Hybrid Random Forest and Reinforcement Learning Framework for Adaptive CPU Resource Management

Pornnapa Panyadee*, Sajja Tanchanpong*, Sirapat Aunkaew*
Kornprom Pikulkaew*‡, Juggapong Natwichai*†, and Suphakit Awiphan*‡
*Information Technology Service Center, Chiang Mai University, Thailand
†Department of Computer Engineering, Faculty of Engineering, Chiang Mai University, Thailand
‡Department of Computer Science, Faculty of Science, Chiang Mai University, Thailand
{pornnapa.p, sajja.t, sirapat.au, kornprom.pikul, juggapong.n, suphakit.a}@cmu.ac.th

***Abstract*—Effective resource management is a challenge for High Performance Computing (HPC) systems. For their jobs, many users request either too many or too few CPU cores, which wastes resources or leads to early job failure. This research proposes an Adaptive CPU Resource Management Framework that combines reinforcement learning optimization with Random Forest prediction to solve this issue. The ERAWAN HPC cluster gathered 3,600 jobs in 2024 to form the dataset. A Random Forest model was trained to estimate CPU efficiency before execution and achieved strong accuracy (R^2 of 0.96 for training and R^2 of 0.85 for testing). For jobs predicted to have CPU efficiency below 70%, the framework applies an iterative adjustment process to optimize CPU allocation. The experimental results show that the proposed framework improves the average CPU efficiency from 68.78% to 78.21% for training and from 66.33% to 76.92% for testing, representing an overall improvement of approximately 13-16%. These results show that the use of predictive modeling with reinforcement learning optimization is an effective method to improve resource utilization, reduce waste, and improve throughput in large-scale HPC environments. Future work will extend the framework to optimize multiple resources and apply it to larger and more diverse HPC clusters for real-time scheduling.**

***Index Terms*—Cluster System, High Performance Computing, Optimization, Prediction, Utilization**

I. Introduction

High-Performance Computing (HPC) systems have become necessary infrastructure for enabling research and innovation. Most fields of computational science require substantial computing resources to support data modeling, validation, and performance evaluation. As a result, HPC has emerged as a critical technology that enables large-scale simulations, big data processing, and experiment acceleration in many fields of research, such as atmospheric science, astrophysics, geoinformation science, and chemistry [1]. Typically, HPC systems consist of high performance computers, high-speed parallel storage systems, and connected compute nodes that deliver large-scale CPU and GPU resources [2]. A key component of every HPC system is the job scheduler, which allocates computational tasks to CPUs, memory, and other shared resources. One of the most popular schedulers is Slurm (Simple Linux Utility for Resource Management), which is reported to be scalable, flexible, and fault-tolerant [3]. However, achieving efficient and fair utilization of HPC resources remains a challenge in heterogeneous and dynamic workloads.

Traditional job schedulers still rely heavily on user-defined resource requests without any verification or adjustment, which very frequently results in poor allocations. Users may underestimate or overestimate CPU, memory, or runtime requirements. Underestimation results in premature job termination, while overestimation results in wasted resources, reduced system performance, and longer waiting times. This constraint is compounded by the scheduler's limited ability to validate user requests, which has a tendency to render Slurm or other workload managers bottlenecks in HPC environments where thousands of jobs compete for limited resources [4], [5].

To address these challenges, data-driven approaches are playing an increased role in managing resources in HPC systems. Machine learning methods, like regression and ensemble models, can predict metrics such as CPU efficiency or job runtime before a task starts. With these predictions, schedulers can make better and more proactive decisions about allocation of resources. However, prediction-based methods have limits because they cannot adjust resources during a job or learn from feedback. Reinforcement Learning (RL) offers a way to overcome this. In RL, an agent interacts with the system, observes its state, takes actions such as changing CPU allocation, and learns from the results. This process helps the scheduler adapt and improve decisions over time without direct supervision. In HPC systems, RL can help balance efficiency and fairness by responding to changing job needs and cluster conditions. Due to these benefits, combining ML prediction with RL optimization can help achieve accurate and flexible resource management.

This study proposes a hybrid framework called hybrid Random Forest and Reinforcement Learning for Adaptive CPU Resource Management. The framework combines Random Forest (RF) prediction with Dyna-Q Reinforcement Learning to achieve adaptive CPU allocation. The RF model provides fast and interpretable predictions of CPU efficiency, while the Dyna-Q agent refines the allocation policy through real

and simulated learning experiences. Dyna-Q is particularly suitable for HPC systems because it integrates model-based planning with direct learning, resulting in faster convergence and higher learning efficiency compared with traditional optimization methods. By combining predictive modeling with adaptive learning, the proposed framework improves resource utilization, reduces wasted CPU cycles, and supports fair scheduling among multiple workloads.

II. Related Work

Efficient job scheduling has been one of the main challenges in High Performance Computing (HPC) systems. The objective is to maintain a good balance between utilization, throughput, and fairness among many different workloads. Traditional methods such as First Come First Served (FCFS), Shortest Job First (SJF), and other priority-based heuristics [6] are easy to implement but often fail to deal with the changing behavior of real workloads. In heterogeneous HPC systems where resource demands vary between CPUs, GPUs, and memory, these static approaches often leave some resources idle, create long queues, and lower the overall performance of the system [7].

A. Machine Learning Prediction

Over the past decade, researchers have increasingly turned to machine learning to improve scheduling efficiency. Predictive ML models can estimate job runtime memory consumption or CPU efficiency before execution helping schedulers make better resource allocation decisions. For instance, Rodrigues et al. [4], Tanash et al. [8], and Gupta et al. [9] showed that supervised and ensemble approaches can accurately forecast resource requirements which helps reduce underestimation errors and improve job completion rates. Other studies have explored the reliability of the job. Banjongkan et al. [10] used decision trees to identify jobs that were likely to fail while Ali et al. [11] showed that AI-assisted scheduling could improve fairness compared to traditional heuristic approaches. Although these ML techniques improve prediction accuracy, most of them still work offline and static, generating one-time predictions without the ability to adjust during job execution.

B. Reinforcement Learning Optimization

To overcome these static limitations, researchers have begun exploring reinforcement learning (RL) and hybrid ML–RL frameworks. Unlike pure prediction models, RL treats scheduling as a sequential decision-making problem in which an agent learns policies that maximize long-term efficiency through trial and feedback. Wang et al. [12] introduced a hierarchical RL (HRL) approach that jointly manages job selection and resource assignment, while Fan et al. [13] developed a deep RL scheduler that improved throughput by more than 40% compared to heuristic baselines. Kolker-Hicks et al. [14] applied RL to improve backfilling strategies and achieved better performance than the classic EASY algorithm. Zhang et al. [15] and Patel et al. [16] also proposed adaptive CPU–GPU management and cloud optimization using hybrid ML techniques. A review by Gu et al. [17] highlighted the growing maturity of deep RL approaches for workload scheduling in both HPC and cloud environments.

C. Hybrid CPU Resource Management Frameworks

Building on these developments, the present study introduces the Hybrid Random Forest and Reinforcement Learning Framework for Adaptive CPU Resource Management (Hybrid RF–RL Framework). This framework merges a Random Forest (RF) predictor with a Dyna-Q reinforcement learning agent to provide adaptive CPU allocation. The RF model offers rapid and interpretable predictions of CPU efficiency, while the Dyna-Q component fine-tunes allocation strategies by combining real and simulated learning experiences. RF was chosen for its robustness against noise, its ability to handle mixed job log features, and its strong performance on tabular data, where training samples are often limited or unevenly distributed [18], [19]. Unlike traditional ML-only or heuristic-based methods, the proposed Hybrid RF–RL Framework achieves both accurate prediction and adaptive optimization, providing a scalable and intelligent resource management solution for production HPC clusters.

III. Hybrid RF-RL Framework

Algorithm 1: Hybrid RF–RL Framework

Input: Job log $\mathcal{D}$ with features $\mathbf{x}$ and target $y = \texttt{CPU\_Efficiency}$

Output: Optimized Q-table and CPU allocation policy

1. Data Preparation: Clean and preprocess job logs.

2. RF Prediction: Train regressor $f_{\text{RF}}(\mathbf{x})$ to predict y and serve as a surrogate environment.

3. RL Optimization: Initialize $Q(s,a) = 0$ and actions $\mathcal{A} = \{\pm 2^k \mid k = 0, \ldots, 6\} \cup \{0\}$.

for *each epoch* **do**

  foreach *job* $\mathbf{x}$ **do**

    Predict baseline $\hat{y}_b = f_{\text{RF}}(\mathbf{x})$

    if $\hat{y}_b < \tau = 70$ **then**

      Select a using an ϵ-greedy policy

      Predict $\hat{y}_n = f_{\text{RF}}(\mathbf{x} + a)$, compute reward

      Update $Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha r$ and perform limited replays.

return Optimized Q-table and predicted improvement $\Delta = \hat{y}_{new} - \hat{y}_{base}$.

The workflow of the Hybrid RF–RL Framework is designed to enable adaptive and data-driven CPU resource management. In the first phase, job log data is collected and preprocessed to remove incomplete records, encode categorical attributes, and normalize numerical features. In the second phase, the RF Prediction is trained to predict CPU efficiency before job execution, providing a surrogate environment that estimates utilization outcomes under different resource configurations.

TABLE I
FEATURES SELECTED FOR MODEL TRAINING

Feature	Type	Description
Username	Text	Username of the job owner
JobName	Text	Descriptive name of the submitted job
Submit_start	Datetime	Submission date and start time of the job
Type	Text	Discipline or research domain of the job (e.g., Chemistry, Computer Science, Engineering, Medicine)
Partition	Text	Partition or queue where the job was submitted
AllocCPUS	Numeric	Number of CPU cores requested by the user
AllocGPUS	Numeric	Number of GPUs requested (if applicable)
ReqMem	Numeric	Total memory requested for job execution
Elapsed	Numeric	Actual runtime of the job (seconds)
State	Text	Final job state (e.g., COMPLETED, FAILED,TIMEOUT)
CPU_Efficiency	Numeric	CPU utilization efficiency (%) (target variable for prediction)

In the third phase, the RL Optimization component interacts with this predictive environment to optimize CPU allocation for jobs predicted to have low efficiency.

A. Data Preprocessing

Data preprocessing involves preparing the dataset collected from the HPC cluster job scheduler. The dataset includes job log information such as submission details, requested resources, and measured CPU utilization. The target variable in this study is CPU Efficiency, defined as the ratio between effective CPU usage and the number of allocated CPU cores. The features selected for model training are listed in Table I. Categorical variables such as Username, JobName, Partition, and Type were converted using label encoding, while numerical features were normalized to maintain a consistent scale. Records containing missing or corrupted values were removed before training to preserve data quality.

B. RF Prediction

This stage focuses on predicting CPU efficiency before job execution. Random Forest Regression is employed due to its robustness against overfitting, ability to handle high-dimensional heterogeneous features, and strong performance in modeling nonlinear relationships between job attributes and CPU utilization [20]. The Random Forest prediction for CPU efficiency of an input $\mathbf{x}$, using B trees, can be expressed as Eq. 1, where $\hat{f}_b(\mathbf{x})$ represents the prediction from the b-th decision tree and $\hat{y}$ is the predicted efficiency value.

$$\hat{y} = \frac{1}{B}\sum_{b=1}^{B} \hat{f}_b(\mathbf{x}) \tag{1}$$

C. RL Optimization

In the final stage, reinforcement learning is applied to adaptively adjust CPU allocations for jobs predicted to have low efficiency. The Dyna-Q algorithm [21], [22] is adopted because it integrates model-based planning with direct learning. Here, the Random Forest regressor functions as a surrogate environment model, enabling simulated efficiency feedback without executing real jobs. Dyna-Q combines real and simulated experiences, allowing efficient policy learning from historical job logs while minimizing costly interactions with the real HPC environment.

A decision threshold is applied to prediction results as follows:

- If $\hat{y} \geq 70\%$, the job proceeds with its current allocation.
- If $\hat{y} < 70\%$, optimization is triggered.

The threshold 70% follows the NERSC workload study [23], which reports widespread underutilization in HPC workloads.

For inefficient jobs, the agent selects an action a from the following.

$$\mathcal{A} = \{\pm 2^k \mid k = 0, \ldots, 6\} \cup \{0\}$$

representing incremental CPU adjustments. After applying a, the environment predicts the new efficiency $\hat{y}_n$, and the reward is defined as:

$$r = (\hat{y}_n - \hat{y}_b) - \alpha_{\text{over}} \cdot \Delta\texttt{Alloc} - \beta_{\text{under}} \cdot \max(0, \tau - \hat{y}_n) \tag{2}$$

where α_{over} and β_{under} penalize over-allocation and low efficiency, respectively. Through iterative Q-updates and planning, the Dyna-Q agent converges toward an allocation policy that enhances overall CPU efficiency and fairness among competing workloads.

IV. PERFORMANCE EVALUATION

This section evaluates the performance of the proposed Hybrid RF-RL Framework for adaptive CPU resource management. The analysis is divided into three parts: experimental setup and framework configuration, evaluation metrics used for prediction and optimization, and overall CPU efficiency improvement compared with baseline methods. The baseline optimization methods are Stepwise Search (brute-force) and Particle Swarm Optimization (PSO).

TABLE II
THE HYPERPARAMETERS USED IN THE CPU-EFFICIENCY PREDICTION AND OPTIMIZATION METHODS.

Module	Hyperparameter	Value
Stepwise	CPU bounds	[1, 64]
	Trigger threshold (τ)	70.0%
	Search strategy	brute-force sweep
PSO	swarmsize	10
	Local bounds	[1,64]
	Penalty weight (λ)	0.25
	Trigger threshold (τ)	70.0%
Dyna-Q	Learning rate (α)	0.15
	Exploration (ϵ)	0.25
	Planning steps (n_{plan})	60
	Action set	{-32,-16,-8,-4,-2,-1,0,+1,+2,+4,+8,+16,+32}
	CPU bounds	[1, 64]
	Under-target penalty (β)	0.30
	Over-allocation penalty (λ)	0.25
	Trigger threshold (τ)	70.0%
	Training epochs	20

A. Experimental Setup

The dataset used in this study was derived from job logs collected from the ERAWAN HPC cluster, the high-performance computing system of Chiang Mai University, Thailand. It comprises 3,600 job records executed during 2024, representing various academic and research workloads. Each record contains the features listed in Table I, with CPU_Efficiency serving as the target variable. Before training, the data were preprocessed to remove incomplete records and normalize the numerical features. The prepared dataset was then split into 70% for training and 30% for testing. All experiments were implemented in Python and executed on the CMU HPC ERAWAN cluster managed by the Slurm workload manager.

The hyperparameters of the framework are summarized in Table II. A Random Forest regressor was employed as the performance prediction model, with the number of trees and tree depth selected to balance prediction accuracy and computational cost. For optimizing CPU allocation, three methods were considered: Stepwise search, PSO, and Dyna-Q algorithm. To ensure a fair comparison, all optimization methods operated within the same CPU allocation limits and were triggered only for jobs with predicted CPU efficiency below a threshold of 70%. In the PSO and RL approaches, penalty terms were incorporated to discourage unnecessary CPU over-allocation and to guide the optimization toward efficient resource utilization.

B. Prediction Accuracy

The accuracy of the Random Forest (RF) model was evaluated using the coefficient of determination (R^2) together with the mean absolute error (MAE) and the root mean square error (RMSE). As shown in Fig. 1, the predicted CPU efficiency values closely align with the actual measurements. During training, the model achieved an R^2 of 0.96 with an RMSE of 6.75 and an MAE of 3.58, which indicates precise predictions and minimal average error.

When tested on unseen data, the model maintained a high level of performance, achieving an R^2 of 0.85. Although slightly lower than the training score, this still reflects strong generalization ability. The RMSE increased to 12.92 and the MAE to 5.78 which is expected because the testing data contain workload patterns not seen in the training set. Despite the wider spread of the predicted values shown in Fig. 1(b), this variation mainly results from the diverse and irregular nature of unseen jobs. Nevertheless, the Random Forest model maintains a high R^2 value indicating stable predictive performance and providing a strong basis for subsequent reinforcement learning optimization. Overall, these results confirm that the RF model effectively captures CPU efficiency trends, offering both accuracy and generalizability required for adaptive resource management in HPC systems.

TABLE III
COMPARISON CPU EFFICIENCY OF OPTIMIZATION METHODS

Method	CPU Efficiency (%)		Improvement(%)	
	Train	**Test**	**Train**	**Test**
Actual Data	68.78	66.33	0.00	0.00
Predicted (RF)	68.96	67.69	+0.26	+2.05
Stepwise Search	78.76	77.34	+14.51	+16.60
PSO Optimization	76.27	74.76	+10.89	+12.71
Proposed Dyna-Q	**78.21**	**76.92**	**+13.71**	**+15.97**

C. Reinforcement Optimization Efficiency

The effectiveness of reinforcement learning optimization was evaluated by comparing the average CPU efficiency across different stages, as illustrated in Fig. 2. The comparison includes the efficiency of the actual data, the predicted efficiency of the RF model, and the optimized results obtained from Stepwise Search, PSO, and the proposed Dyna-Q algorithm.

As shown in the figure, the baseline average CPU efficiency of the actual job logs was approximately 68.8% for the training

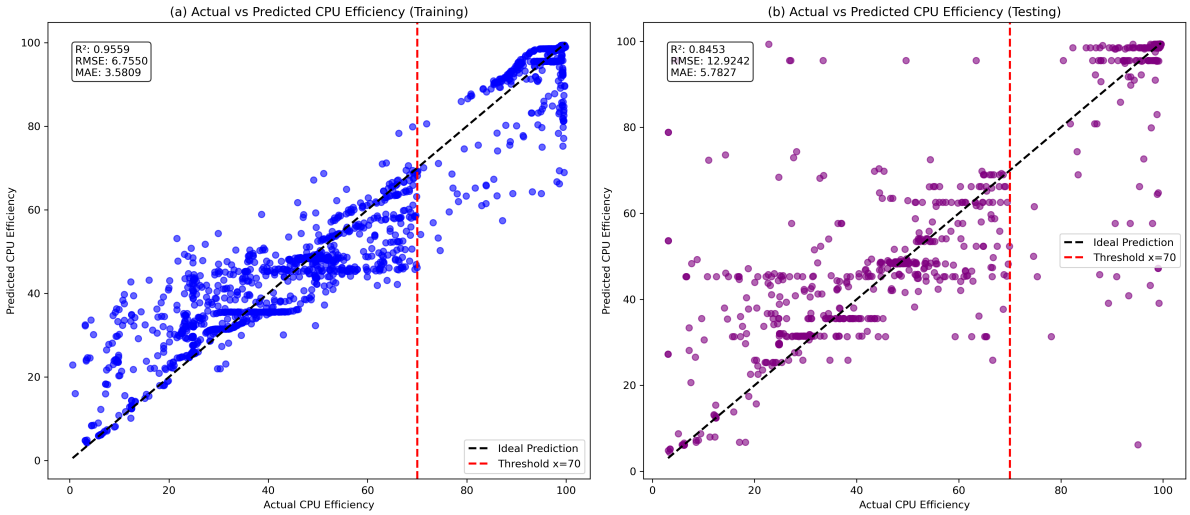

Fig. 1. Relationship between actual and predicted CPU efficiency obtained from the Random Forest model for (a) training and (b) testing datasets.

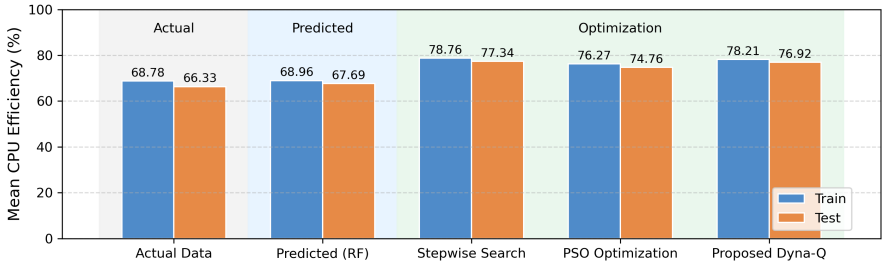

Fig. 2. Comparison of average CPU efficiency for the training and testing datasets across different methods.

set and 66.3% for the test set. After prediction with the Random Forest model, the efficiency improved slightly due to smoothing effects of model generalization. Both optimization methods, Stepwise and PSO, further increased the mean efficiency to around 74 to 79%. The proposed reinforcement-based Dyna-Q achieved the highest performance, reaching 78.2% (training) and 76.9% (testing), corresponding to an improvement of approximately 10 to 16% over the baseline. Table III summarizes the numerical results.

The baseline optimization methods Stepwise Search and Particle Swarm Optimization (PSO) represent static strategies that rely on fixed models trained from historical data and cannot adapt or update during execution. In contrast, the proposed hybrid RF-RL framework introduces a dynamic learning mechanism via the Dyna-Q agent, which continuously refines its allocation policy using both simulated and real feedback. Although the final CPU efficiency of the Hybrid RF–RL Framework is only slightly higher than that of the Stepwise method its adaptive and self-improving nature makes it more scalable and practical for real-time HPC operations.

V. Conclusion

This paper presented a Hybrid Random Forest and Reinforcement Learning (Hybrid RF–RL) Framework for adaptive CPU resource management in HPC systems. The framework combines Random Forest prediction with a Dyna-Q reinforcement learning agent to dynamically optimize CPU allocation based on both historical and real-time feedback. Using 3,600 job records from the CMU ERAWAN HPC cluster, the proposed model achieved high predictive accuracy (R^2 = 0.96 for training and 0.85 for testing) and improved average CPU efficiency by about 13–16% compared with baseline methods. Although the achieved efficiency is close to that of traditional Stepwise optimization, the adaptive learning nature of Dyna-Q makes the framework more scalable and practical for real HPC environments where workloads continually change. These findings demonstrate that combining predictive modeling with adaptive learning enhances both resource utilization and fairness in heterogeneous HPC systems. Future work will extend this approach toward multi-resource optimization (CPU, memory, and GPU) and explore deep reinforcement learning for higher-dimensional decision spaces to enable scalable and intelligent resource management for next-generation HPC environments.

Acknowledgment

This work was supported by the ERAWAN HPC Project, Information Technology Service Center (ITSC), Chiang Mai University, Chiang Mai, Thailand.

REFERENCES

[1] N. R. Council, D. on Earth, D. on Engineering, P. Sciences, and C. on the Potential Impact of High-End Computing on Illustrative Fields of Science, *The potential impact of high-end capability computing on four illustrative fields of science and engineering*. National Academies Press, 2008.

[2] J. J. Dongarra, "An overview of high performance computing and challenges for the future." in *VECPAR*, 2008, p. 1.

[3] A. B. Yoo, M. A. Jette, and M. Grondona, "Slurm: Simple linux utility for resource management," in *Job Scheduling Strategies for Parallel Processing (JSSPP)*, ser. Lecture Notes in Computer Science, vol. 2862. Springer, 2003, pp. 44–60.

[4] E. R. Rodrigues, R. L. Cunha, M. A. Netto, and M. Spriggs, "Helping hpc users specify job memory requirements via machine learning," in *2016 Third International Workshop on HPC User Support Tools (HUST)*. IEEE, 2016, pp. 6–13.

[5] M. Tanash, H. Yang, D. Andresen, and W. Hsu, "Ensemble prediction of job resources to improve system performance for slurm-based hpc systems," in *Practice and Experience in Advanced Research Computing 2021: Evolution Across All Dimensions*, 2021, pp. 1–8.

[6] M. L. Pinedo, *Scheduling: Theory, Algorithms, and Systems*, 5th ed. Springer, 2016.

[7] H. Hussain, S. U. R. Malik, A. Hameed, S. U. Khan, G. Bickler, N. Min-Allah, M. B. Qureshi, L. Zhang, W. Yongji, N. Ghani *et al.*, "A survey on resource allocation in high performance distributed computing systems," *Parallel Computing*, vol. 39, no. 11, pp. 709–736, 2013.

[8] M. Tanash, B. Dunn, D. Andresen, W. Hsu, H. Yang, and A. Okanlawon, "Improving hpc system performance by predicting job resources via supervised machine learning," in *Practice and Experience in Advanced Research Computing 2019: Rise of the Machines (learning)*, 2019, pp. 1–8.

[9] A. Gupta, R. Sharma, and K. Singh, "Improving hpc system performance by predicting job resources via supervised machine learning," in *Proceedings of the IEEE International Conference on High Performance Computing (HiPC)*. IEEE, 2020, pp. 312–321.

[10] A. Banjongkan, W. Pongsena, N. Kerdprasop, and K. Kerdprasop, "A study of job failure prediction at job submit-state and job start-state in high-performance computing system: Using decision tree algorithms [j]," *Journal of Advances in Information Technology*, vol. 12, no. 2, 2021.

[11] H. Ali, F. Khan, and M. Rehman, "Comparative analysis of process scheduling algorithms using ai models," *International Journal of Computer Applications*, vol. 182, no. 43, pp. 10–16, 2021.

[12] L. Wang, M. A. Rodriguez, and N. Lipovetzky, "Optimizing hpc scheduling: a hierarchical reinforcement learning approach for intelligent job selection and allocation: L. wang et al." *The Journal of Supercomputing*, vol. 81, no. 8, p. 918, 2025.

[13] Y. Fan, Z. Lan, T. Childers, P. Rich, W. Allcock, and M. E. Papka, "Deep reinforcement agent for scheduling in hpc," in *2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2021, pp. 807–816.

[14] E. Kolker-Hicks, D. Zhang, and D. Dai, "A reinforcement learning based backfilling strategy for hpc batch jobs," in *Proceedings of the SC'23 Workshops of the International Conference on High Performance Computing, Network, Storage, and Analysis*, 2023, pp. 1316–1323.

[15] L. Zhang, X. Chen, and Y. Liu, "Energy-efficient resource management for federated edge learning with cpu–gpu heterogeneous computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 7, pp. 1521–1534, 2022.

[16] S. Patel, A. Mehta, and V. Gupta, "Evrm: Elastic virtual resource management framework for cloud virtual instances," *Future Generation Computer Systems*, vol. 128, pp. 45–56, 2022.

[17] Y. Gu, Z. Liu, S. Dai, C. Liu, Y. Wang, S. Wang, G. Theodoropoulos, and L. Cheng, "Deep reinforcement learning for job scheduling and resource management in cloud computing: An algorithm-level review," *arXiv preprint arXiv:2501.01007*, 2025.

[18] R. Genuer, J.-M. Poggi, and C. Tuleau, "Random forests: some methodological insights," *arXiv preprint arXiv:0811.3619*, 2008.

[19] H. A. Salman, A. Kalakech, and A. Steiti, "Random forest algorithm overview," *Babylonian Journal of Machine Learning*, vol. 2024, pp. 69–79, 2024.

[20] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees*. Chapman & Hall/CRC, 1984.

[21] R. S. Sutton, "Integrated architectures for learning, planning, and reacting based on approximating dynamic programming," in *Machine learning proceedings 1990*. Elsevier, 1990, pp. 216–224.

[22] L. Zou, L. Xia, P. Du, Z. Zhang, T. Bai, W. Liu, J.-Y. Nie, and D. Yin, "Pseudo dyna-q: A reinforcement learning framework for interactive recommendation," in *Proceedings of the 13th international conference on web search and data mining*, 2020, pp. 816–824.

[23] J. Li, G. Michelogiannakis, B. Cook, D. Cooray, and Y. Chen, "Analyzing resource utilization in an hpc system: A case study of nersc's perlmutter," in *International Conference on High Performance Computing*. Springer, 2023, pp. 297–316.