEEE 13th Int. Conf. High Perform. Switching Routing*, Jun. 2012, pp. 229–234.

[35] S. Usman, S. U. Khan, and S. Khan, "A comparative study of voltage/frequency scaling in NoC," in *Proc. IEEE Int. Conf. Electro-Inf. Technol.*, May 2013, pp. 1–5.

[36] J. Zhan, N. Stoimenov, J. Ouyang, L. Thiele, V. Narayanan, and Y. Xie, "Optimizing the NoC slack through voltage and frequency scaling in hard real-time embedded systems," in *Proc. IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, Nov. 2014, pp. 1632–1643.

[37] D. Zoni, F. Terraneo, and W. Fornaciari, "A DVFS cycle accurate simulation framework with asynchronous NoC design for power-performance optimizations," *J. Signal Process. Syst.*, vol. 83, pp. 357–371, Jun. 2016.

[38] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction*, 1st ed. Cambridge, MA, USA: MIT Press, 1998.

[39] V. Mnih, K. Kavukcuoglu, D. Silver, A. Rusu, J. Veness, M. Bellemare, A. Graves, M. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, andD. Hassabis,"Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[40] H. V. Hasselt, "Double Q-learning," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2010, pp. 2613–2621.

[41] H. van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," in *Proc. 13th AAAI Conf. Artif. Intell.*, 2016, pp. 2094–2100.

[42] M. Fortunato, M. G. Azar, B. Piot, J. Menick, I. Osband, A. Graves, V. Mnih, R. Munos, D. Hassabis, O. Pietquin, C. Blundell, and S. Legg, "Noisy networks for exploration," arXiv:1706.10295, 2017.

[43] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," in *Proc. Int. Conf. Learn. Representations*, 2016.

[44] M. Hausknecht and P. Stone, "Deep recurrent Q-learning for partially observable MDPs," arXiv:1507.06527, 2015.

[45] M. Clark, A. Kodi, and R. Bunescu, "LEAD: Learning-enabled energy-aware dynamic voltage/frequency scaling in NoCs," in *Proc. 55th Annu. Des. Autom. Conf.*, Jun. 2018, Art. no. 82.

[46] R. Sutton, "Learning to predict by the methods of temporal differences," *Mach. Learn.*, vol. 3, no. 1, pp. 9–44, 1988.

[47] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural networks," in *Proc. 28th Int. Conf. Neural Inf. Process. Syst.*, 2015, pp. 1135–1143.

[48] C. Bucilua, R. Caruana, and A. Niculescu-Mizil, "Model compression," in *Proc. 12th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2006, pp. 535–541. [Online]. Available: http://doi.acm.org/10.1145/1150402.1150464
[49] C. Watkins and P. Dayan,"Q-learning," *Mach. Learn.*, vol. 8, no. 3/4, pp. 279–292, 1992.
[50] J. Yin, S. She, M. Oskin, and G. H. Loh, "Toward more efficient NoC arbitration: A deep reinforcement learning approach," in *Proc. IEEE 1st Int. Workshop AI-assisted Des. Architecture*, Los Angeles, CA, Jun. 2018.
[51] J. A. Boyan and M. L. Littman, "Packet routing in dynamically changing networks: A reinforcement learning approach," in *Proc. 6th Int. Conf. Neural Inf. Process. Syst.*, Nov. 1993, pp. 671–678.
[52] S. J. Lu, R. Tessier, and W. Burleson, "Reinforcement learning for thermal-aware many-core task allocation," in *Proc. 25th Edition Great Lakes Symp. VLSI*, May 2015, pp. 379–384.
[53] Y. Bai, V. W. Lee, and E. Ipek, "Voltage regulator efficiency aware power management," in *Proc. 22nd Int. Conf. Archit. Support Program. Languages Operating Syst.*, Apr. 2017, pp. 825–838.
[54] X. Chen, Z. Xu, H. Kim, P. Gratz, J. Hu, M. Kishinevsky, and U. Ogras, "In-network monitoring and control policy for DVFS of CMP Networks-on-Chip and last level caches," in *Proc. IEEE/ACM 6th Int. Symp. Netw.-on-Chip*, Jun. 2012, pp. 43–50.

Quintin Fettes received the BS degree in computer science from Ohio University, Athens, in 2017. He is currently working toward the MS degree in computer science at Ohio University. His research interests include the general field of deep reinforcement learning with a focus on applications in dynamic voltage and frequency scaling (DVFS) in network-on-chips (NoCs), and theoretical work on policy gradient methods and attention networks. He is a student member of the IEEE.

Mark Clark received the BS degree in computer engineering from Ohio University, Athens, in 2016 and is currently working toward the MS degree in electrical engineering at Ohio University. His research interests include network-on-chips (NoCs), dynamic voltage and frequency scaling (DVFS), and power-gating. He is a student member of the IEEE.

Razvan Bunescu received the PhD degree in computer science from the University of Texas at Austin, in 2007, with a thesis on machine learning methods for information extraction. He is currently an associate professor in computer science with Ohio University. His research interests lie in the general area of machine learning, with a focus on applications in computational linguistics, biomedical informatics, computer architecture, software engineering, and music analysis. His research has been funded by grants from the National Science Foundation and the National Institutes of Health. He is a member of the IEEE.

Avinash Karanth received the MS and PhD degrees in electrical and computer engineering from the University of Arizona, Tucson, Arizona, in 2003 and 2006, respectively. He is currently a professor of electrical engineering and computer science with Ohio University, Athens, Ohio. His current research interests include computer architecture, optical interconnects, chip multiprocessors (CMPs), and network-on-chips (NoCs). He was a recipient of the National Science Foundation CAREER Award in 2011, the Best Paper Award at the ICCD 2013 conference and his papers have been nominated for best paper at the IEEE Symposium on NoCs in 2010 and the IEEE Asia and South Pacific Design Automation Conference (ASP-DAC) in 2009. He is a senior member of the IEEE and a member of the ACM.

Ahmed Louri received the PhD degree in computer engineering from the University of Southern California, Los Angeles, CA, in 1988. He is the David and Marilyn Karlgaard endowed chair professor of electrical and computer engineering with the George Washington University, which he joined in August 2015. He is also the director of the High Performance Computing Architectures and Technologies Laboratory. From 1988 to 2015, he was a professor of electrical and computer engineering with the University of Arizona, and during that time, he served six years (2000 to 2006) as the chair of the Computer Engineering Program. From 2010 to 2013, He served as a program director with the National Science Foundation's (NSF) Directorate for computer and information science and engineering. He directed the core computer architecture program and was on the management team of several cross-cutting programs, including: Cyber-Physical Systems; Expeditions in Computing; Computing Research Infrastructure; Secure and Trustworthy Cyberspace; Failure-Resistant Systems, Science Engineering and Education for Sustainability; and Cyber-Discovery Initiative, among others. He conducts research in the broad area of computer architecture and parallel computing, with emphasis on interconnection networks, optical interconnects for scalable parallel computing systems, reconfigurable computing systems, and power-efficient and reliable Network-on-Chips (NoCs) for multicore architectures. Recently, he has been concentrating on: energy-efficient, reliable, and high-performance many-core architectures; accelerator-rich reconfigurable heterogeneous architectures; machine learning techniques for efficient computing, memory, and interconnect systems; emerging interconnect technologies (photonic, wireless, RF, hybrid) for NoCs; future parallel computing models and architectures (including convolutional neural networks, deep neural networks, and approximate computing); and cloud-computing and data centers. He has published more than 160 refereed journal articles and peer-reviewed conference papers, and is the co-inventor on several US and international patents. He is a fellow of the Institute of Electrical and Electronics Engineers (IEEE), a member of IEEE Computer Society (CS) Technical Committee on Computer Architecture, the IEEE CS Technical Committee on Parallel Processing, the IEEE CS Technical Committee on Microprocessors & Microcomputers, and the Optical Society of America.