Power and Memory Efficient High-Speed RL based Run time Power Manager for Edge Computation

Ratnala Vinay[1], Kartik Laad[1], Chandrajit Pal[1], Pradip Sasmal[1], Toshihisa Haraki[2], Chirag Juyal[2]
Mohamed Amir Gabir Elbakri[2], and Amit Acharyya[1]
[1]*IIT Hyderabad, Telangana-502285, India*, [2]Suzuki Motor Corporation, Japan
email:{ee21resch11009, ee22mtech12003}@iith.ac.in, {palchandrajit, sasmal.pradip}@gmail.com,
{haraki, juyal_c, mohamed}@hhq.suzuki.co.jp, {amit_acharyya}@ee.iith.ac.in

***Abstract*—Run-time power management poses severe challenges in modern-day edge computing and the adaptability of these run-time power managers to new workloads has been a major concern. Reinforcement learning (RL) based algorithms are able to address this issue of adaptability to unseen load scenarios in the area of High-performance computing (HPC). However, the performance of RL-based run time power managers on the edge degrades due to the constraints it faces post-deployment. The primary reasons for the performance degradation of the RL-based run time power manager on edge are the random actions taken during the long exploratory phase and the considerable amount of memory required for its smooth execution. This motivated us to propose a power and memory-efficient high-speed RL-based run time manager for the edge computation. This reduces the exploratory time by offline-online co-optimization policy and memory consumption post-deployment by removing sparse states. Subsequently, the proposed methodology is implemented on the edge device Jetson Xavier board. It reduces exploratory time by 36% with a reduction in memory footprint by 40% compared to state-of-the-art approaches with average power savings of 29.21% compared to OS_Performance mode, 26.77% compared to OS_Schedutil mode, 19.24% compared to OS_Ondemand mode and 15.42% compared to state-of-the-art Q-learning on edge computing platforms.**

***Index Terms*—Edge computing, Reinforcement learning (RL), Power manager, Exploratory phase, Memory, Offline, Online, Co-optimization, Sparse state**

I. Introduction

Modern day edge computation platforms are designed to deal with a variety of applications in various verticals ranging from businesses, enterprises, factories, hospitals, banks, automobiles and various established facilities in our daily lives. A greater percentage of these devices are either battery-powered or driven by low-rating power sources. This mandates taking care of energy efficiency. Hence designers are using established power handling frameworks to manage energy consumption for its efficient usage without sacrificing the performance of a system.

Run-time power managers (RTMs) are nowadays used in computing systems to provide proper power optimization decisions in these applications. However, standard heuristic-based run-time power management algorithms often face the issue of improper optimization while handling unseen computing loads arising out of varying applications, as most of them may not have been foreseen during system development. In High-Performance Computing (HPC) systems, this issue of adaptability to new loads unseen during development is resolved by deploying Reinforcement learning (RL) based solutions as reported in some literature which will be mentioned subsequently. However, RL-based power managers on edge devices have their own issues of performance degradation due to prolonged exploratory phase and limited hardware resources like memory availability post-deployment.

A few pieces of literature show the primitive run time power optimizer used to deploy the heuristic-based policy to monitor the power management Application Program Interfaces (API) like Dynamic Voltage and Frequency Scaling (DVFS) and Dynamic Core Scaling (DCS) [1]–[4]. These heuristic-based works perform unexpectedly lower in adapting to real-time unseen workload scenarios. Offline Artificial Intelligence (AI) based policy optimizers were later used where the policy is designed during the development phase, and static inference model is used post deployment [5]–[8]. These studies bring intelligence to decision-making but perform unexpectedly lower to adapt to unseen loads as their working policy remains static post-deployment. To address this issue, online learning-based RTMs were later introduced [9]–[11]. However, they still exhibit performance issues post-deployment and consume a considerable amount of hardware resources and is also a time-consuming phenomenon. Few studies were proposed for online RL-based RTMs on edge devices [12]–[14] that take the underlying hardware constraints into account. However, post-deployment performance and memory management are average.

Further, a case study is performed to assess the impact of the existing methods on unseen loads. A set of benchmark applications from Gemstone benchmark suite [15] are taken and offline methods are trained. Applications other than the trained loads are introduced as test loads. Offline based methods which are heuristic-based like built-in OS modes have shown an average power loss of 18.17% compared to best-case power consumption as they are static and result in improper optimizations. Fully online methods like RL without prior training have shown average performance degradation of 22.19% compared to best-case execution time as they need time to gain experience over new loads. This motivated us to come up with a robust RL-based RTM, handling these post-deployment issues on edge devices, especially in terms

This work is partially supported by the Ministry of Electronics and Information Technology, Government of India and Suzuki Motor Corporation, Japan

of speed, power and memory.

In this study, we propose to introduce a Q-learning-based RL algorithm for run-time power management with our optimization concepts integrated. We have mapped the agent of RL as DVFS acting as an API for power optimization, the environment as CPU utilization and reward as a function that optimises the target policy which is power consumption. Q-learning [16] has two phases namely exploration and exploitation. During the exploratory phase, random decisions are taken across the environment to understand payoffs for different actions and gain experience. The exploitation phase is when the experiences from the table assist in undertaking policy optimization decisions. The combination of exploration and exploitation helps the RL-based run-time managers arrive at proper optimization decisions. Generally, the major issues faced while executing RL algorithms on edge devices are performance degradation post-deployment due to prolonged exploratory time and memory requirements. We address these problems and deployed them on edge.

II. Proposed Methodology

Analysing the implementation phase of the RL algorithms illustrates that when the average reward per episode is above the expected reward threshold, past experiences assist in making decisions (exploitation phase) and when the average reward is below the expected reward threshold, new experiences are gained through random decisions (exploration phase) helping us reach the proper optimization decisions. This is known as the exploration-exploitation trade-off. Edge devices are generally deployed in applications which are deadline critical and the phase of prolonged exploration usually results in a significant number of deadline misses. Another major bottleneck lies in the memory utilisation by the RL approaches that in general exceeds the memory available on the edge device.

A. Proposed exploratory time reduction methodology

While observing Figure 1, it is evident that exploratory phase decisions have a significant impact on performance and become difficult to afford post-deployment. As a part of our work, we propose a strategy to reduce the exploratory time without compromising on the experience required by the RL to take proper optimization decisions. The key steps involved here are illustrated below.

1) Generic reward function and state selection: One major metric to enable an offline-online co-optimization strategy is to have a reward function and state which are transferable post offline training. We use CPU utilization as a state estimation metric and $(IPC^2)/W$ as a reward function. IPC is the instruction per cycle and W is the instantaneous power value. IPC, W and CPU utilization values are generic in their behaviour both in offline and online training. IPC helps us to estimate the performance of the application and instantaneous power helps in estimating the power corresponding to the application. Thus, $(IPC^2)/W$ when used as a reward gives us an estimate of the best power-saving actions with the least amount of performance degradation.

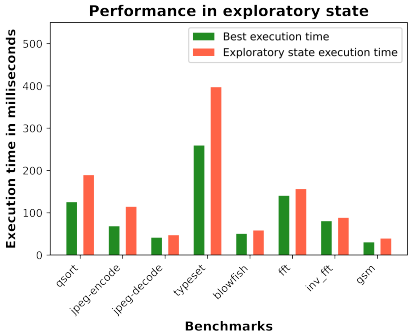


Figure 1: Performance loss in exploratory phase of state of the art methods

2) Offline Training: The purpose of the exploratory phase in RL algorithms is to gain experience in the action space. As a part of our work, we propose the usage of a robust benchmark suite [15] to gain experience. The benchmarks are run in the background on the same target system along with the Q-learning algorithm for handling run-time power. As this phase is designed exclusively to gain experience in the whole offline training phase, random actions are taken and the payoff is computed. The generated table at the end of the offline training phase is coined as the experience table which forms the basis for online retraining.

3) Reward convergence in online training: The experience table formed at the end of offline training forms the basis for online training after deployment. Online training is launched with the offline experience table taken as a reference. Over a window of episodes, the average reward is computed. If the average reward over this window crosses the exploratory reward threshold (β), the online training is said to attain convergence and the algorithm is run in exploitation mode. This mechanism helps to switch to the exploitation phase at the earliest and prevent unwanted exploratory delay cycles. The combination of offline training and reward convergence computation after deployment assists in achieving reduced exploratory phase post-deployment on edge devices.

B. Proposed memory management methodology

Q-table size is an important factor that contributes to the algorithm performance and power consumption in run time. We propose a methodology to reduce the memory footprint of the Q-table that increases the likelihood of the approach being compatible with resource-constrained hardware. The memory management method has three steps as highlighted below.

1) Offline training: This is similar to offline training used for exploratory time reduction. The benchmark suite is executed on the target system and payoffs are computed. We obtain a Q-table with zero and non-zero entries. Especially, for purpose of memory management during offline training a frequency table is also maintained. The frequency table stores the number of times each entry has been visited. Using only absolute rewards may not give us an idea of time spent in each entry. The usage of a frequency table during offline

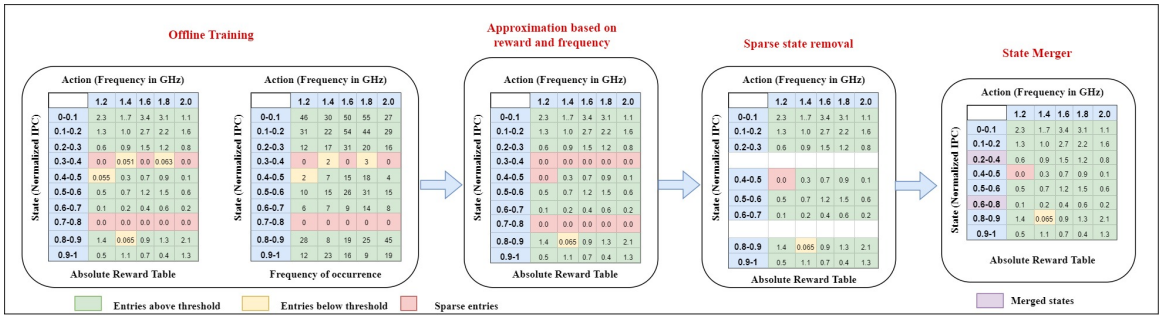


Figure 2: Memory management methodology

learning adds the dimension of estimating the time spent in each entry along with the reward it accumulated.

2) Approximation of entries: A sparse threshold along with a frequency bound is placed to analyze each non-zero entry in the Q-table. Non-zero entries not meeting the sparse threshold and which are below frequency bound are converted to sparse entries. The entries below the sparse threshold (α) and with less frequency of occurrence are those which are explored very less number of times and do not contribute much to the policy optimization post-deployment. Approximation of less explored sparse entries helps us frame states which are fully sparse.

3) Removal of sparse states and state merger: The states which are completely sparse in the offline Q-table are removed. The range of the state removed (sparse state) is merged with the range of the row just above it as shown in Figure 2 so that the overall range of the state estimation variable is maintained even after the sparse state removal. This truncated experience table is used for run-time power management decisions post-deployment on edge.

Algorithm 1 Proposed Methodology

```
for i in Training Benchmarks do
    Compute CPU utilization level
    Take random CPU frequency action
    Compute the reward and update the table
end for
Save the offline experience table
for Each Entry in experience table do
    if Entry < Sparse_Threshold (α) and
      Frequency of occurrence of Entry < Frequency_Bound then
        Entry = 0
    else
        Entry = Entry
    end if
end for
Save the modified experience table
for Row in modified experience table do
    if Row = Sparse then
        Remove the sparse row
    else
        Row = Row
    end if
end for
Save the truncated experience table
for i in Test Benchmarks do
    Load truncated experience table
    if Average_reward < Exploratory_Reward_Threshold (β) then
        Launch exploratory phase
    else
        Launch exploitation phase
    end if
end for
```

III. Results and Analysis

A. Hardware and Benchmarks used

Jetson Xavier board is used as validation platform for the proposed methodology. Algorithm is tested on ARM processor of Jetson Xavier board as a run time program. The validation of the proposed algorithm is done on the Gemstone benchmark suite [15]. Benchmarks: qsort, jpeg, typeset, blowfish, apcm, crc32, fft, inv_fft are used during training phase. Benchmarks: parsec 3.0, gsm, whetstone, dhrystone, parmibench, lmbench, rl are used during testing.

B. Threshold values selection

Sparse threshold, frequency bound and exploratory reward threshold are used as a part of our algorithm implementation. Figure 3a represents variation in average power with change in sparse threshold value(α). The maximum value of the sparse threshold is selected where the average power consumption does not shoot up. Frequency bound is confirmation that an entry is visited rarely. The frequency bound should be a number close to zero. Figure 3b represents a plot of performance degradation to average power consumed with varying exploratory reward threshold (β). β values having optimal performance degradation and power consumption are selected as shown in Figure 3b.

C. Reduction in Exploratory time

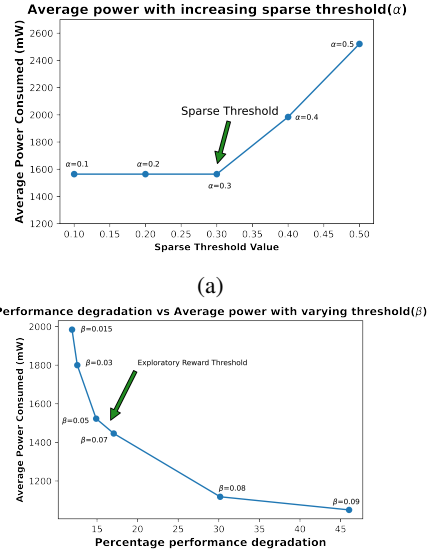
Table. I represents the comparison of the exploratory time of the proposed algorithm with different exploratory thresholds to only online-based methods [12], [13]. The average episode reward was in the range of 0.01 to 0.1. Compared to only the online method, the proposed method was able to decrease the exploratory time on deployment by 36%.

Table I: Exploratory time reduction

Model used	**Exploratory time** (%)
Only online policy [12], [13]	50
Proposed method with threshold at 0.065	18.3
Proposed method with threshold at 0.055	9.5

D. Power savings and performance speed up due to exploratory time reduction

Exploratory time reduction significantly decreases the overall power consumed and improves the performance of the run time power manager as the number of improper optimization decisions is minimized. Figure 4a represents the power



(a)

(b)

Figure 3: Threshold values selection plots

Table II: Comparison of memory utilized with related works

Literature	Q-table Size	Application
Yiming Wang [9]	140 KB	HPC
Hao Shen [5]	120 KB	HPC
M Walker [12]	10 KB	Edge
R Vinay [14]	4 KB	Edge
Proposed Methodology	2.4 KB	Edge

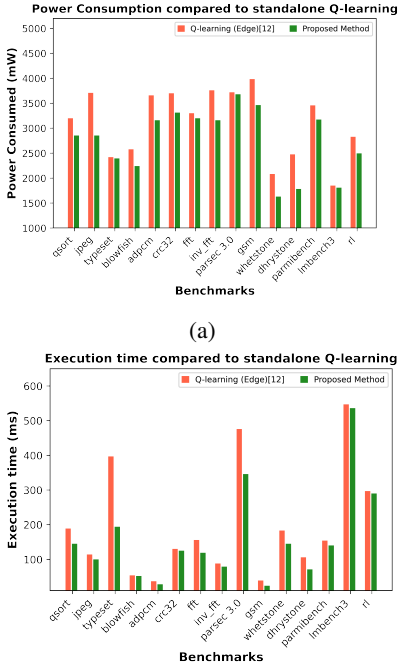
savings and Figure 4b represents the performance speed-up of the proposed method compared to the baseline scheme which is standalone Q-learning. The proposed method brought average power savings of 15.42% and an average performance speed up of 19.3% compared to standalone Q-learning [12] based run time power manager.

E. Reduction in memory of Q-table

Memory foot print of proposed method compared to various competitive methods is given in Table. II. Memory requirement of Q-table per iteration in our proposed methodology is 2.4 KB. A memory reduction of 40% is achieved by proposed memory management policy compared to conventional Q-learning.

F. Power Savings by proposed Run time power manager

Overall power comparison of the proposed method with all other OS-based schemes and Edge device based Q-learning is shown in Figure 5. The average power savings compared to OS_Performance mode is 29.21%, OS_Schedutil mode is 26.77%, OS_Ondemand mode is 19.24% and the state of the Q-learning (Edge) is 15.42%. The overhead of the proposed method is 360 mW which is similar to the overhead of standalone Q-learning. The power overhead is slightly higher than the OS-based methods which are 224 mW. The overhead of the algorithm is taken into consideration in the overall power consumption analysis shown in Figure 5. Idle load estimation and methods like mode switching can further help us to reduce the overhead.



(a)

(b)

Figure 4: Power savings and performance speedup because of exploratory time reduction.

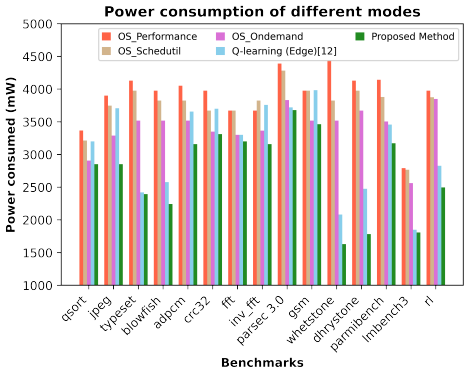


Figure 5: Power consumption of different competitive methods compared to proposed method

IV. Conclusions

In this paper, we proposed a novel power and memory-efficient high-speed Reinforcement Learning based run-time power manager for edge computation. To handle performance degradation caused by RL-based run time power manager post-deployment offline-online co-optimization policy is introduced. The exploratory time was reduced by nearly 36%. Sparsity removal of states was proposed to decrease memory consumption. A memory savings of nearly 40% was achieved compared to the state-of-the-art edge computational approach. Overall the proposed concepts brought an average power savings of 29.21% compared to OS_Performance mode, 26.77% compared to OS_Schedutil mode, 19.24% compared to OS_Ondemand mode and 15.42% compared to conventional Q-learning.

REFERENCES

[1] T. Simunic, L. Benini, A. Acquaviva, P. Glynn, and G. De Micheli, "Dynamic voltage scaling and power management for portable systems," in *Proceedings of the 38th annual Design Automation Conference*, pp. 524–529, 2001.

[2] V. Devadas and H. Aydin, "On the interplay of voltage/frequency scaling and device power management for frame-based real-time embedded applications," *IEEE Transactions on Computers*, vol. 61, no. 1, pp. 31–44, 2010.

[3] J. Hopper *et al.*, "Using the linux cpufreq subsystem for energy management," *IBM blueprints*, 2009.

[4] Z. Mwaikambo, A. Raj, R. Russell, J. Schopp, and S. Vaddagiri, "Linux kernel hotplug cpu support," in *Linux Symposium*, vol. 2, p. 2004, 2004.

[5] H. Shen, Y. Tan, J. Lu, Q. Wu, and Q. Qiu, "Achieving autonomous power management using reinforcement learning," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 18, no. 2, pp. 1–32, 2013.

[6] R. G. Kim, W. Choi, Z. Chen, J. R. Doppa, P. P. Pande, D. Marculescu, and R. Marculescu, "Imitation learning for dynamic vfi control in large-scale manycore systems," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 9, pp. 2458–2471, 2017.

[7] H. Shen, J. Lu, and Q. Qiu, "Learning based dvfs for simultaneous temperature, performance and energy management," in *Thirteenth International Symposium on Quality Electronic Design (ISQED)*, pp. 747–754, IEEE, 2012.

[8] J. L. Conradihoffmann and A. A. Frohlich, "Online machine learning for energy-aware multicore real-time embedded systems," *IEEE Transactions on Computers*, 2021.

[9] Y. Wang, W. Zhang, M. Hao, and Z. Wang, "Online power management for multi-cores: A reinforcement learning based approach," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 4, pp. 751–764, 2022.

[10] G. Dhiman and T. S. Rosing, "System-level power management using online learning," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 28, no. 5, pp. 676–689, 2009.

[11] W. Liu, Y. Tan, and Q. Qiu, "Enhanced q-learning algorithm for dynamic power management with performance constraint," in *DATE 2010*, pp. 602–605, IEEE, 2010.

[12] A. Das, M. J. Walker, A. Hansson, B. M. Al-Hashimi, and G. V. Merrett, "Hardware-software interaction for run-time power optimization: A case study of embedded linux on multicore smartphones," in *ISLPED*, pp. 165–170, 2015.

[13] J. Dai and Z. Liu, "Q-learning based dvfs for multi-core real-time systems," in *The International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery*, pp. 327–336, Springer, 2021.

[14] R. Vinay, P. Sasmal, C. Pal, T. Haraki, K. Tamura, C. Juyal, M. A. G. Elbakri, S. Channappayya, and A. Acharyya, "Light weight rl based run time power management methodology for edge devices," in *2022 29th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, pp. 1–4, 2022.

[15] M. Walker, S. Bischoff, S. Diestelhorst, G. Merrett, and B. Al-Hashimi, "Hardware-validated cpu performance and energy modelling," in *2018 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pp. 44–53, 2018.

[16] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3, pp. 279–292, 1992.