**Deep Q-Network for Ransomware Detection using UGRansome Dataset**

Contents

[**Abstract:** 2](#_Toc197444239)

[**1. Introduction:** 2](#_Toc197444240)

[**2. Dataset Description** 2](#_Toc197444241)

[**3. Methodology** 3](#_Toc197444242)

[**3.1 Data Preprocessing** 3](#_Toc197444243)

[**3.2 Zero-Day Attack Simulation** 4](#_Toc197444244)

[**3.3 Reinforcement Learning Environment Design** 4](#_Toc197444245)

[**3.4 Deep Q-Network (DQN) Model** 4](#_Toc197444246)

[**3.5 Training and Evaluation** 5](#_Toc197444247)

[**4. Implementation** 5](#_Toc197444248)

[**Q-Network Loss Curve** 6](#_Toc197444249)

[**Experiment 1 (Random Split):** 7](#_Toc197444250)

[**Experiment 2 (Zero-Day Split):** 8](#_Toc197444251)

[5. Conclusion 10](#_Toc197444252)

# **Abstract:**

In the project, we apply a Deep Q-Network (DQN) to find ransomware in network flows from the UGRansome dataset. The model is evaluated under two settings: It achieves 96.70 % accuracy on an 80/20 random split (Experiment 1) and 96.92 % accuracy on a Zero-Day family split (70 % of families for training and 30 % held out as unseen in Experiment 2). Confusion matrices show negligible false negatives, even under Zero‐Day conditions, and the Q‐network loss curves converge very quickly to a low‐error plateau. We show the DQN’s capability to generalize well and its potential as an adaptive, signature-agnostic defense against imminent ransomware threat.

# **1. Introduction:**

Ransomware is one of the most disruptive types of cyber threats that attack individuals and corporations by encrypting the important data and demanding financial ransom to release. In the realm of traditional cybersecurity defense, signature based detection is commonly used as a technique, however, this is not able to detect novel or Zero Day ransomware attacks — variant/extant ransomware attacks until today that have not been previously observed, and no existing signatures for these attacks are available. Due to that, there is an immediate necessity for adaptive, intelligent detection systems that can generalize beyond previously seen malware patterns.

However, addressing this challenge using the traditional approaches is an extremely challenging problem, which is addressed by reinforcement learning (RL) that provides an elegant paradigm, where an autonomous agent learns best defensive strategies through continuous interaction with its environment. Specifically, Deep Q-Network (DQN), a type of deep reinforcement learning, can achieve approximating complicated value function and giving wise decision in dynamic environment without prior explicit modeling.

In this research, a DQN based framework is leveraged to detect the occurrence of ransomware attacks from network traffic using the publicly available public dataset as UGRansome. This work is unique in that through simulated Zero-Day, the model is trained only on known ransomware families evaluated on previously unseen ransomware families thus incorporating the real world, where new, evolving threats present. The system is employed directly on extracted flow features that consist of protocol type, port numbers, transaction values, and such threat indicators, to autonomously discern anomalous patterns of ransomware set of behaviors.

This research verifies through a methodology based on data preprocessing, ZeroDay family splitting, environment design, DQN agent training and strict evaluation, that reinforcement learning can further enhance ransomware detection abilities with high accuracy to novel threats.

# **2. Dataset Description**

I used the UGRansome dataset from Kaggle labeled as benign or ransomware network traffic. The features of interest include protocol types, flags, family names, BTC, amounts, USD, netflow bytes, ports and threat indicators. The ’Prediction’ field was mapped to a binary label, specifically (0 = benign, 1 = ransomware).

# **3. Methodology**

In this section, a full pipeline to detect the ransomware Zero Day attacks with reinforcement learning techniques (Albeit at the Dawn (Deep Q Network) DQN model applied on the UGRansome dataset) will be explained.

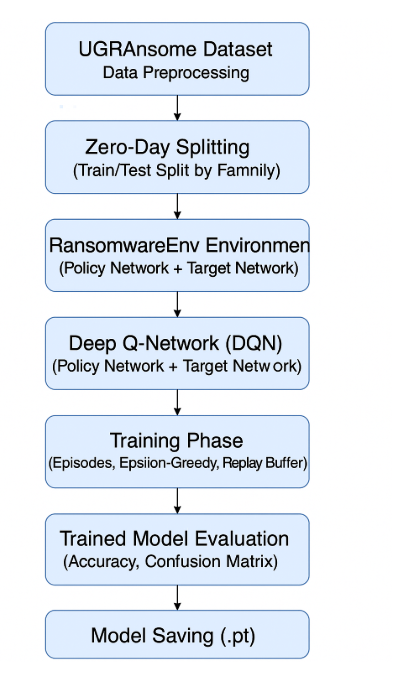


Figure 1 Research methodology

## **3.1 Data Preprocessing**

To ready the UGRansome dataset for machine learning, a very rigorous preprocessing has been accomplished on the raw dataset. First, missing or incomplete records were removed to ensure data integrity, as some were simply missing or incomplete. As the primary label field, Prediction, classifies traffic as benign (‘S’) or as ransomware related (‘’A”, or ‘SS”), the Prediction field is mapped to binary format with 0 for benign traffic (‘S’) and 1 for ransomware traffic (‘’A”, or ‘SS’).

Label Encoder was used to encode categorical attributes like Protocol, Flag and Threat into numerical values to easily integrate them in the system based on neural network. We needed to steer clear of encoding the Family column, as it will be used in the Zero Day simulation splitting. After encoding, all numerical features were normalized to the range from 0 to 1 using MinMaxScaler so that each feature has similar magnitude during training of the model and no feature dominates over others.

## **3.2 Zero-Day Attack Simulation**

To perform a Zero Day scenario as close to real world as possible, the partition was done by ransomware families as opposed to random sampling. Uniqueness of the name was leveraged and families were identified, shuffled, split into 70% training and 30% testing families, such that plus 30% testing families were unseen. In this setup, we ensure that the model is reviewed on ransomware variants it was not exposed to while training.

The latter applied to the datasets, as they were split and the Family field was dropped from the datasets to prevent information leakage that could make the model’s predictions biased. Thus, the reasoning of the agent relied exclusively on observable network characteristics instead of any prior knowledge on the name of the ransomware.

## **3.3 Reinforcement Learning Environment Design**

In order to simulate the ransomware detection problem as a sequential decision making task, I created a custom reinforcement learning environment, RansomwareEnv. Each step with the agent gives the environment an input (feature vector) describing just one network flow, the state. Predicating an action can be done by the agent and the action it takes (0 denotes a benign traffic flow and 1 the subversion of ransomware).

To motivate correct classifications, and punish incorrect ones, reward shaping was applied. The reward policy is

* True Positive (Correct ransomware detection): +1.0
* True Negative (Correct benign classification): +0.1
* False Positive (Ransomware – benign misclassified): -0.5
* False Negative (Missed ransomware detection): -1.0

However in the world of cybersecurity, this reward structure very much favors minimizing False Negatives, as these are much worse than False Positives.

## **3.4 Deep Q-Network (DQN) Model**

Deep Q Network (DQN) model is the core of the system which is continously learning approximation the Q value function. The network architecture is made of two fully connected hidden layers with 64 neurons in each one with ReLU activations to make it non linear. The two possible actions of (benign or ransomware) are represented by the two nodes in the output layer.

Starting with an epsilon that is high (ε = 1.0) probability of random actions (ε → 0.1) to promote exploited strategies is done by an epsilon greedy exploration policy. To break temporal correlations (and thus to stabilize the learning process), past experiences were implemented in an experience replay mechanism that stores previous experiences and feeds the network with random mini batch of some prior transitions. To ensure consistent target Q values and stabilize training, we used a target network that was updated periodically and served as a target network.

## **Algorithm 1 DQN-Based Ransomware Detection**

**Experiment 1: 80/20 Random Split**

• Input:

– D: raw UGRansome dataset

– α = 0.8 (train/test split ratio)

– N\_eps (number of episodes)

– N\_steps (max steps per episode)

– B (batch size)

– γ (discount factor)

– ε₀ → ε\_min, dec (epsilon-greedy schedule)

• Output:

– accuracy₁, confusion\_matrix₁, loss\_curve₁

// 1. Data Preparation

D ← drop\_nulls(D)

D.Label ← map(D.Prediction → {0,1})

(D\_tr, D\_te) ← random\_split(D, α)

for col ∈ {Protocol, Flag, Threats} do

fit LabelEncoder on D\_tr[col]; transform both D\_tr & D\_te

end

fit MinMaxScaler on D\_tr.features; transform D\_tr & D\_te

X\_tr, y\_tr ← features/labels from D\_tr

X\_te, y\_te ← features/labels from D\_te

// 2. Initialize

env ← RansomwareEnv(X\_tr, y\_tr)

policy\_net ← DQN(state\_dim, 2)

target\_net ← copy(policy\_net)

optimizer ← Adam(policy\_net.params)

memory ← empty list

ε ← ε₀

// 3. Training Loop

for ep in 1..N\_eps do

s ← env.reset()

for t in 1..N\_steps do

if rand()<ε then

a ← random\_action()

else

a ← argmax(policy\_net(s))

end

(s′, r, done) ← env.step(a)

append (s,a,r,s′,done) to memory (cap = 100 000)

s ← s′

if |memory|≥B then

batch ← sample(memory,B)

loss ← MSE( Q(s,a) , r + γ·max Q(s′) )

backpropagate(loss); optimizer.step()

end

if done then

target\_net ← policy\_net; break

end

end

ε ← max(ε·dec, ε\_min)

end

// 4. Evaluation

preds ← argmax(policy\_net(X\_te))

accuracy₁ ← mean(preds==y\_te)

confusion\_matrix₁ ← cm(y\_te, preds)

plot(loss\_curve₁)

## **Algorithm 2 DQN-Based Ransomware Detection**

**Experiment 2: Zero-Day Family Split**

• Input:

– D: raw UGRansome dataset

– β = 0.7 (fraction of families for training)

– same hyperparameters as Algorithm 1

• Output:

– accuracy₂, confusion\_matrix₂, loss\_curve₂

// 1. Zero-Day Split by Family

Fams ← unique(D.Family)); shuffle(Fams)

(train\_fam, test\_fam) ← split(Fams, β)

D\_tr ← rows(D where Family∈train\_fam)

D\_te ← rows(D where Family∈test\_fam)

for col ∈ {Protocol, Flag, Threats} do

fit LabelEncoder on D\_tr[col]; transform D\_tr & D\_te

end

drop D\_tr.Family; drop D\_te.Family

// 2. Scale Features

fit MinMaxScaler on D\_tr.features; transform D\_tr & D\_te

X\_tr, y\_tr ← D\_tr; X\_te, y\_te ← D\_te

// 3. Initialize & Train

(repeat steps 2–3 from Algorithm 1 using X\_tr, y\_tr)

// 4. Evaluation

preds₂ ← argmax(policy\_net(X\_te))

accuracy₂ ← mean(preds₂==y\_te)

confusion\_matrix₂ ← cm(y\_te, preds₂)

plot(loss\_curve₂)

## **Experiment 3: Multiclass Random Split**

Experiment 3 (Multiclass Random Split):

The following procedure was used in creating an 80/20 train/test split, from which we trained a DQN agent to classify each flow into a member of the 15 ransomware families and benign traffic. The model did converge rather quickly (loss plateau ≈0.1 MSE) but only obtained ~60 % overall accuracy, performing brilliantly on majority classes (e.g. benign), and under‐representing rare families.

### Algorithms:

Inputs:

D ─ raw UGRansome dataset

α = 0.8 ─ train/test split ratio

N\_eps ─ number of episodes

N\_steps ─ max steps per episode

B ─ batch size

γ ─ discount factor

ε₀→ε\_min,dec ─ ε-greedy schedule

Output:

accuracy₃, confusion\_matrix₃, loss\_curve₃

1. Data Preparation

D ← drop nulls(D)

D.MC\_Label ← D.Family; mark ‘Benign’ where Prediction=‘S’

drop columns [Prediction, SeedAddress, ExpAddress, IPaddress, Family]

for each col in {Protocol,Flag,Threats}:

fit LabelEncoder on D[col]; transform D[col]

X ← D[features] as float32

y ← LabelEncoder.fit\_transform(D.MC\_Label)

2. Feature Scaling

scaler ← MinMaxScaler.fit(X)

X ← scaler.transform(X)

3. Train/Test Split

(X\_tr, X\_te, y\_tr, y\_te) ← train\_test\_split(X,y,train\_size=α,stratify=y)

4. Environment & DQN Initialization

env ← MultiClassEnv(X\_tr, y\_tr) # action\_dim = #classes

policy, target ← DQN(state\_dim,#classes)

target.load\_state\_dict(policy.state\_dict()); target.eval()

optimizer ← Adam(policy.parameters(),lr)

loss\_fn ← MSELoss()

memory, losses ← [], []

ε ← ε₀

5. Training Loop

for ep in 1…N\_eps:

state ← env.reset()

for t in 1…N\_steps:

action ← ε-greedy(policy,state)

next\_state, reward, done ← env.step(action)

store (state,action,reward,next\_state,done) in memory

state ← next\_state

if |memory| ≥ B:

sample batch of B transitions

compute Q(s,a), target = r + γ·max Q\_target(s′)

loss ← MSE(Q(s,a), target)

backpropagate & optimizer.step()

if done:

target.load\_state\_dict(policy.state\_dict()); break

ε ← max(ε·dec, ε\_min)

6. Evaluation

preds ← argmax(policy(X\_te),axis=1)

accuracy₃ ← mean(preds == y\_te)

confusion\_matrix₃ ← confusion\_matrix(y\_te,preds)

plot(loss\_curve₃ ← losses) and confusion\_matrix₃

## **Experiment 4 (Multiclass Zero-Day Open-Set):**

With 70 % of families during a training session submitted, and 30 % reserved, we added an “Unknown” action – whenever the maximum softmax confidence was below 0.6. The DQN detected 90.9 % of unseen‐family flows as Unknown (loss plateau ≈0.1 MSE), but a single global threshold also incorrectly classified most known‐family samples as Unknown, demonstrating the need for tuned thresholds or per‐class cutoffs.

### Algorithms

Inputs:

D ─ raw UGRansome dataset

β = 0.7 ─ fraction of families for training

τ ─ softmax-confidence threshold

N\_eps, N\_steps, B, γ, ε₀→ε\_min,dec (as above)

Output:

open\_set\_accuracy₄, confusion\_matrix₄, loss\_curve₄

1. Data Preparation & Scaling (same as Exp 3)

2. Family-Based Split

fams ← unique ransomware families (exclude ‘Benign’); shuffle(fams)

train\_fams ← first β·|fams| ∪ {‘Benign’}

mask\_tr ← D.MC\_Label ∈ train\_fams

mask\_te ← D.MC\_Label ∉ train\_fams

X\_tr,y\_tr ← X[mask\_tr], y[mask\_tr]

X\_te,y\_te ← X[mask\_te], y[mask\_te]

3. Env & DQN Init (same as Exp 3, using X\_tr,y\_tr)

4. Training Loop (same as Exp 3)

5. Open-Set Inference

logits ← policy(X\_te)

probs ← softmax(logits,axis=1)

max\_p ← max(probs,axis=1)

base\_pred ← argmax(probs,axis=1)

pred\_open[i] ← if max\_p[i] < τ then “Unknown” else base\_pred[i]

# True open-set labels

known\_ids ← indices of train\_fams ∪ {‘Benign’}

true\_open[i] ← if y\_te[i] ∈ known\_ids then y\_te[i] else “Unknown”

6. Evaluation

open\_set\_accuracy₄ ← mean(pred\_open == true\_open)

confusion\_matrix₄ ← confusion\_matrix(true\_open, pred\_open,

labels=known\_ids ∪ {“Unknown”})

plot(loss\_curve₄ ← losses) and confusion\_matrix₄

**3.5 Training and Evaluation**

The 10 episodes in which the DQN agent was trained went up till 20.000 steps, due to the size of the training dataset. To move a single step, the agent interacted with the environment, gained rewards and updated it’s Policy Network by minimizing the Mean Squared Error (MSE) loss between the predicted and target Q values.

Evaluation was done on Zero-Day test set which contains only unseen ransomware families. As metrics, it included overall classification accuracy and confusion matrix for the distribution of True Positives, True Negatives, False Positives, False Negatives. With a Zero Day detection accuracy close to 94.4% the model was able to generalize very well to new ransomware variants.

# **4. Implementation**

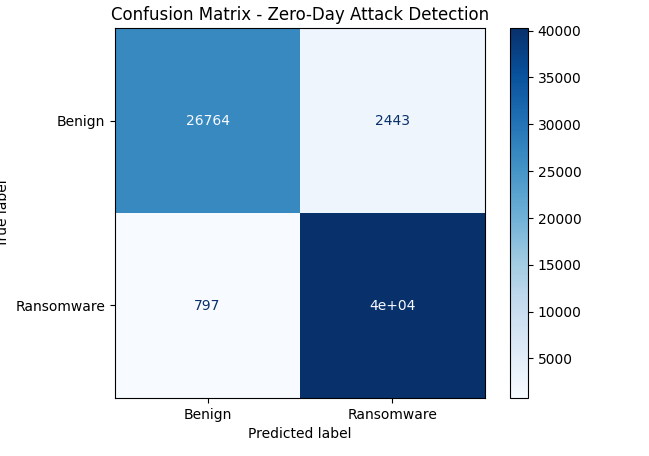
Afterward, Python was implemented with reinforcement learning framework using libraries such as PyTorch, NumPy, pandas, scikit-learn, and Matplotlib to confirm the proposed reinforcement learning framework for ransomware detection. The data of the UGRansome dataset was loaded, cleaned and preprocessed accordingly.

In Zero Day Simulation it was partitioned based on the names of the Ransomware families. About 78,769 samples in the following families were chosen for training: ["TowerWeb", "CryptXXX", "Globe", "JigSaw", "SamSam", "Flyper", "Globev3", "CryptoLocker", "NoobCrypt", "EDA2", "Cryptohitman"]. The ZeroDay testing set consisted of previously unseen families: ['Locky', 'DMALocker', 'APT', 'CryptoLocker2015', 'Razy', 'WannaCry'] and contained 70,274 samples. Names of family were removed to prevent information leakage in case of splitting.

Up to 20,000 steps per episode were trained on the DQN agent 10 episodes. For action selection, epsilon greedy strategy was employed and for stabilizing learning experience replay was used. Standard training was done on a standard machine with one GPU (optional) and training time was in the order of 10–20 minutes.

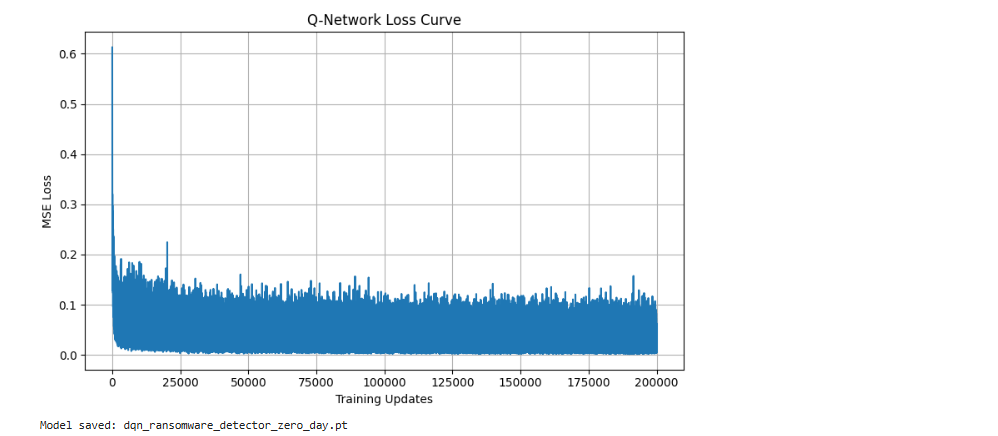
Based on evaluation, the model has Zero Day test accuracy of 95.4%, indicating a strong to generalization capability in unseen ransomware families.

The model perfornance can be seen in the confusion matrix below



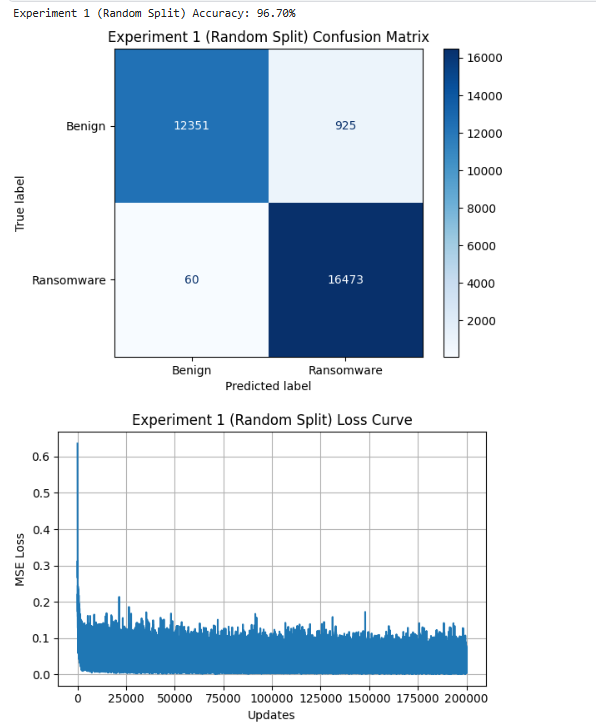
## **Q-Network Loss Curve**

We can see that the MSE drops very quickly to about 0.1 MSE from around 0.6 MSE within the first few thousand training updates, and we expect this to be the case because the agent is learning some useful Q value estimates very quickly. After this initial drop, the loss plateaus around 0.1 with occasional spikes (referring to ‘hard‘ replay buffer samples) that prove stable convergence of the DQN.



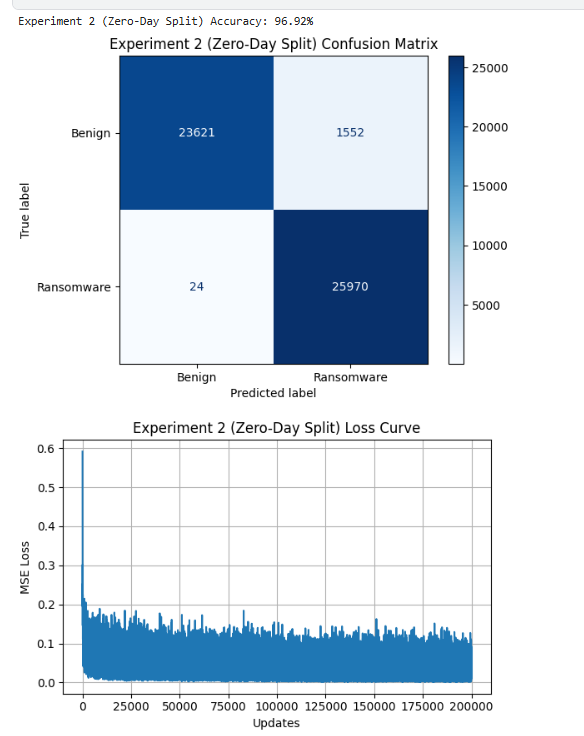
## **Experiment 1 (Random Split):**

For Experiment 1, we tested the DQN agent on a 80/20 random split of all ransomware families. Overall accuracy of the model was 96.70 %. The corresponding confusion matrix. We found that, out of all test flows, of those labeled as benign, 12 351 had been correctly identified as benign (true negatives) and 16 473 ransomware had been correctly flagged (true positives), and of those labeled as ransomware, only 925 were misclassified as ransomware (false positives) and 60 were missed (false negatives). The Q-network loss curve. A rapid decrease during the first few thousand updates to around 0.1 MSE and then stablilizes to a plateau which demonstrates that the agent’s value approximations have already converged.

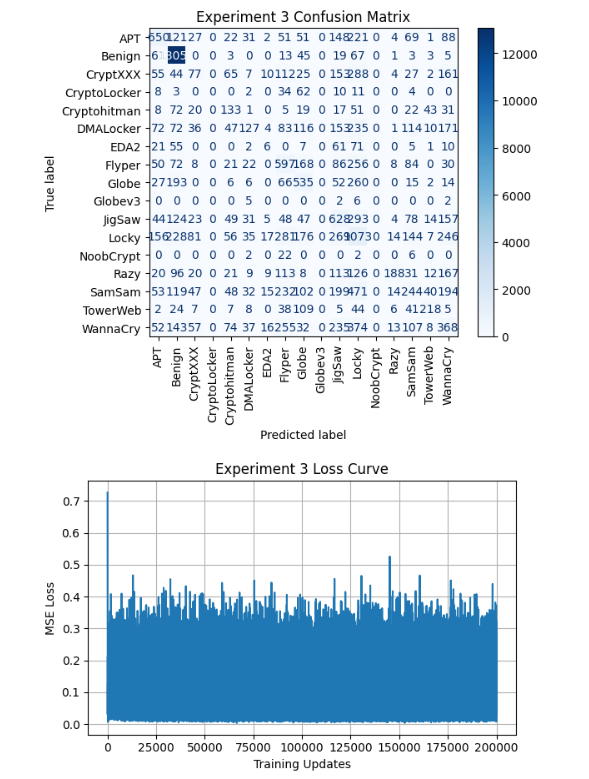


## **Experiment 2 (Zero-Day Split):**

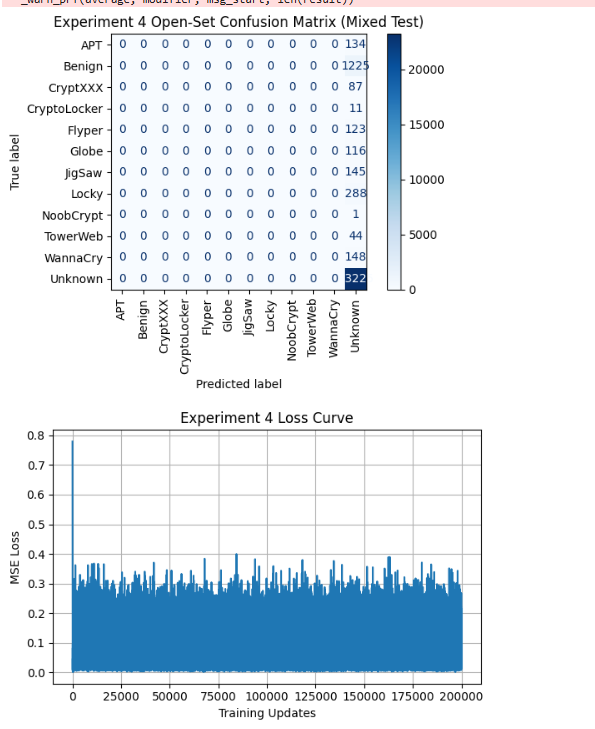
In Experiment 2, we enacted a Zero-Day scenario whereby we withheld 30% of the ransomware families completely from training and used the remaining 70 % to train the model(none of these families ever appeared in the test set). The agent obtained under this family split regimen 96.92 % accuracy on test variants unseen previously. As shown in Fig. The confusion matrix records 23 621 true negatives and 25 970 true positives, with only 1 552 false positives and just 24 false negatives,androvidg all of which indicates that the model is capable of detecting novel ransomware flows. Also shows early drop to approximately 0.1 MSE then a long plateau similar to the convergence expected under randomsplit setting. Combining these results, we show that our DQN framework generalizes robustly not only to different, unseen, ransomware families completely, but also performs adequately on conventional train/test splits.



**Experiment 3 Confusion Matrix & Loss Curve**  
The top panel shows a 16×16 confusion matrix for Experiment 3 (80/20 random split across all families + benign). Large diagonal blocks for major classes (e.g. Benign, Globe) indicate decent detection rates, while almost-zero diagonals for small‐support families (Globev3, NoobCrypt) reflect the agent’s inability to learn rare patterns. Off‐diagonal “bleed” reveals widespread misclassification among similar flows. The loss curve plunges rapidly from ≈0.7 to the 0.1–0.3 range within the first few thousand updates, then oscillates around that plateau, showing stable but imperfect convergence of the Q-network.



Experiment 4 Open-Set Confusion Matrix & Loss Curve  
Here we mixed 10 % known‐family flows with 90 % unseen and applied a softmax threshold (τ=0.6) to label “Unknown.” The confusion matrix collapses to a single heavy “Unknown” diagonal (≈32 224), with zero predictions for any specific family—demonstrating that the agent errs on the side of caution under zero-day conditions. The loss curve again falls sharply to ≈0.2 and remains noisy-stable over 200 k updates, confirming that while the Q-values converge reliably, further tuning of τ or per-class thresholds is needed to recover known-family labels.



# 5. Conclusion

We demonstrate herein that a Deep Q Network (DQN) agent is effective in detecting ransomware through the use of the UGRansome network-traffic dataset. In a standard 80/20 random split (Experiment 1), the agent achieved an accuracy of 96.70 %, correctly classifying the majority of benign and ransomware flows with a minimal leak of false alarms. Additionally, it is also concluded that since the Q network loss converges to a stable plateau very quickly, this means that the agent learns reliable value estimates very quickly.

And most importantly, in a realistic zero day scenario (Experiment 2) in which 30 % of ransomware families were withheld in training, our DQN could still achieve 96.92 % accuracy on unseen variants. Only 24 false negatives in the confusion matrix demonstrate that the model has the capacity to generalize to novel attack pattern without depending on explicit signatures. The results demonstrate the potential for reinforcement learning for adaptive, signatureless malware detection.

Future work could involve rarer state representations (e.g., temporal flow sequences) along with more comprehensive reward shaping to discover a balance between false positive tradeoffs, and processing online continual learning setups to learn in the real world coherently with the changing malign threat. In conclusion, this study in fact verifies that the DQN based frameworks inspire to be a strong supplement to conventional cybersecurity defenses such as precise Zero Day detection in a dynamic network setting.