Adaptive Cooling System Control in Data Center with Reinforcement Learning

Ericha Septya Dinata, Sofia Naning Hertiana, Erna Sri Sugesti
School of Electrical Engineering, Telkom University, Bandung 40257, Indonesia

Article history:

Received Februari 05, 2025
Revised March 06, 2025
Accepted March 21, 2025

Keywords:

Reinforcement Learning;
Data Center;
Monitoring;
Flask;
Energy Efficiency;
Thermal Optimization

Data center cooling system is consuming large amounts of power, which requires effective control to reduce operational costs and deliver optimal server performance. The high power consumption occurs because traditional cooling methods struggle to adapt dynamically to workloads, causing wasteful power consumption. Therefore, this study aimed to explore the use of machine learning methods to improve energy efficiency for data center cooling system. For the experiment, an RL (Reinforcement Learning) model was designed to adjust cooling parameters with dynamic environmental changes. The method focused on optimizing energy efficiency while maintaining stable temperature and humidity control. By applying RL-based control method to PAC system, this study contributed original results that validated the effectiveness of RL-simulated data center environments. Specifically, the stages included developing system model, creating simulations using the PAC control system, and training an RL model with environmental conditions. Data were collected from simulations and analyzed to test the model performance, and the outcomes were presented using a real-time monitoring interface with Flask. The results showed that the RL model achieved an average reward of 4.76 (between -5 and 5), a convergence rate 13.2, a sampling efficiency 10.15, and a stability score 2.6. The model effectively reduced temperature and increased humidity during stressed data center operations. When compared with a fixed cooling system, RL showed superior adaptability to workload variations and reduced unnecessary energy consumption. However, scalability to real data center remained an issue, which required more than simulation validation. In conclusion, the RL-based method optimized efficiency of cooling system, showing the potential to improve energy savings and operational resilience in data center environments.

Corresponding Author:

Sofia Naning Hertiana, School of Electrical Engineering, Telkom University, Bandung 40257, Indonesia
Email: sofinaning@telkomuniversity.ac.id

1. INTRODUCTION

The importance of an effective cooling system in data center was shown by the fire incident at Cyber 1 Building in 2021. Overheating was observed to cause the failure of critical infrastructure, including servers and other electronic components, thereby disrupting business operations.

Data center is critical infrastructure designed to house computer system and their components, such as telecommunications equipment and data storage [1], [2]. These facilities rely on robust system for power supply, environmental control such as air conditioning and ventilation, fire suppression, and physical security to ensure operational continuity and data integrity. Effective cooling system is also essential to prevent overheating, which can cause severe damage to electronic devices and infrastructure [3].

Based on analyzed studies, including energy audits on data center and comparative assessments of data center architectures, thermal and power management show a significant impact on optimizing data center performance and energy consumption [4]. Primary components of data center include IT equipment (servers, storage devices, and switches), power distribution units (PDUs), cooling system, and backup power (UPS and generators) [5]. Among the equipment, Precision Air Conditioner (PAC) is a key element in maintaining thermal balance within server rooms. Compared to conventional air conditioners, PAC units are specifically designed to handle the high heat loads generated by densely packed servers. PAC system uses advanced sensors and controllers to precisely regulate airflow, temperature, and humidity levels [4], [6]-[9].

Despite high accuracy compared to common air conditioner (AC), the majority of PAC system still needs to run and be set manually by user [10], [11]. This process is not suitable for data center because the majority of servers tend to overload at unpredictable condition. Various studies related to Machine Learning or Reinforcement Learning (RL) have explored cooling management in data center. However, the majority still have limitations in terms of dynamic adaptation [12], [13], real-time monitoring [14]–[16], and automated decision-based optimization [17], [18]–[20]. Several strategies have been proposed to increase data center cooling efficiency. For instance, supervised learning algorithms such as regression models and neural networks have been used to predict temperature fluctuations and energy consumption. These models fall short in real-time adjustability in accordance with varying workloads and external environment [21], [22].

RL has shown a promising replacement offering adaptive and autonomous decision-making functionality in the field of cooling control. Deep Q-Network (DQN) and Proximal Policy Optimization (PPO) have been used for adjusting cooling parameters in real-time to reduce energy consumption and provide optimal temperatures [23], [24]. Despite all developments, RL-based methods still have drawbacks of suboptimal sample efficiency, long-converging behavior, and instability to unseen environment changes [25]–[27].

The majority of studies idealized conditions using static airflow models, which do not capture the complex thermodynamics of real data center [28]. Meanwhile, integrating RL with sensor-based real-time monitoring remains a challenge since noisy sensor readings can influence decision accuracy [29]. To address this issue, a RL model was designed to predict the right temperature and humidity based on data center metrics such as CPU, RAM, and disk usage. Aside from the metrics, a monitoring web app was also designed to monitor the server metrics and RL prediction [30]–[32].

RL in data center cooling optimization includes designing an agent that interacts with the environment to learn optimal cooling strategies over time. The framework of RL is generally defined by key components. These include the agent (cooling system controller), the environment (data center conditions such as temperature, humidity, and workload), state (current environmental and operational parameters), action (adjustments to cooling levels), and reward (metrics showing performance, such as energy savings and temperature stability [30], [33].

With a lack of data and an uncertain environment, RL enables cooling system to learn from the environment and automatically adjust temperature and humidity based on server workload [34], [35]. Using a Markov Decision Process (MDP) method, system can optimize energy consumption while maintaining thermal stability [36]–[38]

To ensure effective implementation, this study integrates RL with a web-based monitoring system that allows real-time evaluation and adjustment. The results provide a contribution to the development of an intelligent and adaptive cooling system that minimizes energy waste, thereby preventing damage caused by overheating. Furthermore, this study provides a practical framework for data center optimization. The results are expected to support companies, such as PT Telkom Indonesia, in maintaining their data center infrastructure, ensuring operational continuity, and preventing catastrophic incidents including the Cyber Building 1 fire.

2. METHODS

2.1. Work Process

The work process used in this study is the Iterative Design and Evaluation, a systematic cycle to develop system that include a series of planned processes. The methods include system design, evaluation and refinement, deployment, and optimization. Fig. 1 shows how system is planned until its available for deployment and optimization by ensuring the appropriate method and completed testing [39]–[41].

The first step is system design. In this study, system design concept includes two main components which are a designed RL model and a virtualized architecture using VirtualBox software. The step starts with collecting data from a source to successfully run the virtualization architecture [42].

A set of data was collected from PT Telkom Indonesia for training the machine learning model. Data was collected manually daily through data center administrator, comprising usage of PAC, temperature, and humidity every 3 hours, as a monthly report for the company. With this report, the daily temperature and humidity were combined into a column each and summarized as shown in Table 1.

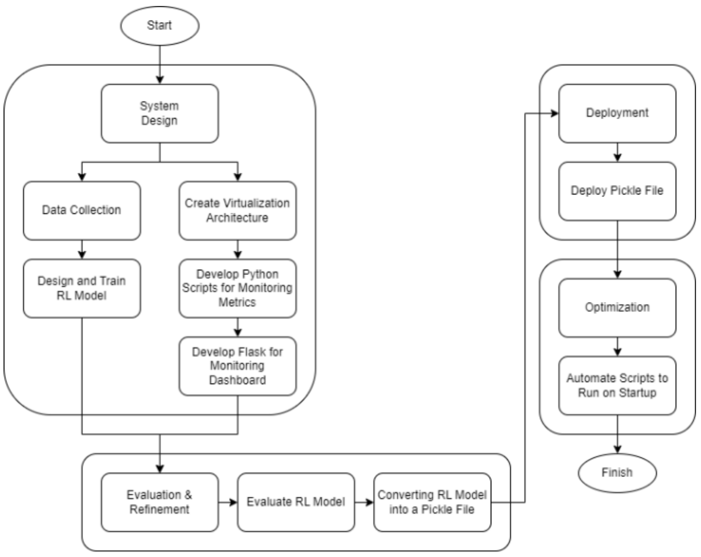
Start
System Design
Data Collection
Create Virtualization Architecture
Design and Train RL Model
Develop Python Scripts for Monitoring Metrics
Develop Flask for Monitoring Dashboard
Evaluation & Refinement
Evaluate RL Model
Converting RL Model into a Pickle File
Deployment
Deploy Pickle File
Optimization
Automate Scripts to Run on Startup
Finish

Fig. 1. Work process stages using iterative design and evaluation

Table 1. Data summary after being collected

Parameter	Values
Minimum Temperature	16.20°C
Maximum Temperature	29.70°C
Minimum Humidity	36.90%
Maximum Humidity	63.10%

RL model was designed using data collected, with PAC temperature ranging from 16.20°C to 29.70°C and humidity of 36.90%-63.10%. The data were used as boundaries for a gym environment, which was made to train RL model because it learned directly from system. When training the model, multiple episodes were tried to find the best evaluation results. Moreover, there were few parameters used when designing the model as shown in Table 2.

Table 2. Design parameters values for RL

Parameter	Values
Alpha	0.1 to 1.0
Gamma	0.90 to 1.0
Epsilon	0.1 to 1.0
Epsilon decay	0.990 to 1.0
Epsilon minimum	0.01 to 0.1
Episode	1000, 10000, and 100000
Max Steps	100

Based on Table 2 in the proposed model, various values of the hyperparameters were tested to discover the optimal settings. Learning rate (α) was examined between 0.1 and 1.0 to explore its influence on the stability of updating the Q value. Furthermore, the discount factor (γ) was tried out between 0.90 and 1.0 to try the extent to which the agent was sensitive in delaying rewards within cooling policy. The exploration rate (ε) was initially fixed in the range of 0.1-1.0, with ε declining from 0.990 to 1.0, thereby controlling the rate of switching from exploration to exploitation. The minimum value of ε was experimented between 0.01 and 0.1 to determine the optimal exploitation boundary. Subsequently, 1000, 10,000, and 100,000 episode experiments were conducted to examine the convergence speed. The maximum step of an episode was 100, which enabled the agent to learn the best cooling policy within a short time horizon.

The virtualization architecture was designed using 3 VMs to balance computational efficiency, resource allocation, and system modularity. Where 2 VMs acted as data center servers and 1 VM as the PAC Control Server. The VM used was Ubuntu Server because of its simplicity, open-source, and serving the purpose of a server. The configuration for all VMs is similar but has different roles. As data center server, the VMs will run a Python script to read its metrics, which are sent to the PAC Control Server for evaluation. For PAC Control Server, the server will receive other VM metrics, calculate the temperature and humidity using machine learning, showing the results on the monitoring dashboard. The interface for all VMs is a bridged network for communication between VMs and host-only, thereby allowing access from the host machine.

The use of 3 VM is to avoid overload on host machine RAM usage. This is because a VM, installed with Ubuntu Server, needs at least 2 GB of RAM. Using 3 VM is sufficient to run a simulation for data center and PAC because a room with more than 1 server can be called data center. With the available RAM on the host machine, it can be used for any other task such as opening the web browser for monitoring.

For monitoring, Python scripts were used, with the potential to read metrics such as CPU, RAM, and disk usage, from any system through dependencies called Psutil. The scripts run on a Python environment manually using a command. When run, the scripts will read the metrics and send to the PAC Control Server in json format. To enhance monitoring capability, Flask was integrated as a microservice framework that served as a fast environment. Since Flask is using Python, it can run the scripts directly from specified routes. In this study, system was designed to read the metrics from other VMs, requiring different routes to receive data from various sources. Additionally, dedicated routes were implemented to show a monitoring dashboard. By using Flask, system was built without reinventing the wheel and able to be shown on a table and graph [43].

The second step is evaluation and refinement. As a machine learning model, RL requires evaluation to ensure a good result before deploying to the server. There are few parameters to define the status of a model. The purpose of the evaluation is to make sure the model has the capabilities to run inside system with high accuracy. By using the first evaluation, the second and third processes can be optimized through variation in some of the model parameters to achieve a better result. However, there are no rules that state the model can only use the latest design. This suggests that the best result can be selected before performing a refinement. The trained model is used in the form of a pickle file, a byte-stream Python object that can be moved in any system. The purpose of pickle file is to maintain a single object that contains the logic of a trained model without the need to give the full script source code. However, pickle cannot be changed or updated directly since it is a byte stream, which requires to be rewritten. Through this method, the evaluation of RL model needs to be determined and ensure suitability for the entire system or re-design.

The third step is deployment. Once the RL model and architecture give a good evaluation, the finalized RL model is saved as a pickle file and deployed on the PAC VM. This deployment enables real-time integration of the model with incoming metrics from the other VMs. System starts operating in a dynamic environment, adjusting PAC parameters based on its learned policy to maintain optimal cooling.

The pickle was made on a host system after achieving a good evaluation. To move the pickle from host machine to guest machine or the PAC Control System, SFTP protocol was used. SFTP ensures secure access and transfer of data between servers or system. Uploading the pickle file to the server only requires user access. However, path where the pickle is uploaded must be defined since Python will read it explicitly and return an error when found.

The fourth step is optimization. After all the virtualization architecture is running and all system works, there is a need for manual execution on every startup. To run system, user needs to type this command beforehand. This command calls Python as an application and instructs it to run the main.py file. If the server keeps restarting, it will take a lot of time to make sure the script is running. Additionally, running Python directly on the server prevents it from accepting commands through the terminal unless accessed via SSH, as the script occupies the server's display session. To optimize this result, system inside an Ubuntu Server is configured to run the Python scripts at start. The configuration for systemd is the user who creates the Python scripts, the Python environment, and working directory. Enabling the optimization will save time from running the Python scripts when the server starts.

2.2. Use Case Diagram

The use case diagram in Fig. 2 comprise data center administrator and staff as actors. Administrator has all access to the server, as well as maintains everything that happens inside the server, while staff can only perform monitoring operations.

Data center metrics that are monitored include CPU, RAM, and disk usage. Each server has unique metrics, which are monitored and sent from a server to the PAC Control Server and collected as a monitoring dashboard. The access for this dashboard is view-only for both data center administrators and staff.

The PAC temperature and humidity were calculated with RL model and changed in real-time according to all metrics server sent to the PAC Control Server. Every incoming metric from any server will change the temperature and humidity directly, as shown on a monitoring dashboard. The access for this dashboard is view-only for both data center administrator and staff [44].

Server configuration was defined as managing the server with both user and root access. The given access is to ensure every server can be monitored by fetching metrics and sending to the PAC Control Server, which is shown on a dashboard. This also includes automatically setting up the scripts for auto-run the whole system when startup and during failure. The access for this configuration is only for data center administrator.

PAC Control Server has multiple endpoints which act as the destination for all server. These endpoints are used by server to offer metrics information, which is sent using POST requests. The metrics are sent in a json format for better readability. Each endpoint is unique to avoid collision between server metrics. Furthermore, new endpoints need to be added manually, ensuring that a fake server will not randomly send more metrics. The access for this operation is only for data center administrator. There is only one RL model used for system in the form of a pickle file. Since the model is made on a different system before importing, access to the server is limited. The pickle file can be updated by retrieving it through SFTP protocol. The access for this is only for data center administrator.

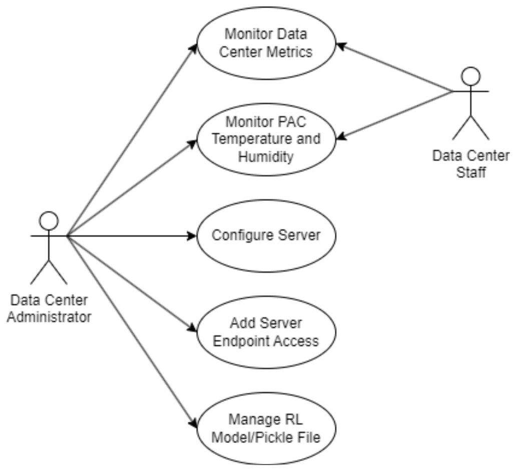
Monitor Data Center Metrics
Monitor PAC Temperature and Humidity
Configure Server
Add Server Endpoint Access
Manage RL Model/Pickle File
Data Center Staff
Data Center Administrator

Fig. 2. Use case diagram

2.3. Study Architecture

System model is split into a few parts as shown in **Fig. 3**, namely VMS A and B, RL model, PAC control server, and dashboard. System starts with 2 VMs designated as VM A and VM B. These VMs are equipped with Python scripts that continuously monitor and collect key system metrics, including CPU, RAM, and Disk usage [45], [46]. The collected metrics are then transmitted to the PAC Control Server through dedicated endpoints defined for each VM. This design ensures real-time data flow from the VMs to control server, enabling dynamic feedback and adaptability. Inside the server, there are 2 components that need to be configured, as shown in Fig. 4

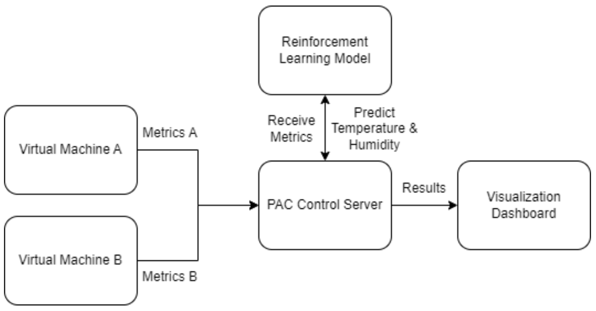

Fig. 3. Study architecture

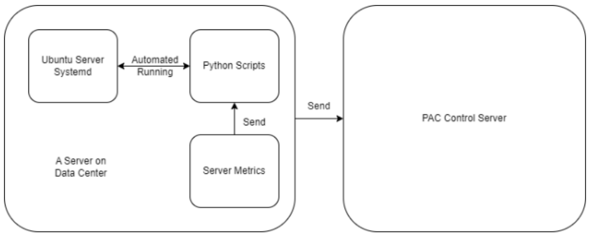

Fig. 4. Architecture on each VMs

Python will act as a service that reads the server metrics and sends to the PAC Control Server endpoint. However, Python needs to be run manually on each startup to ensure the conversion of scripts as Ubuntu service, which allows automatic execution [47], [48]. There is the PAC Control Server which is implemented within an Ubuntu Server VM, integrating both a Flask-based API and RL model. This server acts as the central processing unit for system, receiving real-time metrics from VM A and VM B through API endpoints, as shown in Fig. 5.

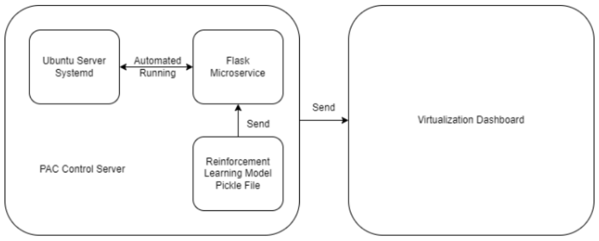

Fig. 5. Architecture on PAC Control Server

Using the metrics, the RL model predicts the optimal temperature and humidity settings for the PAC system. These predictions are dynamically updated to reflect changing conditions, enabling efficient and adaptive cooling control. For the final part, to ensure transparency and usability, the PAC Control Server have a visualization dashboard. This dashboard provides a comprehensive view of all incoming metrics, including CPU, RAM, and Disk usage from VM A and VM B, as well as the temperature and humidity predictions generated by the RL model. Additionally, dashboard allows users to monitor system performance in real-time. [49], [50].

2.4. Implementation of RL

The RL was trained in a virtual environment with an objective to predict then control or adjust the PAC temperature and humidity for servers. The metrics observed were CPU, RAM, and disk usage. By using RL, several steps were carried out, including defining the state, action, and reward, designing the model, performing training, and prediction, as shown in Fig. 6. Other factors that need to be defined including the minimum and maximum temperature and humidity as a range, alongside the random float number to identify CPU, RAM, and disk metrics. Temperature and humidity range are defined to help the agent make decision on how much the prediction range would be as a target. Subsequently, the random float which represents the metrics helps the model to reach multiple training variables in an environment.

The action defined was how the RL agent influenced a particular function. In this case, the action includes increasing and decreasing the temperature and humidity. The reward system correlates with the state and action. An agent will be given a reward when the action taken correlates with the defined state. Additional reward and penalties can also be made to increase the complexity of the reward system. In this context, the reward is given when the action does not pass the threshold range of minimum and maximum temperature and humidity. When the opposite happens, then penalties are given by subtracting the reward.

Gym is an environment created for training the model, which is established by defining the state, action, and reward of an environment and the agent behaviors. This environment is needed because RL model does not learn from historical data but from modified and real environments. Meanwhile, more precise definition of state, action, and reward allows the agent to learn better during the training process.

The model was designed using q-learning algorithm with the help of Python scikit-learn library. This library helped to design the model because it did not need to reinvent the wheel from the beginning. Furthermore, it increased the time taken to design and modify the agent as needed to reach the desired objective. To create the model, a training process was performed, including episodes of steps that required completion. On training, some output such as the metrics used and the prediction made will be shown. This would help developer determine the performance of training process. When a bad result is shown, the gym needs to be re-defined to a better condition or redesign the model. Different gym and agent behavior will have different final results of the model. After the training process is complete, the final step is to predict real or random data that act as server metrics. Whether the value is generated using a script or fetch from a real or virtual server, the output should be as good as the result of the training.

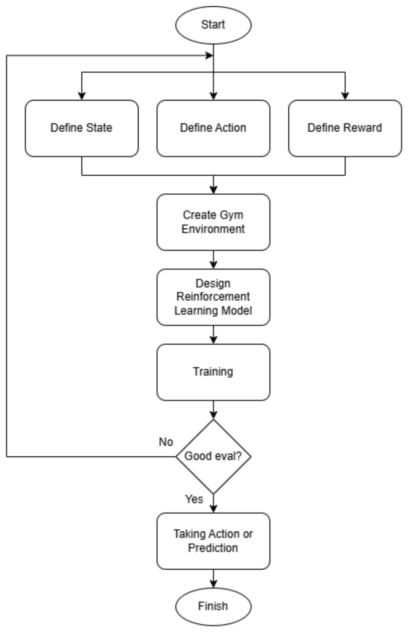

Fig. 6. RL process

3. RESULTS AND DISCUSSION

3.1. RL Evaluation

In RL, there were few parameters that needed to be defined before training, namely alpha, gamma, epsilon, epsilon decay, and epsilon minimum. To find the best result, each parameter was tested with 10x of tests using different ranges based on their characteristic. The test started from alpha to epsilon minimum one at a time from the lowest value of the parameter, with the results shown in Table 3,Table 4,Table 5,Table 6, and
Table 7.

Table 3. Parameter alpha test results

Alpha	Average Reward	Stability	Sample Efficiency	Convergence Rate
0.1	3.88	2.73	4.06	2.25
0.2	3.84	2.75	4.27	1.74
0.3	3.87	2.71	4.45	2.50
0.4	3.88	2.72	4.56	3.48
0.5	3.89	2.67	4.40	3.68
0.6	3.87	2.72	3.93	1.70
0.7	3.88	2.77	4.22	1.49
0.8	3.85	2.82	4.56	4.40
0.9	3.81	2.93	4.38	1.73
1.0	3.77	3.02	4.56	0.88

Table 4. Parameter gamma test results

Gamma	Average Reward	Stability	Sample Efficiency	Convergence Rate
0.1	3.88	2.73	4.06	2.25
0.2	3.88	2.73	4.56	4.55
0.3	3.89	2.66	3.92	1.78
0.4	3.88	2.65	4.11	3.65
0.5	3.84	2.75	1.03	-0.96
0.6	3.87	2.73	4.11	2.68
0.7	3.83	2.78	4.56	1.27
0.8	3.83	2.73	4.56	3.49
0.9	3.85	2.68	3.96	1.67
1.0	3.85	2.71	4.35	1.60

Based on Table 3, alpha 0.8 has the highest convergence rate (4.40), high sample efficiency (4.56), and best stability (2.82). Although not having the highest average reward (3.88), alpha 0.8 of the average reward is still high (3.85) compared to others. Based on these factors, alpha 0.8 appears to be the most favorable overall.

As shown in Table 4, gamma 0.2 has the highest convergence rate (4.55) and sample efficiency (4.56), serving as a strong contender. Additionally, it maintains a solid average reward (3.88.), as gamma 0.7 shows a good performance with the best stability (2.78) and matches the sample efficiency of 4.56. Gamma 0.2 convergence rate and overall balance of metrics, serve as the most favorable option

Table 5. Parameter epsilon test results

Epsilon	Average Reward	Stability	Sample Efficiency	Convergence Rate
0.1	3.88	2.69	4.56	2.06
0.2	3.88	2.73	4.56	4.55
0.3	3.46	3.59	4.57	6.10
0.4	3.27	3.87	3.75	6.36
0.5	3.14	4.12	4.50	8.6
0.6	3.01	4.23	4.05	8.73
0.7	2.91	4.46	3.30	5.31
0.8	2.79	4.51	4.11	6.79
0.9	2.79	4.67	3.99	5.55
1.0	2.79	4.67	3.99	5.55

Based on Table 5, epsilon 0.5 has the highest convergence rate (8.6) and a decent average reward of 3.14, showing strong performance in terms of learning speed. Epsilon 0.6 follows closely with a higher convergence rate (8.73) but a slightly lower average reward (3.01). Meanwhile, epsilon 0.3 offers the best stability (3.59) along with a competitive sample efficiency (4.57) and a reasonable convergence rate (6.1). When balancing all metrics, epsilon 0.6 stands out due to the highest convergence rate (8.73) and solid stability (4.23), despite a lower average reward. Therefore, epsilon 0.6 appears to be the most favorable because of learning efficiency.

Table 6. Parameter epsilon decay test results

Epsilon Decay	Average Reward	Stability	Sample Efficiency	Convergence Rate
0.991	2.86	4.44	4.03	6.28
0.992	2.7	4.56	4.56	7.31
0.993	2.64	4.87	4.42	6.07
0.994	2.39	5.11	3.83	7.70
0.995	2.03	5.4	10.53	14.34
0.996	1.52	5.76	4.28	7.47
0.997	0.92	6.23	3.39	7.02
0.998	0.92	6.23	3.39	7.02
0.999	-1.45	7.09	2.03	4.24
1.000	-2.5	7.05	-0.39	2.22

Table 6 shows that epsilon decay 0.995 has the highest convergence rate (14.34) and a strong sample efficiency (10.53), suggesting excellent learning performance. However, epsilon decay 0.994 offers the best stability (5.11) and a competitive convergence rate (7.7). Decay values like 0.991 and 0.992 maintain decent average reward and stability, without matching the performance of 0.995 in terms of convergence. Balancing these factors, epsilon decay 0.995 appears to be the most favorable due to the convergence rate and sample efficiency.

Based on

Table *7*, epsilon min 0.01 has the highest convergence rate (14.05) and a strong sample efficiency (9.93), showing excellent performance in terms of learning speed and efficiency. It also maintains a competitive average reward of 1.59. Epsilon min 0.02 follows closely with a slightly higher average reward (1.61) and a good stability score (5.8), but the convergence rate (8.78) is lower than that of 0.01. Although higher epsilon min values show improved stability, convergence rate and average reward are lower. Balancing these factors, epsilon min 0.01 appears to be the most favorable overall due to its convergence rate and sample efficiency.

After selecting the best metrics, the test continued by changing the episodes. Based on the selected metrics, Table 8, Table Table ***9***, Table Table ***10***, and Table Table ***11*** show the evaluation results from the RL model with different episodes. The selected episodes were 1000, 10000, and 100000, combined with 100 steps for 1 episode. These experimental values were selected to compare and obtain better episodes for this designed model.

The reward used for this evaluation was in the range of -5 to 5 points, from worse (penalty) to better. Based on the average reward results on Table 8, the closest value to 5 was when using the 100000 episodes

(4.76), followed by 10000 episodes (4.09) and 1000 episodes (1.62). Through this method, using more iteration, which is 100,000 episodes is better than 1,000 and 10,000 episodes.

Table 7. Parameter epsilon minimum test results

Epsilon Minimum	Average Reward	Stability	Sample Efficiency	Convergence Rate
0.01	1.59	5.77	9.93	14.05
0.02	1.61	5.8	4.56	8.78
0.03	1.52	5.87	3.93	8.48
0.04	1.50	5.89	2.54	6.25
0.05	1.34	5.88	3.37	6.86
0.06	1.28	5.93	0.19	2.36
0.07	1.20	6.04	2.67	6.37
0.08	1.05	6.07	1.90	6.18
0.09	0.95	6.18	1.52	4.66
0.10	-1.33	6.19	-1.44	1.58

Table 8. RL average reward evaluation results

Metric	Episodes	Result
Average Reward	1,000	1.62
Average Reward	10,000	4.09
Average Reward	100,000	4.76

Table 9. RL average convergence rate evaluation results

Metric	Episodes	Result
Convergence Rate	1,000	6.66
Convergence Rate	10,000	7.99
Convergence Rate	100,000	13.20

The convergence rate shows how fast the agent was learning. As presented in Table 9, using 100,000 episodes shows the best results (13.20) because higher rate corresponds to faster agent. The best result is followed by using 10,000 (7.99) and 1,000 episodes (6.66). Through this method, using more iteration, which is 100,000 episodes is better than 1,000 and 10,000 episodes.

Table 10. RL average sample efficiency evaluation results

Metric	Episodes	Result
Sample Efficiency	1,000	4.12
Sample Efficiency	10,000	4.43
Sample Efficiency	100,000	10.15

The sampling efficiency shows the effectiveness of the agent when learning, where a higher sample represents better results. Based on Table 10, using 100,000 episodes gives the highest sample efficiency (10.15), followed by using 10,000 episodes (4.43) and using 1,000 episodes (4.12). Through the method, using more iteration of 100,000 episodes is better than 1,000 and 10,000 episodes.

Table 11. RL average stability evaluation results

Metric	Episodes	Result
Stability	1,000	5.74
Stability	10,000	3.09
Stability	100,000	2.60

Stability is measuring how consistent and stable the agent's performance is overtime. This shows that higher value has less fluctuation or performance variations to produce better results. From Table 11, the highest stability was using 1,000 episodes (5.47), followed by 10,000 (3.09) and 100,000 episodes (2.60). By applying this process, using less iteration, which is 1,000 is better than 10,000 and 100,000 episodes.

3.2. Data Center VMs Stress Test

Server metrics that were tested included the increase of CPU, RAM, and disk, both individually, and simultaneously. The changes on each test result aimed to fluctuate the temperature and humidity prediction by RL model. The test was performed by targeting each metric on VM individually with different parameters.

Table 12. Temperature predictions based on CPU tests

CPU Test Size	VM 1 Metric	VM 2 Metric	Target	Avg. Temp. Result
0.5 CPU	33.34%	33.35%	21.62°C	23.50°C
1.0 CPU	52.55%	51.73%	21.62°C	21.53°C
1.5 CPU	72.12%	69.22%	21.62°C	20.42°C

From Table 12, the lowest temperature result was obtained during the stress test using the highest CPU at 1.5 CPU (20.42°C). The result showed that higher CPU usage required the lowest temperature for cooling. Based on Table 13, the highest humidity result was obtained during the stress test at 1.5 CPU (57.65%). This suggested that higher CPU usage corresponded with the greatest requirement for humidity needs.

Table 13. Humidity predictions based on CPU tests

CPU Test Size	VM 1 Metric	VM 2 Metric	Target	Avg. Humid. Result
0.5 CPU	33.34%	33.35%	56.98%	54.94%
1.0 CPU	52.55%	51.73%	56.98%	55.62%
1.5 CPU	72.12%	69.22%	56.98%	57.65%

Table 14. Temperature predictions based on RAM tests

RAM Test Size	VM 1 Metric	VM 2 Metric	Target	Avg. Temp. Result
0.5 GB	39.12%	41.38%	21.62°C	22.51°C
0.7 GB	59.45%	56.82%	21.62°C	21.53°C
1.0 GB	73.40%	70.21%	21.62°C	21.22°C

As shown in Table 14, the lowest temperature result was achieved during the stress test using the highest RAM at 1 GB (21.22°C). The result showed that higher RAM usage produced the lowest temperature required for cooling. From Table 15, the highest humidity result was obtained in the stress test using the greatest RAM at 1 GB (56.47%). This showed that higher RAM usage correlated with greater humidity, thereby preventing system from getting moist.

As shown in Table 16, the lowest temperature result was obtained during the stress test using the highest disk at 6 GB (21.03°C). The results showed that higher disk usage correlated with lower temperature required for cooling. From Table 17, the highest humidity result was obtained during the stress test using the greatest disk at 6 GB (56.97%). The results showed that higher disk usage correlated with greater humidity required to prevent system from getting moist.

Table 15. Humidity predictions based on RAM tests

RAM Test Size	VM 1 Metric	VM 2 Metric	Target	Avg. Humid. Result
0.5 GB	39.12%	41.38%	56.98%	53.32%
0.7 GB	59.45%	56.82%	56.98%	54.65%
1.0 GB	73.40%	70.21%	56.98%	56.47%

Table 16. Temperature predictions based on disk tests

Disk Test Size	VM 1 Metric	VM 2 Metric	Target	Avg. Temp. Result
2 GB	21.54%	19.26%	21.62°C	21.64°C
4 GB	32.04%	29.79%	21.62°C	21.25°C
6 GB	45.34%	42.45%	21.62°C	21.03°C

Table 17. Humidity predictions based on CPU tests

Disk Test Size	VM 1 Metric	VM 2 Metric	Target	Avg. Humid. Result
2 GB	21.54%	19.26%	56.98%	54.35%
4 GB	32.04%	29.79%	56.98%	55.19%
6 GB	45.34%	42.45%	56.98%	56.97%

3.3. Monitoring Dashboard Black-box Text

Black-box testing as shown in Table 18 is a software method where the tester evaluates the functionality of an application without having any knowledge of its internal workings or code structure. The focus is only on the input and output of system. The purpose of this test is to ensure that the software meets user requirements and functions correctly by examining the behavior. Using the black box testing, it can be known that each feature was able to achieve its targets.

Table 18. Black-box test result on monitoring dashboard

Features	Testing	Results	Status
Read VM A metrics and send it to PAC Control Server	Activate Python scripts or run it automatically	VM A able to send its metrics to the PAC Control Server	Achieve
Read VM B metrics and send it to PAC Control Server	Activate Python scripts or run it automatically	VM A able to send its metrics to the PAC Control Server	Achieve
VM PAC Control Server receives VM A & B metrics	Activate Python scripts or run it automatically	VM PAC Control Server able to receive metrics from both VM A and VM B	Achieve
Run RL model on VM PAC Control Server using pickle for temperature and humidity predictions	Activate Python scripts or run it automatically	VM PAC Control Server is able to run RL model’s pickle file to predict the temperature and humidity	Achieve
VM PAC Control Server real-time monitoring on a local web server	Accessing the VM PAC Control Server IP address with the given port	VM PAC Control Server is able to show the results including metrics and predictions in real-time on a local web server	Achieve

3.4. Limitations and Practical Considerations on system

The results indicate that data center cooling system control using RL can improve energy efficiency by adjusting temperature and humidity based on real-time sensor readings, with the conditions of CPU, RAM, and DISK shown in Table 12 to Table 17. Specifically, the application of this model is capable of reducing cooling energy consumption, which is one of the largest items in data center operating costs. By adapting cooling strategy to server operating conditions, this model allows for more dynamic power reduction than static strategies, thereby improving the sustainability of data center operations.

Despite the significant contribution, this model has several limitations that need to be considered in large-scale implementations. The most significant challenge lies in the increasing computational complexity when the number of servers and sensors added increases, leading to longer training durations and computation rates. Additionally, the introduced RL model does not respond to dynamic workload variations in terms of changes in CPU usage or data traffic fluctuations that affect the performance of the developed cooling policy.

In real-world implementations, integration with physical cooling devices and data center facility management software is required, which may cause compatibility issues with existing devices and infrastructure. Airflow, Dust, and Air Quality, as well as other environmental conditions, have not been included in the model calculation, which may affect the accuracy of cooling control under various operating conditions. Therefore, for large data center deployment cases, further evaluation of the model's scalability, computational effectiveness, and robustness to more complex operating conditions is needed.

4. CONCLUSION

In conclusion, the evaluation of the agent's performance using different episodes provided different outputs that allowed identifying the best metric value. The results showed that using 100,000 episodes produced highest average reward of 4.76, followed by 10,000 at 4.09 and 1,000 episodes at 1.62. This was consistent with the convergence rate, where 100,000 episodes also showed the fastest learning speed at 13.20, exceeding the rates for 10,000 and 1,000 episodes at 7.99 and 6.66, respectively. Additionally, sample efficiency was maximized at 100,000 episodes with a score of 10.15, higher than 10,000 episodes at 4.43 and 1,000 episodes at 4.12. This showed that the agent learned more effectively with increased iterations.

The highest stability was 1,000 episodes at 5.74, followed by 10,000 and 100,000 episodes at 3.09 and 2.60, respectively. This showed that increasing the number of episodes could enhance learning outcomes, leading to higher fluctuations in performance. Therefore, 100,000 episodes were considered better for maximizing reward, convergence rate, and learning efficiency, although the stability was not significantly high compared to others.

As the size of the CPU, RAM, and disk components increased, the temperature decreased and the humidity increased. After completing the CPU stress test, the highest CPU usage with 1.5 CPU provided the lowest temperature at 20.42°C and the greatest humidity at 57.65%. When the RAM stress test was performed, the highest RAM usage with 1 GB of RAM produced the lowest temperature at 21.22°C and the greatest humidity at 56.47%. Lastly, when the disk stress test was performed, the highest disk usage with 6 GB of RAM produced the lowest temperature at 21.03°C and the greatest humidity at 56.97%. The black-box testing of the dashboard showed that all features were functioning correctly and were suitable for real-time monitoring and controlling the PAC temperature and humidity.

For further studies, recommendation was made to explore the integration of the proposed model with physical cooling devices and data center facility management software to address potential compatibility issues with existing infrastructure. Additionally, incorporating Airflow, Dust, and air quality into the model could enhance the accuracy of cooling control under diverse operating conditions. Further studies should also focus on evaluating the model's scalability, computational efficiency, and robustness in handling more complex and dynamic data center environments.

REFERENCES

[1] S. Ketabi, H. Chen, H. Dong, and Y. Ganjali, “A Deep Reinforcement Learning Framework for Optimizing Congestion Control in Data Centers,” *in NOMS 2023-2023 IEEE/IFIP Network Operations and Management Symposium,* pp. 1-7, 2023, https://doi.org/10.48550/arXiv.2301.12558.
[2] M. Yenugula, “Data center power management using neural network,” *International Journal of Advanced Academic Studies*, vol. 3, pp. 320–325, Jan. 2021, https://doi.org/10.33545/27068919.2021.v3.i1d.1124.
[3] R. Gunawan, T. Andhika, . S., and F. Hibatulloh, “Monitoring System for Soil Moisture, Temperature, pH and Automatic Watering of Tomato Plants Based on Internet of Things,” *Telekontran : Jurnal Ilmiah Telekomunikasi, Kendali dan Elektronika Terapan*, vol. 7, no. 1, pp. 66–78, Apr. 2019, https://doi.org/10.34010/telekontran.v7i1.1640.
[4] K. Bilal *et al.*, “A Comparative Study Of Data Center Network Architectures,” *26th EUROPEAN conference on modelling and simulation, ECMS*, May 2012, https://doi.org/10.7148/2012-0526-0532.
[5] C. Blad, S. Bøgh, and C. S. Kallesøe, “Data-driven Offline Reinforcement Learning for HVAC-systems,” *Energy*, vol. 261, p. 125290, 2022, https://doi.org/10.1016/j.energy.2022.125290.
[6] M. Biemann, F. Scheller, X. Liu, and L. Huang, “Experimental evaluation of model-free reinforcement learning algorithms for continuous HVAC control,” *Appl Energy*, vol. 298, p. 117164, 2021, https://doi.org/10.1016/j.apenergy.2021.117164.
[7] S. Wassermann, T. Cuvelier, P. Mulinka, and P. Casas, “Adaptive and Reinforcement Learning Approaches for Online Network Monitoring and Analysis,” *IEEE Transactions on Network and Service Management*, vol. 18, no. 2, pp. 1832–1849, 2021, https://doi.org/10.1109/TNSM.2020.3037486.
[8] T. A. Nakabi and P. Toivanen, “Deep reinforcement learning for energy management in a microgrid with flexible demand,” *Sustainable Energy, Grids and Networks*, vol. 25, p. 100413, 2021, https://doi.org/10.1016/j.segan.2020.100413.
[9] H. Che, Z. Bai, R. Zuo, and H. Li, “A Deep Reinforcement Learning Approach to the Optimization of Data Center Task Scheduling,” *Complexity*, vol. 2020, pp. 1–12, Aug. 2020, https://doi.org/10.1155/2020/3046769.
[10] H. Li *et al.*, “Modeling the Relationship Between Air Conditioning Load and Temperature Based on Machine Learning,” in *4th International Conference on Intelligent Control, Measurement and Signal Processing (ICMSP)*, pp. 524–528, 2022, https://doi.org/10.1109/ICMSP55950.2022.9859169.
[11] N. Sulaiman, M. P. Abdullah, H. Abdullah, M. Zainudin, and A. Yusop, “Fault detection for air conditioning system using machine learning,” *IAES International Journal of Artificial Intelligence (IJ-AI)*, vol. 9, p. 109, Mar. 2020, https://doi.org/10.11591/ijai.v9.i1.pp109-116.
[12] J. Hao, D. W. Gao, and J. J. Zhang, “Reinforcement Learning for Building Energy Optimization Through Controlling of Central HVAC System,” *IEEE Open Access Journal of Power and Energy*, vol. 7, pp. 320–328, 2020, https://doi.org/10.1109/OAJPE.2020.3023916.
[13] L. Yu, D. Xie, C. Huang, T. Jiang, and Y. Zou, “Energy Optimization of HVAC Systems in Commercial Buildings Considering Indoor Air Quality Management,” *IEEE Trans Smart Grid*, vol. 10, no. 5, pp. 5103-5113, Oct. 2018, https://doi.org/10.1109/TSG.2018.2875727.
[14] A. Chatterjee and D. Khovalyg, “Dynamic indoor thermal environment using Reinforcement Learning-based controls: Opportunities and challenges,” *Build Environ*, vol. 244, p. 110766, 2023, https://doi.org/10.1016/j.buildenv.2023.110766.
[15] C. Chen *et al.*, “Deep Reinforcement Learning-Based Joint Optimization Control of Indoor Temperature and Relative Humidity in Office Buildings,” *Buildings*, vol. 13, p. 438, Feb. 2023, https://doi.org/10.3390/buildings13020438.
[16] X. Zhong, Z. Zhang, R. ZHANG, and C. Zhang, “End-to-End Deep Reinforcement Learning Control for HVAC Systems in Office Buildings,” *Designs (Basel)*, vol. 6, p. 52, Jun. 2022, https://doi.org/10.3390/designs6030052.
[17] C. Zhou *et al.*, “Simulator-Based Reinforcement Learning for Data Center Cooling Optimization,” *In Deployable RL: From Research to Practice@ Reinforcement Learning Conference*, 2024, https://openreview.net/forum?id=3hZL9Vv0Ay.
[18] Y. Peng *et al.*, “Energy Consumption Optimization for Heating, Ventilation and Air Conditioning Systems Based on Deep Reinforcement Learning,” *IEEE Access*, vol. 11, pp. 88265–88277, 2023, https://doi.org/10.1109/ACCESS.2023.3305683.
[19] T. Bian and Z.-P. Jiang, “Reinforcement Learning and Adaptive Optimal Control for Continuous-Time Nonlinear Systems: A Value Iteration Approach,” *IEEE Trans Neural Netw Learn Syst*, vol. 33, no. 7, pp. 2781–2790, 2022, https://doi.org/10.1109/TNNLS.2020.3045087.
[20] Y. Wang, M. Xiao, and Z. Wu, “Safe Transfer-Reinforcement-Learning-Based Optimal Control of Nonlinear Systems,” *IEEE Trans Cybern*, vol. 54, no. 12, pp. 7272–7284, 2024, https://doi.org/10.1109/TCYB.2024.3485697.
[21] O. Al-Ani and S. Das, “Reinforcement Learning: Theory and Applications in HEMS,” *Energies*, vol. 15, no. 17, p. 6392, 2022, https://doi.org/10.3390/en15176392.

[22] X. Xu, Y. Jia, Y. Xu, Z. Xu, S. Chai, and C. S. Lai, "A Multi-Agent Reinforcement Learning-Based Data-Driven Method for Home Energy Management," *IEEE Trans Smart Grid*, vol. 11, no. 4, pp. 3201–3211, 2020, https://doi.org/10.1109/TSG.2020.2971427.
[23] Y. Wang, Y. Sun, B. Cheng, G. Jiang, and H. Zhou, "DQN-Based Chiller Energy Consumption Optimization in IoT-Enabled Data Center," in *IEEE 23rd International Conference on Communication Technology (ICCT)*, pp. 985–990, 2023, https://doi.org/10.1109/ICCT59356.2023.10419683.
[24] Q. Zhang, C.-B. Chng, K. Chen, P.-S. Lee, and C.-K. Chui, "DRL-S: Toward safe real-world learning of dynamic thermal management in data center," *Expert Syst Appl*, vol. 214, p. 119146, 2023, https://doi.org/10.1016/j.eswa.2022.119146.
[25] Y. Ran, H. Hu, Y. Wen, and X. Zhou, "Optimizing Energy Efficiency for Data Center via Parameterized Deep Reinforcement Learning," *IEEE Trans Serv Comput*, vol. 16, no. 2, pp. 1310–1323, 2023, https://doi.org/10.1109/TSC.2022.3184835.
[26] G. Obaido *et al.*, "Supervised machine learning in drug discovery and development: Algorithms, applications, challenges, and prospects," *Machine Learning with Applications*, vol. 17, p. 100576, 2024, https://doi.org/10.1016/j.mlwa.2024.100576.
[27] Q. Zhang, M. H. B. Mahbod, C.-B. Chng, P.-S. Lee, and C.-K. Chui, "Residual Physics and Post-Posed Shielding for Safe Deep Reinforcement Learning Method," *IEEE Trans Cybern*, vol. 54, no. 2, pp. 865–876, 2024, https://doi.org/10.1109/TCYB.2022.3178084.
[28] Z. Cao, R. Wang, X. Zhou, and Y. Wen, "Toward Model-Assisted Safe Reinforcement Learning for Data Center Cooling Control: A Lyapunov-based Approach," *In Proceedings of the 14th ACM International Conference on Future Energy Systemspp,*pp. 333-346, 2023, https://doi.org/10.1145/3575813.3597343.
[29] G. Wei, M. Chi, Z.-W. Liu, M. Ge, C. Li, and X. Liu, "Deep Reinforcement Learning for Real-Time Energy Management in Smart Home," *IEEE Syst J*, vol. 17, no. 2, pp. 2489–2499, 2023, https://doi.org/10.1109/JSYST.2023.3247592.
[30] T. Hua, J. Wan, S. Jaffry, Z. Rasheed, L. Li, and Z. Ma, "Comparison of Deep Reinforcement Learning Algorithms in Data Center Cooling Management: A Case Study," in *IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pp. 392–397, 2021, https://doi.org/10.1109/SMC52423.2021.9659100.
[31] Z. Chen, J. Hu, G. Min, C. Luo, and T. El-Ghazawi, "Adaptive and Efficient Resource Allocation in Cloud Datacenters Using Actor-Critic Deep Reinforcement Learning," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 8, pp. 1911–1923, 2022, https://doi.org/10.1109/TPDS.2021.3132422.
[32] X. Wang *et al.*, "Deep Reinforcement Learning: A Survey," *IEEE Trans Neural Netw Learn Syst*, vol. 35, no. 4, pp. 5064–5078, 2024, https://doi.org/10.1109/TNNLS.2022.3207346.
[33] D. Lee, S. Koo, I. Jang, and J. Kim, "Comparison of Deep Reinforcement Learning and PID Controllers for Automatic Cold Shutdown Operation," *Energies (Basel)*, vol. 15, p. 2834, Apr. 2022, https://doi.org/10.3390/en15082834.
[34] L. Yu *et al.*, "Deep Reinforcement Learning for Smart Home Energy Management," *IEEE Internet Things J*, p. 1, Dec. 2019, https://doi.org/10.1109/JIOT.2019.2957289.
[35] Y. Li, C. Yu, M. Shahidehpour, T. Yang, Z. Zeng, and T. Chai, "Deep Reinforcement Learning for Smart Grid Operations: Algorithms, Applications, and Prospects," *Proceedings of the IEEE*, vol. 111, no. 9, pp. 1055–1096, 2023, https://doi.org/10.1109/JPROC.2023.3303358.
[36] M. Otterlo and M. Wiering, "Reinforcement Learning and Markov Decision Processes," *Reinforcement Learning: State of the Art*, pp. 3–42, Jan. 2012, https://doi.org/10.1007/978-3-642-27645-3_1.
[37] S. El Hamdani, S. Loudari, S. Novotny, P. Bouchner, and N. Benamar, "A Markov Decision Process Model for a Reinforcement Learning-based Autonomous Pedestrian Crossing Protocol," in *3rd IEEE Middle East and North Africa COMMunications Conference (MENACOMM)*, pp. 147–151, 2021, https://doi.org/10.1109/MENACOMM50742.2021.9678310.
[38] W. Zhan *et al.*, "Deep-Reinforcement-Learning-Based Offloading Scheduling for Vehicular Edge Computing," *IEEE Internet Things J*, vol. 7, no. 6, pp. 5449–5465, 2020, https://doi.org/10.1109/JIOT.2020.2978830.
[39] T. T. Nguyen and V. J. Reddi, "Deep Reinforcement Learning for Cyber Security," *IEEE Trans Neural Netw Learn Syst*, vol. 34, no. 8, pp. 3779–3795, 2023, https://doi.org/10.1109/TNNLS.2021.3121870.
[40] H. Yang *et al.*, "Deep Reinforcement Learning Based Intelligent Reflecting Surface for Secure Wireless Communications," in *GLOBECOM IEEE Global Communications Conference*, pp. 1–6, 2020, https://doi.org/10.1109/GLOBECOM42002.2020.9322615.
[41] A. Singh, R. Akash, and G. R. V, "Flower Classifier Web App Using Ml & Flask Web Framework," in *2nd International Conference on Advance Computing and Innovative Technologies in Engineering (ICACITE)*, pp. 974–977, 2022, https://doi.org/10.1109/ICACITE53722.2022.9823577.
[42] C. Wang, Z. Wang, D. Dong, X. Zhang, and Z. Zhao, "A Novel Reinforcement Learning Framework for Adaptive Routing in Network-on-Chips," in *IEEE 23rd Int Conf on High Performance Computing & Communications; 7th Int Conf on Data Science & Systems; 19th Int Conf on Smart City; 7th Int Conf on Dependability in Sensor, Cloud & Big Data Systems & Application (HPCC/DSS/SmartCity/DependSys)*, pp. 336–344, 2021, https://doi.org/10.1109/HPCC-DSS-SmartCity-DependSys53884.2021.00069.
[43] J. G. Sukumar, M. S. R. Reddy, N. Sambangi, S. Abhishek, and A. T, "Enhancing salary projections: a supervised machine learning approach with flask deployment," in *5th International Conference on Inventive Research in Computing Applications (ICIRCA)*, pp. 693–700, 2023, https://doi.org/10.1109/ICIRCA57980.2023.10220707.

[44] S. Gros and M. Zanon, "Data-Driven Economic NMPC Using Reinforcement Learning," *IEEE Trans Automat Contr*, vol. 65, no. 2, pp. 636–648, 2020, https://doi.org/10.1109/TAC.2019.2913768.
[45] Q. Chou, W. Fan, and J. Zhang, "A Reinforcement Learning Model for Virtual Machines Consolidation in Cloud Data Center," in *6th International Conference on Automation, Control and Robotics Engineering (CACRE)*, pp. 16–21, 2021, https://doi.org/10.1109/CACRE52464.2021.9501288.
[46] M. Duggan, K. Flesk, and E. Howley, "A Reinforcement Learning Approach for Dynamic Selection of Virtual Machines in Cloud Data Centres," *In 2016 sixth international conference on innovative computing technology (INTECH)*, pp. 92-97, 2016, https://doi.org/10.1109/INTECH.2016.7845053.
[47] R. Wang and M. Bonney, "Novel Data Acquisition Utilising a Flask Python Digital Twin Operational Platform," *In Special Topics in Structural Dynamics & Experimental Techniques, Volume 5: Proceedings of the 40th IMAC, A Conference and Exposition on Structural Dynamics*, pp. 7–13, 2023, https://doi.org/10.1007/978-3-031-05405-1_2.
[48] G. Schäfer, M. Schirl, J. Rehrl, S. Huber, and S. Hirlaender, "Python-Based Reinforcement Learning on Simulink Models," *In International Conference on Soft Methods in Probability and Statistics*, pp. 449–456, 2024, https://doi.org/10.1007/978-3-031-65993-5_55.
[49] H. Zhou, K. Jiang, X. Liu, X. Li, and V. C. M. Leung, "Deep Reinforcement Learning for Energy-Efficient Computation Offloading in Mobile-Edge Computing," *IEEE Internet Things J*, vol. 9, no. 2, pp. 1517–1530, 2022, https://doi.org/10.1109/JIOT.2021.3091142.
[50] S. Sierla, H. Ihasalo, and V. Vyatkin, "A Review of Reinforcement Learning Applications to Control of Heating, Ventilation and Air Conditioning Systems," *Energies (Basel)*, vol. 15, p. 3526, May 2022, https://doi.org/10.3390/en15103526.

BIOGRAPHY OF AUTHORS

Ericha Septya Dinata, received her Bachelor's degree in Telecommunication Engineering from Telkom University, Indonesia, in 2023, after learning a Diploma in Telecommunication Technology from the same institution in 2021. She is currently pursuing her Master's degree at the School of Electrical Engineering, Telkom University, Indonesia. She has been working as an Account Manager at Telkom Indonesia. Her research interests include Wirelles Optical Communicaion Network and Machine Learning.
Email: erichaseptyadin@student.telkomuniversity.ac.id, Orcid: 0009-0003-8722-2166

Sofia Naning Hertiana, received her doctorate at the Telkom University in 2014, previously received her master's degree in 2004 at the same university. In 1995 he received a bachelor's degree at Brawijaya University. Currently she works as a permanent lecturer at Telkom University in the electrical engineering study program. Her research interests include the Software Define Network, Traffic Engineering, and Network Engineering. Email: sofinaning@telkomuniversity.ac.id, Orcid: 0009-0001-2303-5754

Erna Sri Sugesti, received her doctorate degree at the University of Indonesia in 2013, previously obtained her master's science degree in 1998 at The Manchester Metropolitan University. In 1991 he received a bachelor's degree at Brawijaya University. Currently she works as a permanent lecturer at Telkom University in the electrical engineering study program. Her research interests include the Wireless Optical Communication, WLAN-over-Fiber, Radio-over-Fiber. Email: ernasugesti@telkomuniversity.ac.id, Orcid: 0009-0006-2344-4028