2nd Report of the project "ns-3 network simulator running Machine Learning Algorithms in python"

Participants:
Gabriel Aldair Villagrán Saucedo
UASLP
267572@alumnos.uaslp.mx

Jose Saul Castillo Ipiña
UASLP
292461@alumnos.uaslp.mx

Advisor
Dr. Pedro David Arjona Villicaña
UASLP
david.arjona@uaslp.mx

Teacher
Dr. Juan Carlos Cuevas Tello
UASLP
cuevas@uaslp.mx

[illegible]

Gabriel Aldair Villagran Saucedo

Pedro David Arjona Villicaña

Abstract
Reinforcement Learning is a branch of Artificial Intelligence that attempts to mimic the human brain allowing it to learn large amounts of data and to become more accurate at performing actions in an environment based on feedback in order to maximize the reward that learns how to make decisions through simulations or in real time **(taking the data that is being processed for testing and training)**. Reinforcement Learning algorithms can learn to make decisions that have never been thought and faster, we can make use of this kind of algorithms through the physical layer that will improve the speed of the packets that are sent and received.

Keywords
RL (Reinforcement Learning), DL (Deep Learning), CLI (Command Line)

---

1 Introduction

In this document you will find a brief explanation of the examples that can be found inside the ns3-ai module. The module allows cross layer optimization which removes boundaries to allow communication between layers, permitting one layer to access the data of another layer to exchange information and enable interaction.
ns3-ai is divided into two different options figure 1:

1. Offline Training
2. Online Training

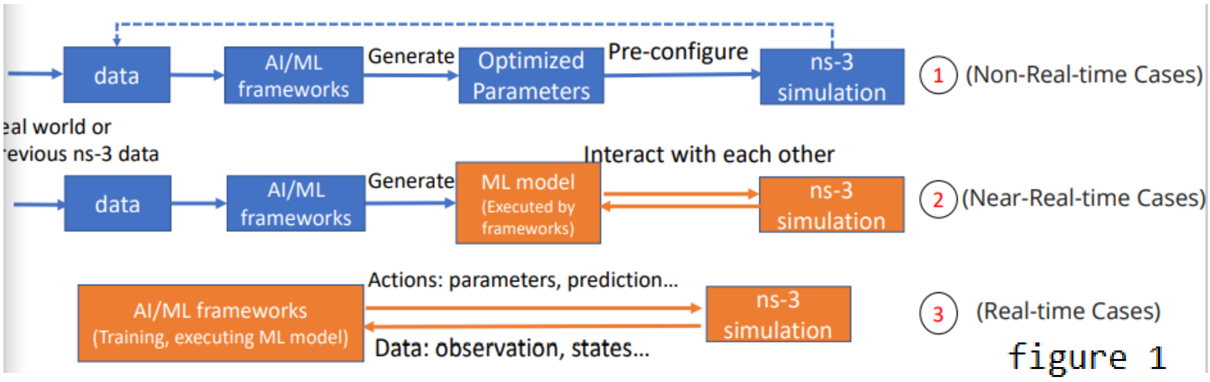


figure 1

The next examples are ready to run inside the module:

1. a_plus_b
2. interface_pattern
3. lte_cqi
4. RL-TCP
5. rate-control
6. q-tcp

Each of these examples works differently and implements distinct AI algorithms, the interference_pattern code, will be detailed explained below.
In this paper you will learn to run in two different ways the examples inside the ns3-ai module, using waf to get the log information and/or the visualization mode, the other

one just running the .py file that is inside the directory scratch, where all the examples need to be run.
Moreover you will find an explanation of the codes and what they are doing when they are executed.

---

2 Offline Training and Online Training

The ns3-ai module has different options to interact with the data:

1. Offline Training:
   - Non Real Time Cases.
   - Near Real Time Cases.
2. Online Training:
   - Real Time Cases.

Offline Training

The model is first trained with offline data (it can be data from the real world or data that is already inside of ns-3 simulator), an offline training is when we already have the data ready to run and to train.test the models.

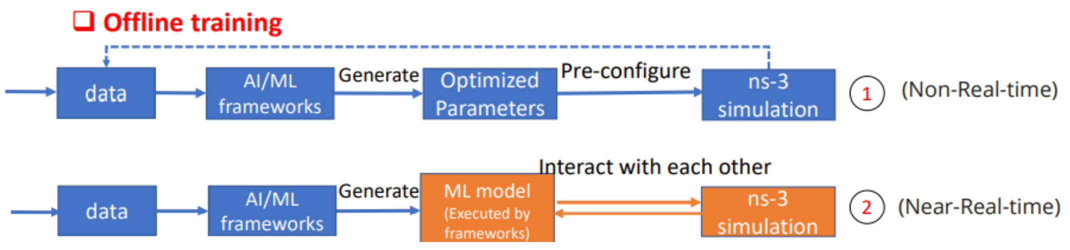


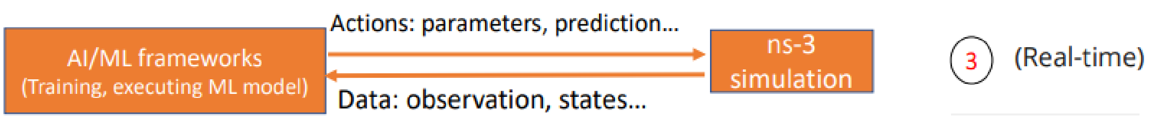
figure 2

There are two ways to use the model in network/simulation which are show on figure 2:

- Configure the parameters before the simulation (power, ACK timeout threshold, sensing power).
- Make decision dynamically during the simulation (Execute the model to adjust the parameters in simulation)

Online Training

Online training are those scenarios that use real time interaction, we can see the diagram on figure 3 for instance, all the data is getting obtained from the simulation when this one is running and at the time that this one is saved is sent to the AI frameworks to do the training and testing .



AI/ML frameworks (Training, executing ML model)
Actions: parameters, prediction...
Data: observation, states...
ns-3 simulation
3
(Real-time)

figure 3

Scenarios such as reinforcement learning, where the model needs to 'learn' as it is executing in the network needs to interact with ns-3 in real time, some of the examples that use reinforcement learning algorithms are:

- CQI
- RL-TCP
- WIFI Rate Control

---

3 Examples Overview

- ***a_plus_b***

This example is very useful, it shows us how the data is exchange between python-based AI frameworks and ns-3, in other words its the helloWorld code of the ns3-ai module.
We have variables a and b that are inside ns-3, then we put them into the shared memory using python to calculate the sum (c = a + b) once c is calculated, we put the value back into the ns-3
Run:
Copy this example to scratch:

```
cp -r contrib/ns3-ai/examples/a_plus_b scratch/
```

once the code is copy to the directory

```
cd scratch/a_plus_b > python3 run.py
```

- ***lte_cqi (cellular network)***

This example is based on NR work (New Radio (NR) network simulator, designed as a pluggable module to ns-3). The objective of this example is to show how to change the source code to use the ns3-ai module

Build and run
We have two options to run this example that are show below:

1. **This option is no longer recommended but is important to understand how it works**

   ```
   pip3 install -r requirements.txt
   cp -r contrib/ns3-ai/example/lte_cqi scratch/
   ./waf --run "lte_cqi"
   ```

   Open another CLI window (terminal) and run the Python code

   ```
   cd scratch/lte_cqi/
   python3 run_online.py
   ```

2. **ns3 code and Python code need to run simultaneously**

```
cd scratch/lte_cqi/
python3 run_online_lstm.py 1
```

- ***q_tcp (Q_Learning)***

Q_Learning is a type of reinforcement learning that performs values- based actions to reach an optimal solution, this reinforcement learning method does not require model knowledge, only observed rewards from many experiments runs.
In this example Q-Learning (SARSA) algorithm is applied into TCP congestion control for real-time changes in the environment of network transmission, by strengthening the learning and threshold size the network can get better throughputs and smaller delay.
The output will be the number of packets received by the right node.

SARSA

SARSA algorithm is a variation of Q-Learning algorithm,does not require model knwoledge instead requires observed rewards from many experiment runs, in SARSA the algorithm choose another action following the same current policy.

- On policy: The learning agent learns the value function according to the current action derived from the policy currently being used.
- Of policy: The learning agent learns the value function according to the action derived from another policy.

In RL the output actions and rewards will determine the performance model.

SARSA Reinforcement Learning - GeeksforGeeks

TCP Congestion Control

TCP uses a congestion window and a congestion policy that avoid congestion. We assumed that only the receiver can dictate the sender's window size.

Build an run

```
cd contrib/ns3-ai/example/q-tcp/
python3 run_q_tcp.py --use_rl --result
```

- ***rate-control***

This example shows how to develop a new rate control algorithm for the wifi model that is in ns-3 using the nas3-ai model.

What is rate control?
Is a mechanism that is able to adapt to various channel conditions and dynamically choose the optimal bitrate for the current conditions. The physical layers devices are capable of transmitting at several different rates. The different rates can use different channel access methods, rate control algorithms adapt the transmission rate dynamically to the changing channel conditions, so the performance of the radio link can be maximized.

IEEE 802.11 Rate Control — INET 4.3.0 documentation (omnetpp.org

What is Thompson Sampling Rate Control (Thompson Sampling)?
Also known as Posterior Sampling or Probability Matching is an algorithm for choosing the actions that address the exploration-exploitation.
An action is performed several times using information that evaluates the actions taken and is called exploration.
Based on the results of those actions, we will get a reward (1) or penalties (0), further actions are performed to improve the performance
Build an run

```
cp -r contrib/ns3-ai/example/rate-control scratch/
cd scratch/rate-control
```

Constant Rate Control

```
python3 ai_constant_rate.py
```

Thompson Rate Control

```
python3 ai_thompson_sampling.py
```

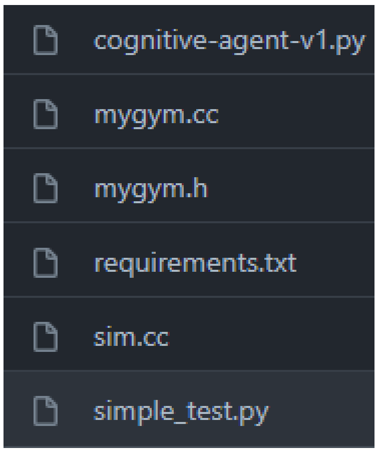
- ***RL-TCP***

Works the same way as q-tcp but the differences is that in this example reinforcement learning (Q-Learning) are applied, the example test the TcpNewReno congestion control algorithm
Build an run

```
cd contrib/ns3-ai/example/rl-tcp/
python3 run_tcp_rl.py --use_rl --result
```

4 Interference_Pattern

The interference_pattern is an example that was originally created for the ns3-gym module, the developers adapted the code for the ns3-ai module, inside the directory of the example we have these different files figure 4.



```
cognitive-agent-v1.py
mygym.cc
mygym.h
requirements.txt
sim.cc
simple_test.py
```

figure 4

4.1 Code analysis

requirements.txt [ns3-ai/requirements.txt at master · hust-diangroup/ns3-ai (github.com)]

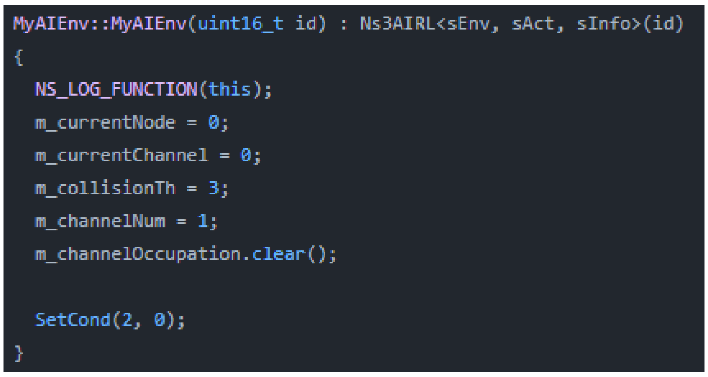
- In **requirements.txt** we have all the packages that need to be installed before running the simulation, in order to do this we execute on the CLI (terminal) the next command: pip3 install -r requirements.txt**.**

mygym.cc [ns3-ai/mygym.cc at master · hust-diangroup/ns3-ai (github.com)]

- What can be found in this file are functions that will help to develop theLO information, some of the functions that can be found here are:

`NS_LOG_COMPONENT_DEFINE("MyAIEnv"); → Define myAIEnv as a component of NS_LOG`

`MyAIEnv::MyAIEnv(uint16_t id) : Ns3AIRL<sEnv, sAct, sInfo>(id)` figure 5



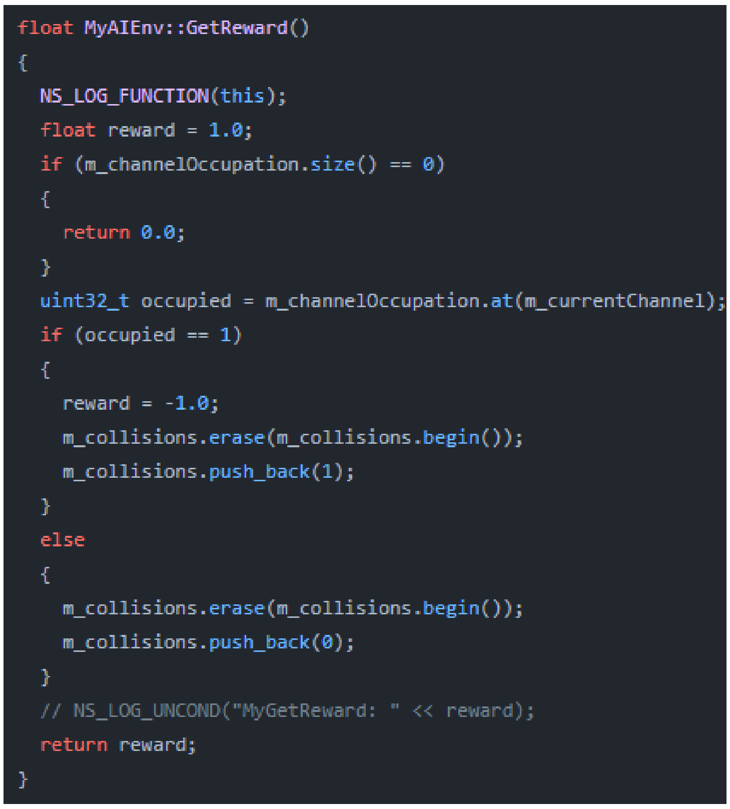
```
MyAIEnv::MyAIEnv(uint16_t id) : Ns3AIRL<sEnv, sAct, sInfo>(id)
{
  NS_LOG_FUNCTION(this);
  m_currentNode = 0;
  m_currentChannel = 0;
  m_collisionTh = 3;
  m_channelNum = 1;
  m_channelOccupation.clear();

  SetCond(2, 0);
}
```

figure 5

in this function we set the parameters that will be used in the simulation

This code is using a **RL algorithm**, the function **float MyAIEnv::GetReward()** that is shown on figure 6 is very important because here are assigned the rewards that will be used in the simulation.



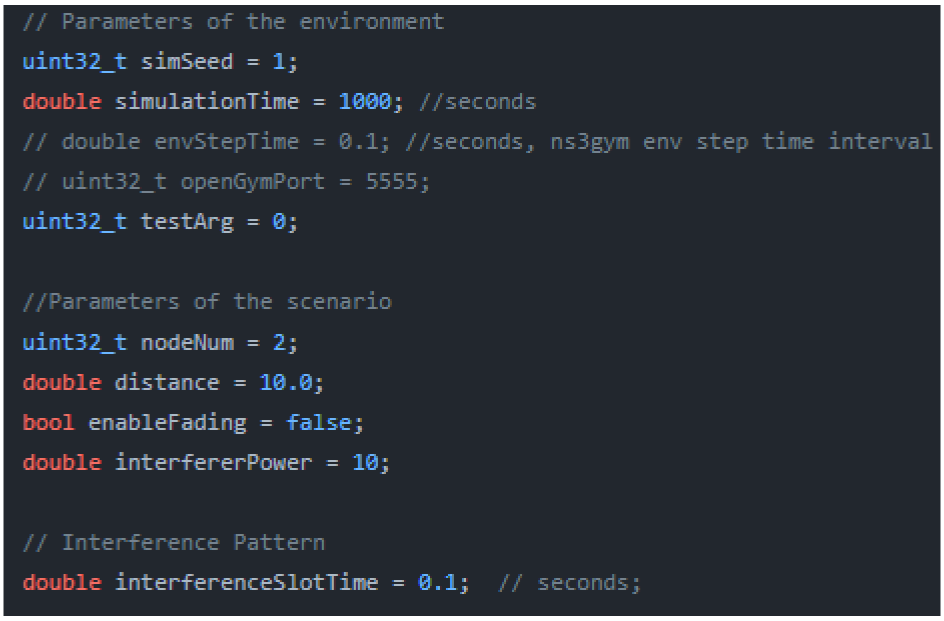
```
float MyAIEnv::GetReward()
{
  NS_LOG_FUNCTION(this);
  float reward = 1.0;
  if (m_channelOccupation.size() == 0)
  {
    return 0.0;
  }
  uint32_t occupied = m_channelOccupation.at(m_currentChannel);
  if (occupied == 1)
  {
    reward = -1.0;
    m_collisions.erase(m_collisions.begin());
    m_collisions.push_back(1);
  }
  else
  {
    m_collisions.erase(m_collisions.begin());
    m_collisions.push_back(0);
  }
  // NS_LOG_UNCOND("MyGetReward: " << reward);
  return reward;
}
```

figure 6

sim.cc [ns3-ai/sim.cc at master · hust-diangroup/ns3-ai (github.com)]

- This code is executed when "**python3 cognitive-agent-v1.py**" is executed, in other words the combination with cognitive-agent-v1.py and sim.cc will run the example, the next function can be found inside the code:

On figure 7 the parameters for the scenario are setting



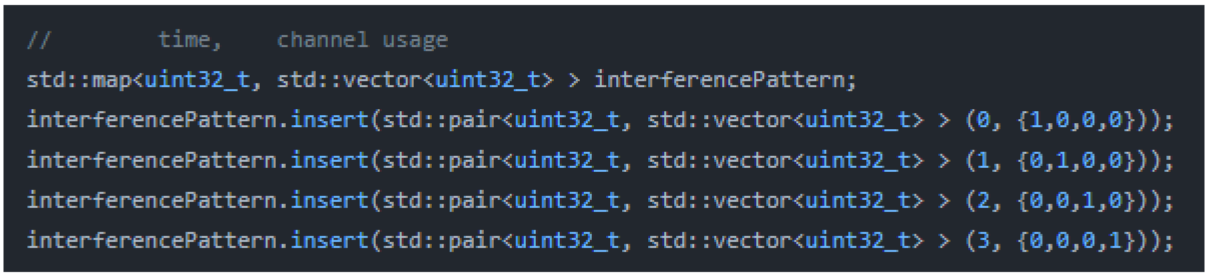
```
// Parameters of the environment
uint32_t simSeed = 1;
double simulationTime = 1000; //seconds
// double envStepTime = 0.1; //seconds, ns3gym env step time interval
// uint32_t openGymPort = 5555;
uint32_t testArg = 0;

//Parameters of the scenario
uint32_t nodeNum = 2;
double distance = 10.0;
bool enableFading = false;
double interfererPower = 10;

// Interference Pattern
double interferenceSlotTime = 0.1;  // seconds;
```

figure 7

In the section that is show in figure 8 the **time and channel** usage is saved in a vector, the first that we have is time 0 with channel usage (vector) {1,0,0,0}



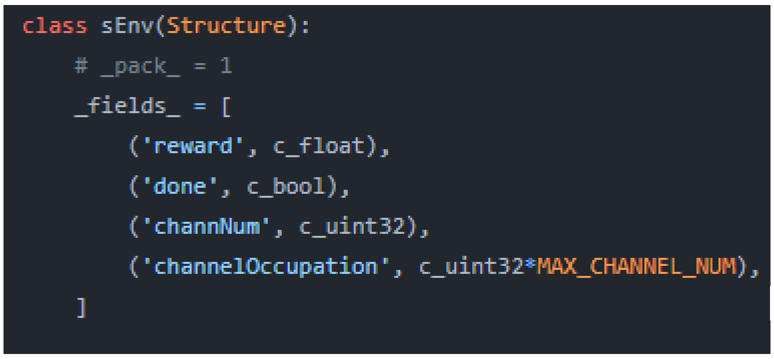
```
//        time,    channel usage
std::map<uint32_t, std::vector<uint32_t> > interferencePattern;
interferencePattern.insert(std::pair<uint32_t, std::vector<uint32_t> > (0, {1,0,0,0}));
interferencePattern.insert(std::pair<uint32_t, std::vector<uint32_t> > (1, {0,1,0,0}));
interferencePattern.insert(std::pair<uint32_t, std::vector<uint32_t> > (2, {0,0,1,0}));
interferencePattern.insert(std::pair<uint32_t, std::vector<uint32_t> > (3, {0,0,0,1}));
```

figure 8

cognitive-agent-v1.py [ns3-ai/cognitive-agent-v1.py at master · hust-diangroup/ns3-ai (github.com)]

- As is mentioned before in this paper, this is the code that needs to be run to execute the example, this are some of the functions that can be found inside the file:

The class **sEnv(Structure)** in figure 9 is defining the structure of the environment, for instance the values of **'reward'** will be a float value**,** for **'done'** the set value will be a boolean



```
class sEnv(Structure):
    # _pack_ = 1
    _fields_ = [
        ('reward', c_float),
        ('done', c_bool),
        ('channNum', c_uint32),
        ('channelOccupation', c_uint32*MAX_CHANNEL_NUM),
    ]
```

figure 9

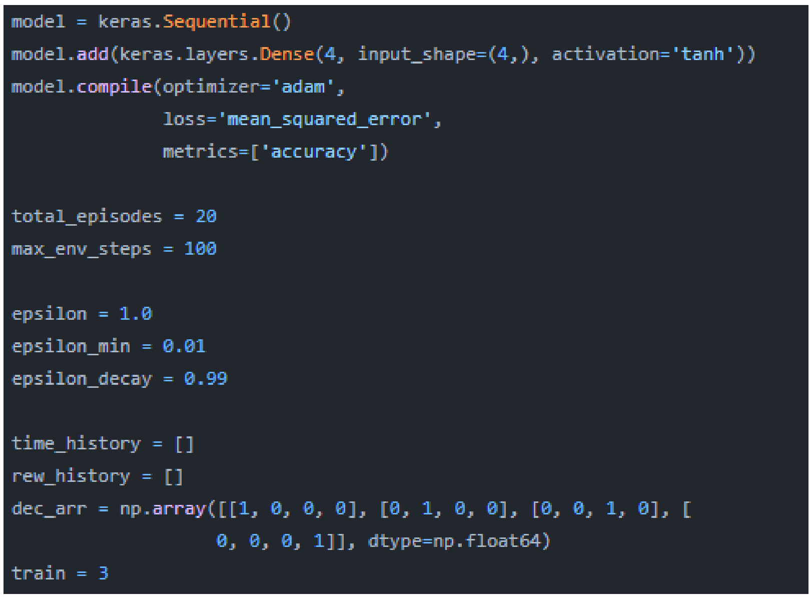
The values of 'rewards', 'done', channNum', channelOccupation', that are in figure 9 are the most important because in this section we are creating a sequential model using **keras.model()** . These models are appropriate to use for a plain stack of layers where each layer has exactly one input tensor and one output tensor.

On figure 10, we have the **keras.layers.Dense function** which implements the operation: output = activation(dot(input, kernel) + bias) where activation is the element-wise activation function, in this case the activation that is used is '**tanh**' which is a function that gave better performance for multi-layer neural networks.

For the compilation is used the **adam** optimizer, there are many optimizers but what they did is uses calculations to adjust weights and biases automatically.

The dec_array is the same as the **time and channel vector** that is described on figure 9

The **total_episodes** are the number of times that the scenario will be trained



```
model = keras.Sequential()
model.add(keras.layers.Dense(4, input_shape=(4,), activation='tanh'))
model.compile(optimizer='adam',
              loss='mean_squared_error',
              metrics=['accuracy'])

total_episodes = 20
max_env_steps = 100

epsilon = 1.0
epsilon_min = 0.01
epsilon_decay = 0.99

time_history = []
rew_history = []
dec_arr = np.array([[1, 0, 0, 0], [0, 1, 0, 0], [0, 0, 1, 0], [
                   0, 0, 0, 1]], dtype=np.float64)
train = 3
```

figure 10

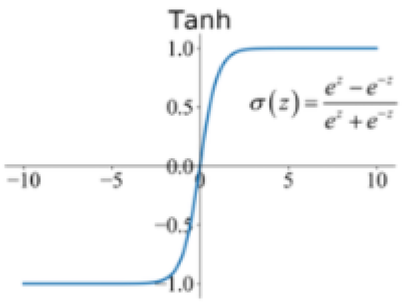
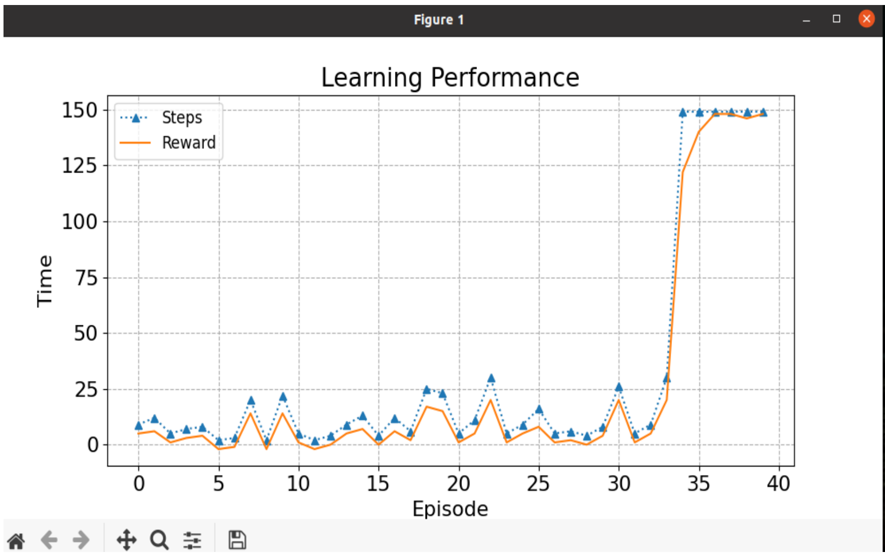
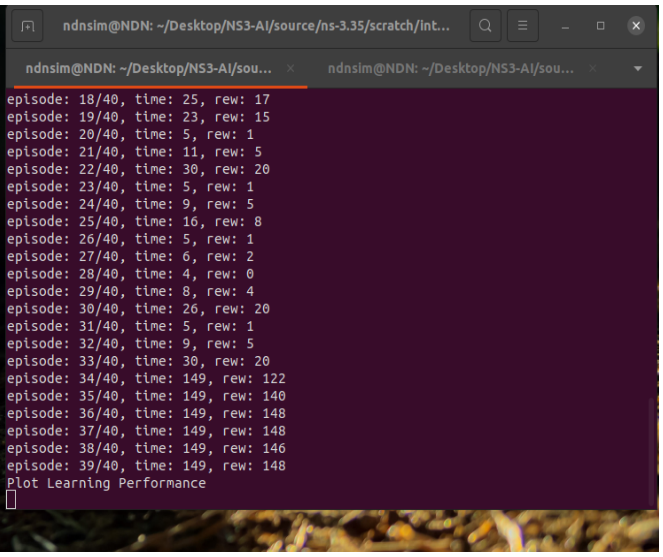


figure 11

Output of the example





```
ndnsim@NDN: ~/Desktop/NS3-AI/source/ns-3.35/scratch/int...
ndnsim@NDN: ~/Desktop/NS3-AI/sou...    ndnsim@NDN: ~/Desktop/NS3-AI/sou...
episode: 18/40, time: 25, rew: 17
episode: 19/40, time: 23, rew: 15
episode: 20/40, time: 5, rew: 1
episode: 21/40, time: 11, rew: 5
episode: 22/40, time: 30, rew: 20
episode: 23/40, time: 5, rew: 1
episode: 24/40, time: 9, rew: 5
episode: 25/40, time: 16, rew: 8
episode: 26/40, time: 5, rew: 1
episode: 27/40, time: 6, rew: 2
episode: 28/40, time: 4, rew: 0
episode: 29/40, time: 8, rew: 4
episode: 30/40, time: 26, rew: 20
episode: 31/40, time: 5, rew: 1
episode: 32/40, time: 9, rew: 5
episode: 33/40, time: 30, rew: 20
episode: 34/40, time: 149, rew: 122
episode: 35/40, time: 149, rew: 140
episode: 36/40, time: 149, rew: 148
episode: 37/40, time: 149, rew: 148
episode: 38/40, time: 149, rew: 146
episode: 39/40, time: 149, rew: 148
Plot Learning Performance
```

5 Executing the examples codes

There are two different ways to execute the codes:

- Using waf

waf is a python-based build tool that is implemented in ns-3, using the parameters of this tool we can get a visualization or generate LOG files for the examples that we want to run.
The LOGs are very important because they will help us to understand what the example is doing

Generate LOG files

```
NS_LOG = ./waf --run ”lte_cqi”
```

Generate and save the LOG files

```
NS_LOG = ./waf --run ”lte_cqi” > debug.out 2 >& 1
```

Get a visualization of the example

```
NS_LOG = ./waf --run ”lte_cqi” --vis
```

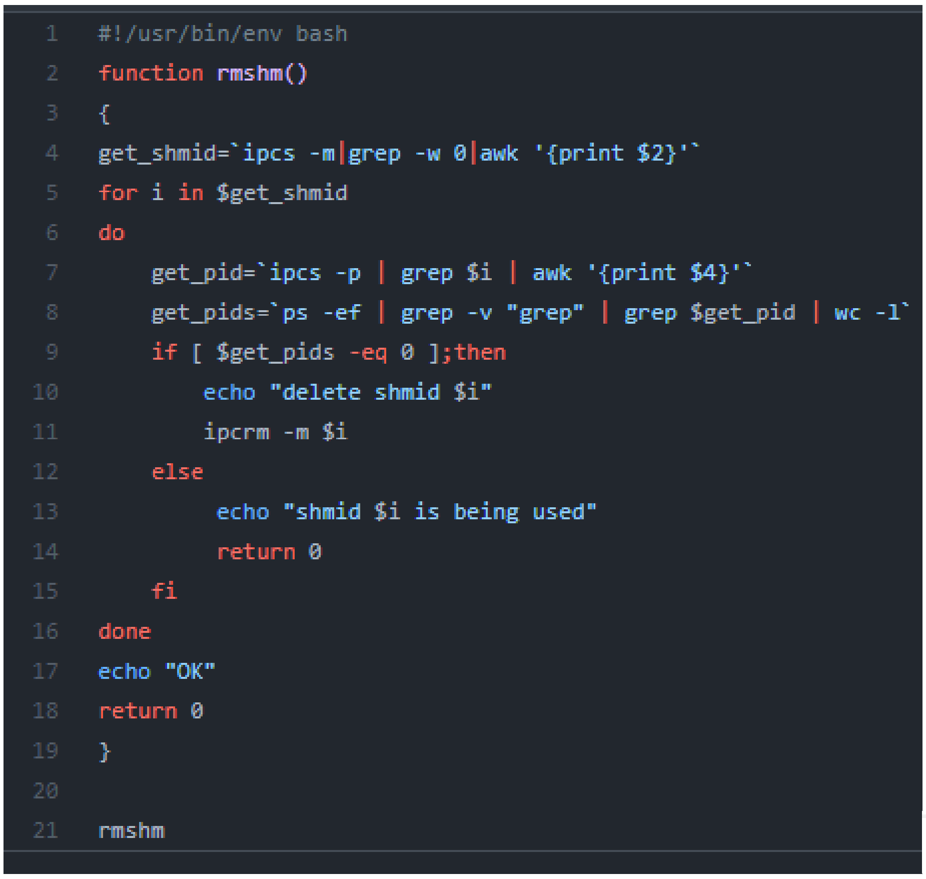
-Executing the python code from the directory

There are some examples that needs to be executed from the code directly, in the directory of the example that needs to be run, we are gonna find **.c, .cc, .h and .py** files, the **.py** files are the ones that need to be run, for instance running the **rlt-tcp** example:

```
cd scratch
cd rl-tcp
python3 run_rl_tcp.py
```

6 Release the memory

There will be times that the shared memory pool is not successfully released (this is done every time when the simulation is finished) in case that this happened is necessary to run the script taht can be found on figure 12 and is on the next github repository [ns3-ai/freeshm.sh at master · hust-diangroup/ns3-ai (github.com)]



```
#!/usr/bin/env bash
function rmshm()
{
get_shmid=`ipcs -m|grep -w 0|awk '{print $2}'`
for i in $get_shmid
do
    get_pid=`ipcs -p | grep $i | awk '{print $4}'`
    get_pids=`ps -ef | grep -v "grep" | grep $get_pid | wc -l`
    if [ $get_pids -eq 0 ];then
        echo "delete shmid $i"
        ipcrm -m $i
    else
         echo "shmid $i is being used"
         return 0
    fi
done
echo "OK"
return 0
}

rmshm
```

figure 12

7 Installation Guide

In this link you can find a detailed installation guide of ns-3 and ns3- ai module using ubuntu 20.04 [ns3-ai/NS3-AI Installation Guide (Ubuntu20.04).pdf at main · GabrielVillagran/ns3-ai (github.com)]

Conclusions

The module works in 2 different ways, most of the examples that uses RL/DL algorithms are in online training, is important to generate logs because they will be useful to understand how the code is working and will be helpful when we need to analyze the code of the examples or even if we want to do our own example.
Most of the examples can generate LOGs but if an example can't generate them we can run the code and the output still be enough to understand how it works.
The example that is analyzed in this paper is interface_control and as is mentioned before, the examples made use of the optimizer '**adam'** this optimizer is one of the most used in this area and is an stochastic gradient descent method that is based on adaptive estimation of first-order and second-order moments

Bibliography

Communications-Caching-Computing Tradeoff Analysis for Bidirectional Data Computation in Mobile Edge Networks (nsnam.org)

hust-diangroup/ns3-ai: Enable the interaction between ns-3 and popular frameworks using Python, which mean you can train and test your AI algorithms in ns-3 without changing any frameworks you are using now! (github.com)

5G-LENA module | 5G LENA (cttc.es)

An introduction to Q-Learning: reinforcement learning (freecodecamp.org)

Introduction to Thompson Sampling | Reinforcement Learning - GeeksforGeeks

Waf | ns-3 (nsnam.org)

Tanh Activation Explained | Papers With Code

The Sequential model (keras.io)

Dense layer (keras.io)

Adam (keras.io)