Investigation on reinforcement learning for dynamic firewall configuration and its effect on edge devices

ERIK GUSTAVSSON & LUDWIG LUNDSTEDT
MASTER´S THESIS
DEPARTMENT OF ELECTRICAL AND INFORMATION TECHNOLOGY
FACULTY OF ENGINEERING | LTH | LUND UNIVERSITY

Investigation on reinforcement learning for dynamic firewall configuration and its effect on edge devices

Erik Gustavsson
er5007gu-s@student.lu.se
Ludwig Lundstedt
lu4524lu-s@student.lu.se

Department of Electrical and Information Technology
Lund University

Supervisor: Debajyoti Das

Examiner: Thomas Johansson

June 2, 2026

Printed in Sweden
Tryckeriet i E-huset, Lund

Abstract

Cybersecurity has never been more relevant than it is today, as attacks become increasingly sophisticated. To keep devices safe from threats, a new type of smart firewall has become an active area of research. This type of firewall can dynamically configure itself based on incoming network traffic using reinforcement learning. This research has shown promising results; however, its impact on resource usage on edge devices with limited hardware resources has not been properly investigated. In this thesis, we will investigate how different reinforcement learning models (DQN, Double DQN, Dueling DQN, and 3DQN) and levels of complexity affect the performance of smart firewalls when deployed on edge devices. The results indicate that models with 10k-50k parameters achieve the optimal trade-off between F1 score, memory usage, and inference time.

Popular Science Summary

In today's world, the number of electronic devices used in homes, companies, and society continues to grow. As a result, potential opportunities for cyberattacks are also increasing.

Furthermore, as cybersecurity evolves to protect individuals from attacks, the attackers find new ways to bypass these protections, resulting in a constantly shifting landscape. This, combined with the growing number of attack entry points as more electronic devices are added, creates a fast-moving, complex environment.

These connected electronic devices, commonly known as Internet of Things (IoT) devices, are primarily protected by firewalls, rules within the devices that decide what traffic is allowed or blocked. Some of these IoT devices are referred to as edge devices, meaning devices where data is either created or processed. Continuously updating firewall rules on such devices is challenging, especially since different networks carry different traffic, and rules vary across networks and within devices.

To address this, researchers have introduced a smart firewall that learns from analyzing local network traffic and automatically adjusts firewall rules as needed. However, research on this topic is new and non-standardized, making it difficult to compare progress across studies. These smart firewalls also use machine learning to make decisions, a rapidly evolving field where new techniques are continually emerging but have yet to be applied in this context.

This work will therefore implement both established and emerging techniques and evaluate them in terms of memory usage, inference time, and firewall accuracy. The goal is to provide a benchmark to guide future research in the field and to indicate which implementation is most suitable for system needs.

Table of Contents

List of Figures

List of Tables

Chapter 1
Introduction

The number and sophistication of cyberattacks have increased in recent years, driven by the rapid development of digital technologies. As new tools and services are introduced, the number of potential attack vectors and exploits continues to grow [1]. Today, companies, cities, and organizations utilize the Internet of Things and cloud-edge computing to achieve efficient and optimized services. However, as more edge nodes are added, the number of interconnected systems increases, as do cybersecurity risks [2].

Firewalls are one of the primary technologies for protecting individuals and organizations from such threats. Traditional firewalls operate by filtering network traffic based on predefined rules. However, these systems struggle to adapt to the increasing volume and evolving nature of modern cyberattacks [3].

To address these limitations, recent research has introduced intrusion detection systems for edge networks that use machine learning and artificial intelligence to identify and respond to threats in real time. These approaches have demonstrated improved effectiveness in identifying complex attack patterns but typically require more computational resources than traditional solutions. Furthermore, this research area lacks standardized datasets, benchmarks, and evaluation criteria [4].

As an alternative approach, recent work has investigated the use of machine learning to automatically configure firewall rules. In particular, reinforcement learning has been explored as a method for dynamically updating firewall rules on a per-packet basis to provide real-time protection against attacks [3].

These models are trained either on a central processing unit or on the edge device itself. The trained model's weights are then used to decide which actions to take, that is, how to alter the firewall. In this paper, all models were trained on a separate computer and then transferred to the edge device for deployment and evaluation.

Although research in this area remains limited compared to work on machine learning-based intrusion detection systems, it is likely to face challenges similar to those in IDS research, including the need for efficient models and the lack of standardized evaluation frameworks.

This creates challenges for deploying smart firewalls in resource-constrained environments, such as edge devices. In these settings with limited computational power, energy constraints, and latency requirements, the feasibility of complex models is reduced. Particularly when additional services on the device must not interfere with its normal behavior [4].

This is problematic when combined with the lack of standardized datasets and evaluation frameworks, which makes comparison across the literature difficult. As a result, there is no clear notion of which methods are most suitable under different system constraints. This paper, therefore, investigates how different reinforcement learning models and levels of complexity affect the performance of smart firewalls deployed on edge devices, with a focus on camera-based systems. To guide this investigation, the following research questions are explored:

- How does reducing model complexity influence the memory and CPU usage of reinforcement learning–based smart firewall models on edge devices?
- How does reduced model complexity affect the ability of reinforcement learning–based smart firewalls to make correct decisions?
- What trade-offs exist between resource efficiency and firewall effectiveness when deploying reinforcement learning models on edge devices?

Chapter 2
Background

2.1 Reinforcement learning

Reinforcement learning (RL) is a machine learning method that learns what to do by mapping situations to actions to maximize a reward.

This method learns by having an agent interact with its environment. The agent has a set of actions to choose from, and when selecting an action to maximize its reward, it directly interacts with and influences its environment. To maximize this reward, the agent must both exploit what it already knows to make effective choices and explore new actions to make better choices in the future. The agent's behavior at a given time depends on the policy. A policy is a form of mapping from perceived states to the actions it can take in those states. At each time step, a reward signal is sent; this value defines how good or bad this action was. It is this cumulative reward received at each time step that the agent tries to maximize over the long run.

Another signal used is a value function, which defines what is good over time. The value of a state is the total reward the agent can receive from it. Whereas the reward signal defines the immediate reward that can be received, the value function is the total amount of rewards the agent can expect from that state.

Some RL methods use an environment model to approximate the environment's expected behavior. This is necessary to predict the following states and their rewards and plan which path to take. RL methods that use planning are called model-based methods. On the other hand, model-free methods learn purely by trial and error [5].

2.2 Dynamic Programming and Temporal-Difference

Dynamic Programming (DP) and Temporal-Difference (TD) are two distinct families of methods that aim to solve similar problems. Both the control problem, which concerns finding the optimal policy, and the prediction function, which estimates the value function V.

DP works by maintaining a perfect model of the environment, then using it to compute value functions and policies by simulating states and actions and propagating expected returns. This is more of a theoretical approach that works for a limited number of problems and approaches, especially tailored to them. In

practice, it requires high computational power, and having a perfect model of the environment is unrealistic.

TD, on the other hand, works by estimating the value function and optimal policy. It learns directly from experiences, i.e., previously observed transitions in the environment, and does not need a model. It works by following the policy and choosing an action. After choosing this action, it receives an immediate reward and uses the value function to see how good this new state is in the long run. An example formula for TD is defined as follows:

$$V(s_t) \leftarrow V(s_t) + \alpha \left[R_{t+1} + \gamma V(s_{t+1}) - V(s_t)\right] \tag{2.1}$$

$V(s_t)$ is a measurement of how good the current state is in the long run. The following term:

$$\alpha \left[R_{t+1} + \gamma V(s_{t+1}) - V(s_t)\right] \tag{2.2}$$

It is called the TD error and represents the difference between the predicted value and the observed return. Alpha is a constant used to reduce the impact of the TD error on the value function, since otherwise it risks completely overwriting the initial approximation in the presence of noise. Gamma is used to determine how much the value function discounts future cumulative rewards relative to immediate rewards [5].

2.3 On-policy and off-policy methods

There are two main methods by which a reinforcement learning agent learns from its environment: on-policy and off-policy. On-policy methods iteratively improve the policy currently being used to make decisions in the environment. It does this by taking actions in response to the environment's state. The outcome of the actions is then observed and used to update and improve the same policy. Off-policy methods work similarly to on-policy methods in the sense that they observe the outcome of an action; however, they do not necessarily improve the policy that is generating the agent's behavior. Instead, they can try to improve another policy, separate from the behavior policy, often called the target policy [5].

2.4 Q-learning

Q-learning is a model-free off-policy TD control method. The agent learns the action-value functions for a given state from experience and updates them using TD. Q-learning uses a Q-table that maps each state to all possible actions. Each state-action pair has a correlated Q-value indicating how good it is to take that action in the long run. For the behavior policy, an epsilon-greedy policy is often used. This is the probability of taking a random action versus the best action, to balance exploration and exploitation. Epsilon is then decreased with each iteration, since exploration is less valuable later in the training phase. Meanwhile, the target policy is the best action to be taken in the next state. The best action is the one with the highest Q-value. The Q-values are then updated using the following formula. [5]

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right] \tag{2.3}$$

2.5 Types of attacks

2.5.1 Denial-of-Service

Denial-of-Service (DoS) is an attack that consumes the target's resources, such as bandwidth, by sending a massive amount of traffic to the target. This leads to poor performance of the target's service or, in the worst case, causes the service to go down, preventing legitimate users from accessing it. In the network model used in this work, a DoS attack could target the edge device itself, for example, by sending a large number of connection requests or packets to the camera via the router from the Internet. This attack can be prevented by limiting the number of requests per IP address [6][7].

2.5.2 Distributed Denial-of-Service

Distributed Denial-of-Service (DDoS) is a type of DoS attack in which traffic originates from multiple machines simultaneously, generating a volume of traffic greater than that of an ordinary DoS attack. See 2.7.1 on prevention[8].

2.5.3 Dictionary Brute Force

Dictionary brute-force attacks attempt to access a device by trying words from a list of likely usernames and passwords to find the correct login credentials. In the context of the described network, an attacker on the Internet could target the SSH login interface. This attack can be mitigated by using complex passwords and limiting the number of allowed attempts [9].

2.5.4 Spoofing

Spoofing attacks attempt to impersonate a device by faking its IP/MAC address to appear as a trusted source and carry out malicious activities. In the network model used here, spoofing could be used against the camera or router to make malicious traffic appear to originate from a legitimate internal device or a trusted external service. By using strong authentication, this kind of attack can be prevented. [10]

2.5.5 Mirai

Botnet attacks involve infected hosts that are remotely controlled. Mirai is a type of botnet attack that infects IoT devices, turning them into bots. The attack exploits devices with weak security. Once a device is infected, the attacker can launch an attack. This is often done in coordination with other infected devices, resulting in a DDoS attack. In the described network model, the camera is a realistic target for this type of attack if it is exposed to the Internet through the router and uses weak authentication or outdated software. This type of attack can

be mitigated by keeping devices up to date and using strong authentication. To prevent the DDoS component of the attack, one can follow the same prevention methods described in Section 2.7.1 [11].

2.5.6 Vulnerability Scan

Vulnerability scanning is a reconnaissance attack in which the attacker scans various components to identify weak security configurations on a device, such as open ports and outdated service versions. In this network context, an attacker may scan the router's forwarded ports to discover whether the camera is reachable. To prevent this kind of attack, one can disable unused ports, block malicious IPs, and ensure software is up to date [12].

2.5.7 Cross-site scripting

Cross-site scripting (XSS) is a type of code injection attack in which an attacker injects malicious scripts into web applications due to poor input validation in the website's forms. This attack could be relevant if the camera had a web-based interface with insecure input fields. To prevent this type of attack, proper input form controls are needed, such as sanitizing all user input [13].

2.6 Attack selection

Given the nature of the attacks and common mitigation techniques, not all are detectable by a firewall. Furthermore, some attacks are not expected to be prevented by a firewall. For this reason, the following attacks will not be considered in this paper:

- XSS - Is not a firewall-related problem.
- Spoofing - Primarily a protocol-related problem.
- Dictionary Brute Force - Password requirements and limitations on the service side to ensure password integrity.

The list below states attacks included in the paper:

- Vulnerability Scan
- Mirai
- Distributed Denial-of-Service
- Denial-of-Service

2.7 Evaluation metrics

In order to assess the performance of the model, the following evaluation metrics were [14]:

- True Negatives (TN): The number of benign flows correctly let through.

- True Positives (TP): The number of attack flows correctly blocked.
- False Negatives (FN): The number of attack flows incorrectly let through.
- False Positives (FP): The number of benign flows incorrectly blocked.

- Recall: The number of attack flows blocked in relation to all attack flows.

$$\text{Recall} = \frac{TP}{TP + FN}$$

- F_1 score: The harmonic mean between precision and recall.

$$F_1 = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

- Accuracy: The number of correct predictions in relation to the total number of predictions.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

- Precision: The number of correctly predicted attack flows in relation to the total number of flows predicted as attacks.

$$\text{Precision} = \frac{TP}{TP + FP}$$

2.8 Artificial Neural Networks

Artificial Neural Networks (ANNs) are networks consisting of multiple layers of connected nodes, often called neurons. Each node has directed weighted edges to other nodes. The learning process in ANNs consists of multiple steps: first, each neuron computes a weighted sum of its input signals, then applies a nonlinear activation function to the result, and finally passes the output to the next layer. After the final layer, a prediction is made, and the error between the prediction and the true target value is quantified using a loss function. The error is then sent back into the network, updating each neuron in every layer [5].

2.9 Deep Q-Network

A deep Q-network (DQN) is a type of reinforcement learning model that combines Q-learning with ANNs. DQN differs from Q-learning by approximating the Q-value with a neural network rather than selecting the largest value from a table, and was introduced by Mnih et al.[15] , who modified the Q-learning algorithm in two main ways: by adding experience replay and a second network. Experience replay is a method that stores the model's experience at each time step in memory for later reuse, helping break correlations between consecutive samples. By adding a second network, they could split the work into two neural networks: an online network and a target network. The online network makes a Q-value prediction

and compares it to the target value from the target network, which is the desired Q-value. The comparison computes the error as a loss value, which is then used to update the online network's weights. The online network is therefore updated continuously, while the target network is a copy of the online network and updates its weights only occasionally. This promotes a more stable learning phase [5].

2.10 Dueling Deep Q-Network

In many states, it is unnecessary to estimate the value of each action to make a choice. This is the case because several actions have little or no effect, resulting in similar outcomes. Wang et al.[16] build on Mnih et al. research on Deep Q-Network and introduce a so-called dueling network architecture to address this problem. Compared to Mnih et al., which uses a single sequence of fully connected layers after the convolution layer, Wang et al. introduce two sequences of fully connected layers. These streams are separated, one for estimating the value and the other for estimating the advantage functions. After which, they are combined to produce a single Q function.

The formula used is given in Equation 2.4.

$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + \left(A(s,a;\theta,\alpha) - \frac{1}{|\mathcal{A}|} \sum_{a'} A(s,a';\theta,\alpha) \right) \tag{2.4}$$

Here θ denotes the parameters of the convolutional layers, and α and β denote the parameters of the two fully connected streams. The term

$$A(s,a;\theta,\alpha) - \frac{1}{|\mathcal{A}|} \sum_{a'} A(s,a';\theta,\alpha) \tag{2.5}$$

is referred to as the centered advantage function. It shows how much better or worse this action is compared to the average. Adding V+A directly would yield an unidentifiable decomposition, making it impossible to trace each parameter's contribution to the result. Furthermore, this also lacks constraints on the decomposition, which may cause parameters to drift to larger values, making learning unstable. Hence, the centered advantage function is used to ensure the advantage function has zero mean.

Equation 2.4 is implemented within the network, with V and A computed without any algorithmic modifications. This means it shares the same input-output interface as standard Q networks and can be easily combined with other Q-learning algorithms such as DDQN.

2.11 Double Deep Q-Network

DQN can be biased toward overestimation because it uses the same values to select and evaluate actions. Van Hasselt et al. [17] tackle this problem by updating the two networks individually via random sampling of experiences, resulting in two sets of weights. The online network weight, denoted θ, is used to determine the greedy

policy, denoted $\arg\max_a Q(S_{t+1}, a; \theta_t)$. The target network's weights, denoted θ', are used to determine its Q-value using the formula:

$$Q\big(S_{t+1}, \arg\max_a Q(S_{t+1}, a; \theta_t); \theta'_t\big) \tag{2.6}$$

The formula for the Q-learning target can be realized as:

$$Y_t^{\text{DoubleQ}} \equiv R_{t+1} + \gamma\, Q\big(S_{t+1}, \arg\max_a Q(S_{t+1}, a; \theta_t); \theta'_t\big) \tag{2.7}$$

2.12 Double Dueling Deep Q-Network

To reap the benefits of both Dueling and Double DQN, one can combine them into a single model. This type of model is simply called Double Dueling Deep Q-Network (3DQN) [18].

Chapter 3
Previous work

3.1 Previous work on dynamic firewall configuration

3.1.1 Firewall Configuration with Q-learning

Yang et al.[19] propose a strategy for an adaptive firewall based on Q-learning, treating the firewall configuration problem as a Markov Decision Process. The states represent observed network traffic, and rewards are given when the model takes an action that maintains the network's security and operational efficiency. Since the model iteratively interacts with the environment, it learns an optimal firewall policy to update firewall rules in response to changing attack patterns. This study shows that the model outperforms static firewalls in adaptability; however, it does not address the hardware constraints of edge-based cameras.

3.1.2 Firewall Configuration with Proximal Policy Optimization

Jha[11] presents a model similar to that used by Yang et al., treating the firewall configuration problem as a Markov Decision Process. However, in Jha's work, the model is implemented with Proximal Policy Optimization (PPO) rather than Q-learning. The model is then evaluated in a simulated network environment and demonstrates improved adaptability and detection performance compared to static and heuristic firewall policies. The report notes that, in a real-world environment, the model would require significant computing resources when retrained. However, no further hardware limitations are discussed, particularly regarding deployment on resource-constrained edge devices or independently networked cameras.

3.2 The thesis contribution to existing research

The thesis is expected to advance understanding of how different RL firewall models with varying levels of complexity affect the performance of smart firewalls deployed on edge devices in camera-based systems. The primary outcome of this thesis is a heuristic comparison of different implementations to evaluate the trade-offs between these two metrics and the importance of model selection.

Chapter 4
Methodology

4.1 Dataset

For training, the dataset CIC_DIAD2024 from the Canada Institute for Cybersecurity was used. It has two subdirectories, one being flow-based and the other packet-based. The flow-based features were extracted using CICFlowMeter, while the packet-based data points were extracted using Packet-per-Packet analysis from Pcap files. Both of these directories contain seven traffic categories: six attack categories (DDoS, Recon, Web-based, Brute Force, Spoofing, and Mirai) and one benign category. In this thesis, a flow-based dataset was used because it contains more information per sample.[20]

4.2 Feature selection

Within the flow-based dataset, 84 features were initially extracted. Reducing the dimensionality of the feature space is important, as a large number of features increases the risk of overfitting. A reduced feature set also decreases the number of parameters, resulting in faster training and inference.

To address this, a variant of *Recursive Feature Elimination* (RFE) was implemented, similar to the one proposed by Guyon et al.[21]. In the original work of Guyon et al., a linear Support Vector Machine (SVM) was trained to obtain feature weights. The weights indicate the importance of each feature and are used to remove the least important one. After a feature is removed, the SVM is retrained, and the process repeats until a desired number of features remains.

In this work, the approach is modified in two main ways. First, instead of using an SVM, a Random Forest classifier was used to estimate feature importance. Unlike linear SVM, it allows for the capture of non-linear relationships between features. Second, feature elimination is guided by a performance-based criterion. Instead of directly removing the least important feature, the model is retrained and evaluated without it. The feature is removed only if the resulting decrease in F1 is no more than 1%. If the feature is not removed, the process continues with the next-least-important feature. This process is done until no features can be removed without violating the threshold constraint.

The results showed that 22 of the original 84 features could not be removed without a significant impact on performance. Furthermore, two additional features

were added for correct firewall functionality: whether the source IP address or the destination port is currently blocked. This led to a final list of 24 features. The features can be seen below:

Feature	**Feature**	**Feature**
Flow Duration	Flow Bytes/s	Flow Packets/s
Total Length of Fwd Packet	Total Length of Bwd Packet	Fwd Packet Length Max
Fwd Packet Length Min	Fwd Packet Length Mean	Bwd Packet Length Max
Bwd Packet Length Mean	Flow IAT Mean	Flow IAT Max
Fwd IAT Max	Fwd IAT Min	Bwd IAT Total
Packet Length Max	Packet Length Mean	Packet Length Std
Fwd Segment Size Avg	Bwd Segment Size Avg	FWD Init Win Bytes
Fwd Packets/s	IP Blocked	DST Port Blocked

Table 4.1: Network traffic features used in the model

4.3 Data preprocessing

In the dataset, different feature categories appear: *numerical* (e.g., packets/s and flow duration), *categorical* (e.g., protocol), or *numerical-identifiers* (e.g., src- and DST port).

4.3.1 Numerical features

The numerical features vary a lot in size. Since the sizes of these features determine their impact on the model, it is common to scale them down so each has an equal impact. These values were normalized using Standard Scaler. It subtracts each value's mean, then divides the result by that value's standard deviation. The result is a distribution with a mean of 0 and unit variance; see equation 4.1 for the formula [22].

$$x' = \frac{x_i - \bar{x}}{s} \tag{4.1}$$

4.3.2 Numerical-identifiers

From the feature list, only two numerical identifiers were used: Src IP and Dst Port. The Src IPs are unique, and converting them to numerical values does not add any value on its own. Also, there is the risk of new IPs occurring during evaluation, whose impact is unknown. For these reasons, neither IPs nor Ports appear as features; instead, one-hot encoding indicates whether the current IP or port is blocked by the firewall. 0 indicates it is not blocked and 1 indicates it is present in the firewall rules.

4.3.3 Categorical features

Categorical features were initially considered and encoded using multi-hot. This approach converts specific protocols to binary arrays of a set length. Notably, the representation space is limited by the vector size; only the most common packet types received individual representations, while the rest were grouped. However, for the final model configuration, no categorical features were used.

4.4 Data splitting

Firstly, three datasets were created: training, evaluation, and testing. Training received 70% of the total data, 10% for evaluation, and the remaining 20% for testing. The split between attack and benign samples for each dataset was 20% attack and 80% benign traffic. All attacks were equally represented in order to avoid bias. This data split is necessary to avoid overfitting and ensure an unbiased, well-generalized model. For this reason, the training dataset was used only to train the RL model, the evaluation dataset to monitor early stopping criteria (as described in section 4.8), and the testing dataset for final model evaluation. The number of samples per attack and dataset can be seen below:

Attack	Train		Eval		Test	
	A	B	A	B	A	B
DDoS_Ack	8680	34720	1240	4960	2480	9920
DDoS_HTTP_Flood	8680	34720	1240	4960	2480	9920
DDoS_ICMP	8680	34720	1240	4960	2480	9920
Mirai	8680	34720	1240	4960	2480	9920
Vul_scan	8680	34720	1240	4960	2480	9920
DoS_HTTP	8680	34720	1240	4960	2480	9920
DoS_SYN	8680	34720	1240	4960	2480	9920
DoS_UDP	8680	34720	1240	4960	2480	9920

Table 4.2: Dataset distribution per attack class. A = attack samples, B = benign samples.

4.5 Model architecture

To find the most suitable DQN model for resource-constrained edge nodes, four DQN variants were implemented. These were the normal DQN, Double DQN, Dueling DQN, and 3DQN.

The assumption was that the number of dimensions and layers of the neural network would significantly affect the model's performance, both in evaluation metrics such as F1 score and in computational resource requirements. Therefore, the decision was made to create 12 variants of each model, with layer counts ranging from 1 to 3 and dimension sizes set to 32, 64, 128, or 256.

The network first processes the input state vector, of the same size as the number of features, and then passes it through the first hidden layer, with input states and an output size of 32, 64, 128, or 256. Then, depending on whether the model had one or two extra layers, the result would go into the next layer with input sizes 32, 64, 128, or 256 and an output size matching the input, either once or twice. DQN and Double DQN differ from 3DQN in that they maintain a single unified stream that directly estimates Q-values.

As for the 3DQN architecture, the network consists of a shared feature-extraction layer and two separate fully connected streams that estimate the state-value and action-advantage functions. However, compared with the architecture presented by Wang 2016, which used convolutional layers to process images, the selected implementation used fully connected layers throughout the model. This change was made because the environment's state is represented by numerical features rather than images.

The output is then split into two streams: a state-value stream and an action advantage stream, both represented as hidden layers with input matching the hidden dimension and an output of size 128. The two output streams have their corresponding output layers: the state-value stream's output layer has size 1, and the other has size actions. The outputs are then used to calculate the final Q-values for each action using Q(s,a)=V(s)+A(s,a)-mean(A). After each hidden

layer, the Rectified Linear Unit (ReLU) activation function is applied to introduce non-linearity into the model.

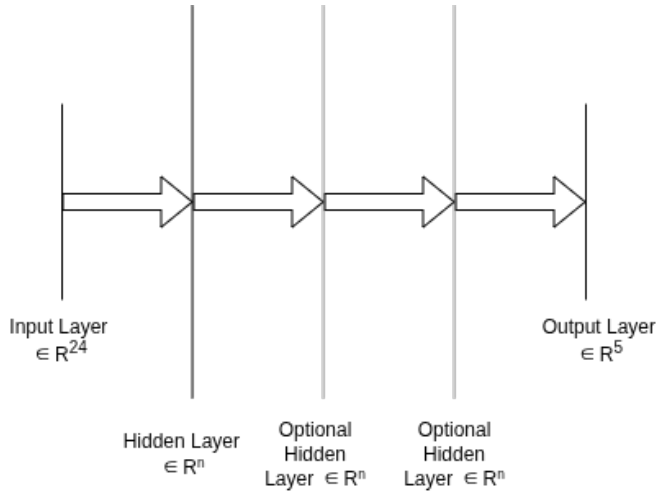


Figure 4.1: Normal DQN architecture with two optional hidden layers

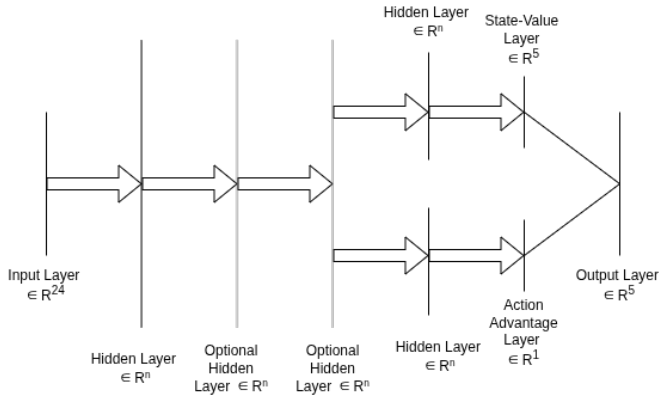


Figure 4.2: 3DQN architecture with two optional hidden layers

4.6 Simulated firewall

In this work, a simulated firewall is used to block potentially dangerous connections. The firewall contains two lists: one of blocked IP addresses and one of blocked ports.

It consists of two rules: block traffic if the source IP is on the blocked IP address list, and block traffic if the port is on the port blocklist. If neither rule is met, traffic is allowed to pass through the firewall.

For each observed traffic flow, the RL agent can modify the configuration of the firewall by choosing one of the following actions:

1. No operation
2. Block IP address
3. Remove block on IP address
4. Block port
5. Remove block on port

With these actions, the agent can dynamically update the firewall rules during execution.

4.7 Hyperparameter optimization

To determine hyperparameters such as the learning rate, rewards, epsilon decay, hidden dimensions, and batch size, the Python library Optuna was used. Specifically, the TPESampler from Optuna was utilized, which performs Tree-Structured Parzen Estimation (TPE), a variant of Bayesian optimization.

It searches for optimal parameters in a tree-structured search space. It models the distribution of parameter values associated with good and bad outcomes, sampling promising areas to improve performance while also exploring. More specifically, it selects potential candidate areas by maximizing the ratio of the probability of good outcomes to the probability of bad outcomes.

This process stops when a certain number of episodes have been reached, and the best-performing parameter configuration achieved so far is returned [23].

4.8 Model training

The reinforcement learning agents were trained using the architectures described in section 4.5. During training, the model interacts directly with a simulated firewall that can block IP addresses and ports, and rules can be added to specify which IP addresses and ports to block. The reward function is designed to give a reward depending on whether it performed the correct action, that being blocking malicious traffic or allowing benign traffic, and a negative reward for incorrect actions such as blocking benign traffic or allowing attacks. The model parameters were optimized using the Adam optimizer, and mean squared error (MSE) was used as the loss function.

To balance exploration and exploitation during training, an epsilon-greedy strategy was implemented. At each decision step, the action was chosen according to this strategy, with epsilon decreasing across episodes. In addition, a replay buffer was used. Replay buffers are an important part of Deep Q-learning to break correlations between consecutive samples, resulting in smoother learning and reducing the risk of oscillations or parameter divergence. It was implemented as suggested by Mnih et al. [15]: at each decision step, save the chosen action and its outcome, namely the next state and the rewards. Initially, during this sampling

period, no parameters are updated until a certain threshold is reached. Once it is reached, optimization starts and proceeds in batches of randomly sampled data from the replay buffer at each step.

Early stopping is also commonly used to prevent overfitting during supervised training of neural networks. This method involves choosing a stopping criterion that indicates when the model has stopped improving and is beginning to overfit the training data. A commonly used signal is the validation error, as described by Prechelt[24]. In this work, the F1-score was used instead as a stopping criterion. The training loss is computed using the mean squared error (MSE) between predicted and target Q-values, which measures the accuracy of Q-value estimation. Since the Q-function represents the expected future reward for a given action, minimizing this loss does not necessarily correspond to improved classification performance. Therefore, the F1-score was used, as it better reflects the objective of this paper: detection of malicious traffic while limiting false positives. Training was evaluated periodically on the validation set, and early stopping was triggered if the F1-score did not improve for a specified number of consecutive evaluations. After which, training was terminated, and the best-performing model was returned.

The full set of parameters used is shown in Table 4.3.

Parameter	Value	Description
Episodes	1000	Upper limit of training episodes
Discount factor (γ)	0.937	Scale factor of future rewards in the Bellman update
Learning rate	0.0022648	Step size used by the Adam optimizer
Batch size	128	Number of transitions sampled per training update
Replay buffer size	$25 \times \lvert\text{train set}\rvert$	Maximum number of stored transitions
Minimum replay size	1000	Training begins after this many transitions
Initial ϵ	0.8	Initial exploration probability
ϵ decay	0.988	Exponential decay applied each episode
Minimum ϵ	0.029	Minimum value for exploration rate
Target network update	1000 steps	Frequency of copying weights to target network
Evaluation interval	5 episodes	Frequency of F1 validation evaluation
Early stopping patience	30 evaluations	Training stops if F1 does not improve
Warm-up episodes	0	Early stopping starts after this many episodes
TP	15	Reward for correctly blocking malicious traffic
TN	5	Reward for correctly allowing benign traffic
FN	-25	Penalty for incorrectly allowing attack
FP	-15	Penalty for incorrectly blocking benign traffic

Table 4.3: Training hyperparameters used for the DQN agent

4.9 Network

A communication network is a system that enables interconnected nodes to exchange information using standardized protocols via communication links, such as electronic components. The network traffic consists of individual packets that together form flows. The packet flows have features such as duration and length. Together, these features may characterize certain types of network attacks [25]. The simulated network consists of a local network with 105 IoT devices connected to the internet.

4.10 System architecture

The system architecture considered in this thesis will comprise an edge node, in this case a camera, equipped with a simulated firewall and a reinforcement learning agent that makes decisions. A firewall works by following a set of rules to determine whether packets sent to the camera should be blocked. The firewall will be dynamic, meaning the agent will configure the set of rules based on the type of traffic that reaches the camera.

4.11 Edge deployment setup

To run inference on the model when deployed on the camera, the C API Larod was used via calls to the Larod-client service. The Larod architecture consists of two main components: liblarod, a shared library, and the Larod service, a Linux system service that communicates with liblarod. Larod wraps other frameworks, such as TensorFlow Lite, and lets users run models on different devices, including GPUs and CPUs [26].

The model trained on the local machine was converted to TensorFlow Lite format, as the Larod service expects. TensorFlow Lite is a framework developed by Google for deploying machine learning models on edge devices. It converts models, in this case a Keras model, into a TensorFlow Lite (tflite) model, an optimized FlatBuffer format. This format was designed for performance-critical applications. It provides direct access to serialized data without unpacking or parsing, reducing memory consumption.

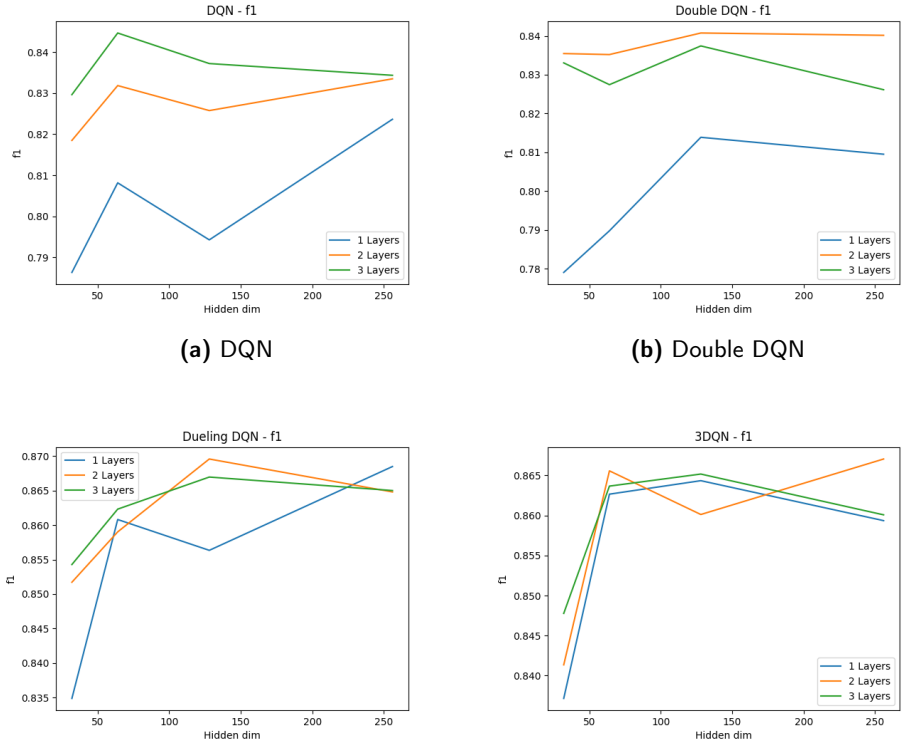
To execute the model on the edge device, a Python script was developed. The script loads the same test data used during training, which has been prepared in advance, and processes it one sample at a time. Each sample is then written to a separate CSV file because the Larod interface can accept only one line of input per inference step.

The inference output is stored in a bin file, processed by the Python script, and converted into a Q-matrix.

Chapter 5
Results

5.1 Pruning result

Instead of training all models, a few of the most promising were selected for full training to evaluate the F1 score. To find these models, all were trained on 2000 malicious samples per attack, and the F1 score for each model was saved. For each implementation type, two models that achieved a high F1 score while minimizing model complexity were selected for full training. The results of the probing are shown in Figure 5.1.



(a) DQN **(b)** Double DQN

(c) Dueling DQN **(d)** 3DQN

Figure 5.1: Comparison of models.

Based on the result provided in Figure 5.1, the following model complexities were chosen:

- DQN - 2 layers, 64 dim
- DQN - 3 layers, 64 dim
- Double DQN - 2 layers, 128 dim
- Double DQN - 3 layers, 32 dim
- Dueling DQN - 1 layer, 64 dim
- Dueling DQN - 2 layers, 128 dim
- 3DQN - 1 layer, 64 dim
- 3DQN - 2 layers, 128 dim

The F1 score per episode for the chosen models is shown in Figure 5.2. Meanwhile, the final F1 scores per attack type are shown in Tables 5.1 and 5.2.

5.2 Result for chosen models

5.2.1 F1 score graphs during training for pruned model configurations

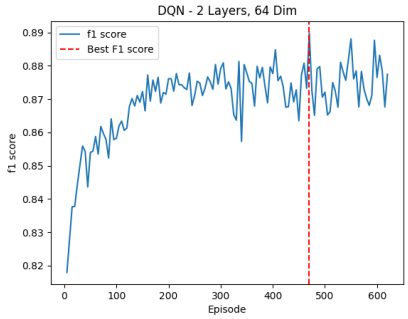
The results of the fully trained models are displayed in Figure 5.2. The red line in the graph represents the episode on which the best model was achieved. The model achieved at these points is returned. The results show similar F1 scores across models despite different configurations.

5.2.2 Attack evaluation for pruned model configurations

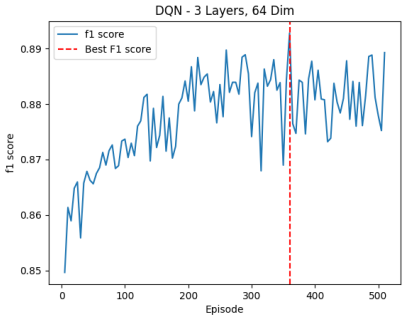
The configurations are displayed using the architecture name combined with two numbers- The first number is the number of layers, and the second number is the number of dimensions.

Model	**DDoS_Ack**	**HTTP_Flood**	**DDoS_ICMP**	**Mirai**
(a) DQN 2×64	0.9705	0.9614	0.7717	0.7347
(b) DQN 3×64	0.9705	0.9416	0.7938	0.8216
(c) Double DQN 2×128	0.9683	0.9530	0.7890	0.8005
(d) Double DQN 3×32	0.9641	0.9526	0.7675	0.7321
(e) 3DQN 1×64	0.9688	0.9507	0.7742	0.7635
(f) 3DQN 2×128	0.9597	0.9555	0.8080	0.8080
(g) DuelDQN 1×64	0.9624	0.9614	0.7897	0.8116
(h) DuelDQN 2×128	0.9674	0.9696	0.7993	0.8195

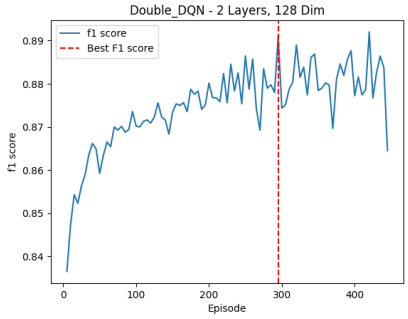
Table 5.1: Per-attack F1 score comparison across models (Part I).



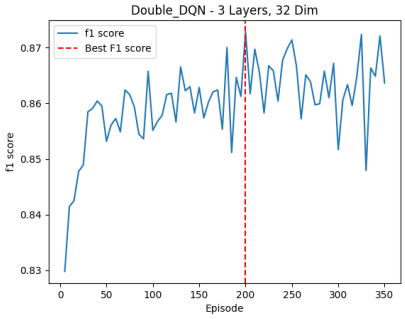
(a) DQN (2 layers, 64 dim)



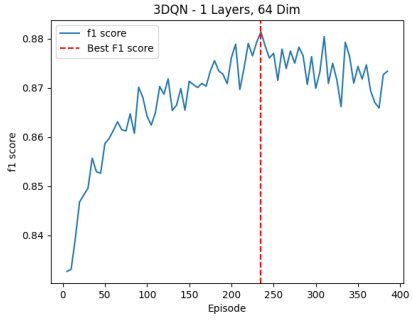
(b) DQN (3 layers, 64 dim)



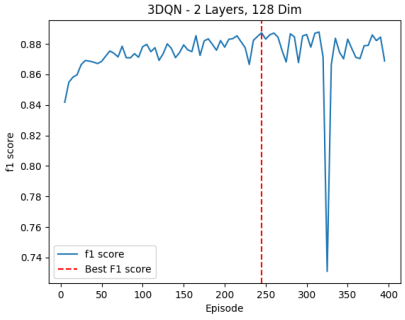
(c) Double DQN (2 layers, 128 dim)



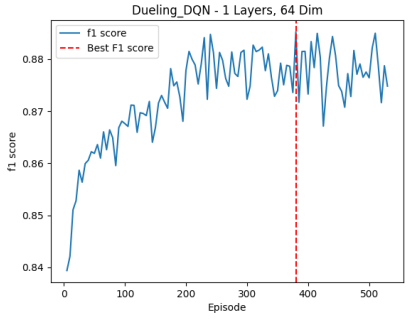
(d) Double DQN (3 layers, 32 dim)



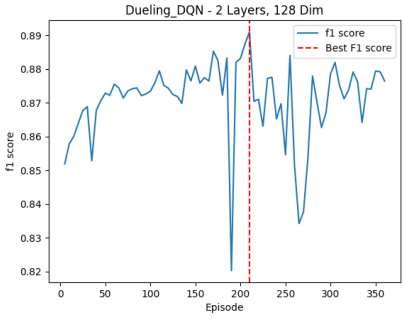
(e) 3DQN (1 layer, 64 dim)



(f) 3DQN (2 layers, 128 dim)



(g) Dueling DQN (1 layer, 64 dim)



(h) Dueling DQN (2 layers, 128 dim)

Figure 5.2: Training F1 score for the pruned model configurations.

Model	Vul_scan	DoS_HTTP	DoS_SYN	DoS_UDP
(a) DQN 2×64	0.5941	0.9575	0.9719	0.9627
(b) DQN 3×64	0.6523	0.9526	0.9689	0.9669
(c) Double DQN 2×128	0.6200	0.9539	0.9747	0.9647
(d) Double DQN 3×32	0.5904	0.9517	0.9681	0.9563
(e) 3DQN 1×64	0.5430	0.9603	0.9731	0.9582
(f) 3DQN 2×128	0.6264	0.9531	0.9649	0.9621
(g) DuelDQN 1×64	0.6053	0.9498	0.9735	0.9656
(h) DuelDQN 2×128	0.6006	0.9612	0.9775	0.9717

Table 5.2: Per-attack F1 score comparison across models (Part II).

The attacks DDoS_ICMP, Mirai, and Vulnerability scan show worse performance than the rest.

5.3 Performance comparison between all models and configurations

The inference time when deployed on a camera is shown in Figure 5.3 and is the mean time of 1000 inference samples. The results are grouped in two main ways: first, by the number of layers; and, within each layer group, by the number of dimensions, with the inference time per model configuration on the Y-axis.

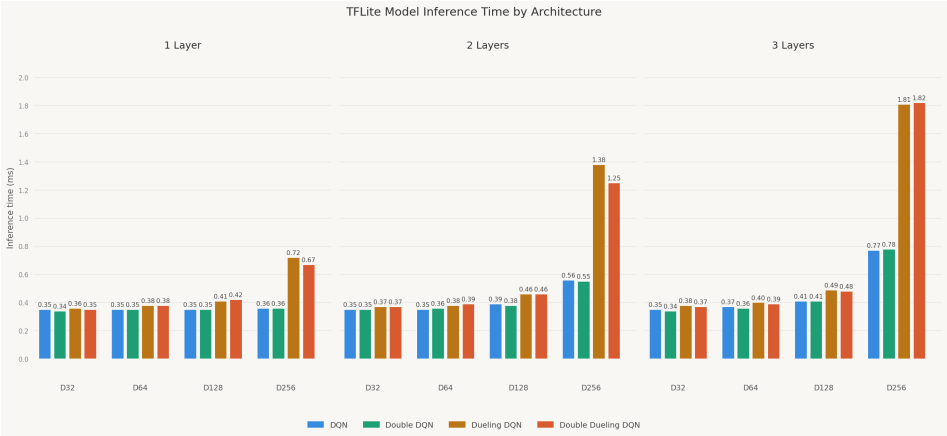


Figure 5.3: Inference time comparison

The graph shows that the number of dimensions and layers has little impact on inference time for models with fewer than 256 dimensions. However, when

the count increases to 256, a significant change is evident: the models' initial inference time increases, as does the scaling between layers. This effect is even more pronounced in the dueling architectures.

5.3.1 Model size

Below, the model size per model configuration is shown. It follows the same pattern described in section 5.2, except that the Y-axis now displays the TensorFlow model size in kilobytes.

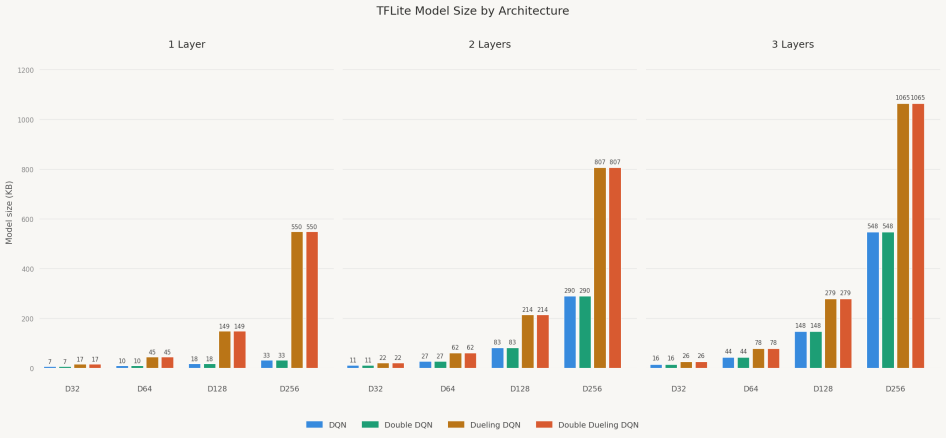


Figure 5.4: Model size comparison

Figure 5.4 shows that, for each model, size increases with layer count and dimensionality, regardless of the architecture. However, using higher dimensions leads to faster scaling of model size across layers. The use of architecture also has a significant impact on results, with dueling architectures increasing initial model size compared to non-dueling architectures. Another pattern observed is that across all layer-and-dimension combinations, the dueling architectures have the same model size. The same is true for the non-dueling architectures.

5.3.2 Number of Parameters

Below is the number of parameters per model configuration. It follows the same grouping as before, but the Y-axis displays the number of parameters.

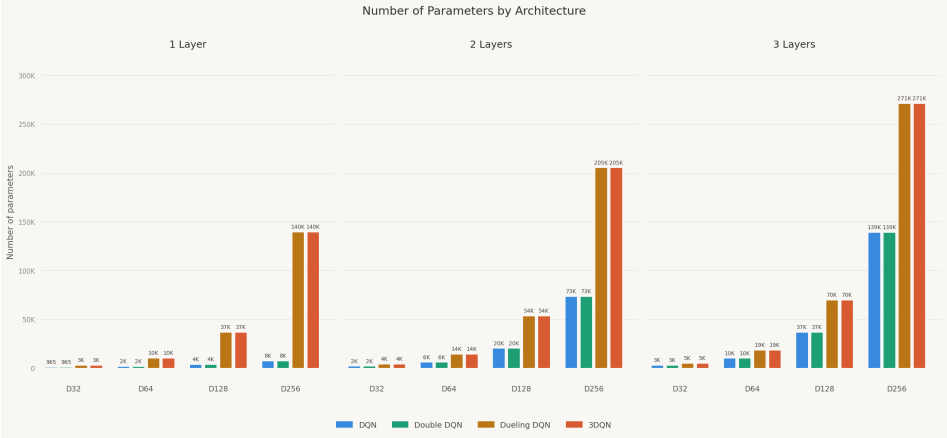


Figure 5.5: Number of parameters comparison

Figure 5.5 shows that the number of parameters scales with both the number of layers and the dimensionality. The architecture affects how many parameters a model has from the start; the dueling architecture has more.

Chapter 6
Discussion

6.1 Interpretation of results

The results in Tables 5.1 and 5.2 show that the chosen models perform very well on all DoS and DDoS attacks except DDoS_ICMP, and much worse on Mirai and vulnerability scan, despite the datasets being balanced. One reason for this might be that attacks such as DoS_HTTP, DoS_SYN, and DoS_UDP have very distinct traffic patterns, for example, repetitive request types and sudden traffic surges, therefore becoming easy to detect. In contrast, attacks like Vul_scan and Mirai generate traffic that resembles legitimate network activity. There is also the complexity aspect of the attacks. Mirai and Vulnerability scan attacks are often divided into multiple stages, each with distinct network behavior. Therefore, it becomes harder to detect than DoS attacks, which are uniform and predictable.

Additionally, since DoS-based attacks are similar and can be grouped together, one could say they account for 80% of the total dataset. The RL agent wants to generalize better to the most common attack, which in that case would be DoS-based attacks, since the model optimizes cumulative reward rather than per-class classification accuracy.

Another factor is the dataset's features. Of 84 features, 24 were selected for model training. While feature selection is applied to remove redundant features, it may also remove features that, when isolated, do not affect the model but, when combined with other features, do. This leads to the final features being biased towards more easily detectable attacks, i.e., the DoS-based ones.

6.2 Research questions

6.2.1 How does reducing model complexity influence the memory and CPU usage of reinforcement learning–based smart firewall models on edge devices?

According to the data in Figure 5.4, two primary factors significantly affect model memory consumption: the number of dimensions and the model architecture.

For lower-dimensional models, increasing the number of layers leads to slower growth in model size. For example, the transition from DQN 1x64 to DQN 2x64 represents a 170% increase in size, from 10KB to 27KB.

In contrast, the higher-dimensional models, DQN 1x256 to DQN 2x256, are substantially larger, ranging from 33KB to 290KB. This results in a significantly steeper growth rate, approximately 779%. This indicates that higher dimensionality not only increases baseline memory consumption but also accelerates the rate at which memory consumption scales with model depth.

The second factor is model architecture. The models can be divided into two categories: dueling architectures (Dueling DQN and 3DQN) and non-dueling architectures (DQN and Double DQN). Generally, the dueling models require more memory, and this difference grows with increasing dimensionality. However, they scale more slowly with increasing layer depth than non-dueling models do. All these patterns are also evident in Figure 5.5, which shows the number of parameters and indicates a strong correlation with model size.

This connection is probably due to the fact that the majority of the memory is occupied by learned weights and biases, along with their connections. Operations performed during inference are also saved, with only minor variations across architectures, resulting in only small differences in overall memory usage. Hence, increasing the number of parameters by increasing the number of dimensions and layers primarily affects memory consumption.

Inference time shows similarities to memory usage, especially at higher dimensions, resulting in longer inference times. However, the overall increase in inference across layers is relatively modest compared to the growth in memory usage, except for the dueling models with 256 hidden dimensions, which show significant growth as the number of layers increases.

As a result, the relationship between parameters and memory consumption and inference time is less direct. While more parameters imply more computations, the dominant factor appears to be the operations performed on them, not the parameter count itself. As a result, the model architecture plays a more crucial role at inference time than the number of dimensions and layers.

6.2.2 How does reduced model complexity affect the ability of reinforcement learning–based smart firewalls to make correct decisions?

The results from the initial scan (Figure 5.1) indicate that model complexity has a limited impact on the overall F1 score. An exception, however, can be observed for non-dueling architectures, where single-layer models consistently perform worse than their two- and three-layer counterparts.

This suggests that these configurations may be under-parameterized, limiting their ability to capture relevant patterns in the data. In contrast, the dueling architectures do not show the same sensitivity. A likely explanation is that these architectures introduce more parameters, enabling them to model important relationships even with lower dimensionality and depth.

When evaluating the models trained on the full dataset (Figure 5.2), the overall F1 score remains very similar across all architectures and complexity levels.

However, when evaluating the per-attack results (Tables 5.1 and 5.2), some variation in performance is observed. Mainly, differences are observed in the Mirai attack class, where some models perform noticeably worse than others. The results do not provide a clear explanation for why this is the case, but it is hypothesized

that it is related to the number of parameters. Since all models that performed worse had fewer parameters, this suggests that lower-parameterized models struggle with this task.

6.2.3 What trade-offs exist between resource efficiency and firewall effectiveness when deploying reinforcement learning models on edge devices?

Models with lower complexity, such as fewer layers and dimensions, require much less memory and CPU, making them suitable for devices with hardware limitations. However, a model with more than 10,000 parameters achieves a higher F1-score than models with fewer parameters for discrete attacks, especially Mirai.

Although more complex models can learn more detailed relationships in the data, they come with a cost: increased memory usage and inference time. The cost is generally much higher than the F1-score gain, indicating that more complex models yield diminishing returns in terms of performance per unit of resource.

When comparing the tables and figures in Section 5, it can be concluded that, in this context, where memory and CPU usage are crucial to limit, DQN and Double DQN are better choices when given enough parameters (i.e., above 10,000), as in the DQN 3x64 model. These models can capture the relationships needed to achieve good performance against attacks like Mirai, while maintaining significantly lower inference time and model size than more complex architectures.

6.3 Limitations

As of now, the model is trained on a local computer with simulated training data. This is not ideal because it makes the training unrealistic, since the training data is most likely not exactly the type the model will see when deployed.

As cyberattacks become more sophisticated, the RL agent will need to be updated to handle previously unseen attacks. This type of model update is not supported, as it was considered out of scope for this thesis.

A limitation was also found in the number of samples per dataset per attack. As with all machine learning, the size of the training data significantly impacts how well the model generalizes. However, the total number of training points was largely limited by the amount of benign data, since the datasets required four times as much benign data as attack data.

6.4 Conclusion

In this paper, a smart firewall was implemented using reinforcement learning techniques. Implementations using DQN, Double DQN, Dueling DQN, and 3DQN were evaluated to analyze differences in memory size, inference time, and firewall accuracy, with the goal of determining which model complexities and architectures are best suited for edge node deployment. The inference was analyzed on a camera edge node to see how the models perform in a realistic environment. The results show that, in this environment, DQN and Double DQN require significantly less

memory and inference time than Dueling DQN and 3DQN, while achieving similar accuracy. Furthermore, no improvement in firewall accuracy was observed with higher model complexity, indicating that complexities ranging from 10k to 50k parameters are optimal for the trade-off between F1 score, memory usage, and inference time.

Chapter 7
Future work

7.1 Inference on a real AXIS network

An interesting continuation of this work is to evaluate the pre-trained model on an actual local network. Since the data used for training and testing in this thesis come from the CIC IoT-DIAD dataset, the reported results may overestimate the model's ability to generalize to other traffic environments. It is therefore unclear how robust the Smart Firewall is when deployed on traffic generated by actual devices in a real environment. To perform such an evaluation, traffic would be collected from a real environment and processed by CICFlowMeter to match the model's expected features. This would enable assessment of the model's ability to generalize to new environments.

7.2 Combine anomaly detection with Smart Firewall

If the firewall policy generalizes poorly when deployed in new network environments, it may be necessary to retrain the model on traffic collected from the network where it will be deployed. In such a setting, an anomaly classifier could be used to label collected traffic and use it as training data for the Smart Firewall.

This introduces several interesting research questions. Firstly, since anomaly classifiers are not perfect, it is important to examine how labeling errors affect the Smart Firewall's performance. Secondly, the full pipeline would need to be optimized for the limited hardware on edge nodes. Areas of particular interest in this regard include how much data is needed for optimal performance, whether any extra hardware is needed for specific tasks such as training, and what the requirements for such hardware would be.

7.3 Per-packet model

The current model operates on traffic flows generated by CICFlowMeter. This representation provides richer information per sample, simplifying training and making traffic patterns more noticeable. This setup, however, is not a good representation of a real-world deployment, since data has to be captured and preprocessed before use. In a live network, this step would introduce additional

computational overhead and latency.

Therefore, an important next step would be to train and deploy the model using per-packet data instead. Such a study would compare the models' performance and determine how to adjust them to meet the new requirements.

7.4 Increase firewall rules

This work has primarily been a proof of concept, focusing on the viability of deploying a dynamically changing firewall on an edge device and comparing commonly used reinforcement learning methods and models for this task. As a result, more emphasis was placed on learning methods and models rather than the firewall environment itself.

It would therefore be relevant to expand the set of firewall actions and rules. Examples of such actions include thresholds, connections/s, or other rules commonly used in firewalls. More advanced architectures such as Double DQN and Dueling DQN often perform better in environments with a more complex action space. Hence, it would be interesting to investigate whether performance differences between models become more pronounced in such an environment.

7.5 Reward shaping and penalties

Another area for improvement is to refine the reward functions and add additional penalties. As shown in the results, the model performs better against DDOS and DOS attacks than against other attack types, such as Mirai and vulnerability scanning. Shaping the reward function so that successful mitigation of the more difficult attacks is more rewarded could encourage the agent to focus more on these during training.

In addition, penalties could be introduced for unnecessary or ineffective actions. During probing, it was observed that the agent tended to perform invalid actions, such as removing an inactive rule or adding a rule that already existed. These actions consume resources on the edge device without providing any benefit. Penalizing such actions while keeping NOOP penalty-free could encourage the model to learn a more efficient policy without redundant actions.

7.6 Post-training quantization

When deploying the model on an edge device for firewall purposes, it must be lightweight enough to fit within hardware resources, computationally efficient enough not to interfere with the camera's normal operation, and fast enough to keep latency low. A promising technology for improving these properties is post-training quantization. By reducing the precision of weights and activations from float-32 to float-16 or int-8, it is possible to reduce model size and inference times with minimal effects on accuracy.

Initial exploratory tests suggest that quantization may have a larger impact on DQN than other architectures. This suggests that DQN may require a more

carefully implemented model for quantization to be viable. A study of techniques for quantization robustness and trade-offs between accuracy and inference time would therefore be a valuable continuation of this thesis.

Bibliography

[1] E. Igbekele, V. Ekele, E. Omonigho, and P. Nwachukwu, "Overview of recent cyberattacks: A systematic review," in *Proceedings of the International Conference on Science, Engineering and Business for Sustainable Development Goals (SEB-SDG)*, Omu-Aran, Nigeria: IEEE, Apr. 2023. DOI: 10.1109/SEB-SDG57117.2023.10124473.

[2] M. B. A. M. N. M. A. B. I. B. M. Derdour, "Cyberattack detection in smart cities using ai: A literature review," in *2025 International Conference on Networking and Advanced Systems (ICNAS)*, IEEE, 2025. DOI: 10.1109/ICNAS68168.2025.11298103.

[3] J. T. K. K. V. V. V. N, "Neuro-adaptive firewalls: Integrating deep reinforcement learning for predictive network defence," in *Proceedings of the 2026 International Conference on Advanced Ubiquitous Computing (ICAUC)*, 2026. DOI: 10.1109/ICAUC68182.2026.11440956.

[4] M. M. Rahman, S. Al Shakil, and M. R. Mustakim, "A survey on intrusion detection system in iot networks," *Cyber Security and Applications*, vol. 3, p. 100 082, 2025. DOI: 10.1016/j.csa.2024.100082.

[5] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. Cambridge, MA: MIT Press, 2018.

[6] M. Soliman and M. A. Azer, "Web application api blind denial of service attacks," in *2018 14th International Computer Engineering Conference (ICENCO)*, 2018, pp. 249–253. DOI: 10.1109/ICENCO.2018.8636115.

[7] T. Mahjabin, Y. Xiao, G. Sun, and W. Jiang, "A survey of distributed denial-of-service attack, prevention, and mitigation techniques," *International Journal of Distributed Sensor Networks*, vol. 13, no. 12, p. 1 550 147 717 741 463, 2017. DOI: 10.1177/1550147717741463.

[8] G. Sujatha, Y. Kanchhal, and G. George, "An advanced approach for detection of distributed denial of service (ddos) attacks using machine learning techniques," in *2022 3rd International Conference on Smart Electronics and Communication (ICOSEC)*, 2022, pp. 821–827. DOI: 10.1109/ICOSEC54921.2022.9951944.

[9] C. Adams, "Dictionary attack," in *Encyclopedia of Cryptography, Security and Privacy*, S. Jajodia, P. Samarati, and M. Yung, Eds., Cham: Springer, 2025, pp. 637–638, ISBN: 978-3-030-71522-9. DOI: 10.1007/978-3-030-71522-9_74.

[10] P. R. Babu, D. L. Bhaskari, and C. H. Satyanarayana, "A comprehensive analysis of spoofing," *International Journal of Advanced Computer Science and Applications*, vol. 1, no. 6, pp. 157–160, 2010. DOI: 10.14569/IJACSA.2010.010623. [Online]. Available: http://dx.doi.org/10.14569/IJACSA.2010.010623.

[11] C. Kolias, G. Kambourakis, A. Stavrou, and J. Voas, "Ddos in the iot: Mirai and other botnets," *Computer*, vol. 50, no. 7, pp. 80–84, 2017.

[12] K. R. Houerbi, K. Salamatian, and F. Kamoun, "Scan surveillance in internet networks," in *NETWORKING 2009, LNCS 5550*, IFIP International Federation for Information Processing, 2009, pp. 614–625.

[13] A. Hannousse, S. Yahiouche, and M. C. Nait-Hamoud, "Twenty-two years since revealing cross-site scripting attacks: A systematic mapping and a comprehensive survey," *arXiv preprint arXiv:2205.08425*, 2022. DOI: 10.48550/arXiv.2205.08425. [Online]. Available: https://arxiv.org/abs/2205.08425.

[14] X. Zhang, "A study on the selection of performance evaluation metrics for machine learning methods," in *Proceedings of the ICCBD+AI Conference*, 2024.

[15] V. Mnih et al., "Playing atari with deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015. DOI: 10.1038/nature14236.

[16] Z. Wang, T. Schaul, M. Hessel, H. Van Hasselt, D. Silver, and N. De Freitas, "Dueling network architectures for deep reinforcement learning," *arXiv preprint arXiv:1511.06581*, 2016.

[17] H. van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," *arXiv preprint arXiv:1509.06461*, 2015.

[18] N. Khan, A. Y. Al-Tamimi, A. Bermak, and I. M. Khalil, "Adaptive malware detection using sequential feature selection: A dueling double deep q-network (d3qn) framework for intelligent classification," *CoRR*, vol. abs/2507.04372, 2025. DOI: 10.48550/arXiv.2507.04372. arXiv: 2507.04372 [cs.LG].

[19] M. Yang, Z. Yang, and Q. Yang, "Adaptive firewall strategy generation and optimization based on reinforcement learning," *Informatica*, vol. 49, no. 33, 2025. [Online]. Available: https://informatica.si/index.php/informatica/article/view/9363.

[20] Canadian Institute for Cybersecurity, *CIC IoT-DIAD 2024 Dataset: A Dual-Function Dataset for IoT Device Identification and Anomaly Detection*, https://www.unb.ca/cic/datasets/iot-diad-2024.html, Accessed: 2026-05-29, 2024.

[21] I. Guyon, J. Weston, S. Barnhill, and V. Vapnik, "Gene selection for cancer classification using support vector machines," *Machine Learning*, vol. 46, no. 1-3, pp. 389–422, 2002.

[22] R. de Amorim et al., "The choice of scaling technique matters for classification performance," *arXiv:2212.12343*, 2022, Accessed: 2026-02-10. [Online]. Available: https://arxiv.org/abs/2212.12343.

[23] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," *arXiv preprint arXiv:1907.10902*, 2019.

[24] L. Prechelt, "Automatic early stopping using cross validation: Quantifying the criteria," *Neural Networks*, vol. 11, no. 4, pp. 761–767, 1998. DOI: 10.1016/S0893-6080(98)00010-0.

[25] S. Raghavan, *Communications networks*, in *Encyclopedia of Wireless and Mobile Communications*, Springer, 2011. DOI: 10.1007/978-1-4419-1153-7_135. [Online]. Available: https://link.springer.com/rwe/10.1007/978-1-4419-1153-7_135.

[26] Axis Communications. "Introduction for app developers (larod api)." Accessed: 2026-04-10. [Online]. Available: https://developer.axis.com/acap/api/src/api/larod/html/md__opt_builder-doc_larod_doc_introduction-for-app-developers.html.

Printed by Tryckeriet i E-huset, Lund 2026

LUND
UNIVERSITY

Series of Master´s theses
Department of Electrical and Information Technology
LU/LTH-EIT 2026-1131
http://www.eit.lth.se