ns3-ai: Fostering Artificial Intelligence Algorithms for Networking Research

Conference Paper · June 2020

DOI: 10.1145/3389400.3389404

7 authors, including:

Hao Yin
University of Washington Seattle
3 PUBLICATIONS **0** CITATIONS
SEE PROFILE

Yayu Gao
Huazhong University of Science and Technology
36 PUBLICATIONS **150** CITATIONS
SEE PROFILE

Xiaojun Hei
Huazhong University of Science and Technology
117 PUBLICATIONS **2,197** CITATIONS
SEE PROFILE

NS3-AI: Enable Applying Artificial Intelligence to Network Simulation in ns-3

Hao Yin, Pengyu Liu, Lytianyang Zhang, Liu Cao, Yayu Gao, and Xiaojun Hei
Department of Electrical Engineering, University of Washington, Seattle WA, 98915
Email: {haoyin, lyutiz, liucao}@uw.edu
School of Electronic Information and Communications, Huazhong University of Science and Technology, Wuhan, China, 430074
Email: {eic_lpy, yayugao, heixj}@hust.edu.cn

ABSTRACT

Recently, artificial intelligence (AI) algorithms have been widely used in many fields thanks to the advances in processing speed and data acquisition and storage, and several applications to computer networking have been described. Performance evaluation of network systems using AI techniques can be conducted using ns-3, and such studies can be facilitated if ns-3 is able to interact with existing open-source AI frameworks. In the past year, an ns-3 extension module called ns3-gym connecting ns-3 with the OpenAI Gym toolkit has been published; this module makes use of Zero MQ sockets as an interprocess communications (IPC) mechanism. In this paper, we present ns3-ai, a newly designed module between ns-3 and multiple Python-based AI frameworks, to provide efficient and high-speed data exchange between AI algorithms and ns-3. This module uses a shared memory implementation for IPC, which we demonstrate on a small benchmark example to achieve improvements in IPC transfer speed of up to 50-100 times that of ns3-gym on the same scenario. We also describe our high-level interface designed to improve the abstraction between ns-3 and different AI frameworks and provide an example use case based on a 5GNR scenario. We hypothesize that this new framework may offer performance advantages over the approach in ns3-gym for cases in which large amounts of data must be transferred between ns-3 and the AI framework.

CCS CONCEPTS

• **Networks → Network simulations**.

KEYWORDS

AI, ns-3, network simulation

ACM Reference Format:
Hao Yin, Pengyu Liu, Lytianyang Zhang, Liu Cao, Yayu Gao, and Xiaojun Hei, Department of Electrical Engineering, University of Washington, Seattle WA, 98915, Email: {haoyin, lyutiz, liucao}@uw.edu, School of Electronic Information and Communications, Huazhong University of Science and Technology, Wuhan, China, 430074, and Email: {eic_lpy, yayugao, heixj}@hust.edu.cn. 2020. NS3-AI: Enable Applying Artificial Intelligence to Network Simulation in ns-3 . In *WNS3 '20.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145

WNS3, June, 2020, MD, USA

ACM ISBN 978-1-4503-9999-9.
https://doi.org/10.1145

1 INTRODUCTION

With the quick development of modern technology, communication networks have developed into a complicated and colossal system. The new network technology, like 5G commits to provide higher performance and improvement, but the density and complexity of the network will further increase, and more new requirements and problems will also emerge. At the same time, with the growth of computing performance, the field of artificial intelligence has also made rapid expansion. Algorithms like machine learning, deep learning, and reinforced learning have been tried and applied in various fields, and they have also been widely used in the research of communication networks. The combination of the next-generation network and AI algorithms merges in many areas, such as big data-driven optimization problem in 5G [1], the integration of AI and blockchain into 5G [2] and so on. Although AI algorithms have been widely used in network research, there are still many problems and challenges in the actual training and testing process.

The first challenge of applying AI algorithms to the network is the requirements of a large amount of data. Although real networks can produce massive data, these data are often expensive to monitor and collect. What's more, even if the data can be obtained through methods such as spectrum observatory [3], it is difficult to test in a real network with well-trained models. Because the hardware devices used in real networks are not compatible with AI models, the current application of AI algorithms is usually based on simple link-level simulations, testing only a portion or a small module in the network. So the second challenge is how to test and validate the AI models in networks. Besides, the current research on AI algorithms is based on the researchers' own simulation tools, and it is troublesome for others to reproduce and verify the results. On some issues of widespread concern, such as the coexistence of WiFi and LTE in unlicensed bands, the research of network algorithms needs to consider the openness and transparency of the system. Therefore, simulation and testing on the open-source system-level network simulation platform are of great significance for the development of AI algorithms in the network.

From the discussion above, the open-source, system-level network simulation tool like ns-3 can be applied to the development

and test of the utilization of AI algorithms. It is closer to the real network environment and can build various types of simulation scenarios. The users are able to access all the data to interact with AI models in real-time. As it is an open-source tool, researchers can quickly reproduce and verify the algorithm, which contributes to the expansion of AI in the network study. However, most AI algorithms are likely to rely on open source frameworks such as TensorFlow and PyTorch, which provides a flexible and more natural way to create AI models as well as to validate the algorithms. The ns-3 simulator is an open-source networking simulation tool implemented by C++ and wildly used for network research and education. These two parts are developed independently and extremely hard to integrate, so we propose to connect these two tasks with data interaction. In a word, we build a flexible research tool to study the AI algorithms in the network. The specific contributions of this work include:

- Design and implement a shared memory module between open-source AI frameworks and ns-3 with data interaction, which enables run and test the AI algorithms in networking simulator in the same environment called **ns3-ai**.
- Provide a high-level interface in both Python and C++ sides for different algorithms, which makes it easier to adapt to different requirements.
- Present examples to implement and test AI algorithms in network simulation.

This paper is organized as follows. In Section 2, we introduced the background and motivation of this work. Then, we explain the architecture and implementation of the ns3-ai module in Section 3. In Section 4, we provide the examples and benchmarking results of the ns3-ai module. Finally, we conclude the paper in Section 5.

2 BACKGROUND AND MOTIVATION

2.1 AI frameworks and algorithms in communication networks

Artificial intelligence algorithms are one of the most critical new wireless radio technology paradigms for the next-generation networks. Next-generation wireless networks are expected to support extremely high data rates and radically new applications, which require assisting the radio in intelligent adaptive learning and decision making [4]. In recent papers [5][6], the authors summarized the most striking recent accomplishments that machine learning techniques have achieved with respect to classical approaches and proposed research guideline on networking with machine learning to help motivate researchers to develop innovative algorithms.

An AI framework is a tool that assists users in performing training methods and developing algorithms. Its appearance has lowered the barriers to entry for AI. Users don't necessitate to start coding from a complex neural network but use existing models as needed. It is precisely because of the development of various libraries and frameworks that it has become a friendly and convenient field, making people begin their artificial intelligence journey. Compared to the more efficient but complex C/C ++, Python has become the popular programming language used by mainstream frameworks for its simplicity, programmer-friendliness, and high development efficiency. TensorFlow [7] and PyTorch [8] are two popular AI frameworks in recent years. TensorFlow is a deep learning framework that supports various platforms such as Linux, Windows, Mac, and even mobile devices. TensorFlow provides the most comprehensive API among all deep learning frameworks, including basic vector-matrix calculations, various optimization algorithms, convolutional neural networks, and loops, etc. It is fast, flexible, and suitable for large-scale applications at the product level, making it easy for every developer and researcher to use artificial intelligence to solve diverse challenges. PyTorch is the python front end of the Torch [9] calculation engine, which not only provides Torch's high performance but also provides better support for the GPU. The difference between PyTorch and Torch is that it is not just a package, but a framework that is deeply integrated, which makes PyTorch more flexible in network construction.

2.2 The ns-3 simulator

The ns-3 simulator is a discrete event network simulator for network systems, primarily for research and educational purposes [10]. It is an open-source project developed with C ++, an object-oriented programming model. The ns-3 simulator uses several core concepts and abstractions, which can be well mapped to the way computers and networks are constructed, that is, nodes are the most basic entities connected to the network. There is a container class for applications, protocols, and network devices. An application is a user program that generates a stream of packets. Protocols represent the logic of network and transport-level protocols, such as TCP and OLSR routing. A network device is an entity connected to a channel abstracted as a primary communication sub-net. For a network system, some necessary preparations must be made, including installing network devices in nodes and connecting them to channels, assigning appropriate MAC addresses, and configuring the protocol stacks of all nodes. The ns-3 simulator simplifies the tedious work behind easy-to-use APIs. Based on the core concepts, the ns-3 community has developed a large number of network protocols (such as IP, TCP, UDP) and communication technologies (such as Ethernet, WiFi, LTE, WiMAX), and has modeled the physical layer operations in detail. Besides, ns-3 provides a variety of statistical models for wireless channels, mobility, and traffic generation. ns-3 can also interact with external systems (such as a real-time LTE test platform), applications (such as using direct code execution technology), and libraries. All these features are essential for the research of the network and the application of AI algorithms.

Piotr Gawłowicz and Anatolij Zubow from Technische Universität Berlin have developed a module, ns3-gym, to provide an interface in Reinforced Learning between ns-3 and OpenAI Gym [11]. In ns3-gym, OpenAI Gym mainly provides a standard that allows access to the state and performs actions in the environment. The environment is wholly defined in the simulation scenario so that the Python code is independent of the environment, and the implementation of the proxy can be easily exchanged while maintaining the repeatability of the environmental conditions. The ns3-gym uses ZMQ sockets for the data transmission between ns-3 and OpenAI Gym. However, facing the next generation of network, the density of the network is rapidly increasing, which a large amount of data may require to transmit from ns-3 to AI modules for training or testing. It is crucial to reduce the time consuming as well

as supporting colossal data transmission, which is hard to be done by sockets. Besides, ns3-gym is based on the construction of the OpenAI Gym. It uses the API provided by the OpenAI gym, but the drawback is that it is only for reinforcement learning algorithms, so the ns3-gym may be limited when using other methods like deep learning. It is complicated for users familiar with other frameworks to have specific integration.

2.3 Motivation

The main goal of our work is to provide a fast interface between ns-3 and other Python-based AI frameworks to develop and test the AI algorithms on open-source, system-level network simulation tool instead of simple link-level simulation or costly testbed. Moreover, inspired by the ns3-gym module, we implement a new toolkit to shorten the transmission time and support the exchange of large amounts of data. Therefore, we design a new method using shared memory instead of the socket to reduce the transmission time. Besides, different high-layer interfaces for deep learning and reinforced learning are developed to integrate with AI algorithms more conveniently.

There have been other frameworks developed over the years for Inter-process communication (IPC) between Python and C++ programs like Boost Interprocess and Boost Python. However, these mature modules may provide more features and functions, but the cost of learning and redesign of the high-level interface is also higher. Besides, linking other components or libraries in ns-3 is troublesome sometimes, especially for the new users. In most use cases in implementing AI algorithms in network simulator, users only demand to consider pure data exchange instead of complex control information. So we design a new shared memory structure focusing more on handling the data transmission, and also easy to modify for different data types.

3 IMPLEMENTATION

The ns3-ai module consists of two components, the ns-3 interface developed by C++ and the AI interface developed by Python. This module provides a high-level interface for the quick development of DL/RL algorithms as well as the core module to transfer data from one C++ program to another Python program. As highlighted in Section 2, the key concept of interaction between AI frameworks and ns-3 is to enable data to transfer in multiple processes. Each process has a different user address space, and the global variables of any process cannot be seen in the other process, so to exchange data between processes must pass through the buffer in the kernel (e.g., process A puts data from user space to the kernel buffer, process B reads the data from the kernel buffer). Several ways can be adopted to the communication between processes such as pipe, socket (used by ns3-gym), and shared memory. In the next generation of wireless communication networks, the density and complexity of networks are risen rapidly, leading to an increase in data and training time. Thus, the ability to exchange a large volume of data in a short time should be considered first, which motivates us to choose shared memory as the core module to transfer data. In the following subsections, we describe the system architecture and explain the shared memory pool and high-level interface in detail.

3.1 System Architecture

The system architecture of running AI algorithms in ns-3 is shown in Fig. 1. The ns-3 simulator and AI frameworks are running in different processes. The data transfer is mainly required in two situations - sending data for training and testing the AI model in ns-3. The ns-3 simulator is used to set up the network and topology, which generates data for the training in AI algorithms. The AI frameworks provide functions to train the models using the data from the ns-3 and then to return the data from the trained model back to ns-3 for testing.

The ns-3 simulator provides environments for testing AI algorithms crossing the whole network layers by building simulation scenarios. No matter you only want to predict a specific value like channel condition indicator (CQI) in LTE, or you want to establish a scheduling algorithm for self-driving cars, you could create a scenario in ns-3 and replace the value or decision through the interface. For the examination of a well-trained model or algorithm, it seems that it inserts a function in ns-3. Because the codes of ns-3 are all open-sourced and well documented, the examination could be performed in any inner layers or modules. Besides, the ns-3 simulator could also serve as the data generator. For instance, thinking of a simple cellar network, users may be connected or disconnected to the base station or move from one side to another. We concern about the location and speed of the user in a higher view but also desire to acknowledge its channel condition and user experiences. Then we could train a model to make the scheduling decision based on this information. In ns-3, it is straightforward to build a scenario from the user view and also receive the lower layer information to feed the deep learning model. Thus ns-3 could be regarded as a data generation tool as well as a test tool for various requirements.

Different AI frameworks using Python scripts are all supported to be integrated with this module. By applying the interfaces in the Python side, the users could extract the data from the shard memory than continue to train the model or return the results for testing. It only impacts the procedure of data processing and testing methods without interfering with the internal processing of the AI algorithms. Therefore it is convenient to rerun the existing algorithms in ns-3.

The ns3-ai module interconnects the ns-3 and AI frameworks by transferring data through the shared memory pool. The memory can be accessed by both sides and controlled mainly in ns-3. From data using for training or testing to the control signals, all the information transmitting through the two parts could be handled by the ns-3 module. In this way, users could define different memories for complex usages, as all kinds of messages could convey through the ns3-ai module.

3.2 Shared Memory Pool

The creation and maintenance of shared memory are done in ns-3 because it's convenient and simple to operate memory in C++ directly. The data is transferred through separate memory blocks in the memory pool created at the beginning of each simulation, so different kinds of data could use different memory blocks. The structure of the shared memory pool is shown in Fig. 2. The shared memory pool consists of three structures in continuous memory,

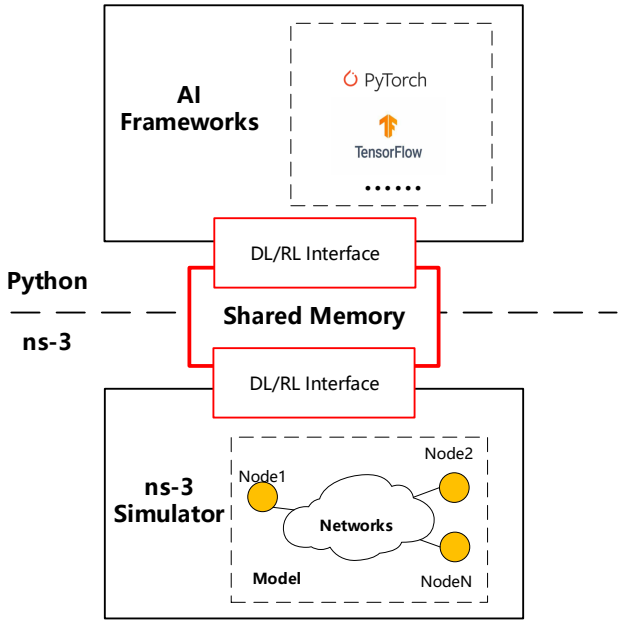

Figure 1: System Architecture.

the memory blocks, the control blocks, and the main control block. The main control block is in charge of the whole memory pool and updates its version after each modification, like requiring a memory block that happened in the memory pool. The control blocks and the memory blocks are one-to-one correspondence, i.e., each time when you acquire a new memory block from the head, a new control block corresponding to it will be created automatically in the tail of the memory pool. The control block includes information such as the size and the address of the memory block. It is readable but could not be changed until the free of the block. The memory block is readable and writable memory with the lock indicator (version and next version pairs) and the pointer to the actual start address of the data.

3.2.1 Reading/writing lock and synchronization. The reading/writing lock is implemented by two eight bits variables version and the next version. For every process that wants to access or modify the data, it compares the version variable and the next version variable. If they are the same, it means that the memory is reachable. Then it adds one to the next version atomically to lock the memory and also add one to the version after its operation to the memory to unlock the memory.

The version pairs also achieve the synchronization between reads and writes. For the most common use case, the ns-3 generates the data and then writes it to the shared memory, and the AI part reads the data from the shared memory to train the module. The action done on the ns-3 side is always writing, and the other side is always reading, that the action takes place alternately. So the version in ns-3 is always even, and the other side is always odd. In this way, the actions are related to the version values, that some methods like modulation can be applied to synchronize between reads and writes. If reading and writing happen on both sides, we recommend using two different memory blocks so that the same method is also useful in these cases. The version of the memory acts as the signal to tell different processes the current state of the memory block, which provides different methods to synchronize by determining the version value in the memory block.

3.2.2 Cooperation between processes. The polling mechanism is expected to access the memory and pending for data transmission. In most cases, we devise an AI algorithm to resolve some issues in the networks, and the ideal test of the AI algorithm is to plug in or replace a particular algorithm. In other words, in the network simulation with AI algorithms, the simulation and accomplishing AI algorithms are performed sequentially rather than in parallel. By applying the pooling mechanism to wait for new results before continuing, it avoids the synchronization of time in each process. By this means, the results can be reproduced, and it seems like we solely plug in a new AI algorithm in the simulation procedures. The processes wait for each other until the data required arrives for the subsequent step, so users don't have to concern about the virtual time management in ns-3 or the control of the different processes.

Because we introduce the version variable to make it easy to identify the status of the memory by the processes, it is not necessary to develop another method to convey the control signal. Every process is required to maintain the memory on its version, which likewise increases the stability of the system and avoids conflict. It may introduce some extra calculations of the memory size and version management before the simulation, so we plan to also implement the control channel in the future for more complex cases, but currently, it is sufficient to deal with most scenarios.

3.3 High-level Interface

The dimension and type of the data usually vary from training to testing, and most time users want to send some self-defined data structure rather than a single value. It is simple to transmit and get a single value from one side to another while much harder to put or get some self-defined data structure in Python directly. Thus we provide a high-level interface to enable the transmission of different types of data in both Python and C++ side.

In the C++ side, a template class for data is used, so the developers are able to put any structure into the memory directly. In the Python side, we develop a shared memory extension module using C/C++, which has the same functions as in the C++ side. Constructing extensions for Python requires three main steps: creating application code, using boilerplate to wrap the code, and compiling and testing. The application code is similar to the memory pool management code in ns-3, but without the extra modules or namespace in ns-3. It contains all the structures and functions needed to manage and access the shared memory pool. The code for the interface is called the "boilerplate" code, and it is an essential part of the interaction between the application code and the Python interpreter. The template is mainly divided into four steps:

- Contains Python header files.

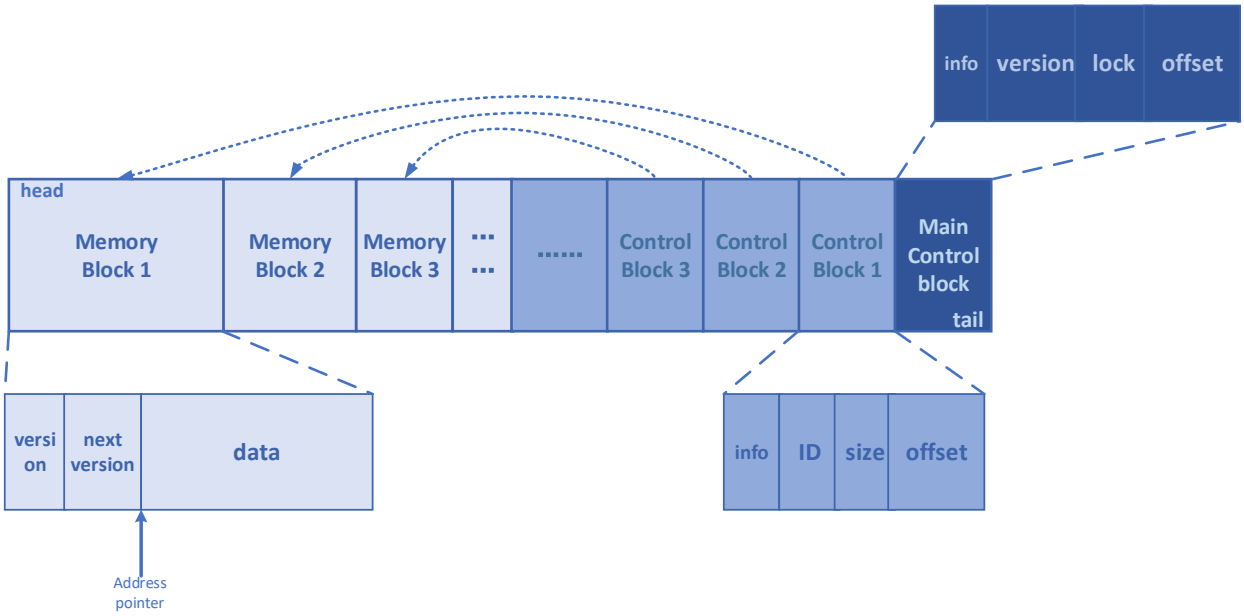

Figure 2: Memory Structure.

- Adds a wrapper function of type *PyObject** for each function of each module.
- Adds an array of specific type such as *PyMethodDef* for each module.
- Adds module initialization function *void initModule ()*.

Then the main task of compilation is done by the *setup()* function in *setup.py* script, which puts these functions together with the Python library to compile and generate the corresponding Python module. In this way, the installation of the ns3-ai Python interface is straightforward, that users require to download the modules and run the *setup.py* script, and it will automatically install the modules for users. Finally, users can import this module and use it in any AI-based framework. Though we do not support using pip to install this module as ns3-gym, it is not hard and flexible for the future update.

For some widely applied algorithms in the network like Reinforced Learning (RL) and Deep Learning (DL), more specific data structures and interfaces are also provided to make this module easy to use. For RL modules, similar to ns3-gym, functions that modify the environment set or get actions and create simulation information are supported to simplify the tasks to develop an RL algorithm and exchange data. For DL modules, new functions to extract features for training, feedback prediction results, build up different targets, and create simulation information are presented to train and run some prediction models by deep learning more efficiently. These functions illustrate the everyday tasks to modify the memory or transfer the date and hide them behind easy to use interfaces.

4 EXAMPLES AND BENCHMARKING

In this section, we provide an example to explain how to use this module and build up specific scenarios to examine deep learning algorithms in ns-3. The example expects to modify some source code in ns-3 to train and predict data via a deep learning method. This work is based on ns3 and 5G NR module developed by CTTC[12]. Besides, we implemented the same example for both ns3-ai and ns3-gym modules to test the stability and transmission speed, as well as illustrate the usage of RL interfaces.

4.1 CQI Prediction in NR with LSTM

4.1.1 Motivation. In cellar systems, channel conditions can seriously affect communication quality. Therefore, one of the critical technological advances is to improve channel estimation and enhance modulation schemes continuously. The channel quality indicator (CQI) is one of the most important parameters to determine the data rate achievable for multimedia transmission. However, due to the transmission delay from the user to the base station, real-time CQI is not always available in the base station. Unlike LTE, the channels of 5G networks may be more dynamic and complex, which makes it more challenging to estimate CQI accurately. In 5G new radio (NR) systems, more complex and variable channels may severely limit the throughput performance of 5G systems. Currently, 5G networks are still under active development, aiming to provide large data rates of 100 Mbps to 1 Gbps at any time, anywhere, even in high-speed mobile environments with speeds up to 500 km/h. In this example, we propose a CQI prediction method in a deep learning method through a Long Short-Term Memory (LSTM) algorithm. To directly verify the CQI prediction algorithms in 5G

new radio (NR) systems, an online training module is proposed to reduce the negative impact of outdated CQI on the degradation of communication quality, especially in high-speed mobility scenarios.

4.1.2 Using LSTM to Predict CQI. Long Short-Term Memory (LSTM) is well suited to deal with issues that are highly time series related and can evaluate and predict changes in CQI at the BS side in the past CQI series. Therefore, we use LSTM as the core network to predict the subsequent CQI. The key function of LSTM is to have a certain memory effect, which can remember the historical characteristics of CQI, and then use this information to predict future CQI. As shown by 3, this module consists of a fully connected layer, an LSTM layer, a prediction layer, and an update module. Connect fully connected layers for feature extraction and then implement an LSTM layer to extract temporal information. Finally, the prediction layer sums the output of the vectors by the LSTM layer to generate the predicted value of the CQI. The input layer is a CQI vector from $CQI_{real}(t-\tau-N)$ to $CQI_{real}(t-\tau)$, where N represents the data length used for each prediction. N_{FC} Neural units are used for fully connected layers and the output of the module is a vector of length N_{LSTM}.

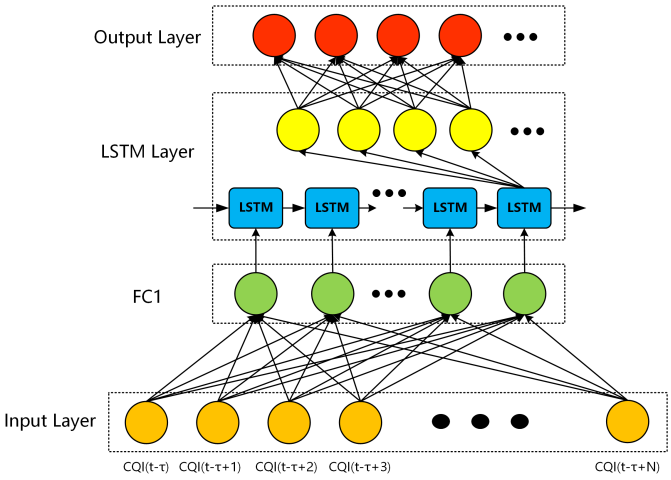

Figure 3: LSTM architecture.

4.1.3 Online Training Method. Because the distribution of the CQI sequence changes over time in the wireless channel, the trained model is time-sensitive, so an online learning model is introduced to update the predictions in time. When the MAC layer starts scheduling users, it processes the CQI reported by the user. Since the generation and transmission of CQI are time-consuming (use τ to represent the delay of all CQI arrivals), traditional scheduling will use $CQI_{real}(t-\tau)$ as the input CQI ($CQI_{delay}(t)$) Current time t. Therefore, we want to use the algorithm predicted CQI_{pred} instead of $CQI_{real}(t-\tau)$.

The ns3-ai is used to realize this online training module. The ns-3 provides the data as the input of the deep learning algorithm to train the model through ns3-ai and holds itself until getting the feedback data from the DL algorithms. In this way, it only transfers the data from both sides without having any impact on the original simulation process, which looks like simply adding an algorithm directly into the simulator.

Table 1: Simulation Parameters

Parameter	Value
Power	20 dBm
Numerology	1
Bandwidth	100MHz
Scenario	UMa
Data Rate	64Mbps
RLC Mode	UM
Moving Module	Constant Speed
Speed	10m/s 20m/s 30m/s ...
Scheduler Algorithm	Random Robin

Table 2: Training Module Parameters

Parameter	Value
Batchsize	20
Pre_train Times	20
LSTM Unit	30
Input Length (K)	200
Data Length Using for Prediction (N)	40
Activation Function	Selu

4.1.4 Results. The ns-3 simulation platform can be used for system-level simulation and data generation. In the simulation scheme, four users are attached to the base station as interfering users. Another user, marked as a mobile user, is crossing the base station from outside the coverage area. The speed of mobile users varies in different scripts, which gives us different results. This scheme is completed to examine the performance in high-speed situations, and multiple users are attached to the base station to test downlink scheduling performance. Select the millimeter-wave channel module and set the numerology to 1 to reduce simulation time. The delay τ of the CQI used by the BS is set to 0.5 ms to cause a delay equal to only one slot, so it is close to the true CQI. The simulation scenario is shown in 4, and the parameters are shown in table 1. The parameters used by the training module are shown in table 2.

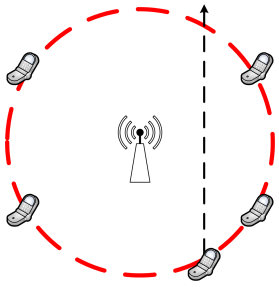

Figure 4: Graphic illustration of simulation scenario: Four interfere users and one users move fast across the BS for the prediction of CQI.

Using the data generated from ns-3, we test the prediction accuracy of the scenario mentioned above, and the result is shown in Fig. 5. The prediction accuracy indicates that the CQI predicted is equal to the real CQI that feeds back after the delay. During each scheduling procedure at time t, we predict a CQI value $CQI_{preduct}(t)$ instead of using theCQI_{delay}. After several slots, the eNB will receive the actual value of the CQI $CQI_{real}(t)$, then we compare it with $CQI_{preduct}(t)$ whether they are the same to calculate the accuracy. As the speed increases, the channel condition changes more rapidly, so the CQI is hard to predict, which causes a drop in prediction accuracy. Two different prediction methods are applied as the prediction module, the FNN and LSTM. The LSTM module is more reliable than the FNN module, which has a prediction accuracy above 50%. Because LSTM is designed to deal with issues that are highly time series related like CQI, the prediction result is better than FNN. Even if the speed is increasing, FNN could not provide a reliable prediction while LSTM is still working.

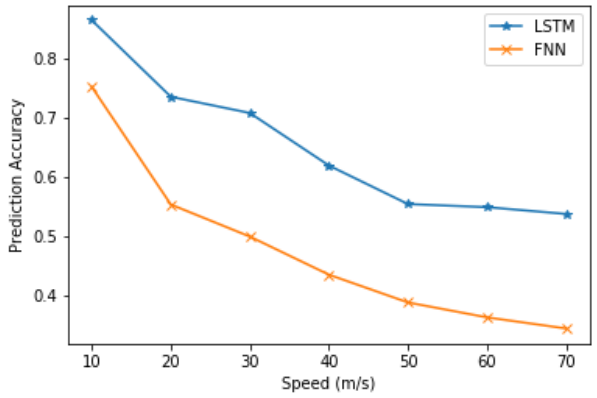

Figure 5: Prediction Accuracy versus speed.

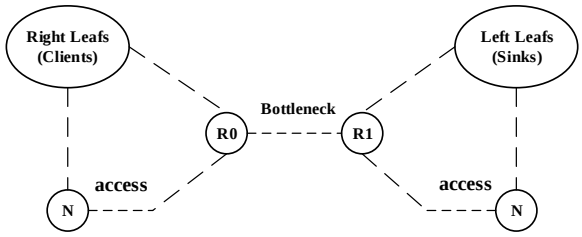

Figure 6: TCP example topology.

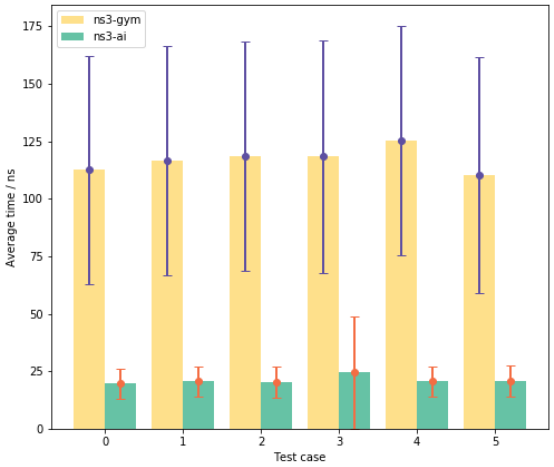

Figure 7: Average transmission time from ns-3 to Python.

4.2 Benchmarking

To compare the transmission time with ns3-gym, we implement the same environment in Opengym and ns-3. The motivation of the RL-TCP examples is to apply the RL algorithm to TCP congestion control. For real-time changes in the environment of network transmission, the network can get better throughputs and smaller delays by strengthening the learning management sliding window and threshold size. However, running RL algorithms may have an impact on the accuracy of the actual time-consuming in the data exchange module, so a simplified model is applied to calculate the time cost by the interaction module. The benchmarking example is shown in Fig. 6. The dumbbell-like topology simulation scenario is set up in ns-3. There is only one leaf node on each side, and two routers R0, R1 in the middle link. The TCP buffer is 4 Mbps, receiving and sending rate is 2Mbps respectively. The left leaf node sends packets to the right node through the link. The routers initialize a routing table about the nodes in the simulation to sense all the other nodes. The result is the number of packets received by the right node. During the process of TCP simulation in ns-3, it sends the state values of TCP such as ssThresh, congestion window, segment size, etc. to the RL training module with an extra tag of the current time. The RL module only reads these data and then using its current time to minus the time tag to calculate the transmission time from ns-3. After reading the tag, it will rewrite the tag and update the memory version. Then the ns-3 side could also read the tag and do the calculation to determine the transmission time from the RL module.

The benchmarking results are shown in Fig. 7 and Fig. 8. The transmission time of ns3-ai is six times faster than ns3-gym from ns-3 to Python and is even 50 to 100 times faster from Python to ns-3. The performance of ns3-gym is stable while in ns3-ai, the time-consuming is related to the direction. The reason is that in Python, reading data requires extra steps to lock memory, which is slower than directly uses C++. These parts will be optimized in the next release. From these two figures, it clear the ns3-ai module is much faster, which contributes to reducing the simulation time especially when the data volume is huge.

The total simulation time differs from scenarios. For this RL TCP example, the data volume is not huge, so the reduction of the data

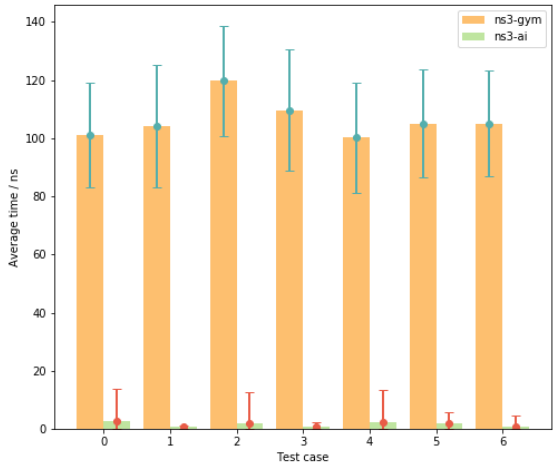

Figure 8: Average transmission time from Python to ns-3.

exchange time may not be significant compared with the simulation time. However, in the next-generation wireless network like 5G, the density of the network increases rapidly. Like the new layout of numerology in 5G, the time slot could be less than 0.03 ms. If you want to test AI algorithms for downlink scheduling, more data changing will happen that takes more time. In this kind of situation, the reduction of time should be considered to optimize the simulation time of the combination of AI algorithms and next-generation networks. What's more, as discussed in section 1, ns-3 can be served as a data generator for AI. In these kinds of scenarios, data changing is the main time-consuming part, so that the reduction of times matters more.

5 CONCLUSION

In this paper, we introduce ns3-ai, a new toolkit for data transmission between the ns-3 and AI frameworks based on shared memory. The transmission speed is seven times faster from ns-3 to AI frameworks and 50-100 times faster the other way than the current ns3-gym toolkit from the benchmarking results. Moreover, it also supports more AI frameworks than ns3-gym with high-level DL and RL interface. A specific CQI prediction example is presented for the test and usage of the ns3-ai module. We are formulating a series of test cases, including downlink scheduling problem in LTE/NR, and optimize the lock method to further reduce the transmission time from ns-3 to Python side.

6 ACKNOWLEDGMENTS

The authors would like to express my special thanks to Prof. Henderson for helping release the ns3-ai module and the review of this paper as well as to Prof. Roy for supporting me to continue this work in UW. Secondly, we would also like to thank other students, Keshu Liu, Yingying Han, and Xiaojun Guo from Dian Group in HUST who also help us to run some simple tests and provide feedback!

REFERENCES

[1] K. Zheng, Z. Yang, K. Zhang, P. Chatzimisios, K. Yang, and W. Xiang. 2016. Big data-driven optimization for mobile networks toward 5g. *IEEE Network*.

[2] Y. Dai, D. Xu, S. Maharjan, Z. Chen, Q. He, and Y. Zhang. 2019. Blockchain and deep reinforcement learning empowered intelligent 5g beyond. *IEEE Network*.

[3] S. Roy, K. Shin, A. Ashok, M. McHenry, G. Vigil, S. Kannam, and D. Aragon. 2017. Cityscape: a metro-area spectrum observatory. In *2017 26th International Conference on Computer Communication and Networks (ICCCN)*, 1–9. DOI: 10.1109/ICCCN.2017.8038427.

[4] C. Jiang, H. Zhang, Y. Ren, Z. Han, K. Chen, and L. Hanzo. 2017. Machine learning paradigms for next-generation wireless networks. *IEEE Wireless Communications*.

[5] G. Zhu, D. Liu, Y. Du, C. You, J. Zhang, and K. Huang. 2020. Toward an intelligent edge: wireless communication meets machine learning. *IEEE Communications Magazine*.

[6] M. Wang, Y. Cui, X. Wang, S. Xiao, and J. Jiang. 2018. Machine learning for networking: workflow, advances and opportunities. *IEEE Network*.

[7] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2016. Tensorflow: a system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. (November 2016).

[8] 2017. *Introduction to pytorch. Deep Learning with Python: A Hands-on Introduction*.

[9] Ronan Collobert, Samy Bengio, and Johnny Marithoz. 2002. Torch: a modular machine learning software library, (November 2002).

[10] Thomas R. Henderson, Mathieu Lacage, and George F. Riley. 2008. Network simulations with the ns-3 simulator. In *In Sigcomm (Demo)*.

[11] Piotr Gawłowicz and Anatolij Zubow. 2019. ns-3 meets OpenAI Gym: The Playground for Machine Learning in Networking Research. In *ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM)*. Miami Beach, USA.

[12] Natale Patriciello, Sandra Lagen, Lorenza Giupponi, and Biljana Bojovic. 2019. An improved mac layer for the 5g nr ns-3 module. In *Proceedings of the 2019 Workshop on Ns-3* (WNS3 2019). Association for Computing Machinery, Florence, Italy, 41–48.