National College of Ireland

Configuration Manual

MSc Research Project

MSc in Artificial Intelligence

Vishnu Kumar Javvaji

Student ID: 23153539

School of Computing

National College of Ireland

Supervisor: Rejwanul Haque

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Vishnu Kumar Javvaji		
Student ID:	23153539		
Programme:	MSc in Artificial Intelligence	**Year:**	2023-2024
Module:	MSc Research Practicum		
Lecturer:	Rejwanul Haque		
Submission Due Date:	12/08/2024		
Project Title:	Adaptive Network Intrusion Detection Using Deep Reinforcement Learning		
Word Count:	……1400…… **Page Count:** ……10……		

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Vishnu Kumar Javvaji
Date:	12/02/2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on the computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Vishnu Kumar Javvaji
Student ID: 23153539

1. Introduction

This project is oriented to the development of a Network Adaptive Intrusion Detection System using Deep Reinforcement Learning and, in particular, a Deep Q-Network. The main objective of the study is to increase the accuracy of NIDS by realising the power of DRL in detecting and reacting to network intrusions in real time.

The DQN model is trained in this paper using the CICIDS 2017 dataset consisting of all variations of normal and malicious network traffic: several classes of attacks, such as Brute Force, Denial-of-Service, Infiltration, Botnet, and so on.

These two critical techniques, experience replay and epsilon decay, are used within the model to make it robust and more effective in learning. This is coming from the fact that experience replay allows the model to learn from previous experiences, by memorising and sampling them in a random way within recent experiences, breaking the correlation among them. Epsilon decay will ensure the model explores different actions sufficiently in the beginning stages of training before slowly shifting to policy exploitation.

2. Setting Up the Environment

2.1. Link to Google Colab

Google Colab is a free cloud service for running Jupyter notebooks aimed at helping students and working professionals execute their scripts in the safari of a web browser. Especially handy for my machine learning projects, where powerful GPUs accelerate model training. Here's how it's done:

Open Google Colab:

Open any browser in your Computer and navigate to search bar and search for google colab

Login With gmail credentials to start.

Add a new notebook:

Once the colab is opened Click on +New notebook.

Runtime configuration:

At the very top of the Colab window, click Runtime > Change runtime type.

Select the appropriate option from the dropdown in the Hardware accelerator section. For example, you can choose GPU. Click "Save." This is essential when one wants to train deep learning models, as it will be done more quickly.

2.2 Installation of Appropriate Libraries

You will need to install quite a few Python libraries running this project successfully. It is a set of libraries that makes it possible for you to do all your data processing, model building, training, and evaluation. You should be able to install them by just running this command in a Colab code cell.

```
!pip install numpy pandas matplotlib &&
!pip install -q scikit-learn torch seaborn
```

This command performs the following installation:

numpy: For numerical operations.
Pandas: Manipulation and analysis of data.
matplotlib: Plotting and visualisation.
scikit-learn: To use for data preprocessing and model evaluation. torch:
PyTorch library to build and train the DQN model.
seaborn – Better looking visualisations.

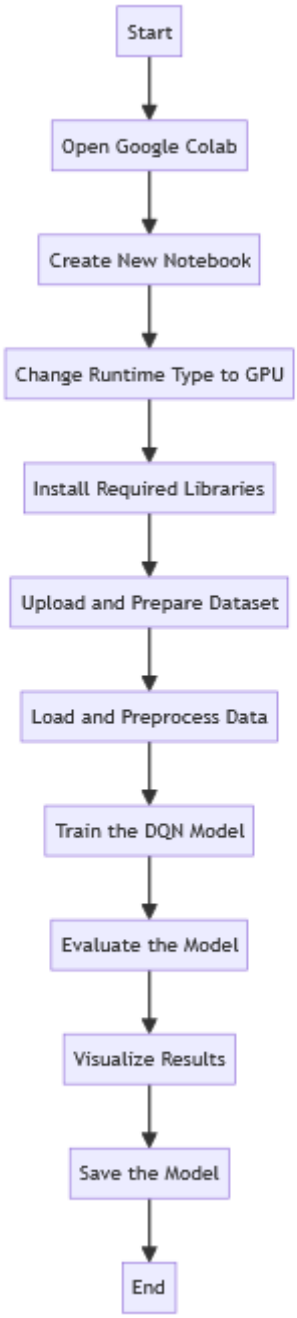

Fig 1: Workflow of Setting Up the Environment and Running the Model

3. Uploading and Preparing Data

3.1 Steps for Uploading Necessary Files

The dataset is uploaded and prepared using the following steps:

Download the dataset:

Download the dataset from CICIDS 2017 Dataset: The dataset must be in the format of a CSV file.(https://paperswithcode.com/dataset/cicids2017)

Load Dataset on Colab:

On the left sidebar of your Colab notebook, click on the folder.

You click on the upload button, which is an upward arrow inside of a file, to upload a data set from your local machine into the COLAB environment.

Check the Upload:

After uploading, ensure this dataset is in your working directory by running:

!ls

This command will list all the files in the working directory so you can be sure your data set is properly uploaded.

3.2 Updating Folder Structures as Needed

Once you upload the files, proceed to make sure that your code correctly matches up all the file path locations in the Colab space.

For instance, if you have a data wrangling that you named dataset1_ids.csv, you can load it into a pandas DataFrame as:

```
        return x
import pandas as pd

file_path = 'dataset1_ids.csv'
dataset = pd.read_csv(file_path)
```

Update all the paths related to your code in correspondence to the dataset and other files.

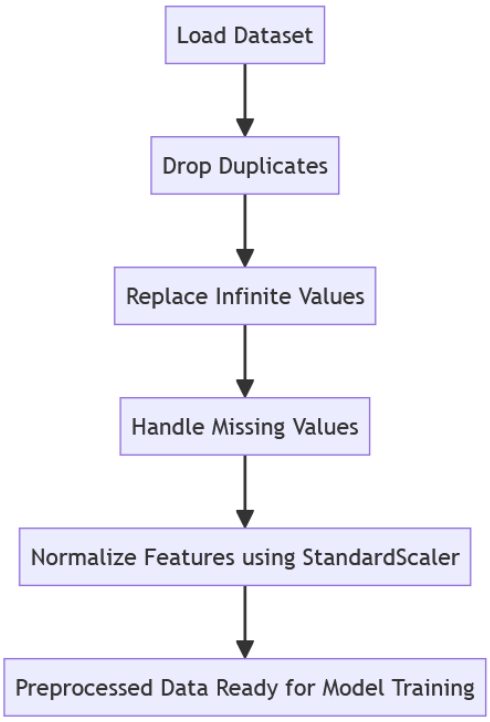

Fig 2: Data Preprocessing Workflow

4. Running the Model/System

4.1 Instructions for Running the Model

Now, let's go ahead in steps, after an appropriate setup of the environment and data preparation, with running the DQN model.

Load and Preprocess Data:

Load the dataset into a DataFrame using the pandas library.

Clean data from duplications, missing values, and values normalisation.

Sample Code:

```
# Data Cleaning: Removing duplicate entries and handling missi
dataset = dataset.drop_duplicates()
dataset.replace([np.inf, -np.inf], np.nan, inplace=True)
dataset = dataset.dropna()

# Feature Extraction: Selecting relevant features and labels
features = dataset.drop(columns=['Label'])
labels = dataset['Label']

# Convert labels to categorical and then to integer codes
labels = labels.astype('category')
label_mapping = dict(enumerate(labels.cat.categories))
labels = labels.cat.codes

# Normalization: Standardizing the feature values
scaler = StandardScaler()
features_normalized = scaler.fit_transform(features)
```

DQN Model Training:

Define the DQN model with the architecture that includes an input layer, a hidden layer, and an output layer.

Train the model using experience replay and epsilon decay over a few epochs. While you are training the model, make sure to train it using a validation set for evaluating its performance.

Example training loop:

```
# Training loop with epsilon decay
for epoch in range(num_epochs):
    dqn_model.train()
    correct_train = 0
    total_train = 0
    epoch_loss = 0
    for X_batch, y_batch in train_loader:
        optimizer.zero_grad()
        outputs = dqn_model(X_batch)
        loss = criterion(outputs, y_batch)
        loss.backward()
        optimizer.step()
```

Model Evaluation:

Check the evaluation metrics post-training, which include accuracy, confusion matrix, and ROC curve.
Example scoring code:

```
# Confusion matrix for the DQN model on test data
dqn_model.eval()
with torch.no_grad():
    outputs = dqn_model(X_test_tensor)
    _, predicted = torch.max(outputs, 1)
    cm = confusion_matrix(y_test, predicted)
```

Visualise Results:

Plot the training and validation accuracy and loss and show the ROC curve using Matplotlib.

Sample code for visualisation:

```
# Plot training and validation accuracy
plt.figure(figsize=(12, 6))
plt.plot(range(1, num_epochs+1), training_accuracy, label='Training Accuracy')
plt.plot(range(1, num_epochs+1), validation_accuracy, label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.title('Training and Validation Accuracy')
plt.show()
```

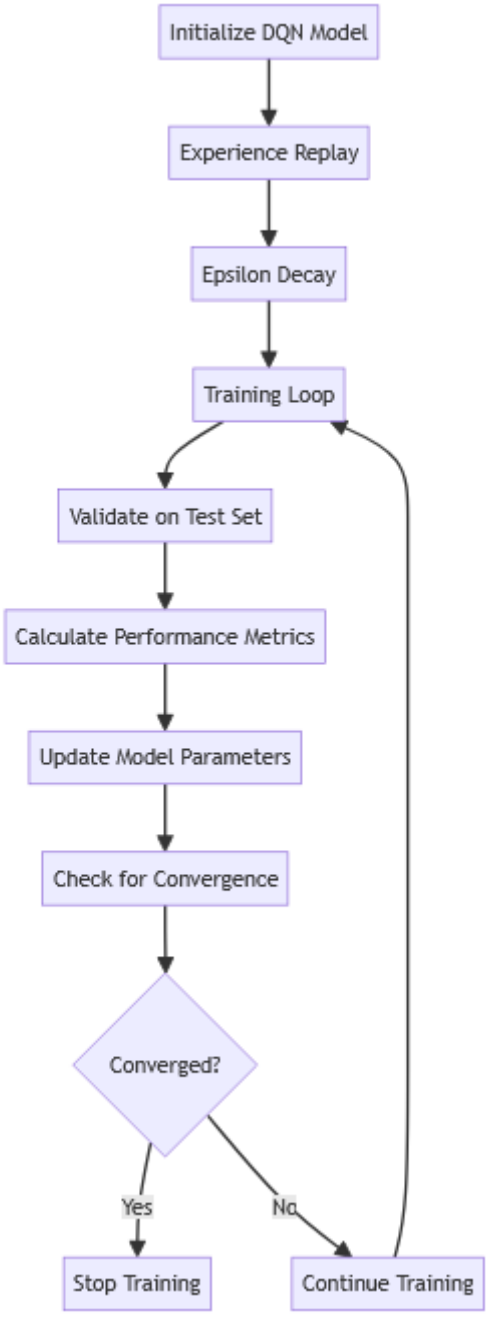

Fig 3: DQN Model Training Process

4.2 Saving and Loading Models

It is important to save the trained model so that you can reuse it without retraining. Here's how to save and load the model:

Saving the Model:

```
torch.save(dqn_model.state_dict(), 'dqn_model.pth')
```

Loading the Model:

```
dqn_model.load_state_dict(torch.load('dqn_model.pth'))
```

By saving the model, you ensure that you can deploy or further evaluate it without having to go through the training process again.

5. User Interface Overview

5.1 Explanation of User Interface

The user interface for this work majorly involves the outputs and visualisations as exhibited under the Colab notebook. Here are the main walkthroughs of those:

Confusion Matrix:

The confusion matrix presents the number of correct and incorrect predictions for each class, indicating where the model may go wrong. This part is important to correctly understand the performance displayed by the model both for normal and malicious network traffic.

ROC Curve:

The ROC curve plots the trade-off of the true positive rate across different threshold settings against the false positive rate. The Area Under the Curve (AUC) provides a measure of performance, in which high values indicate good discrimination of classes.

Precision and Loss Graphs:

Keep these plots on the screen to observe accuracy and model improvements during training, which allows you to monitor the learning process. Consistent improvement in validation accuracy with decreasing loss seems to be effective learning.

5.2 Tips for Effective Use

Keep count while moving:

The accuracy and loss plots should be monitored periodically during training to observe if the learning is actually being done effectively by the model. If overfitting is observed—specifically, an increase in the training accuracy while the validation accuracy drops off—consider early stopping or hyperparameter adjustment.

Evaluate Regularly:

Use confusion matrices and ROC curves at different stages of training so that you are able to continue gauging the classification performance of your model on a continuous basis. It helps in recognising problems early and making appropriate changes.

Restoration, if needed:

Notice how the model performs across classes using the confusion matrix. Indeed, a high false positive rate suggests a need for more tuning, especially in a sensor-intensive mode of application like NIDS.

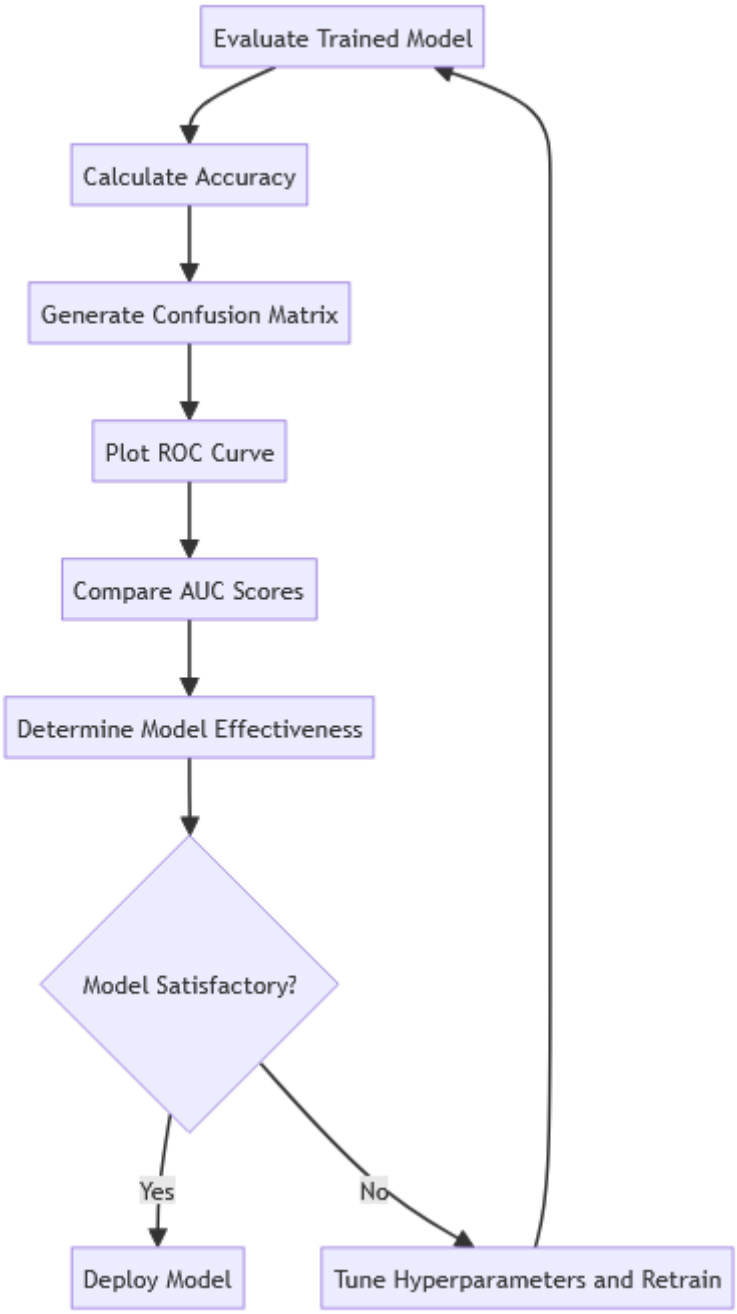

Fig 5: Troubleshooting Flow

6. Troubleshooting

6.1 Common Issues and Solutions

File Path Errors:

In case of any 'no such file' kinds of error, double-check that you have specified your file paths correctly. To be safe from all the file location-based problems in Colab, use absolute paths.

Memory Errors:

Memory problems might occur while attempting to work with very large datasets or complex models. One can try to defeat such limitations by reducing the batch size or working with a smaller subset of the data.

Library Conflicts:

Make sure you have all the additions and most up-to-date modules installed: If you notice that there are discrepancies with these, you can upgrade them by using pip:

!pip install --upgrade library_name

Model Convergence Issues:

If a model is not converging or its loss is fluctuating, this could be improved by decrementing or incrementing values, such as the learning rate, batch size, or epochs, and testing the variations accordingly. Also, exploring new neural network architectures that give improvable performances is worthwhile.

7. Conclusion

This manual is a step-by-step guide to the configuration, implementation, and debugging of an Adaptive Network Intrusion Detection System running on the principle of Deep Reinforcement Learning. You should have the ability to execute the DQN model to ensure the proper updating of the target and current Q-values in improved network intrusion real-time detection with this system. It embodies yet another historic improvement in the cybersecurity landscape through harnessing the power of DRL toward adaptation to ever-changing and threat-evolving environments.

References

Google. (n.d.). Google Colaboratory. Retrieved August 11, 2024, from https://colab.research.google.com/

Scikit-learn developers. (n.d.). Scikit-learn: Machine learning in Python. Retrieved August 11, 2024, from https://scikit-learn.org/

TensorFlow developers. (n.d.). TensorFlow: An end-to-end open-source machine learning platform. Retrieved August 11, 2024, from https://www.tensorflow.org/

Hunter, J. D., Droettboom, M., & Caswell, T. A. (n.d.). Matplotlib: Visualisation with Python. Retrieved August 11, 2024, from https://matplotlib.org/

Python Software Foundation. (n.d.). PyPI: The Python Package Index. Retrieved August 11, 2024, from https://pypi.org/