DEGREE IN TELECOMMUNICATION TECHNOLOGIES ENGINEERING

Bachelor's final project

GENERALIZATION OF CFR-RL

Author
Carlos Marí Noguera

Director
Amy Zhang
The University of Texas at Austin

Madrid
June 2024

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título

GENERALIZATION OF CFR-RL

en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el

curso académico 2023/24 es de mi autoría, original e inédito y

no ha sido presentado con anterioridad a otros efectos.

El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido

tomada de otros documentos está debidamente referenciada.

Fdo.: Carlos Marí Noguera Fecha: 14/ 06/ 2024

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Fdo.: Amy Zhang Fecha: 14/ 06/ 2024

ACKNOWLEDGEMENTS
Texas Advanced Computing Center (TACC) The University of Texas at Austin

DEGREE IN TELECOMMUNICATION TECHNOLOGIES ENGINEERING

Bachelor's final project

GENERALIZATION OF CFR-RL

Author
Carlos Marí Noguera

Director
Amy Zhang
The University of Texas at Austin

Madrid
June 2024

Generalización de CFR-RL

Autor: Marí Noguera, Carlos.
Director: Zhang, Amy.
Entidades Colaboradoras: ICAI - Universidad Pontificia de Comillas, The University of Texas at Austin

RESUMEN DEL PROYECTO

Introducción

Las redes definidas por Software [1] son un planteamiento de arquitectura de red emergente que separa el plano de control del dispositivo de red del plano de datos. El controlador de red tiene un control centralizado de la red y actua como el sistema operativo. Este planteamiento permite que la red sea controlada centralmente, habilitando nuevas posibilidades de invesitgación en el ámbito de las SDN por las siguientes razones. En primer lugar, debido a que el controlador tiene una vista global de la red, este puede recolectar información sobre el estado de la red a tiempo real, y esta información puede ser utilizada para optimizar diferentes aspectos como el enrutado. En segundo lugar, el controlador es capaz de controlar los recursos de red mediante la instalación de reglas de envio en los swtiches que estan bajo su control [2].

Aunque se hayan abierto nuevas posibilidades para la optimización de red, esta tarea no es trivial. Solucionar un problema de Ingeniería de Tráfico (TE) en terminos de rendimiento y retraso puede ser complejo. El objetivo de la Ingeniería de Tráfico es ayudar a los administradores de red y a los proveedores del servicio de Internet (ISPs) a optimizar el rendimiento de la red mediante una asignación de recursos y tráfico eficiente. Otros algoritmos como [3] basado en MultiProtocol Label Switching (MLPS) son capaces de obtener un funcionamiento casi óptimo mediante en reenrutado de la mayor cantidad de flujos posbiles, sin embargo estas soluciones no tienen en cuenta resultados negativos como puede ser el retraso punto a punto o paquetes desordenados. De otra manera métodos actuales basados en aprendizaje de máquina como [4] [5], son capaces de obtener resultados

cuasi-óptimos sin impactar negativamente a la red, pero estos no son capaces de generalizar a diferentes topologias de red.

Definición del proyecto

El objetivo del proyecto es el diseño e implementación de un modelo basado en aprendizaje reforzado (RL), capaz de aprender una política de selección de flujos críticos, posteriormente estos flujos críticos seran reenrutados óptimamente mediante la resolución de un problema de programación lineal. El objetivo del agente de aprendizaje reforzado es optimizar la selección de los flujos críticos con el propósito de minimizar la máxima utilización de vínculo (MLU). Este proyecto diferirá del modelo de CFR-RL [4], ya que el objetivo es el desarrollo de un modelo capaz de generalizar a diferentes topologías de red y patrones de tráfico. La red neuronal a utilizar, deberá ser ligera y la latencia obtenida debido al proceso de inferencia deberá ser despreciable cuando es comparada con el retraso punto a punto. Los resultados de este proyecto serán considerados positivos, si somos capaces de desarrollar un modelo capaz de obtener mejores resultados que el algoritmo TOP-K Critical en topologias y matrices de tráfico en las que no ha sido entrenado.

Se ha elegido un algoritmo basado en selección en vez de un algoritmo de enrutamiento ya se cree que resultará más simple el aprendizaje de una política de selección capaz de generalizar, que el aprendizaje de una política de enrutamiento. En cada paso t, el modelo recibirá la matriz de tráfico en ese mismo instante (TM_t) y la topología actual de la red y seleccionará K flujos críticos que serán reenrutados.

Descripción del problema

El desarrollo de este algoritmo de aprendizaje reforzado profundo tiene como objetivo solucionar un problema de Ingeniería de Tráfico, para eso se ha transformado el problema de TE en un problema de aprendizaje reforzado mediante el uso de una Cadena de Markov con estados, acciones y recompensas.

Definición del problema de aprendizaje reforzado

El uso de una cadena de Markov para describir el problema nos habilita el uso de algoritmos de aprendizaje reforzado para solucionarlo. El problema de RL varia del presentado en [4], ya que ha sido modificado para facilitar la generalización a multiples topologías. La cadena de Markov es definida de la siguiente manera:

Estados: Los estados representan la entrada a nuestra red neuronal, este estado debe contener toda la información necesaria para aprender la política deseada.

El estado propuesto (s_t), estará formado por la matriz de tráfico en el instante t (TM_t) y la topología de la red. Debido a la naturaleza continua de los elemtenos de la matriz de tráfico, hay un número infinito de TMs, por esa razón se ha decido que se usará una red neuronal para solucionar el problema.

Acciones: Para cada estado s_t, el modelo seleccionara K flujos críticos. Hay un total de $N * (N-1)$ flujos en una red creado un espacio de acciones considerablemente grande. Debido a que se seleccionarán K accciones por paso, se ha decidido utilizar el siguiente espacio de acciones $A = \{0, 1, ..., (N * (N-1)) - 1\}$ inspirados por [6] y [4]. Por cada paso, se muestrearan K acciones diferentes $(a_t^1, a_t^2, ..., a_t^K)$. Se entrenará a nuestra red nueronal para proporcionar una distribución categórica de la cual se muestreará.

Recompensas: Tras el muestreo, los flujos seleccionados serán enrutados de manera óptima y se obtendrá la utilización máxima de vínculo (MLU). Para conseguir el objetivo de generalización se ha diseñado una nueva señal de recompensa. Se ha optado por normalizar el valor del MLU mediante la comparación con el resultado del algortimo TOP-K Critical.

$$r = \frac{MLU_{TOP_K} - MLU}{MLU_{TOP_K}} + 1 \tag{1}$$

Esta recompensa se utilizará para actualizar la red neuronal.

Algoritmo: Para actualizar la red neuronal se ha optado por el algoritmo REINFORCE [7] con el uso de una base. Se llevará un registro de las recompensas obtenidas en cada estado y se computará la base $V(S_t)$ como la media de las experiencias previas en ese mismo estado. Se actualizará la red neuronal mediante la siguiente función.

$$\theta_t = \theta_{t-1} + \alpha[\delta + \epsilon H(\pi(A_t|S_t, \theta_{t-1}))]\nabla \ln [\pi(A_t|S_t, \theta_{t-1})] \tag{2}$$

Donde

$$\delta = G - V(S_t) \tag{3}$$

$$H(p(x)) = -\sum p(x) \ln p(x) \tag{4}$$

$$\begin{aligned} \pi(A_t|S_t, \theta_{t-1}) =& \pi(a_t^1|S_t, \theta_{t-1}) \times \pi(a_t^2|S_t, \theta_{t-1}) \times \\ & ... \times \pi(a_t^K|S_t, \theta_{t-1}) \end{aligned} \tag{5}$$

La implementación base de REINFROCE ha sido modificada para tener en cuenta la selección de multiples acciones y se ha añadido un pequeño bonus de entropía para favorecer la exploración. ϵ es un hiperparámetro, se ha elegido el valor 0.01, el mismo que en [4] y otras opciones no han sido exploradas.

Red Neuronal

Para calcular las probabilidades, se ha decidido utilizar una red neuronal convolucional (CNN) [8] debido a la estructura 2D de las matrices de tráfico y de adyacencia.

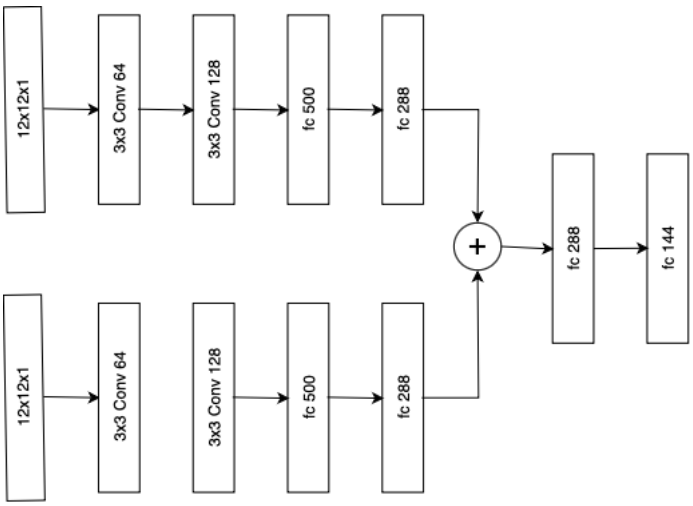

Figure 1: Red Neuronal Diseñada

Generación de Datos

Debido a precauciones de seguridad hay escasez de matrices de tráfico reales públicas, esto ha llevado al estudio e investigación de la generación de matrices de tráfico sintéticas. Se ha decidido generar datos sintéticos basandose en el modelo de gravedad [9] que esta inspirado por el modelo gravitacional de Newton. Debido a que el objetivo de este proyecto es evaluar la generalización se han utilizado datos reales de la red de Abilene [10] para probar los resultados.

Resultados

Para evaluar los resultados de generalización del modelo, se ha comprobado el rendimiento del modelo en una topología y distribución de tráfico no presentes en los datos de entrenamiento. Cabe destacar, que la topología era similar a una de

las topologias en los datos de entrenamiento, con el cambio de un enlace, simulando un fallo en la red. Para poder evaluar el rendimiento de el modelo se comparará con los resultados obtenidos por el algoritmo TOP-K Critical que selecciona los flujos más grandes de los nodos más congestionados.

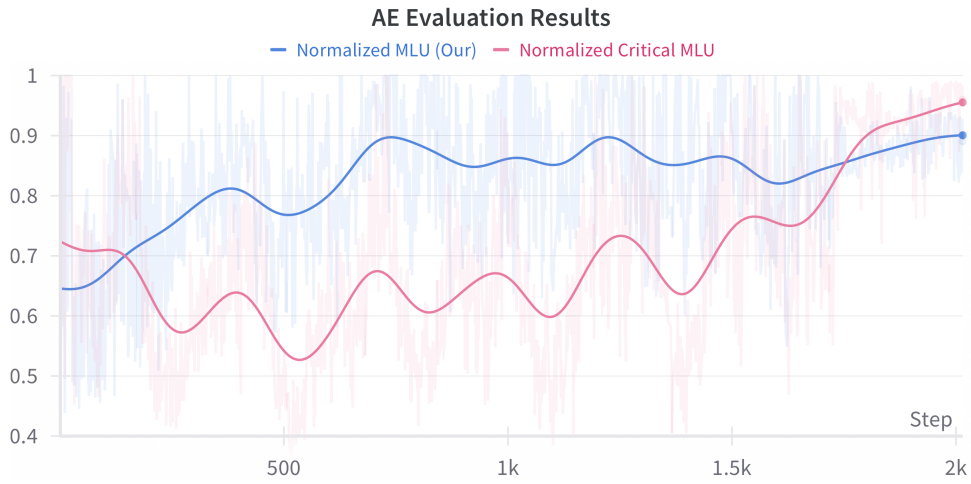

Figure 2: Evaluación de los resultados

Se puede observar que el modelo muestra mejor rendimiento que el método basado en reglas TOP-K Critical, en datos en los que no ha sido entrenado.

Conclusiones

Se ha desarrollado un modelo capaz de generalizar a diferentes distribuciones de tráfico y topologias, modelos anteriores como CFR-RL no eran capaces de sobrepasar métodos heurísticos con simplemente ser entrenados en datos sintéticos. Se ha demostrado la utilidad de los datos artificiales y como estos pueden servir para entrenar modelos de aprendizaje de máquina. Sin embargo, este modelo no es capaz de generalizar a topologias completamente diferentes, solo es capaz de vencer a métodos basados en reglas cuando ha sido entrenado en topologías similares. Recomendaciones de trabajo futuro incluyen intentos de sustituir el valor de base utilizado en el algoritmo REINFORCE, ya que actualmente limita el número de topologías diferentes en las que se puede entrenar.

Referencias

[1] Nick McKeown et al. “OpenFlow: Enabling Innovation in Campus Networks”. In: *SIGCOMM Comput. Commun. Rev.* 38.2 (Mar. 2008), pp. 69–

74. ISSN: 0146-4833. DOI: 10.1145/1355734.1355746. URL: https://doi.org/10.1145/1355734.1355746.

[2] Sakir Sezer et al. "Are we ready for SDN? Implementation challenges for software-defined networks". In: *IEEE Communications Magazine* 51.7 (2013), pp. 36–43. DOI: 10.1109/MCOM.2013.6553676.

[3] Yufei Wang and Zheng Wang. "Explicit routing algorithms for Internet traffic engineering". In: *Proceedings Eight International Conference on Computer Communications and Networks (Cat. No.99EX370)*. 1999, pp. 582–588. DOI: 10.1109/ICCCN.1999.805577.

[4] Junjie Zhang et al. "CFR-RL: Traffic Engineering With Reinforcement Learning in SDN". In: *IEEE Journal on Selected Areas in Communications* 38.10 (2020), pp. 2249–2259. DOI: 10.1109/JSAC.2020.3000371.

[5] Yi-Ren Chen et al. "RL-Routing: An SDN Routing Algorithm Based on Deep Reinforcement Learning". In: *IEEE Transactions on Network Science and Engineering* 7.4 (2020), pp. 3185–3199. DOI: 10.1109/TNSE.2020.3017751.

[6] Hongzi Mao et al. "Resource Management with Deep Reinforcement Learning". In: *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*. HotNets '16. Atlanta, GA, USA: Association for Computing Machinery, 2016, pp. 50–56. ISBN: 9781450346610. DOI: 10.1145/3005745.3005750. URL: https://doi.org/10.1145/3005745.3005750.

[7] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Second. The MIT Press, 2018. URL: http://incompleteideas.net/book/the-book-2nd.html.

[8] Yann Lecun and Yoshua Bengio. "Convolutional Networks for Images, Speech and Time Series". In: *The Handbook of Brain Theory and Neural Networks*. Ed. by Michael A. Arbib. The MIT Press, 1995, pp. 255–258.

[9] Matthew Roughan. "Simplifying the Synthesis of Internet Traffic Matrices". In: *SIGCOMM Comput. Commun. Rev.* 35.5 (Oct. 2005), pp. 93–96. ISSN: 0146-4833. DOI: 10.1145/1096536.1096551. URL: https://doi.org/10.1145/1096536.1096551.

[10] Ying Zhang. *Abilene TM*. Online: cs.utexas.edu/ yzhang/research/AbileneTM/.

Generalization of CFR-RL

Author: Marí Noguera, Carlos.
Director: Zhang, Amy.
Collaborating Entities: ICAI - Universidad Pontificia de Comillas, The University of Texas at Austin

ABSTRACT

Introduction

Software Defined Network (SDN) [1] is an emerging network architecture approach that separates the control plane of the networking device from its data plane. The SDN network controller has a centralized control of the network and can act as the network's operating system. This approach allows the network to be centrally controlled, enabling the possibility of new research on network optimization due to the following reasons. First, as the controller has a global view of the network, it can collect live information of the network state, this information can be used to optimize different aspects of the network such as routing. Secondly, the controller is able to manage network resources by installing and forwarding rules in the switches under its control [2].

Although new opportunities appear for network optimization, it is still not trivial to solve a Traffic Engineering (TE) problem in terms of delay and throughput. The goal of TE is to help network administrators and Internet Service Providers (ISPs) optimize network performance and resource utilization by an efficient allocation of traffic. Other traffic algorithms such as [3] based on Multiprotocol Label Switching (MLPS) achieve near optimal performance via rerouting as many flows as possible and do not take in mind negative results, such as packets out of order, or end-to-end delay. On the other hand current Machine Learning based methods such as [4] [5], achieve near optimal performance, while impacting the network as little as possible, but are not able to generalize to different network topologies.

Project Definition

The objective of the problem is the design and implementation of a Reinforcement Learning based scheme that learns a selection policy and reroutes the selected critical flows to minimize the maximum link utilization (MLU) of a given network. This project will vary from CFR-RL [4] as the objective is to develop a model that is capable of generalizing to different network topologies and traffic patterns, as other approaches such as CFR-RL struggle if the topology or traffic pattern is modified. The designed artificial neural network must be lightweight, and the latency added by running inference on this model must be negligible when compared to the total end-to-end delay. The results of this project will be considered successful if the average MLU is lower than heuristic based methods such as TOP-K Critical.

A selection based algorithm has been selected rather than a fully-fledged RL routing algorithm as learning a policy capable of generalizing selection over multiple policies will not be as complex as learning a routing policy. At every time step t, the model will receive the Traffic Matrix at that time step (TM_t) and will select K flows to be rerouted.

Problem Description

The development of this Deep Reinforcement Learning algorithm aims to solve a TE problem, this Traffic Engineering problem has been transformed into a Reinforcement Learning problem by converting it to a Markov Chain with states actions and rewards.

Reinforcement Learning Problem

By adapting the Markov Chain framework in the definition of the problem we are capable of using Reinforcement Learning algorithms to optimize it. The RL problem formulation from [4] has been modified to facilitate the generalization to multiple topologies. The Markov Chain is defined in the following way.

States The state space represents the input of the neural network, this state space must contain all the information needed to learn the desired policy, the state (s_t), will be formed by the Traffic Matrix at time step t (TM_t) and the topology of the network. As there is an infinite number of observations due to the continuous nature of a traffic matrix, an Artificial Neural Network will be used to approach the problem.

Actions For every state s_t the model will select K critical flows. There is a total of $N*(N-1)$ flows in a network creating a large action space. As the model will select K actions per time step, the action space is defined in the following way $A = \{0, 1, ..., (N*(N-1))-1\}$ inspired by [6] and [4]. For every time step K different actions will be sampled $(a_t^1, a_t^2, ..., a_t^K)$. The ANE will be trained to output a Categorical Distribution from which the actions will be sampled.

Rewards After the sampling the flows, the selected critical flows will be rerouted and the maximum link utilization will be obtained (MLU). To accomplish the generalization objective a different reward function will be defined. To account for multiple topologies, the MLU utilization will be normalized by comparing it to the link utilization obtained by the heuristic based method TOP-K Critical.

$$r = \frac{MLU_{TOP_K} - MLU}{MLU_{TOP_K}} + 1 \tag{6}$$

This reward will be used to update the neural network.

Algorithm To update the neural network the REINFORCE algorithm [7] algorithm with baseline will be used. A registry of state specific rewards will be kept to compute a baseline $V(S_t)$ based on the average the previously obtained rewards. The policy will be updated via the following function:

$$\theta_t = \theta_{t-1} + \alpha[\delta + \epsilon H(\pi(A_t|S_t, \theta_{t-1}))]\nabla \ln [\pi(A_t|S_t, \theta_{t-1})] \tag{7}$$

Where

$$\delta = G - V(S_t) \tag{8}$$

$$H(p(x)) = -\sum p(x) \ln p(x) \tag{9}$$

$$\begin{aligned} \pi(A_t|S_t, \theta_{t-1}) =& \pi(a_t^1|S_t, \theta_{t-1}) \times \pi(a_t^2|S_t, \theta_{t-1}) \times \\ & ... \times \pi(a_t^K|S_t, \theta_{t-1}) \end{aligned} \tag{10}$$

The base REINFORCE implementation has been modified to account for multiple action selection and a small entropy bonus to encourage exploration. ϵ is a hyperparameter with value 0.01, this value has been chosen as it was the final value in [4], and it has not been tuned.

Artificial Neural Network

In order to calculate the probabilities in REINFORCE a Convolutional Neural Network (CNN) [8] will be used. Due to the 2D structure of Traffic Matrices and Adjacency Matrices it was believed that this would be the best approach.

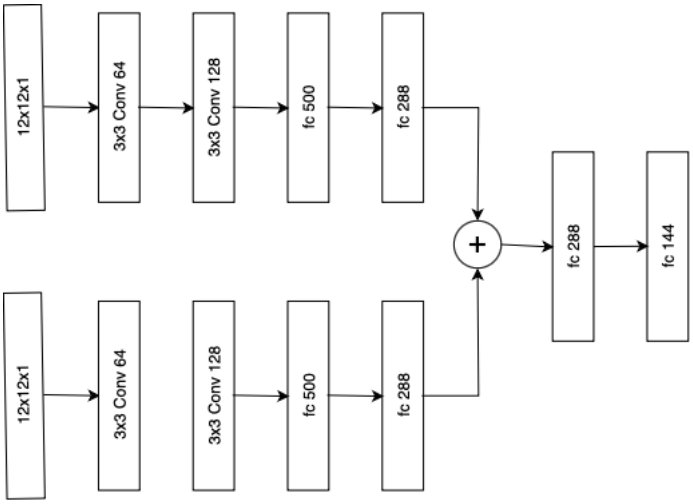

Figure 3: Designed Neural Network

Data Generation

Due to security concerns there is lack of real traffic matrices, this has led to the research of the generation of realistic synthetic traffic matrices. The training data will be generated following the gravity model [9] which is inspired by Newton's gravity model. As the objective of this project is to test generalization, the testing data of the experiment will originate from the Abilene network [10], one of the few public traffic matrices datasets.

Results

To test the generalization capabilities of the designed model, the performance of the model will be evaluated in a topology and traffic data not present in the training datasets. However, this topology is a modification of a topology present in the training dataset with the simulation of a link failure. To assess the performance of the model, it will be compared to the TOP-K Critical approach, that selects the biggest flows out of the congested nodes.

It can be observed that the developed model is able to outperform heuristic based methods on a topology it had not seen before.

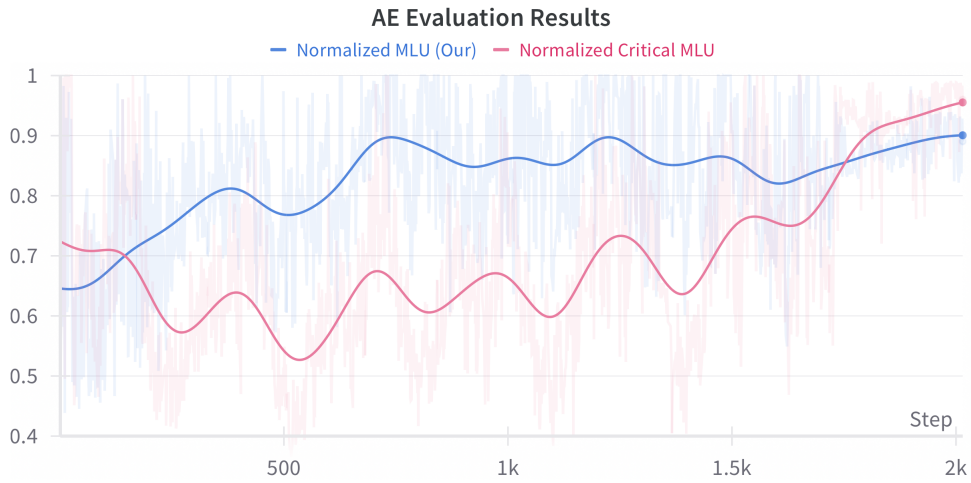

Figure 4: MLU Evaluation in generalization

Conclusions

This project has resulted in the development of a model capable of generalizing to different types of traffic and topologies, previous selection based model such as CFR-RL struggled when trained on synthetic data. With the obtained results the usefulness of this synthetic data has been proved. However, this model is not able to generalize to completely different topologies, it is only capable of outperforming heuristic based methods when it has been trained in similar topologies. Future work may include attempts to develop a model capable of generalizing in different topologies, for this another baseline approach must be selected as the current one limits the number of topologies being used during training.

References

[1] Nick McKeown et al. "OpenFlow: Enabling Innovation in Campus Networks". In: *SIGCOMM Comput. Commun. Rev.* 38.2 (Mar. 2008), pp. 69–74. ISSN: 0146-4833. DOI: 10.1145/1355734.1355746. URL: https://doi.org/10.1145/1355734.1355746.

[2] Sakir Sezer et al. "Are we ready for SDN? Implementation challenges for software-defined networks". In: *IEEE Communications Magazine* 51.7 (2013), pp. 36–43. DOI: 10.1109/MCOM.2013.6553676.

[3] Yufei Wang and Zheng Wang. "Explicit routing algorithms for Internet traffic engineering". In: *Proceedings Eight International Conference on Computer Communications and Networks (Cat. No.99EX370)*. 1999, pp. 582–588. DOI: 10.1109/ICCCN.1999.805577.

[4] Junjie Zhang et al. "CFR-RL: Traffic Engineering With Reinforcement Learning in SDN". In: *IEEE Journal on Selected Areas in Communications* 38.10 (2020), pp. 2249–2259. DOI: 10.1109/JSAC.2020.3000371.

[5] Yi-Ren Chen et al. "RL-Routing: An SDN Routing Algorithm Based on Deep Reinforcement Learning". In: *IEEE Transactions on Network Science and Engineering* 7.4 (2020), pp. 3185–3199. DOI: 10.1109/TNSE.2020.3017751.

[6] Hongzi Mao et al. "Resource Management with Deep Reinforcement Learning". In: *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*. HotNets '16. Atlanta, GA, USA: Association for Computing Machinery, 2016, pp. 50–56. ISBN: 9781450346610. DOI: 10.1145/3005745.3005750. URL: https://doi.org/10.1145/3005745.3005750.

[7] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Second. The MIT Press, 2018. URL: http://incompleteideas.net/book/the-book-2nd.html.

[8] Yann Lecun and Yoshua Bengio. "Convolutional Networks for Images, Speech and Time Series". In: *The Handbook of Brain Theory and Neural Networks*. Ed. by Michael A. Arbib. The MIT Press, 1995, pp. 255–258.

[9] Matthew Roughan. "Simplifying the Synthesis of Internet Traffic Matrices". In: *SIGCOMM Comput. Commun. Rev.* 35.5 (Oct. 2005), pp. 93–96. ISSN: 0146-4833. DOI: 10.1145/1096536.1096551. URL: https://doi.org/10.1145/1096536.1096551.

[10] Ying Zhang. *Abilene TM*. Online: cs.utexas.edu/ yzhang/research/AbileneTM/.

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction

Software Defined Networking (SDN) [1] has emerged as a revolutionary network architecture paradigm that decouples the control plane from the data plane in networking devices. In an SDN environment, a centralized network controller acts as the brain of the network, enabling intelligent and dynamic control over the network resources. This centralized control opens up new avenues for network optimization research due to several key advantages.

Firstly, the SDN controller has a global view of the network, being able to collect real-time information about the state of the network. This comprehensive network visibility enables the controller to make informed decisions and optimize various aspects of the network such as routing. Secondly the controller has the ability to manage resources by installing and enforcing forward rules on the switches under its control [2]. This control of the network facilitates efficient resource allocation and traffic management.

However, despite all these opportunities presented by SDNs for network optimization, solving Traffic Engineering problems (TE) in terms of throughput and delay remains a complex challenge. The primary goal of TE is to aid network administrators and Internet Service Providers (ISPs) in optimizing network per-

formance and resource allocation through efficient traffic allocation. Traditional solution such as those based on Multiprotocol Label Switching (MPLS) [3], are able to achieve near optimal performance by rerouting as many flows as possible. However, these approaches often overlook the potential negative consequences such as packet reordering or increased end-to-end delay.

On the other hand, recent advancements in Machine Learning (ML) have enabled new routing optimization techniques. ML-Bases methods, such as [4] and [5], have demonstrated promising results achieving near-optimal performance while minimizing the impact on the network. However, a significant limitation of these approaches is their inability to generalize to different network topologies. They often rely on memorizing network patterns, or external information such as day of the week, and struggle to adapt to changes in traffic distributions or topology modifications, such as link failures. For this reason, these models require retraining whenever the network undergoes updates, making them impractical and expensive to deploy in real-world scenarios.

To address this challenges, we propose the development of a Reinforcement Learning (RL) based model based on the work of CFR-RL [4]. Our objective is to create a model capable of generalizing to different topologies and traffic patterns, overcoming the limitations of current state-of-the-art approaches. By enabling the model to adapt to changes in the network environment without the need for extensive retraining, we aim to make it more practical and cost-effective for real-world deployment.

Our proposed solution relies on the fact that optimizing network routing in terms of maximum link utilization can be formulated as a Linear Programming (LP) problem. However, directly solving the LP problem can be computationally expensive and not feasible for real-time decision-making. To avoid this issue, our model will select K flows to be optimally rerouted and will receive a reward based

on the obtained MLU compared to heuristic approaches.

Through this project, we aim to develop an adaptable Reinforcement Learning model capable of generalizing to unseen topologies and traffic patterns.

The remainder of this thesis is organized as follows: Chapter 2 provides an overview of the related work on the field of Traffic Engineering, Chapter 3 will contain the description of the problem as well as the methodology and available resources. Chapter 4 will contain the obtained results and the analysis of this result and Chapter 5 will contain the conclusion with an overview of the work and possible research directions.

Chapter 2

State-of-the-art

The advent of Software Defined Networking (SDN) [1] [2] has started a new era of network optimization and traffic engineering (TE). SDNs have revolutionized the ways networks are designed, managed and controlled by decoupling the control plane from the data plane. This architectural shift has opened up new areas of research and innovation, enabling the development of advanced techniques for network optimization and Traffic Engineering.

The centralized control and global network visibility provided by SDNs have caused a significant increase in research efforts focused on exploring novel methods for optimizing network performance and resource allocation. Researchers have used the capabilities of SDNs to design and implement TE strategies that were previously challenging or infeasible in traditional network architectures.

In this chapter, we will discuss the various approaches and methodologies proposed in the literature for TE and routing optimization in the context of SDNs. We will organize the discussion into three different sections, first there will be an introduction to SDNs protocols and architectures, with emphasis on the OpenFlow and ONOs protocols. Next we will discuss the current machine learning approaches, which have gained popularity in the recent years, and lastly we will

focus on traditional routing algorithms and how these algorithms have changed with the arrival of SDNs.

2.1 Software Defined Networks

Software defined networks introduced by McKeown et al. [1] in 2008 were designed with the idea of proposing a major architecture change in network topology to allow researchers to run experiments in heterogeneous switches via the OpenFlow architecture. SDNs were developed with the idea of allowing for new ideas in computer networks to be tested and lowering the entry barrier for new ideas encouraging innovation. As at the time, most of the research carried out on networks remained untested due to the lack of infrastructure. McKeown et al. designed a new switch feature that extended its programmability with the goal of persuading commercial equipment vendors and being able to install this switches in campus networks to enable research.

To clarify the varied concept of SDNs, the ONF (Open Networking Foundation) defined Software Defined Networks as the following *"In the SDN architecture, the control and data planes are decoupled, network intelligence and state are logically centralized, and the underlying network infrastructure is abstracted from the applications."* [11].

Since the release of the OpenFlow architecture many other approaches have been proposed with their benefits and disadvantages, this includes Open Network Operating System [12] or P4-Runtime [13] by the Open Network foundation, or can be implemented via other broader protocols such as NETCONF by Huawei.

2.1.1 OpenFlow Architecture

During their research [1] found that most modern Ethernet switches contain flow-tables that run at line-rate and identified a common set of functions that run in many switches and routers from different vendors. OpenFlow provides an open protocol to program the flow tables found in switches and routers. Via this protocol researchers are able to control their own flows, allowing them to try new routing protocols, security models or even alternatives to IP, while not disturbing the production traffic.

They define three main characteristics and OpenFlow switch must have, a Flow Table (1) with an action associated with each flow entry, allowing the user to tell the switch how to process the flow. A Secure Channel (2) connecting the switch to a remote process (the controller), that allows communication between switch and controller using the OpenFlow Protocol (3). This protocol provides a standard way of controller-swtich communication through which researchers can define entries in the flow table without programming or having direct access to the switch. Each of these entries in the Flow-Table will have three fields, a packet header that defines the flow, an action that will decide how this packet will be processed, and Statistics which keep track of the number packets or bytes for each flow.

There is a distinction between different OpenFlow switches, a switch will be defined as "Type 0" when it is only able to perform the basic actions and only has the required elements. The definition of a "Type 1" switch was not and has not (to the best of our knowledge) been specified yet, as it was left to the development of third parties, some mentioned capabilities are rewriting portions of headers and mapping packets to priority classes.

2.1.2 Other Architectures

Many other approaches, both open-source and proprietary, have been proposed for managing Software Defined Networks. In this section we will primarily focus on the Open Network Operating System (ONOs) architecture and protocol developed by the Open Network Foundation (ONF). ONOs, along with the OpenFlow protocol, is supported by a significant number of contemporary SDN switches.

ONOs, similar to OpenFlow provides the control plane functionality for SDNs, enabling efficient network management. Due to its open-source nature, it has served as a foundation for the development of many additional solutions. The primary objective of ONOs was to create a practical abstraction that operates across multiple servers in a distributed manner, enabling control plane scale-out and fault tolerance while maintaining a global network view. This global network perspective facilitates broader research opportunities that were not available with OpenFlow alone.

The design of ONOs was guided by the following requirements.

- **Throughput:** Up to 1M request/second
- **Latency:** 10-100 ms event processing
- **Size:** Up to 1 TB of data
- **Availability:** 99.99% of service availability.

The ONOs protocol and architecture successfully achieved these goals, providing a more versatile and high level implementation of SDNs compared to alternative approaches like OpenFlow. By leveraging the global network view offered by ONOs, researchers and network operators can gain a comprehensive understanding of the network state and make informed decisions based on this knowledge.

Our work will be built upon the assumption of utilizing the ONOs protocol and architecture, allowing us to benefit from the global network view it provides. This choice enables us to explore advanced network management techniques and develop innovative solutions that leverage the capabilities of ONOs.

2.1.3 Software Defined Networking Issues

Software Defined Networks have emerged as a promising paradigm for controlling modern networks. However, as SDNs gain popularity and are deployed in various environments, they face several challenges than need to be addressed. Sakir et al. [2], provide a comprehensive discussion of the current issues faced by SDNs, which can be broadly grouped into two main areas: scalability and security. In the following sections we will describe each of this issues in more detail, exploring the impact on SDN performance and reliability, and discussing potential solutions proposed in the literature.

Scalability Issues

SDNs were initially conceived as a research tool for campus networks, rather than a comprehensive solution for handling all network traffic. As networks scale up and the number of nodes increases, the communication between switches and controllers can become problematic. These issues may arise due to latency or limited channel capacity, hindering the efficient operation of the network. One potential solution to address this scalability challenge is the deployment of multiple controllers. However, this approach introduces a new set of complexities and requires careful coordination among the controllers. In their work, the authors of [14] explore these and other scalability issues in depth, proposing a range of hardware and software modifications to mitigate the problems. Their solution aims to enhance the scalability of SDNs, enabling them to handle larger networks and higher

traffic volumes without compromising performance or reliability.

Security Issues

The ability to control the actions of individual switches for each network flow is a powerful feature that offers significant benefits for researchers. However, this capability also introduces potential security vulnerabilities. In the event of a malicious attack, an adversary could exploit this control to render the network inoperable by manipulating the flow of traffic. Furthermore, if the network traffic is not properly encrypted, the attacker could gain unauthorized access to sensitive information, leading to serious privacy concerns. By rerouting traffic as desired, the attacker could intercept, monitor, or even modify the data transmitted across the networks. This highlights the importance of implementing robust security measures in SDNs, such as strong authentication mechanisms, encryption protocols, and access control policies, to prevent unauthorized access and protect the confidentiality and integrity of network traffic.

2.2 Machine Learning Based Methods

With the growth in popularity of Machine Learning and Deep Learning, there has been an uproar in methods based on this to optimize routing in networks. Because of the nature of this paper we will classify these methods in Reinforcement Learning based methods and non-Reinforcement Learning based methods.

2.2.1 Reinforcement Learning Based Methods

Routing has been historically one commonly researched topic in Reinforcement Learning with algorithms such as [15] and [16] being published during the 1990s. These methods, provided great theoretical results, but due to algorithm and net-

work limitations at the time and were only discussed on theoretical settings and simulations, causing the initial uproar of research to decline. With the advance of technologies and the development of software defined networks, many new approaches started emerging, as what was not able to be done before due to technological limitations now became possible. This led to the research of new approaches to routing, we can find two main strategies on modern Traffic Engineering problem-solving, selection and rerouting.

The first method selection will focus on selecting a small percentage of flows and optimally rerouting this flows, it has been proven that if the selection of the flows is correct near-optimal performance can be achieved with minimal network interference. One popular selection approach is Critical Flow Routing-Reinforcement Learning (CFR-RL) [4].

The second method is the traditional routing problem, where a Reinforcement Learning agent will focus on rerouting each individual flow, when this approach is successful it can achieve great results in transfer speed, a popular method that uses this approach is Reinforcement Learning-Routing (RL-Routing). [5].

Reinforcement Learning-Routing

Reinforcement Learning-Routing (RL-Routing) [5] by Chen et al. approaches the TE problem with a rerouting approach optimizing in terms of throughput and delay presented as an alternative to Open Shortest Path First (OSPF) [17] and Least Loaded [18] routing algorithms. The scalability issue is addressed by deploying one agent per every switch, in some previous works more than one agent was required per switch causing scalability issues. This approach introduces novel terms to their state space such as *link trust level*, *switch throughput rate*, and *link-to-link switch rate*. With the introduction of this state space they achieved 2.5× and 2.11× speedups over OSPF and LL on average.

This method however also includes the day of the week and part of the day in its state space, this causes this method to be highly reliant on memorizing traffic patterns of the network and making it unsuitable for generalization.

RL-Routing provides a clear example on why selection methods have been chosen for our attempt in routing generalization, rerouting methods depend on memorization obtaining excellent results in one network, and we believe it would result harder to train a rerouting method to be able to generalize.

Critical Flow Rerouting-Reinforcement Learning

Critical Flow Rerouting-Reinforcment Learning (2020) is a selection algorithm that trains a Deep Reinforcement Learning model on a single topology traffic matrix obtaining near-optimal results. This model focuses on selecting only around 10% of flows and will optimally reroute them via the solution of a Linear Programming (LP) Problem. This method is capable of achieving near-optimal performance in terms of minimizing MLU running inference in traffic matrices it has not been trained on and unlike many other traditional routing models it has no significant impact on the end-to-end delay This model does not have access to any network information aside from the traffic matrices and needs to be fully retrained if there is a slight change on the traffic topology or traffic patters of the network as it heavily relies on memorization of traffic patters and the effects each flow has on the network. We have carried out tests on this method changing the traffic patterns and the results obtained were not much better than the base routing algorithm (ECMP), proving the memorization of traffic patterns carried out by the model.

Zhang et al. also designed a Linear Programming Problem through which they could optimally reroute the selected flows in terms of minimizing maximum link utilization. We will be using this same formulation, and will be explained later in the methodology section.

2.2.2 Traditional Machine Learning Methods

Software Defined Networks have also enabled research on non-RL settings, approaches such as [19], explored energy efficient routing algorithms in Wireless Sensor Networks. Tcherak et al., presented a novel routing algorithm Naïve Bayes for Software-Defined Wireless Sensors (NB-SDWSN) based on the conditional probability theorem. Via the use of this algorithm, the controller will choose the best neighboring node towards the end node based on variable parameters such as the battery level or previous results. Via this approach, NB-SDWSN was able to outperform Dijkstra's shortest path routing algorithm in terms of lower packet loss and higher throughput.

Different alternatives have been proposed, Barletta et al. [20] proposed the use of a Random Forest Classifier to predict the bit rate error of non-established light paths in fiber optic communication obtaining an improvement over previous methods on synthetically generated data.

Shen et al. [21] explored the combination of the Least Loaded algorithm [18] with a light assistance from a Machine Learning model. Via including a supervised Naïve Bayes classifier based on historical network snapshots that attempt to predict potential circuit blocking. This approach allowed to reduce the blocking probability of service connection requests outperforming LL and Shortest Path (SP) algorithm, while also providing a parallel computing framework to implement parallel learning in the network.

2.3 Heuristic based methods

Aside from the use of neural networks and machine learning to learn a policy or optimize routing, there is still research carried out from an algorithmic and mathematical background on network optimization and routing.

2.3.1 Traditional routing algorithms

Before the advent of Software Defined Networks, the routing problem was approached with limited information, some of the most common and widely used routing algorithms are Open Shorted Path First (OSPF) [17] Least Loaded [18] and Border Gateway Protocol [22]. ECMP [23] will also be discussed due to its relevance in this project.

Equal-Cost Multi-Path

[23] Equal-Cost Multi-Path (ECMP) routing is a widely adopted technique in modern network infrastructures to achieve load balancing and improve network performance. ECMP allows routers to distribute traffic across multiple paths of equal cost, effectively utilizing network resources and reducing congestion.

In other approaches, routers typically select a single best path to forward traffic towards a destination. However, ECMP introduces the ability to leverage multiple paths simultaneously, as long as they have the same cost or metric. By employing ECMP, network administrators can take advantage of the available bandwidth across multiple links and enhance overall efficiency of the network.

Open Shortest Path First

The Open Shortest Path First (OSPF) protocol, designed for medium and large-scale network routing, has become a widely adopted standard in the field of network communication. OSPF-enabled routers exchange link state information, allowing each router to maintain a comprehensive understanding of the network topology. One of the most significant features of OSPF is its ability to partition a network into multiple areas while ensuring connectivity through a central area known as the backbone. This hierarchical structure allows for efficient routing and scalability,

making OSPF particularly suitable for large and complex networks.

In the context of inter-domain routing, OSPF plays a crucial role in connecting different Autonomous Systems (AS). An AS refers to a group of networks under a single administrative domain, often managed by a single entity such as an Internet Service Provider (ISP) or a large organization. Each AS may employ its own internal routing protocols, tailored to their specific requirements and network characteristics. However, to communicate and facilitate communication between different AS, OSPF serves as a common protocol for exchanging routing information.

Border Gateway Protocol

The Border Gateway Protocol (BGP) has emerged as the predominant external routing protocol in the Internet ecosystem. BGP views the Internet as a collection of interconnected Autonomous Systems, each identified by a unique 2-byte number. This protocol employs a distance metric to determine the optimal route between AS, ensuring efficient and reliable communication across the global network.

One of the key features of BGP is its ability to exchange routing information between AS. When a BGP router sends a message to its neighboring routers, it includes an additional field that records the AS traversed along the path. This information enables routers to build a comprehensive view of the network topology, facilitating informed routing decisions and improving the overall performance and stability of the Internet.

BGP routers continuously monitor the network for any updates or changes. Whenever a router detects a modification in the network state, such as link failure or the introduction of a new AS, it quickly communicates information to its neighboring routers using TCP messages. This proactive approach ensure that routing tables are kept up to date, allowing for quick convergence and minimizing

the impact of network distributions.

Least Loaded Routing Algorithm

The Least Loaded (LL) Routing Algorithm emerged as a dynamic and adaptive approach to load balancing in modern network systems. This algorithm aims to efficiently distribute traffic across multiple servers or paths based on their current load or capacity, ensuring optimal resource utilization and minimizing the risk of overloading any single server or path.

The core principle behind the LL Routing algorithm is to route incoming requests or traffic to the server or path that is currently handling the least amount of load. By continuously monitoring the load levels of each server or path, the algorithm can make informed decisions and dynamically adjust the routing strategy to maintain a balanced distribution of traffic. Via directing traffic to the least loaded server or path the algorithm helps prevent bottlenecks and ensures all resources are utilized effectively. This load balancing approach improves the overall system performance, reduces response times and increases the user experience.

Another advantage of this approach is its scalability. As the network grows and the number of servers and paths increases, the algorithm can seamlessly adapt and continue to provide efficient load balancing. This scalability is particularly important in large-scale systems where the demand for resources can fluctuate over time.

The model to be developed will resemble LL Routing in some aspects, as it will select a number of flows to reroute to reduce the maximum link utilization of the system, while penalizing longer routes.

2.3.2 SDN-Enabled routing algorithms

This section includes traditional routing algorithms that have been enabled by the invention of Software Defined Networks, as for the Reinforcement Learning based methods we can divide this section into two clearly defined sections routing algorithms and selection algorithms with the same distinction as in the previous section.

Multiple routing techniques have been presented since the introduction of SDNs [24] [25], Bin et al. presented a Resource Scheduling algorithm for ForCES (Forwarding and Control Element Separation) networks [26] to meet flexibility, programmability and scalability demands in node resources. This algorithm relies on the selection of a cost or a priority and then scheduling to meet the demands of the user via an economic model.

Jain et al. presented B4 [27] a private WAN connecting Google's data centers across the planet, achieving near 100% link utilization via the use of the OpenFlow model, in contrast to the average 30-40 % obtained in normal WANs due to failures and re-routings. To achieve this they defined two main points to focus their solution about, based around their unique requirements.

- **Failures:** Accepting failures as inevitable and common events.
- **Central Control:** Switch hardware that exports simple interfaces to program forwarding table entries under central control.

Martínez et al. designed a path computation element-based strategy for Label Switched Paths re-optimization in Optical Networks [28] and many other approaches have been presented.

For simplicity reasons we will focus on the two heuristic selection algorithms developed by Zhang et al. [4] in CFR-RL, as we will be comparing our results to these approaches as they follow the same K item selection approach.

The TOP-K approach will select the biggest K numbers from the traffic matrix, a simplified version of the algorithm is depicted below:

Listing 2.1: TopK Algorithm

```
def get_topK_flows(tm):
    f = {}

    # We store all the indixes in a dictionary
    for idx,source, destination in pairs:
        f[idx] = tm[soruce][destination]

    # We sort the list
    sorted_f = sort(f)

    # We select the candidate flows,
    cf = []
    for i in range(self.max_moves):
        cf.append(sorted_f[i])

    return cf
```

The TOP-K Critical algorithm is an evolution of the previous algorithm, in which the biggest flows of the congested links will be selected. This approach however requires information on the ECMP distribution and the shortest paths of the network.

Listing 2.2: TopK Critical Algorithm

```
def get_critical_topK_flows(tm):
    # Get link loads of ECMP and normalize
    link_loads = self.ecmp_traffic_distribution(tm_idx)
    critical_links = sort(link_loads /self.link_capacities)

    cf_potential = []
    for idx in range(num_pairs):
        for path in shortest_path[idx]:
            if path intersects with critical_links:
                cf_potential.append(idx)
```

```
                break

    # return the topK flows of the selected critical flows
    return self.get_topK_flows(cf_potential)
```

Chapter 3

Description

In this chapter, we will provide a comprehensive description of the project, containing its objectives, methodologies and implementation strategies. We begin by clearly defining the problem that our project aims to address and highlight the significance of finding a solution.

We will first state the purpose of this project and the motivations behind it and state the problem we aim to solve. Then we will define the Reinforcement Learning problem and the formulation of the Traffic Engineering problem as a Markov Chain, defining the state and action space and the rewards, as well as discussing the different choices.

After establishing the problem, we will continue setting specific and measurable objectives that we seek to accomplish. These objectives will serve as guide to evaluate the success and effectiveness of our defined approaches.

To tackle the problem, we will explore various approaches, and will define the different neural network architectures that have been used through the project, and state the positives and negatives of each of the approaches.

Next we will discuss one of the main problems in traffic engineering, the lack of reliable data and will discuss the different approaches used for generating real-

istic and representative network traffic data. This will involve exploring current datasets as well as different traffic and topology generation models.

Finally, we will discuss the Reinforcement Learning algorithm, and the modifications that have been carried out to the update and discuss the different attempted approaches and define the Linear Programming Problem used through all our project to find and re-route critical flows.

3.1 Motivation

With the raise of Software Defined Networks there has been an increasing number of publications on the optimization of routing on large scale networks and datacenters as described in the previous section. All this prior research has focused on the optimization of a single network achieving near-optimal performance.

We can define the term of optimal performance as Optimal Routing, in terms of Maximum Link Utilization (MLU), can be achieved via solving a Linear Programming (LP) Problem. This solution, is not suitable for deployment as the time and compute needed to solve the LP problem are greater than those required in SDN Controllers and the latency caused by the process would not be insignificant when compared to the end-to-end delay.

Most current research avoids optimizing routing on a generalized case, which can be useful for both datacenters and large scale networks, as they may experience link failure or update links capacities, while also benefitting smaller networks that may not have enough resources to train a Deep Reinforcement Learning model in their network.

The development of a model that is capable of generalizing to multiple networks, would greatly benefit network administrators and would also include a reduction in power consumption as there would be no need to retrain a model

from scratch every single time there is an update on the network.

3.2 Problem specification and objectives

This section defines the problem we aim to solve and the objectives to be met by the end of the project. Additionally, we introduce the Markov Chain used to formulate this problem as a Reinforcement Learning problem.

3.2.1 Problem Specification

The primary objective of this project is to optimize routing in Software Defined Networks (SDNs) by minimizing the Maximum Link Utilization (MLU). To achieve this, we will develop a Deep Reinforcement Learning (DRL) model capable of optimizing routing across multiple network topologies. In order to convert the routing problem into a Reinforcement Learning we will proceed to define a Markov Chain.

States

In a Markov Chain, the state captures the relevant information at each time-step and must adhere to the Markov Property, which stipulates that the future state only depends on the current state and not on the trajectory of the current episode. The state of the designed approach will be formed by two components a Traffic Matrix and the network topology.

The traffic matrix represents a snapshot of a network at a given time-step, showing all the ongoing flows and their source, destination and size. By including the traffic matrix to our state, we will ensure that our model has access to all the information about the current flows of the network.

However, the traffic matrix does not provide a complete picture of the network environment as it has no information about the topology. To allow our model to make informed routing decisions and achieve generalization capabilities, we will need to provide information about the underlying network topology. We will incorporate the network topology into the state space, which will include information about the links connecting the nodes and their different capacities.

Actions

The actions space of our model has been inspired by the approach presented in [4]. At each state s_t, the model selects K critical flows to optimize. However, as the number of flows grows quadratically with the number of nodes in the network, the action space becomes significantly large. This large action space poses a challenge in our case as we need to perform multipole actions per time-step, and creating combined actions would only increase the problem.

To tackle this issue, we adopt a strategy inspired by [6] and define the actions space as $A = \{0, 1, ..., (N \times (N-1) - 1)\}$ where N represents the number of nodes in the network. Instead of considering all possible combinations of actions we will sample K distinct actions $(a_t^1, a_t^2, ..., a_t^K)$ from this action space without replacement. By sampling from the same distribution, we can effectively address the large action space issue while avoiding the computational complexity associated with combined actions.

This approach, at least in theory, would allow us to handle large-scale networks with a significant number of flows. It is worth noting that the choice of K, the number of critical flows to be rerouted is an important hyperparameter in our model, and it should be carefully tuned to strike a balance between optimization and the overhead of rerouting too many flows, and the other effects this may have such as end-to-end delay.

Rewards

After executing each action, the model will receive a reward signal that is used as feedback for the learning process. In our defined environment, the model will optimally route the selected flows and obtain the resulting Maximum Link Utilization (MLU). However, using the MLU value directly as a reward has limitations, as the quantities can vary significantly between different networks or even within the same network at different time points. To address this issue, we designed a new reward that provides a more consistent feedback signal.

We decided to compare the obtained MLU with the MLU achieved by the heuristic method described in the previous chapter TOP-K Critical. We will compare our obtained MLU with the MLU obtained by the heuristic method TOP-K Critical (MLU_{TOP_K}).

The reward is defined by the following equation:

$$r = \frac{MLU_{TOP_K} - MLU}{MLU_{TOP_K}} + 1 \tag{3.1}$$

This reward offers several advantages, it provides a normalized measure of the model's performance compared to heuristic methods. By adding 1 to the reward we accelerate the learning process by having positive and negative numbers, theoretically this should not matter in the long term due to the presence of a baseline, but we will still observe a speedup during the training. With this reward we are also able to quickly assess the performance of the model during training.

3.2.2 Project Objectives

To evaluate the effectiveness and practicality of our solution for optimizing routing in SDNs, it is crucial to establish clear and measurable objectives. These objectives will serve as benchmark to assess the performance of our model and determine its

suitability for real-world deployment. In this section, we identify and define the key objectives that we aim to achieve by the end of the project.

Our primary focus is the design of a model that not only minimizes the MLU but also exhibits strong generalization capabilities and operates efficiently in terms of computational resources. By setting these objectives, we ensure that our solutions addresses the key challenges and requirements of optimizing routing in diverse environments.

In the following subsections, we will explain each objective in detail, explaining their significance and criteria for determining their successful completion.

Generalization Capability

The primary objective of our project is to develop a model that demonstrates strong generalization capabilities. We aim to create a model that can effectively optimize routing in various network topologies given a fixed number of nodes. Achieving this goal would validate that the model is not merely memorizing specific traffic patterns but rather learning to make intelligent routing decisions based on the given information of the network.

It is important to note that memorizing traffic patterns is not inherently negative. In fact, it can be positive in scenarios where there the network exhibits consistent and predictable behaviour and in cases where we just want to adapt to a single network topology. However, our focus is on generalizing, for which we want to avoid memorization.

We will consider our generalization to objective to be achieved when our model is capable of outperforming TOP-K Critical on data not seen before during the training, this can be achieved in different manners.

- **Only Traffic:** The model being capable of generalizing in never seen before traffic on topologies it has been trained on.

- **Traffic and Topology** The model being capable of generalizing in never seen traffic in never seen topologies.

It is important to note, that for both, we will require the model to be able to perform in multiple topologies. For the second objective, we would also include being able to perform on modifications of topologies seen before to test for adaptability of topology updates or link failure.

Negligible Delay and memory usage

Another crucial objective of our project is to develop a model that ca be seamlessly integrated into real-world scenarios without introducing significant delays or consuming excessive memory resources. In practical SDN deployments need to operate efficiently on SDN controllers, ensuring it does not affect the overall performance of the network.

To achieve this objective, we will focus on designing a compact neural network architecture. By minimizing the size of the neural network, we can reduce the computational overhead and memory requirements associated with running the model on SDN controllers. A smaller model also translates to faster inference times, which is critical for real-time routing optimizations.

It is important to acknowledge that the real world performance of the solution may vary on factors such as the value of K, or the size of the network. As the network scales and the value of K increases, the computational complexity and requirements of the Linear Programming Problem will grow.

To define a success criterion for this objective, we will aim for our model to perform inference in less that half the time it takes to solve the designed LP Problem, which will be showed in the following sections. This benchmark ensures that the model does not introduce significant delay into the overall routing process.

Optimizing the model's architecture implementation will be a key focus to meet this objective, for deployment some techniques such as pruning [29] or quantization [30] may be explored and considered an option as long as they do not cause any other issues or worsen the performance of the model.

3.3 Methodology

This section will contain details about the tools and techniques that are to be used in order to obtain a model that will attempt to meet the established objectives. We will proceed to explain the techniques used for data generation, the approach to be used to design a neural network and possible optimizations for the deployment of this network.

3.3.1 Data

One of the biggest challenges in the topic of Traffic Engineering is the lack of realistic data, due to security concerns almost no network shares information about their traffic matrices and topology. Due to this reason there has been discussion on the generation of synthetic data and how well it represents realistic data. Prior research such as CFR-RL and RL-Routing does not mix the usage of real and synthetic data, when trained on real data it is tested on real data, and when trained on synthetic data it will be tested on synthetic data. In this project we will attempt to train on synthetic data in our case we will use the Gravity method [9], to generate our synthetic traffic matrices. This method is still being used on state-of-the-art research [31] and is the one that best represents the real data that we have access to from the Abilene network [10].

Gravity Model

The gravity traffic model is based on Newton's law of gravitation. In this approach the traffic from each source to each destination is modelled as a random process and assumes independence between flows. In the ideal case we assume that the sum of the traffic leaving nodes and the traffic entering nodes it is equal 3.2. For simplicity reasons we will assume we are only generating one traffic matrix (TM). The value at $TM_{i,j}$ shows the traffic from node i that has node j as final destination (inside the network), and it does not represent the next jump.

$$T^{total} = \sum_{i=1}^{n} T_i^{in} = \sum_{i=1}^{n} T_i^{out} \tag{3.2}$$

Where T_i^{in} is the traffic that will ingress node i and T_i^{out} is the traffic that will egress node i. This equation only states that there is no packet that exits the network.

The gravity model is defined in the following way

$$T(n_i, n_j) = T^{total} \frac{T^{in}(n_i)}{\sum_k T^{in}(n_k)} \frac{T^{out}(n_j)}{\sum_k T^{out}(n_k)} \tag{3.3}$$

Which can be simplified via the use of matrices in the following way:

$$P = T \times p_{in} p_{out}^T \tag{3.4}$$

This way we have simplified the synthesis of traffic matrices, as we will only have $2N$ random variables instead on N^2, we will generate the ingress and egress values of every node via sampling from an exponential distribution.

$$T_n = \lambda e^{-\lambda x} \tag{3.5}$$

No importance has been given to the parameter of the exponential as long as

it is the same value for every node, for simplicity reasons we chose to stick with 1.

There is only one value to be discussed, the total amount of traffic, this value is not important for the algorithm per se as the values will be normalized, the importance of this value falls in the scale of the maximum link utilization, but as this value is also normalized we have chosen to simply choose this value, so it resembles the distribution in the only real data we had access to, the Abilene Network, the distribution of this was a uniform distribution, which we copied in mean and limits.

Topology generation

While the number of real topologies freely available online is much greater than the number of traffic matrices, synthetically generated topologies are being used as there has been no evidence of this performing better or worse than real network topologies as far as there is a diverse enough dataset. We have focused on generating three different types of topologies, resembling autonomous-systems (AS) [32], modifications of the Abilene topology to test link failure robustness and random topologies.

3.3.2 Algorithm

In order to solve the Reinforcement Learning problem proposed previously a modified version of the REINFORCE algorithm [7] has been chosen, this was the algorithm used previously on [4] and our early testing results were positive, for which no few approaches have been discussed.

REINFORCE is a policy-gradient algorithm, where we learn a policy instead of a state-value function, via learning this policy we attempt to simplify the learning of our DRL model. The algorithm 3.6 is a modified version of REINFORCE with the objective of adding a baseline, the entropy and multiple action selection per

step. Our policy will be updated via performing gradient ascent on the following equations.

$$\theta_t = \theta_{t-1} + \alpha[\delta + \epsilon H(\pi(A_t|S_t, \theta_{t-1}))]\nabla \ln [\pi(A_t|s_t, \theta_{t-1})] \tag{3.6}$$

$$\delta = G - V(s_t) \tag{3.7}$$

$$H(p(x)) = -\sum p(x) \ln p(x) \tag{3.8}$$

$$\begin{aligned} \pi(A_t|s_t, \theta_{t-1}) =& \pi(a_t^1|S_t, \theta_{t-1}) \times \pi(a_t^2|S_t, \theta_{t-1}) \times \\ & ... \times \pi(a_t^K|S_t, \theta_{t-1}) \end{aligned} \tag{3.9}$$

Where the value of the function $V(s_t)$, equates to the average of the reward of all previous experiences in the given state s_t, the ϵ is a hyperparameter which determined the importance of the entropy, we have chosen 0.01 as stated by [4], we believe this value has little importance in the end results.

There was an attempt of using actor-critic methods [7] with little success due to the complexity of learning a value function in these scenarios. An actor-critic method may still be a considerable approach in other solutions where there is an infinite amount of data, as there would be no need of keeping a baseline value for every traffic matrix of every topology.

3.3.3 Linear Programming Problem

In order to evaluate the performance of the performance of our results we will compare the obtained results with the optimal solutions. The mathematical for-

mulation of this solution originates from the CFR-RL paper, [4] where the idea of using the optimal solution to evaluate the problem comes from. We will start by the explanation of the used nomenclature.

Nomenclature

$G(V, E)$	Network with V nodes and E directed edges.	
$c_{i,j}$	The capacity of link (i, j)	$(i, j) \in E$
$l_{i,j}$	The traffic load on link (i, j)	$(i, j) \in E$
$D^{s,d}$	The traffic demand from source to destination	$(s, d) \in V, s \neq d$
$\sigma_{i,j}^{s,d}$	The percentage of traffic demand from source s	$(s, d) \in V, s \neq d$
	to destination d routed on link $\langle i, j \rangle$.	$(i, j) \in E, \langle s, d \rangle \in f_K$
f_K	The selected critical flows.	
U	Maximum Link Utilization	

Algorithm

By the default the traffic will be routed using the ECMP algorithm and the route the selected flows f_K will follow will be defined by conducting an explicit routing optimization for the selected critical flows $< s, d > \in f_k$. Given a network $G(V, E)$ with the set of traffic demands $D^{s,d}$ for all selected critical flows ($\forall \langle s, d \rangle \in f_K$) and the background link load $\{\hat{l}_{i,j}\}$ contributed by the links routed via ECMP obtain the explicit routing ratios $\{\sigma_{i,j}^{s,d}\}$ for each critical flow, so that the maximum link utilization U is minimized. To search for all possible under-utilized paths the routing problem is formulated as the following optimization:

$$minimize \qquad U + \epsilon \times \sum_{\langle i,j \rangle \in E} \sum_{\langle s,d \rangle \in f_K} \sigma_{i,j}^{s,d} \tag{3.10}$$

subject to:

$$l_{i,j} = \sum_{\langle s,d \rangle \in f_K} \sigma_{i,j}^{s,d} \times D^{s,d} + \hat{l}_{i,j} \qquad i,j : \langle i,j \rangle \in E \tag{3.11}$$

$$l_{i,j} \leq c_{i,j} \times U \qquad i,j : \langle i,j \rangle \in E \tag{3.12}$$

$$\sum_{k:\langle k,i \rangle \in E} \sigma_{k,i}^{s,d} - \sum_{k:\langle i,k \rangle \in E} \sigma_{i,k}^{s,d} = \begin{cases} -1 & \text{if } i = s, \\ 1 & \text{if } i = d \\ 0 & otherwise. \end{cases} \qquad i \in V, sd : \langle s,d \rangle \in f_K \tag{3.13}$$

$$0 \leq \sigma_{i,j}^{s,d} \leq 1 \qquad s,d : \langle s,d \rangle \in f_K, i,j : \langle i,j \rangle \in E \tag{3.14}$$

Via the solution of this optimization problem designed by Zhang et al. [4] we are able to optimally reroute the selected flows minimizing link utilization. According to their paper "the epsilon term in equation 3.10 is needed to avoid optimal solutions that will include unnecessary long paths to avoid congested links, where ϵ ($\epsilon > 0$) is a sufficiently small constant to ensure the minimization of U takes higher priority" [33].

This LP problem will be solved by the use of the LP solver Gurobi [34] and the SDN controller will receive the selected actions for the different flows.

3.3.4 Artificial Neural Network

It was decided to use an Artificial Neural Network to solve this Reinforcement Learning problem, due to the nature of traffic matrices and their continuous nature, the use of the Reinforce Algorithm instead of a state-value method and the complexity of the problem inclined the balance in favour of DRL. Due to the de-

ployment objectives established in previous sections we are limited in the number of parameters and Floating point operations (FLOPs). For this reason it was decided that the designed model must be smaller than 100 MB prior quantization, using 32 bit precision this equates to 25 million parameters.

This number may seem big, but both pruning [29] and quantization [30] may be performed to reduce the size of this model substantially while achieving a similar performance.

Neural Network Architectures

There have been multiple architecture design approaches in order to find an appropriate architecture to solve the problem. Three major approaches were considered, using a Convolutional Neural Network [8], using a traditional Multilayer Perceptron (MLP), or using an architecture based on attention blocks [35].

The use of regularization techniques such as layer normalization [36] has been found to have little to no effect in the convergence rate of the agent, experiments with and without layer normalization were performed, and we found no significant differences.

Multilayer Perceptron The Multilayer Perceptron represents the original approach to deep learning, this neural networks consist of fully connected layers, where each layer will contain a specified amount of neurons. In a fully connected network, every single neuron of the first layer will be connected to every single neuron of the second layer, normally followed by non-linear activation functions, for this experiment Gaussian Error Linear Units (GELUs) [37] provided the best results. The Multilayer perceptron designed for this task can be seen in figure 3.1

Convolutional Neural Network Convolutional Neural Networks were developed to address the scaling problems of MLPs. The problems were caused due

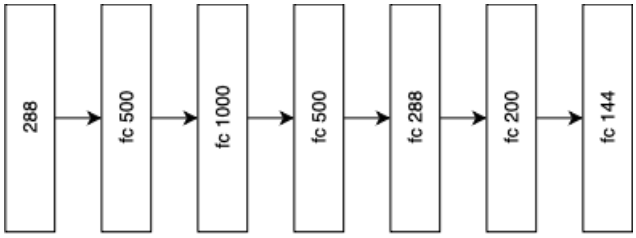

Figure 3.1: Multilayer Perceptron

to the large number of parameters required to analyze images, for this reason the CNN architecture was defined with images in mind. The weights of a CNN do not scale with the size of the input, this scale with the size of the kernel. A kernel can be imagined as a shape, normally squared that gets convoluted with the image, this approach allows locating features in an image, specially if features are in proximal. To attempt to solve our problem we defined a CNN that can be seen in figure 3.2, the two inputs could have been configured as channels, but this caused worse results.

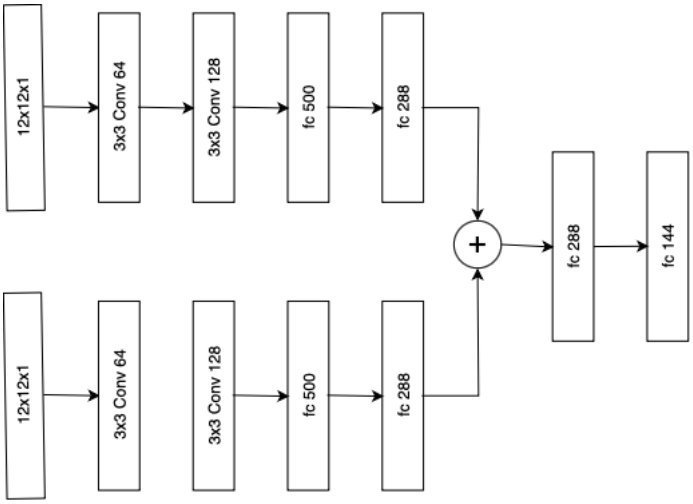

Figure 3.2: Convolutional Neural Network

Attention Attention models have recently gained traction due to their uses in Natural Language Processing and Computer Vision tasks with surge of foundation

models. This models can have billions of parameters and can fully benefit from the advantages of the transformer architecture. This architecture focuses on computing some approximate of the correlation between two different parts, in NLP the parts could be words or tokens, while in computer vision parts could be single pixels or a patch pixels as presented in the Visual Transformer (VIT) [38]. Due to the low scale of our input (12x12x2) we decided that each single value could be the equivalent of a token and designed the following architecture 3.3.

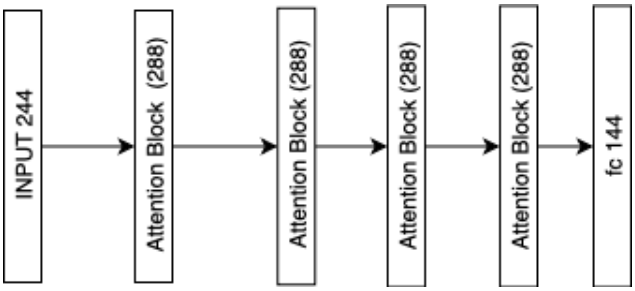

Figure 3.3: Attention-Based Neural Network

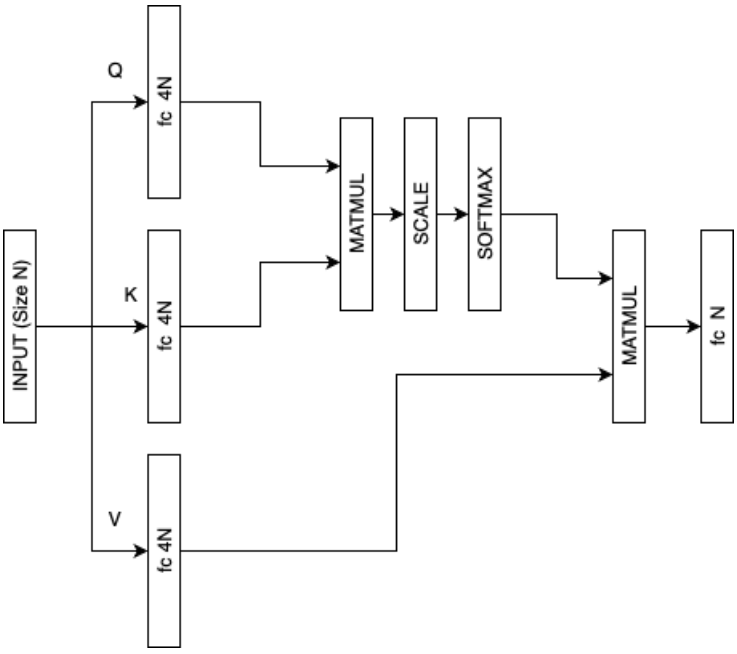

Figure 3.4: Attention Block

Previous work had been carried out on the use of transformers in planning, obtaining better results than the A* algorithm, [39]. Due to this we believed that this problem could have benefited from an attention-based architecture.

Weight initialization

Reproducibility is a major known issue in Reinforcement Learning, specially in DRL [40]. Completely different learning curves may arise from slight changes such as different seeds, different trajectories or in this case weight initialization. When attempting to replicate the results obtained in the CFR-RL paper [4], the reported results could not be matched even when the environment seeds and neural network were identical, the only difference being the library used to generate the neural network. Pytorch [41] and Tensorflow [42] use different weight initialization techniques, causing different results in DRL problems. For this reason the weights are initialized in a customized way [43] [44], which has been found to be useful in most cases. The algorithm for weight initialization is as follows:

Listing 3.1: Weight initialization

```
def weight_init():
    """Custom weight init for Conv2D and Linear layers."""
    if layer is linear:
        init.orthogonal(m.weight.data)
        bias = 0
    elif layer is conv:
        bias = 0
        mid = shape // 2
        gain = nn.init.calculate_gain('relu')
        init.orthogonal(m.weight[:, :, mid, mid], gain)
```

3.3.5 Training

To maximize the efficiency of the training we have followed a classical Reinforcement Learning approach were multiple agents are collecting experiences interacting with the environment and a central agent is updating the weights and distributing them across the different agents. A simplified version of the training loop of a normal agent and the central agent can be found next.

Listing 3.2: Training loop of an agent

```
def run_agent():
    # Load the weights from central agent
    model_weights = model_weight_queues.get()
    model.load_state_dict(model_weights)

    run_idx = 0

    while True:
        # Get experience
        state = game.get_state()
        actions = model(state, mat)
        reward = game.step(actions)

        run_idx += 1
        if run_idx == 10:
            # Share experience
            experience_queue.put(experience)
```

```
        # Load weights from central agent
        model_weights = model_weight_queues.get()
        model.load_state_dict(model_weights)
        run_idx = 0
```

There were some experiments during which the action selection was overriden during the early stages of training and there was a chance of carrying out the action selected by TOP-K Critical. This led to some beneficial results but also to bigger variance in the results, this will be portrayed in detail in the results section.

Listing 3.3: Training loop of the central agent

```
def central_agent():
    algorithm = Reinforce()
    for step in range(training_steps):
        # Share the weights
        model_weights = model.get_weights()
        model_weight_queues.put(model_weights)
        # Collect experience from agents
        experience = experience_queues.get()
        # Update the weights
        model.update(experience)
```

Note that this code is not equal to the one in the GitHub repository and some of its complexity has been abstracted for clarity reasons.

This training is being executed on a distributed manner, where every agent equates to one thread of the CPU, due to hardware limitations most agents have been trained with 15 agents, which also equates to 15 topologies

During most runs the number of traffic matrices assigned to every agent varied

between 200 and 1000, we believe that the lower the number is the better unless we start overfitting to our traffic matrices. By keeping the number of traffic matrices low, we can keep the baseline updated and speed up the training.

3.3.6 Deployment

In this section possible approaches for deployment will be discussed, as stated above in the objectives we are aiming to develop a model that can be deployed in SDN controllers, for this reason the designed neural networks all attempt to be as simple as possible. We will proceed to discuss possible approaches on how we may adapt the model further after training, some of this approaches may require further fine-tuning to adapt to the results.

Quantization

Quantization is a technique used to reduce the precision of the weights in a machine learning model, typically the weights are converted from floating-point numbers to integers with lower bit-width [30]. The primary motivation behind quantization is to reduce the memory footprint and computational complexity of the model, while maintaining its performance and accuracy. Quantization could provide us with the following benefits:

- **Memory Efficiency:** By transforming our weights from 32-bit floats to 8-bit integers we are able to reduce by 4 the memory footprint of our model.
- **Faster Inference:** Fixed point operations are faster than floating point operations.
- **Energy Efficiency:** Most of the power consumption of neural networks is caused by moving weights between different memory types, by reducing the size we are able to reduce the consumption.

By adapting quantization we would be able to reduce the memory footprint of our model and reduce the inference time of this model. This approach should be considered for the final results and evaluation.

Pruning

Pruning is a technique used to reduce the size and complexity of a machine learning model by removing the redundant or less useful components, such as weights, neurons or entire layers [29]. The goal of pruning is to create a more compact and efficient model while preserving its performance. The process of pruning can be divided into two different categories: Structured Pruning and Unstructured pruning.

Structured Pruning Structured pruning is a technique that aims to reduce the complexity of the model by removing entire structures, for example removing full channels from a convolutional layer. This approach does not allow for as much pruning as the unstructured pruning approach, but it does not need specialized hardware. However, this approach can still be challenging, and we may not be able to obtain as much performance as we expect.

Unstructured Pruning Unstructured pruning involves removing individual weights or connections from the model based on certain connections, normally their magnitude. This approach results in a sparse model with irregular connectivity. Due to the irregular connectivity this approach would require the use of specialized hardware, which would not be optimal for our use case.

3.4 Resources

The development of the project will rely on a diverse set of resources, including real-world datasets, computing infrastructure, and software tools to ensure efficiency, reproducibility, and practicality for our proposed solutions.

3.4.1 Datasets

To validate the effectiveness of our approaches in real-world scenarios, we will utilize the ABILENE network dataset [10]. The ABILENE network is a high performance backbone network that connects various educational and research institutes across the United States 3.5. This dataset provides a rich collection of traffic matrices (TMs) that captures real-world traffic patterns and dynamics observed in the network. By using this dataset, we can evaluate the performance of our model under realistic network conditions and demonstrate the applicability of our solution.

In addition to the Abilene dataset, synthetic traffic data will be generated following the gravity definition described on the previous section. This synthetic datasets will allow us to explore a greater amount of topologies and traffic matrices, and by training on synthetic data and testing on real data we can potentially prove the usefulness of this data.

3.4.2 Computing Resources

To support the computational demands of our project, we will utilize a combination of personal and high-performance computing resources. For local development and evaluation, a personal laptop will be used for its accessibility.

However, to handle more computationally intensive tasks, such as long training

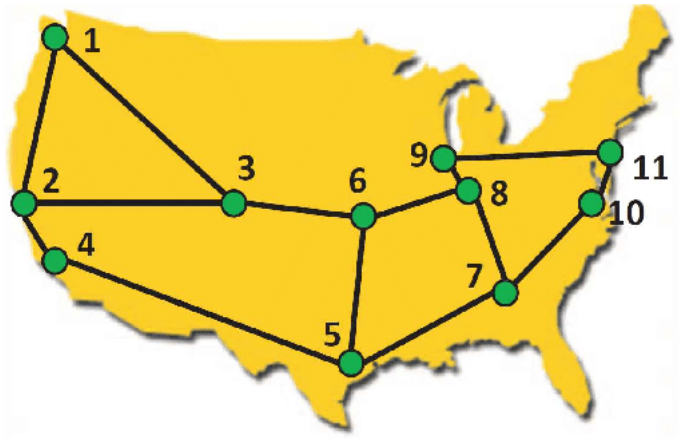

Figure 3.5: Abilene Network Topology

Model	Intel Xeon Platinum 8280 ("Cascade Lake")
Hardware threads per core:	56 cores on two sockets
Hardware threads per core:	1
Clock rate:	2.7 GHz nominal
RAM:	192GB (2933 MT/s) DDR4
Cache:	32KB L1 data cache per core; 1 MB L2 per core; 38.5 MB L3 per socket Each socket cache up to 66.5 MB (L2 + L3)
Local storage:	144GB /tmp partition on a 240GB SSD.

Table 3.1: Specifications of a Compute Node from Frontera

runs or multiple agents running in parallel, the Frontera Supercomputer from the Texas Advanced Computing Center will be used. Frontera is currently one of the world's most powerful academic supercomputers, offering a massive amount of computing power and parallel processing capabilities. By using this system, we are able to do an in-depth exploration of different models and approaches.

3.4.3 Software Tools and Frameworks

To facilitate the development and evaluation of our models, we will employ a range of software tools and frameworks. The primary machine learning framework for

this project will be PyTorch [41], as it offers a flexible and intuitive interface for building and training neural networks.

In addition to Pytorch Tensorflow [42] will be used to observe and reproduce the CFR-RL results, as they shared a codebase using tensorflow.

For real-time visualization of training results, Weights and Biases (wandb) will be used [45], as it will allow us to remotely view the live results of training runs, useful when training on a server such as Frontera.

Finally, for version control and code sharing we will use GitHub, all the code and training data will be publicly available at the GitHub repository. Git allows us to keep track of the different updates of the code.

By using these technologies, we aim to develop a robust and practical solution for network optimization in Software Defined Networks. The combination of these resources will allow us to conduct experiments, validate our approaches and test the efficacy of our approach in realistic scenarios.

3.5 Market Analysis

Software Defined Networking has emerged as a transformative technology in the networking industry, gaining significant traction in recent years. The global SDN market has experienced substantial growth, and it is expected to continue upward in the following years.

Talk about the data center market, the number of SDN switches being sold every year, amount of data sent per second/hour/day in a datacenter, importance of datacenters. [46] at USD 28.2 billion in 2023 and projected to reach USD 120.5 Billion by 2032, with a projected Compound Annual Growth Rate (CAGR) of 17% during the specified period. The increasing demand for automation, scalability and simplified network management is driving the adoption of SDNs across various

industries.

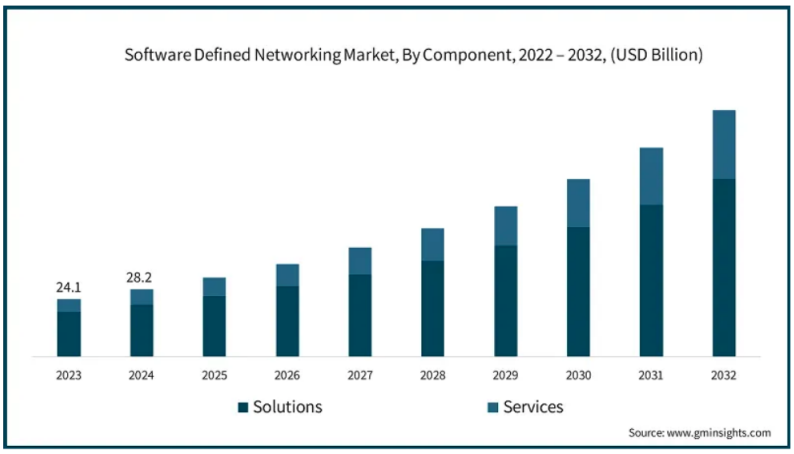

Figure 3.6: Market Projection

The Software Defined Networking market can be segmented on the components, controllers, switches and services. Across this the controllers hold the largest market share due to their crucial role in orchestrating the network resources.

In terms of end-users, the SDN market serves a wide range of industries, including telecommunications, cloud service providers and enterprises, being the enterprises' industry the largest adopter of SDN technology.

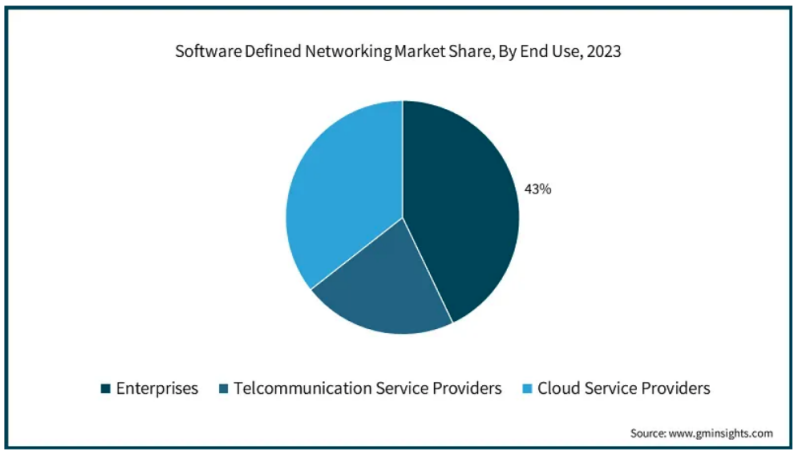

Figure 3.7: Market Share by end use

Geographically North America hold the largest share of the SDN market, followed by Europe and the Asia-Pacific region. This last region is expected to witness the highest growth during the following years, driven by the increasing adoption of cloud computing and datacenter consolidation in countries like China, India or Japan.

The major players in the SDN market include Arista Networks, Cisco Systems, Dell Technologies, Huawei Technologies, IBM, Intel Corporation and Oracle Corporation among others.

Currently, the most popular solution comes by the hand of Cisco-Meraki in Software Defined-Wide Area Networks (SD-WAN) and non-enterprise products are lacking.

Chapter 4

Results

In this section, we present the results of our study on Generalizing Critical Flow Rerouting-Reinforcement Learning. We conduct a series of experiments to evaluate the performance of our proposed approach under various exploration strategies, including Autonomous Exploration, Heuristic-Assisted Exploration and Heuristic-Guided Initial Exploration. With these strategies we aim to research different ways of adding domain knowledge in the exploration of our agent through defined heuristic algorithms and how this domain knowledge affects the convergence speed and generalization capabilities of the agent.

Another aspects of generalizing are also explored, some of these aspects are the effects of training on synthetically generated traffic matrices, training on different alterations of the same topology, training on completely different topologies in all cases the evaluation and generalization results will be on unseen traffic data, and the cases where the testing topology was present on the training data will be clearly indicated. This is being tested as in cases where we want to train for link failure inside a network it makes sense to train on the real topology as well as the topology with some failures.

For clarity, we will divide the results into three different sections, the archi-

tecture results, which will contain information about the final neural network and relevant data about it, the training results where we will share the learning curves of different approaches and the testing results where we will compare the results of our model against an optimal model.

Finally, we will continue with an in-depth analysis of the obtained results and a discussion on the current capabilities of the model, and explore the viability of the proposed model.

The training and evaluation results sections will be divided into the different approaches for exploration.

In detail results will be provided in the appendix B.

4.1 Architecture Results

This section contains the results of the explored architecture. In order to choose between the different architectures we trained each of the different architectures in different settings. The ConvNet architecture brought the best results, the results of the other architectures will be displayed in the appendix B. The specifications of the different architectures can be seen in table 4.1.

Model	Parameters	Size (32-bit)	Size (8-bit)	Average Time (ms)
ConvNet	19,074,892	76.3 MB	19 MB	1.27
SimpleMLP	1,374,620	5.5 MB	1.3 MB	0.08
Attention	5,361,540	21.2 MB	5.3 MB	0.52

Table 4.1: Selected Architecture

The average time shown in the table are prior quantization and on a laptop CPU, if the model was quantized and compiled for the final architecture we expect to see better results.

It must be noted that for the Attention model we attempted more deep architectures with a bigger number of parameters, and we were not able to match the results of the Convolutional Neural Network approach, even with these results we think that there is some attention based architecture that has not been found capable of matching the results of the CNN.

4.2 Training Results

This section will contain all the training results. We explored different approaches for training to see capabilities for adaptability to link failure, capability of training in multiple topologies and also different Reinforcement Learning approaches such as different approaches (baseline vs no baseline) or different neural network architectures. For each approach we will provide with two different graphs, the first one will be the average reward against the update steps, each of the update steps equates to 10 training steps, as the agents take 10 steps before sharing their experience with the central agent. The second figure will include three different variables in the y-axis, the part of actions that were better than the TOP-K Approach, the part of actions that was worse, and the amount of actions that was equal (in MLU) to the TOP-K Critical approach, two different actions may still provide the same MLU. As these quantities are normalized they will add up to 1. Additionally, all the graphs have been smoothed for clarity reasons. As a reminder to the reader, the reward function 3.1 will be equal to 1 when the resulting MLU is equal to the TOP-K Critical MLU, it will be smaller than 1 when it is worse, and greater than 1 when it is better.

A factor to keep in mind is that the designed reward is not centered in 0, as the maximum link utilization can be infinitely worse but only 100% smaller, with this last case not being realistic. For this reason, we may have a better performing

model that has an average reward close but smaller than 1. For this reason we have added the second graph, to be able to really assess the performance during training.

4.2.1 Autonomous Exploration

Autonomous exploration is the most intuitive approach to the problem, not giving any information to the agent on what policies it should learn, by using this approach we expect the learning process to be slower than the rest of methods, but we expect our agent to be able to find a more diverse policy. This non-guided exploration can be clearly seen on AlphaGo Zero [47], that was only trained in self-play and used no human data, in contrast to AlphaGo that was trained on a dataset of games by expert. The model that was trained only on self-play (AlphaGo Zero) was able to beat the AlphaGo model 100-0 in 100 games.

To speed up the process we will divide the training into two parts, the first part takes approximately 25,000 steps where the agent will train normally updating the baseline. For the second part of the training we will generate new traffic matrices (same topologies), which also equals to deleting the baseline. We will force the agent to adapt to those newly generated traffic matrices. For clarity reasons we will only include the second part of the training in the graph, the first part will be included in appendix B.

The results of this approach can be seen in figures 4.1 and 4.2, we consider the obtained results positive, as the obtained reward is higher than one and the percentage of actions that are better than the heuristic method is almost approaching half of the actions.

In the generalization subsection we will evaluate the true performance of this approach and how it really compares to an assisted exploration.

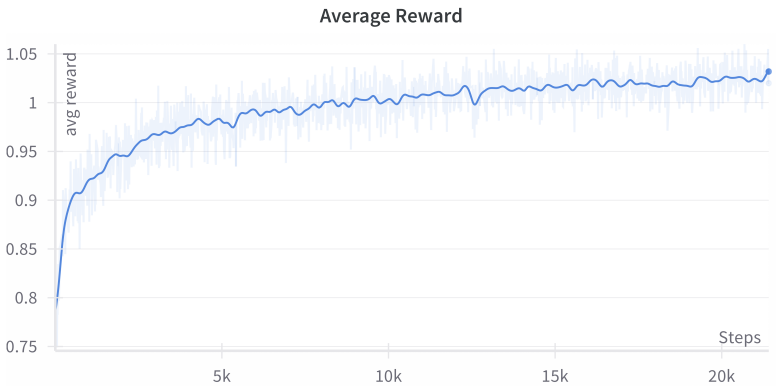

Figure 4.1: Autonomous Exploration Reward

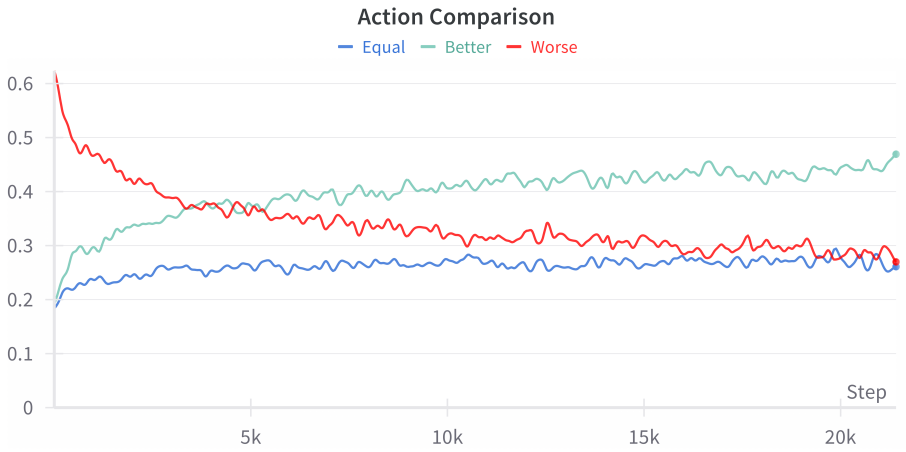

Figure 4.2: Autonomous Exploration Action Comparison

4.2.2 Heuristic-Assisted Exploration

To address exploration vs exploitation problem one of the selected approaches was that during all the training, the agent would have a chance of being updated by the TOP-K Critical result and a chance of being able to select its own action. The training results of this approach can be seen on figures 4.4 and 4.3.

These results were obtained training on different versions of the Abilene traffic matrix on synthetic data generated via the gravity model, five hundred traffic

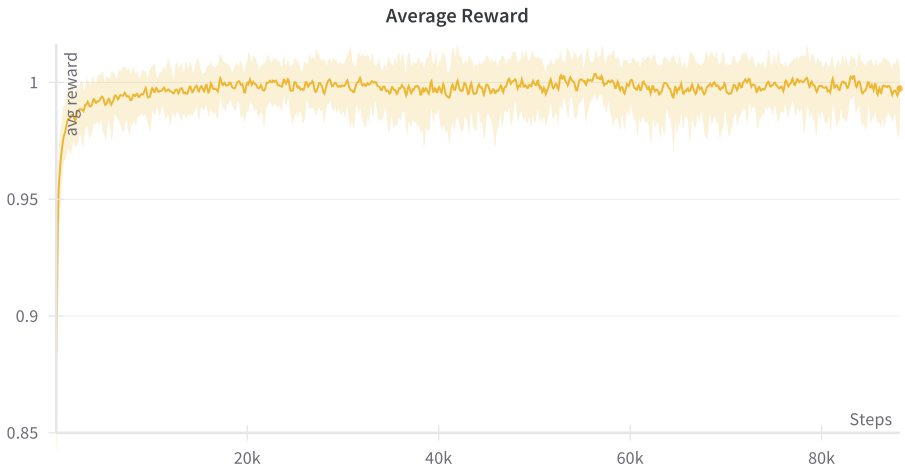

Figure 4.3: Heuristic-Assisted Exploration Reward

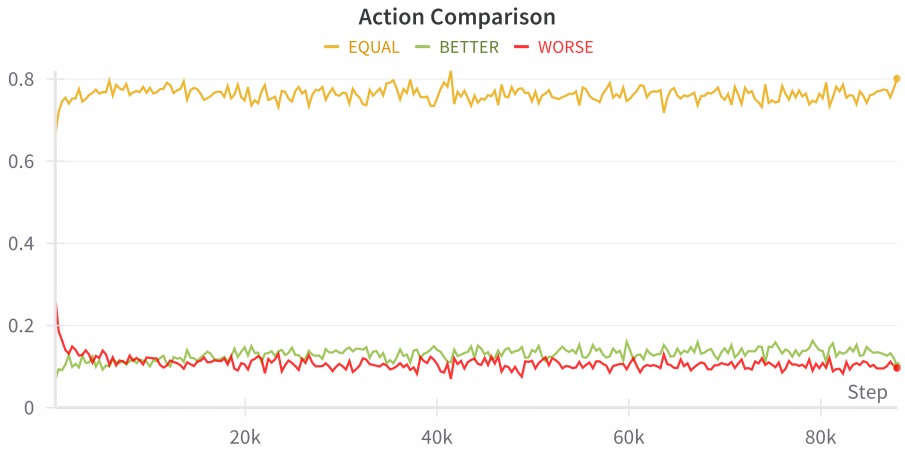

Figure 4.4: Heuristic-Assisted Exploration Action Comparison

matrices were generated for each of the different 15 agents that were running.

We can observe that the model is able to in a good part imitate the actions of the TOP-K Critical algorithm, and in very few occasions obtain better or worse results. The obtained results are not negative per se, and the generalization results will be needed to decide on the capabilities of this model.

4.2.3 Heuristic-Guided Initial Exploration

To address the issues found in the Heuristic-Assisted Exploration method, we adopted this approach, in which the agent will have a chance of taking the same action as the Top-K Critical Method, this chance however will decrease or disappear after a given number of steps. The specified number of steps has been decided by trial and error, and we found it to be extremely complex to estimate due to the stochasticity of gradient based methods in reinforcement learning, as it depended on the run.

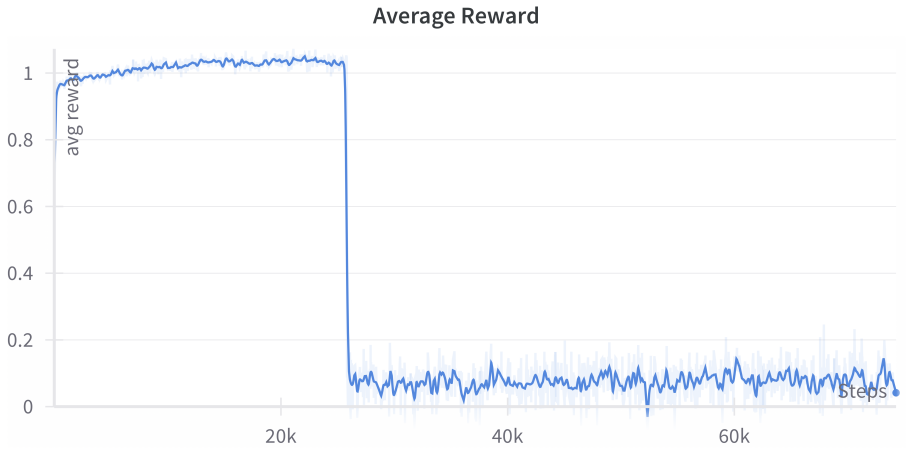

Figure 4.5: Heuristic-Guided Initial Exploration Reward

This method was guided for the first 5,000 updating steps (50,000 training steps), in which it had a chance of updating with the heuristic action. It can be observed in figures 4.5 and 4.6, that this seemed successful as for approximately the first 25,000 updating steps the model kept updating and improving, however after that it suffered from some sever catastrophic forgetting and was not able to improve any further. This catastrophic forgetting has been present in all of our attempts of HGIE, even when the chance of taking the action is slowly decreased. We believe that with further exploration on ways to implement this method the

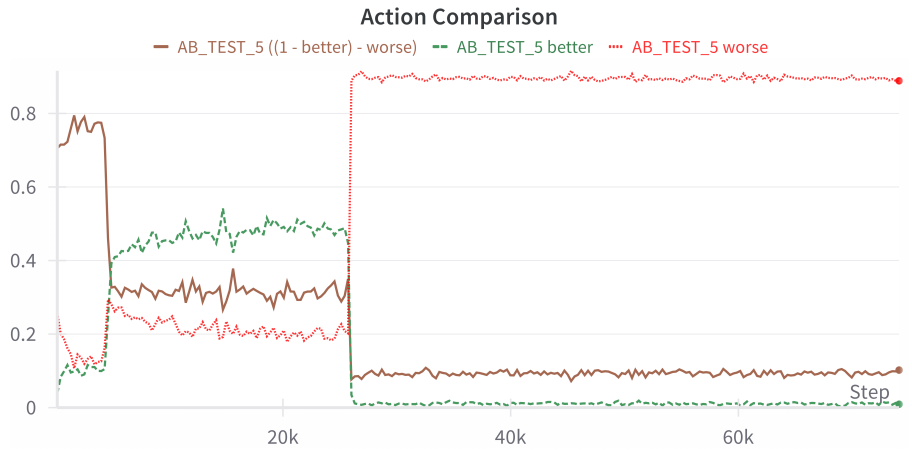

Figure 4.6: Heuristic-Guided Initial Exploration Action Comparison

issue could be solved, but we have not been able to do so.

4.3 Generalization Results

In this section the results of the previous exploration methods will be tested for their different generalization capabilities defined previously in this thesis; their capability to work on not seen before data, whether this is traffic data or topology data will be signaled in the traffic. For either of those, it is important to note that, previous methods such as CFR-RL struggled when trained on synthetic traffic data and tested on real traffic data, let alone a different topology. We will divide our results in the same sections as the previous section, but on top of that we will divide each of those parts into two different parts, each addressing a different type of generalization.

For each evaluation we will have two different graphs, the first graph will show the normalized maximum link utilization of the approach, and the maximum link utilization of TOP-K Critical, the scale of this graph is different from the one shown in training, for this graph the value of 1 will equate to the optimal solution

in terms of MLU, the second graph will show the normalized delay, where 1 is the value obtained via a solution focusing on optimal delay, the higher the value is the better.

For topology generalization we will test our results in a graph generated by the Newman-Watts-Strogatz approach [48]. We generated a graph connected to its two closest neighbors and a 0.3 chance of connection.

4.3.1 Autonomous Exploration

We expect this approach to present the best results, as we have been the model has been able to produce a non-guided policy without any influence.

Traffic Generalization

To test traffic generalization we are going to test the performance of the model trained without any guidance on data from the Abilene topology.

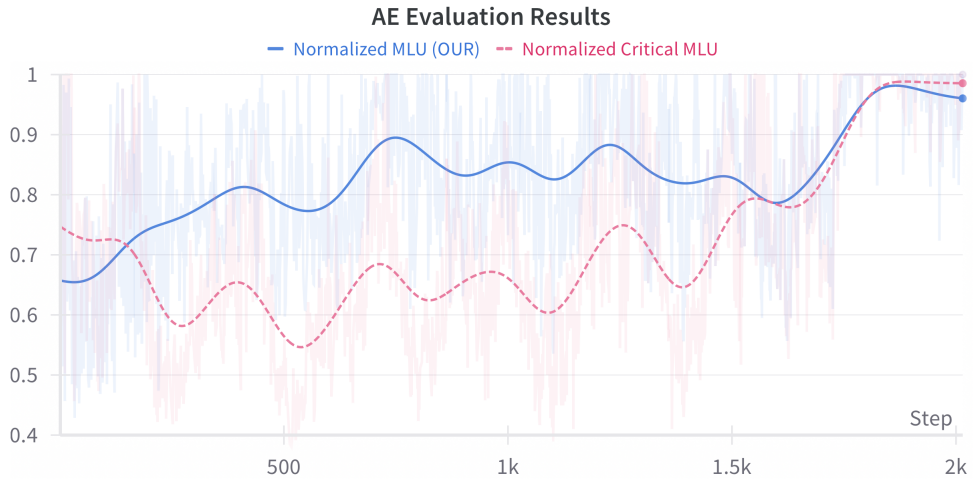

Figure 4.7: AE MLU Evaluation

As it can be observed in figures 4.7 and 4.8, this model is able to surpass the

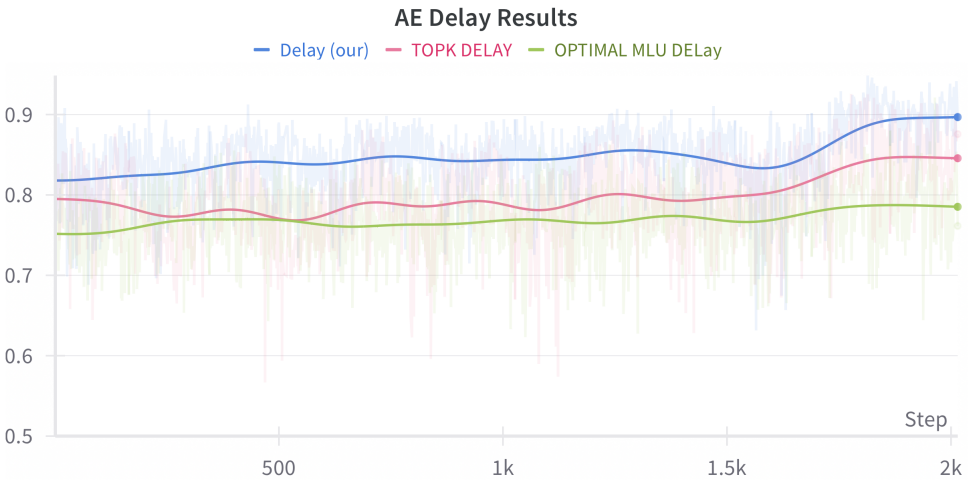

Figure 4.8: AE Delay Evaluation

heuristic method. We consider the results obtained in this section a success as we have been able to outperform the heuristic based method in both delay and maximum link utilization.

Topology Generalization

To test the topology generalization capabilities of this model, we will modify the Abilene topology by removing a link, this modification was not present in the training dataset, by removing this link we can assess the performance of our model in cases of link failure.

The results of this approach can be observed in figures 4.10 and 4.9, with these results we confirm that our agent has been able to adapt to topologies it has not been trained on, in comparison with the results of the previously discusses models [4] and [5], we have created an agent that is not only able to adapt to a different distribution of traffic matrices in a network, but also adapt to different topologies.

However, this agent is still not able to adapt to completely different topologies.

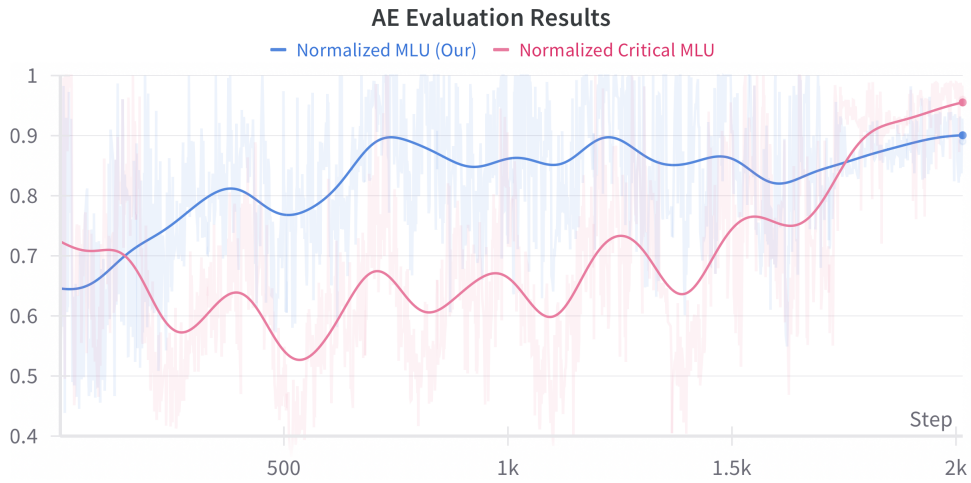

Figure 4.9: AE MLU Evaluation in different topology

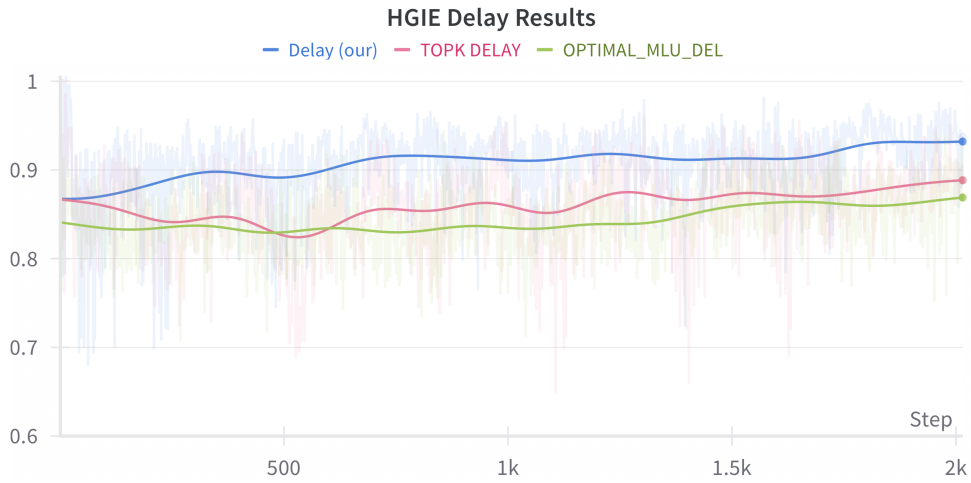

Figure 4.10: AE Delay Evaluation in different topology

4.3.2 Heuristic-Assisted Exploration

This method consisted on having a chance of selecting a TOP-K option during the complete training process. Unlike with other methods, by being able to choose a TOP-K Action during all the training process it made assessing the true per-

formance of this method difficult, as we were not aware of how the model would perform when it was not guided.

Traffic Generalization

In this example we tested this approach on the Abilene topology (it was on the training data) on real traffic (it was only trained on synthetic data). The results of ECMP were included for comparative reasons.

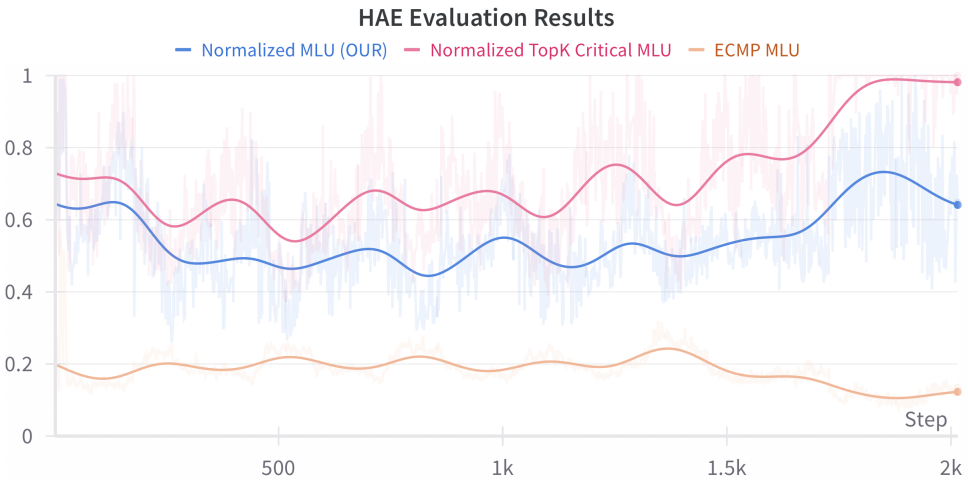

Figure 4.11: Heuristic-Assisted Exploration MLU Evaluation

As it can be seen in the results, this approach did not result in positive results, we believe that the policy was highly unstable, and if a part of autonomous exploration was added these issues may be solved. As the results obtained in this section are already negative, the topology generalization section will be skipped, as we believe it would be a waste of compute resources, as the task of generalizing to different topologies in significantly more complex than the task of adapting to different traffic patterns.

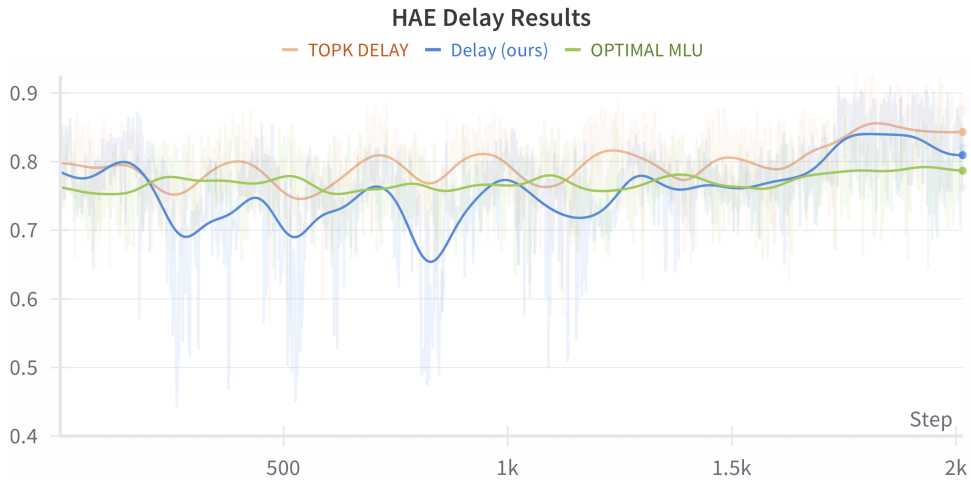

Figure 4.12: Heuristic-Assisted Exploration Delay Evaluation

4.3.3 Heuristic-Guided Initial Exploration

For this section we will include the results of the better checkpoint we have had access to, later checkpoints suffered from the catastrophic forgetting providing significantly worse performance in terms of MLU minimization.

Traffic Generalization

For this traffic generalization we will test the performance of a model trained on synthetic traffic on the Abilene topology and modifications of this topology in the Abilene topology with real traffic, a positive performance in this part would signify that we have developed a model capable of adapting to new traffic patterns.

As it can be observed in figures 4.14 and 4.13, the obtained results are close to the ones obtained by the heuristic method. It can be observed that at the last 500 evaluation steps there is an apparent difference, however in this evaluation scenarios the non-Normalized values are significantly smaller than in the rest of the graph, so the difference may not be as apparent when deployed. Regarding

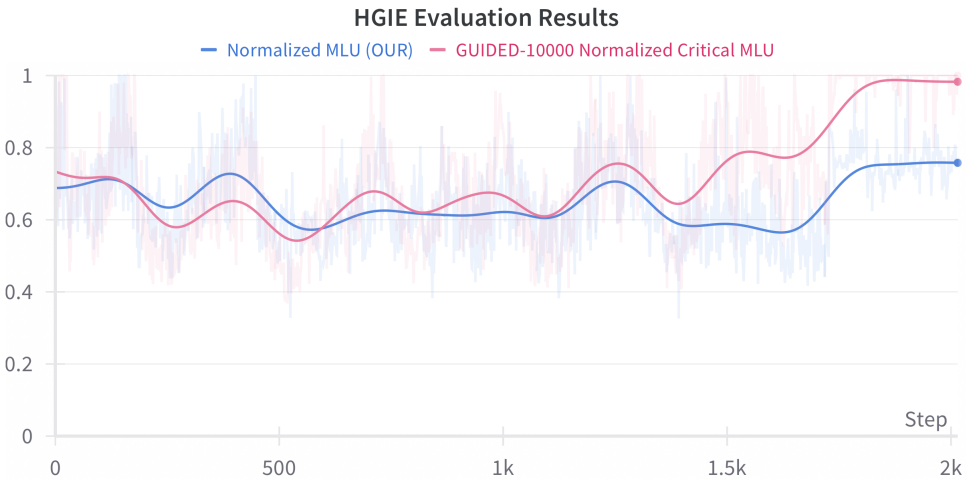

Figure 4.13: Heuristic-Guided Initial Exploration MLU Evaluation

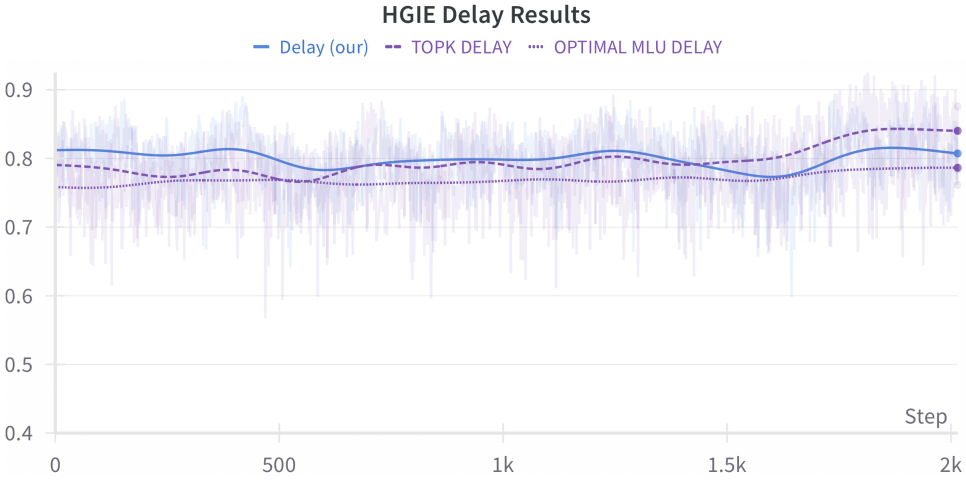

Figure 4.14: Heuristic-Guided Initial Exploration Delay Evaluation

delay, we also find really similar performance. Via this experiment we have proven that training on synthetically generating data can be useful, at least for the Abilene Network, to make a broader statement more networks would need to be explored and compared.

However, we believe there is still room for improvement as we are really similar

to the heuristic method, and still relatively far from optimal performance. We believe that for this slight difference, training a reinforcement learning model would not be practical, and we would still prefer using the heuristic method.

Topology Generalization

Given the slightly positive results obtained in the previous section, we will proceed to evaluate the performance of the model in topology generalization, a priori, we would not expect this to be better than heuristic method, as it has only been trained on topologies similar to the Abilene traffic matrix, and we will now proceed with a randomly generated topology.

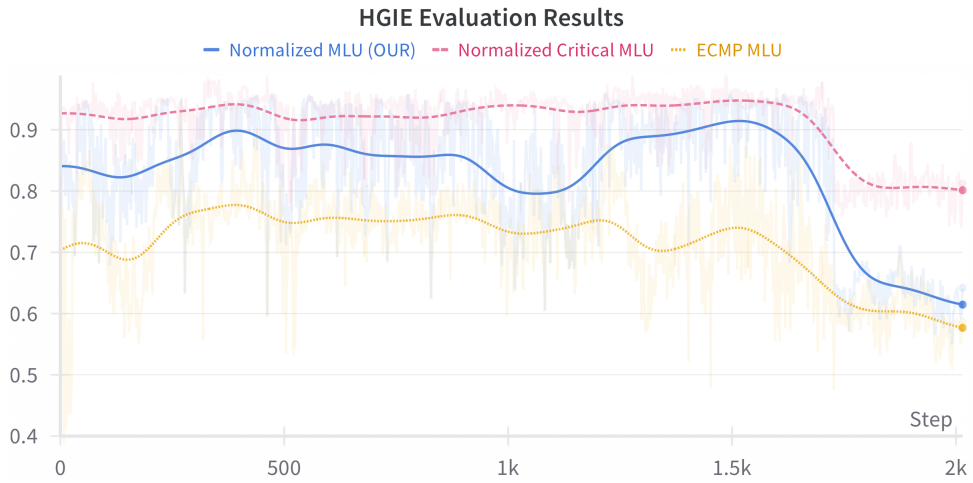

Figure 4.15: HGIE MLU Evaluation in different topology

As it can be observed in figures 4.15 and 4.16, the obtained results are worse than the ones obtained by TOP-K Critical, we believe these results were caused due to only training in topologies that were modifications of the Abilene topology. However, the results obtained regarding MLU are better than expected, as for the first part the results are really similar and are closer to TOP-K Critical than they

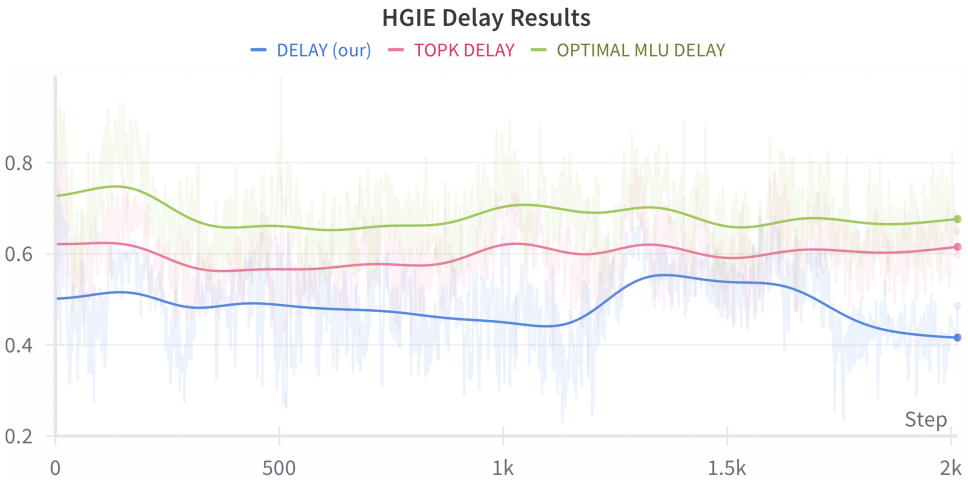

Figure 4.16: HGIE Delay Evaluation in different topology

are to the base method ECMP. We believe that can signify that with a better training dataset the agent may be able to perform in different topologies.

4.4 Result Analysis

This section will contain the discussion on the results obtained in the previous section, we will divide this analysis into three different subsections, the discussion results obtained on the training data, and how this results compare to other approaches, we will then discuss the results obtained on generalization, and finally we will discuss the results relevant to deployment such as end-to-end delay and the latency of our model. On this first section we will also provide details about the training process.

4.4.1 Training

For training, we are measuring how well our approach is adapting to the training data, and how quickly it learns a policy. When comparing our results to the CFR-

RL paper [4], we can see that our approach takes more time, approximately 4 times more steps, to learn a policy that is able to outperform heuristic methods. This is expected as the previous model was only learning information about one topology and one traffic distribution, whereas we are training a model to be able to be usable in different topologies and distributions.

In terms of the quality of the trained policy, via this approach or model is slightly worse than CFR-RL on training data, we do not think these results are important as it only happens on the training distribution, where CFR-RL obtains near-optimal results for every traffic matrix. In any case this shows that there is still room for improvement for our model and the obtained results can be improved.

To obtain the final model, we have trained different models for approximately 1000 hours, 500 hours in the Frontera supercomputer and 500 hours in personal computers, most of this exploration has been focused on testing different architectures, testing the replicability of the results and different approaches. The final model was trained on a personal computer (M3 Pro MacBook Pro), for a total of twenty hours, making this experiment replicable as long as one has a minimum amount of compute.

Due to the small batch size and dimensionality of the input, all training has been performed on the CPU, as the overhead of moving the data to the GPU was greater than the benefits, so in systems where there is a separate VRAM and RAM using the CPU for training would still be recommended.

During our attempts at autonomous exploration, we found that our model started to plateau somewhere between 0.85 and 0.9 average reward and accidentally found that stopping the training when all the advantages are zero, (the model is no longer updating), and restarting the training with new traffic matrices, did result in achieving better models. We believe this problem could have been addressed with another approaches such as a different learning rate schedule. However, we

believe that a better model could be achieved by designing a better baseline. There were attempts on training an actor critic model [7], but the attempts were not successful.

4.4.2 Evaluation

In this section we will discuss the performance of the model on tasks that it had not been trained on, we will mainly focus on the performance of the Autonomous Exploration approach as it was the approach that obtained the better results. The results obtained in traffic generalization and topology generalization will be discussed in different section, a comparison to the capabilities of other approaches will also be provided.

Traffic Generalization

To test the generalization capabilities of the model, we run will run inference on a different distribution of data that it had been trained on. To train the model, we generated synthetic data based on the gravity model, shown on equations 3.2 3.3. With this approach we can also assess the usefulness of using synthetic data to train Reinforcement Learning models in traffic engineering.

The traffic generalization results can be observed in figure 4.7. These results prove that the developed model is able to generalize to different traffic patterns, and it is not memorizing current patterns. One of the main problems of CFR-RL [4] was its inability to perform when the slightest change to the traffic distribution was performed. It is important to note that when trained on real traffic data from the Abilene network, and assessed on other types of real traffic data the model showed some deterioration regarding results but still was able to outperform the TOP-K Critical approach in most approaches. We believe that these results are caused by the cyclical nature of real network data, same reason as approaches

such as RL Routing [5] work at all, the traffic data tends to be really similar when comparing different days or different weeks. This memorization of patterns but said methods however are also the reason for which they are unable to adapt to other types of traffic, even when the gravity model is the model that resembles Abilene the most, said methods perform better than heuristic methods when they are trained on this distribution and tested on the real distribution or vice versa.

Topology Generalization

We will proceed to discuss the results on topology generalization. This task is more complex than traffic generalization, as to be able to adapt to different topologies the designed neural network must be able to have some understanding of the underlying topology, and how each of the flows of the traffic matrix correlates to the links of the topologies, and not only that it must also be able to differentiate the different links of the network depending on their capacity.

To test the capabilities we ran inference of the model on different topologies, one of them is an adapted version of the Abilene topology, present in the network, the other one is a randomly generated topology following the approach presented in [48], in which we defined that every node must have at least two neighbors and a 30% probability of having more links. This topology is similar to small scale networks.

Our solution was able to adapt to the modified topology and obtain similar results as it had obtained on its training data, and was capable of outperforming heuristic based methods. This can be seen in figure 4.9, where our approach performs better than the heuristic method in most of the evaluation.

When we tested our solution in a completely different topology, it was not able to perform as well as TOP-K Critical, we believe that a better model could be achieved if a more diverse set of topologies was added to the training data. For this

to happen a different approach should be considered for the baseline, as currently the number of topologies is equal to the number of threads available in order to keep track of the baseline.

In conclusion, we were able to develop a with some topology generalization capabilities, as it is currently able to adapt to link failure or link addition events. This model can still be useful for single networks in datacenters due to its fault tolerance.

4.4.3 Deployment

We will proceed to discuss the results obtained in our approach regarding deployment, this results include model latency, introduced end-to-end delay and how it compares to other methods, model dimensions and the floating point operations required. We will focus on the results obtained by the Convolutional Neural Network, which has been the architecture that achieved the reported results.

As it can be seen in table 4.1, the CNN model is capable of running inference in approximately 1 ms, as it was stated in the objectives, this time is negligible when compared to the time it takes to reroute the flows, for this reason we can consider that we have met the objective. The results have been obtained on a M3-Pro MacBook Pro, running on CPU without any optimization, quantization or compilation. For this reason we believe that if this model were to be compiled for the controllers' architecture we could achieve better or similar performance than the results obtained here.

Another important aspect is the memory, currently the quantized model is expected to fit in less than 20 MB, this would make it easily deployable in most modern systems and in the strange case in which the model could not be deployed in the controller it could be deployed in some small capacity microcontroller at the cost of adding some latency to our model.

4.5 Limitations

While our proposed model has demonstrated promising results in optimizing routing in SDNs, it is important to acknowledge and discuss its limitations. This section will highlight the scenarios where our model may not provide improvements over existing heuristic methods.

4.5.1 Performance in Homogeneous Networks

One notable limitation of our model is its performance in homogeneous networks, where all the links have the same capacity. In such cases, the TOP-K Critical approach, has been observed to achieve near-optimal results. Consequently, independently of our model's performance, there is little reason to use another method.

The performance of our model becomes more evident in heterogeneous networks, even if a single link presents a different capacity than the rest our model should be capable of outperforming the TOP-K Critical approach. Heterogeneous networks are not uncommon, an example of these types of networks is the Abilene network, exhibiting different capacities among links. For this reason, we believe that the developed model is still useful, as heterogeneous networks are real and common.

There may be the case where our model is capable of significantly outperform heuristic models in a homogeneous networks, however we believe these cases are few and uncommon, normally caused by obscure topologies which would not reflect real world networks.

4.5.2 Adaptability to significant topology change

Another limitation of our model lies in its ability to adapt to significant changes in the network topology. Our model has been primarily trained and evaluated on networks that are similar to the ones found in the training phase. While it has shown improved performance in these networks, its adaptability to completely different networks remains a challenge.

If a network undergoes substantial modifications, such as the addition or removal of numerous links, our model may struggle to maintain its performance advantages.

In conclusion, the network is able to generalize to similar topologies, the model will be able to continue performing as expected in case of link failure or updates to the network or simply traffic distribution changes, however if there is a major network disturbance that modifies the topology of the network significantly, the model will not be able to respond as expected. For cases like this, having a system that detects network outages may be encouraged, with the presence of a system like this, the controller could change from our model to a heuristic approach in extreme cases.

4.5.3 Deployment considerations

It is important to consider the practical aspects of deploying a model in real-world SDN environments. While our model has shown promising results in simulations and experiments, the performance of this model may vary in real scenarios when there are others aspects to keep in mind, for example the sample rate, how often will we choose critical flows?

Other aspects such as real world latency need to be studied, how fast will the controller be able to install this forwarding rules in the different switches.

For reasons like this, the deployment of theoretical models such as this one need to be carefully evaluated in real world scenarios before deploying it in production environments. Moreover, the integration of our model with existing SDN controllers and networks systems may require additional development efforts and compatibility considerations. Even if this model provided optimal results in simulations, it would be useless if it was not possible for it to be seamlessly deployed in current network solutions.

Chapter 5

Conclusions

In this chapter, we present the conclusions drawn from our research on optimizing routing inn Software Defined Networks (SDNs) using a Deep Reinforcement Learning approach. We will reflect on the methodology employed, the key findings and results obtained and potential implications of our work. Furthermore, we provide recommendations for future studies highlighting areas that warrant further investigation and exploration.

Through this thesis, we have explored the challenges associated with routing optimization in SDNs and proposed a novel DRL-based solution to address the challenges. By using DRL, our model has been able to make intelligent decisions that minimize network congestion and improve overall network performance.

We will first begin by going over the methodology, continue with the results and finalize with future study recommendations.

5.1 Methodology Conclusions

In this section we will summarize the conclusions drawn from our methodology and approach to solving the routing optimization problem in SDNs. We will divide this

section onto the sections of our methodology that we believe are worth mentioning and carefully explain our thoughts on the approach and things that could have been done better.

5.1.1 Data

Data started as one of the major problems of this project, multiple approaches at data generation where attempted with the gravity model being the most successful one. With this approach we have been able to obtain positive results, and we have not seen a case where we have gotten negative results (on traffic generalization), however in order to be sure, approaches where there are multiple ways of generating data could be explored.

We expected to need a higher complexity model in order to be able to generalize to multiple topologies, however we did not expect how different topologies can bring really different results, and in some cases just changing the capacity of a single link of a network can cause completely different results, for better or for worse. Regarding topology, it would have been better to be able to train on a large quantity of topologies at the same time, also generated by different approaches.

5.1.2 Algorithm

The REINFORCE algorithm provided surprisingly positive results, this algorithm was used for the results it provided in CFR-RL. Some attempts were made at training an actor-critic method to get rid of the baseline, this approaches however did not bring positive results. The actor-critic method consists on having a separate neural network that gives a score to every state and could be used as a replacement for the baseline.

Most of the problems with our approach originate from this baseline, this baseline currently limits the number of topologies we can train at the same time with

the CPU threads, as there is a need to keep track of the experience.

5.1.3 Artificial Neural Network

Due to the complexity and characteristics of the problem, the use of a neural network was not really a choice. The final architecture was based on a combination of convolutional layers and fully connected layers. We believe that there must be a simpler architecture that is capable of obtaining better results. Aside from the study of different architectures, there was no further exploration on different architectures, the use of different activation functions, different optimizers or the use of different learning rate schedulers remain unexplored. Further exploration may benefit from the use of Neural Architecture Search (NAS) [49], to evaluate and compare different architecture approaches.

The current architecture was initialized as a modified version of the architecture presented in [4] and was increased in complexity as it was required. We believe that this model could benefit from graph neural network models to obtain a better representation of the underlying network topology and its characteristics.

5.2 Results Conclusions

This section will contain insights on the obtained results, what were the obtained results and the completion status of the established objectives.

Our initial objective was the development of a model capable of generalizing to different network topologies as we deemed this research could bring potential benefits to the field of SDN routing, up to the point, most of the current research on the uses of RL had focused on the development of a model capable of outperforming heuristic based methods in single network topologies. However, there was little to no evidence of how a model that was trained on synthetic data would perform on

real scenarios, the few topologies that had available data, used part of that data from training as it resulted on better results on the testing phases, and in the cases where there was no real data, most research just trained and tested on the same distribution (not the exact same data, but similar) of data.

We can identify two main contributions of this project. The first, we have proved the usefulness of synthetic data, and how, at least on our case, this data can be used to train Deep Reinforcement Learning models, and secondly the development of a DRL model capable of outperforming heuristic based methods on a set of similar topologies and capable of adapting to network events such as link failure or adding new links. This model may not be as useful as a model that is capable of generalizing to any topology.

While it is true that our model, is able to generalize to small subset of topologies, as they must have the same number of nodes and be similar, we believe it is a step in the right direction.

5.3 Future study recommendations

This section will contain suggestions for future studies based on the experience of this project, such as training suggestions, approach suggestions or general ideas.

Firstly, the model is currently being trained completely on the CPU of the device, we found some issues while attempting to move part of the training to the GPU, no importance was given to these issues as there was not a significant speedup, so decided to train on CPU. However, if a larger model with more agents and bigger agents were to be deployed, the use of GPU would be recommended, some approach such as A3C [50], would be recommended, where the agents are collecting experience on CPU and the central agent is simply updating the models on GPU.

Another topic that may be explored is a variable value of K, currently the selected K corresponds to around a 10% of the network flows, however in some occasions this may be too much or too little, it may be a good idea to train a model capable of selecting different values of K for different situations, for this a penalty penalizing a high number of K should be included to avoid just selecting every single flow.

Another area that may provide some positive results, is the use of a baseline different from the average of previous experiences, we had no positive result incorporating a different baseline on multiple topologies, but in CFR-RL, the AC method was able to produce positive results, slightly worse than REINFORCE. We think that using a neural network to predict the value of the state would be the best approach, as this would allow using as many topologies as desired, as there would be no need to keep track of all previous experiences and could just generate different topologies dynamically. We think that this approach would be required to train a model capable of generalizing to any desired topology.

Finally, a part that has been only mentioned slightly is the topology size, we would recommend the use of some different neural network such as a Variational AutoEncoder (VAE) [51] formed by only convolutional layers, with this approach, we would be able to train on a much larger amount of network topologies, we have no idea if this approach would result successful, but it is one of the few approaches that would allow to create a model capable of generalizing in topologies of different sizes.

Bibliography

[1] Nick McKeown et al. "OpenFlow: Enabling Innovation in Campus Networks". In: *SIGCOMM Comput. Commun. Rev.* 38.2 (Mar. 2008), pp. 69–74. ISSN: 0146-4833. DOI: 10.1145/1355734.1355746. URL: https://doi.org/10.1145/1355734.1355746.

[2] Sakir Sezer et al. "Are we ready for SDN? Implementation challenges for software-defined networks". In: *IEEE Communications Magazine* 51.7 (2013), pp. 36–43. DOI: 10.1109/MCOM.2013.6553676.

[3] Yufei Wang and Zheng Wang. "Explicit routing algorithms for Internet traffic engineering". In: *Proceedings Eight International Conference on Computer Communications and Networks (Cat. No.99EX370)*. 1999, pp. 582–588. DOI: 10.1109/ICCCN.1999.805577.

[4] Junjie Zhang et al. "CFR-RL: Traffic Engineering With Reinforcement Learning in SDN". In: *IEEE Journal on Selected Areas in Communications* 38.10 (2020), pp. 2249–2259. DOI: 10.1109/JSAC.2020.3000371.

[5] Yi-Ren Chen et al. "RL-Routing: An SDN Routing Algorithm Based on Deep Reinforcement Learning". In: *IEEE Transactions on Network Science and Engineering* 7.4 (2020), pp. 3185–3199. DOI: 10.1109/TNSE.2020.3017751.

[6] Hongzi Mao et al. "Resource Management with Deep Reinforcement Learning". In: *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*. HotNets '16. Atlanta, GA, USA: Association for Computing Machinery, 2016, pp. 50–56. ISBN: 9781450346610. DOI: 10.1145/3005745.3005750. URL: https://doi.org/10.1145/3005745.3005750.

[7] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Second. The MIT Press, 2018. URL: http://incompleteideas.net/book/the-book-2nd.html.

[8] Yann Lecun and Yoshua Bengio. "Convolutional Networks for Images, Speech and Time Series". In: *The Handbook of Brain Theory and Neural Networks*. Ed. by Michael A. Arbib. The MIT Press, 1995, pp. 255–258.

[9] Matthew Roughan. "Simplifying the Synthesis of Internet Traffic Matrices". In: *SIGCOMM Comput. Commun. Rev.* 35.5 (Oct. 2005), pp. 93–96. ISSN: 0146-4833. DOI: 10.1145/1096536.1096551. URL: https://doi.org/10.1145/1096536.1096551.

[10] Ying Zhang. *Abilene TM*. Online: cs.utexas.edu/ yzhang/research/AbileneTM/.

[11] Gunjan P Tank, Anmol Dixit, and Alekhya Vellanki. "Software Defined Networks: The New Norm for Networks". In: 2017. URL: https://api.semanticscholar.org/CorpusID:53052362.

[12] Pankaj Berde et al. "ONOS: towards an open, distributed SDN OS". In: *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*. HotSDN '14. Chicago, Illinois, USA: Association for Computing Machinery, 2014, pp. 1–6. ISBN: 9781450329897. DOI: 10.1145/2620728.2620744. URL: https://doi.org/10.1145/2620728.2620744.

[13] Pat Bosshart et al. "P4: programming protocol-independent packet processors". In: *SIGCOMM Comput. Commun. Rev.* 44.3 (July 2014), pp. 87–95. ISSN: 0146-4833. DOI: 10.1145/2656877.2656890. URL: https://doi.org/10.1145/2656877.2656890.

[14] Soheil Hassas Yeganeh, Amin Tootoonchian, and Yashar Ganjali. "On scalability of software-defined networking". In: *IEEE Communications Magazine* 51.2 (2013), pp. 136–141. DOI: 10.1109/MCOM.2013.6461198.

[15] Justin Boyan and Michael Littman. "Packet Routing in Dynamically Changing Networks: A Reinforcement Learning Approach". In: *Advances in Neural Information Processing Systems*. Ed. by J. Cowan, G. Tesauro, and J. Alspector. Vol. 6. Morgan-Kaufmann, 1993. URL: https://proceedings.neurips.cc/paper_files/paper/1993/file/4ea06fbc83cdd0a06020c35d50e1e89a-Paper.pdf.

[16] Shailesh Kumar and Risto Miikkulainen. "Dual Reinforcement Q-Routing: An On-Line Adaptive Routing Algorithm". In: 1997. URL: https://api.semanticscholar.org/CorpusID:10985533.

[17] John Moy. *OSPF Version 2*. RFC 2328. Apr. 1998. DOI: 10.17487/RFC2328. URL: https://www.rfc-editor.org/info/rfc2328.

[18] Ling Li and A.K. Somani. "Dynamic wavelength routing using congestion and neighborhood information". In: *IEEE/ACM Transactions on Networking* 7.5 (1999), pp. 779–786. DOI: 10.1109/90.803390.

[19] Amine Tcherak, Samia Loucif, and Mohamed Ould-Khaoua. "On Efficient Routing for SDN-Based Wireless Sensor Networks". In: *2023 24th International Arab Conference on Information Technology (ACIT)* (2023), pp. 1–6. URL: https://api.semanticscholar.org/CorpusID:268542830.

[20] Luca Barletta et al. "QoT estimation for unestablished lighpaths using machine learning". In: *2017 Optical Fiber Communications Conference and Exhibition (OFC)*. 2017, pp. 1–3.

[21] Gangxiang Shen et al. "Machine Learning-Assisted Least Loaded Routing to Improve Performance of Circuit-Switched Networks". In: *CoRR* abs/1804.08403 (2018). arXiv: 1804.08403. URL: http://arxiv.org/abs/1804.08403.

[22] Yakov Rekhter, Susan Hares, and Tony Li. *A Border Gateway Protocol 4 (BGP-4)*. RFC 4271. Jan. 2006. DOI: 10.17487/RFC4271. URL: https://www.rfc-editor.org/info/rfc4271.

[23] Christian Hopps. *Analysis of an Equal-Cost Multi-Path Algorithm*. RFC 2992. Nov. 2000. DOI: 10.17487/RFC2992. URL: https://www.rfc-editor.org/info/rfc2992.

[24] Rashid Amin et al. "A Survey on Machine Learning Techniques for Routing Optimization in SDN". In: *IEEE Access* 9 (2021), pp. 104582–104611. DOI: 10.1109/ACCESS.2021.3099092.

[25] Alaitz Mendiola et al. "A Survey on the Contributions of Software-Defined Networking to Traffic Engineering". In: *IEEE Communications Surveys & Tutorials* 19.2 (2017), pp. 918–953. DOI: 10.1109/COMST.2016.2633579.

[26] Zhuge Bin et al. "Resource scheduling algorithm and ecnomic model in ForCES networks". In: *China Communications* 11.3 (2014), pp. 91–103. DOI: 10.1109/CC.2014.6825262.

[27] Sushant Jain et al. "B4: experience with a globally-deployed software defined wan". In: *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*. SIGCOMM '13. Hong Kong, China: Association for Computing Machinery, 2013, pp. 3–14. ISBN: 9781450320566. DOI: 10.1145/2486001.2486019. URL: https://doi.org/10.1145/2486001.2486019.

[28] R. Martínez et al. "Experimental validation of active frontend — Backend stateful PCE operations in flexgrid optical network re-optimization". In: *2014 The European Conference on Optical Communication (ECOC)*. 2014, pp. 1–3. DOI: 10.1109/ECOC.2014.6963969.

[29] Hongrong Cheng, Miao Zhang, and Javen Qinfeng Shi. *A Survey on Deep Neural Network Pruning-Taxonomy, Comparison, Analysis, and Recommendations*. 2023. arXiv: 2308.06767 [cs.LG].

[30] Amir Gholami et al. *A Survey of Quantization Methods for Efficient Neural Network Inference*. 2021. arXiv: 2103.13630 [cs.CV].

[31] Leon Poutievski et al. "Jupiter Evolving: Transforming Google's Datacenter Network via Optical Circuit Switches and Software-Defined Networking". In: *Proceedings of ACM SIGCOMM 2022*. 2022.

[32] Ahmed Elmokashfi, Amund Kvalbein, and Constantine Dovrolis. "On the Scalability of BGP: The Role of Topology Growth". In: *IEEE Journal on Selected Areas in Communications* 28.8 (2010), pp. 1250–1261. DOI: 10.1109/JSAC.2010.101003.

[33] Yufei Wang, Zheng Wang, and Leah Zhang. "Internet traffic engineering without full mesh overlaying". In: *Proceedings IEEE INFOCOM 2001. Conference on Computer Communications. Twentieth Annual Joint Conference of the IEEE Computer and Communications Society (Cat. No.01CH37213)*. Vol. 1. 2001, 565–571 vol.1. DOI: 10.1109/INFCOM.2001.916782.

[34] Gurobi Optimization, LLC. *Gurobi Optimizer Reference Manual*. 2023. URL: https://www.gurobi.com.

[35] Ashish Vaswani et al. "Attention Is All You Need". In: *CoRR* abs/1706.03762 (2017). arXiv: 1706.03762. URL: http://arxiv.org/abs/1706.03762.

[36] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. *Layer Normalization*. 2016. arXiv: 1607.06450 [stat.ML].

[37] Dan Hendrycks and Kevin Gimpel. *Gaussian Error Linear Units (GELUs)*. 2023. arXiv: 1606.08415 [cs.LG].

[38] Alexey Dosovitskiy et al. *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*. 2021. arXiv: 2010.11929 [cs.CV].

[39] Lucas Lehnert et al. *Beyond A*: Better Planning with Transformers via Search Dynamics Bootstrapping*. 2024. arXiv: 2402.14083 [cs.AI].

[40] Peter Henderson et al. *Deep Reinforcement Learning that Matters*. 2019. arXiv: 1709.06560 [cs.LG].

[41] Adam Paszke et al. *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. 2019. arXiv: 1912.01703 [cs.LG].

[42] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[43] Denis Yarats et al. *Improving Sample Efficiency in Model-Free Reinforcement Learning from Images*. 2020. arXiv: 1910.01741 [cs.LG].

[44] Lechao Xiao et al. *Dynamical Isometry and a Mean Field Theory of CNNs: How to Train 10,000-Layer Vanilla Convolutional Neural Networks*. 2018. arXiv: 1806.05393 [stat.ML].

[45] Lukas Biewald. *Experiment Tracking with Weights and Biases*. Software available from wandb.com. 2020. URL: https://www.wandb.com/.

[46] Global Market Insights. "Software defined networking market size & share report". In: (2023). URL: https://www.gminsights.com/industry-analysis/software-defined-networking-sdn-market.

[47] David Silver et al. *Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm*. 2017. arXiv: 1712.01815.

[48] M.E.J. Newman and D.J. Watts. "Renormalization group analysis of the small-world network model". In: *Physics Letters A* 263.4 (1999), pp. 341–346. ISSN: 0375-9601. DOI: https://doi.org/10.1016/S0375-9601(99)00757-4. URL: https://www.sciencedirect.com/science/article/pii/S0375960199007574.

[49] Colin White et al. *Neural Architecture Search: Insights from 1000 Papers*. 2023. arXiv: 2301.08727.

[50] Mohammad Babaeizadeh et al. *Reinforcement Learning through Asynchronous Advantage Actor-Critic on a GPU*. 2017. arXiv: 1611.06256.

[51] Diederik P Kingma and Max Welling. *Auto-Encoding Variational Bayes*. 2022. arXiv: 1312.6114.

Appendix A

Alignment with Sustainable Development Goals

Our research on optimizing routing in SDNs using DRL techniques aligns with several Sustainable Development Goals (SDGs). By developing intelligent and efficient routing strategies, our work contributes to the following SDGs.

A.1 9 - Industry, Innovation and Infrastructure

This SDG focuses on building resilient infrastructure, promoting inclusive and sustainable industrialization and fostering innovation. We believe our solution, is aligned with this objective for the following reasons:

A.1.1 Target 9.1

The target 9.1 of this SDG states *Develop quality, reliable, sustainable and resilient Infrastructure, including regional and transborder infrastructure, to support economic development and human well-being, with a focus on affordable and equitable access for all.*

Our research focuses on optimizing network infrastructure, to ensure reliable and efficient data transmission. By improving the network performance and reducing congestion, our work contributes to the development of sustainable and resilient communication Infrastructure.

A.1.2 Target 9.5

The target 9.5 states *Enhance scientific research, upgrade the technological capabilities of industrial sectors in all countries, in particular developing countries, including, by 2030, encouraging innovation and substantially increasing the number of research and development workers per 1 million people and public and private research and development spending.*

Our approach focuses on improving the technological capabilities of SDNs, which are increasingly being used in different sectors. By improving the capabilities we can encourage growth of industrial sectors.

A.2 12 - Responsible Consumption and Production

The twelfth sustainable development goal, focuses on ensuring sustainable consumption and production patters, and reducing the carbon footprint of developed countries. Our research aligns with this development goal in the following manner.

A.2.1 Target 12.2

The target 12.2 states *By 2030, achieve the sustainable management and efficient use of natural resources.*

Our approach aligns with this target as it follows a trend in research of making

models generalizable, instead of training a model for every single topology as it was previously being done, we developed a model that can be trained only once and deployed on multiple topologies, reducing the carbon footprint of the approach.

A.2.2 Target 12.a

The target 12.a states *Support developing countries to strengthen their scientific and technological capacity to move towards more sustainable patterns of consumption and production*

Our research aligns with this target, as with previous approaches such as CFR-RL, a different model had to be trained for every single network topology, if industry from developing countries wanted to adopt this approach it would have needed to train a model for every single network. With our approach, they may be able to use pretrained models, or if they have the capabilities train a model on their own for the desired topologies.

Appendix B

Results and Reproducibility

This section will contain the results of experiments that were not relevant enough to be part of the thesis. More than 300 training runs were performed and this section will not contain the results of all runs, but only the ones that are considered relevant.

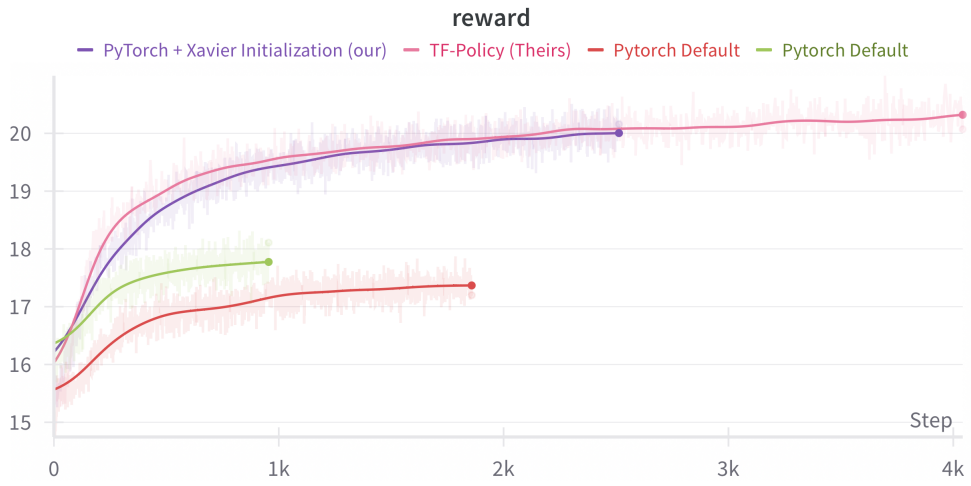

Figure B.1: Attempt to replicate CFR-RL

The first experiments were based around replicating the results obtained by CFR-RL, the TensorFlow results are the ones obtained by the code provided in the

GitHub repository, the ones in PyTorch are the ones that were obtained replicating their experiment. Replicating the results resulted challenging, as PyTorch provided us with really inconsistent results, after careful experimentation we discovered that the weight initialization used in PyTorch and the weight initialization used in TensorFlow are different, when that issue was addressed we were able to replicate their results.

Once the results of the paper were replicated we decided to attempt some other approach by using an actor-critic algorithm.

Figure B.2: Actor Critic Algorithm

It can be observed that even if this approach peaks higher than the REINFORCE approach it quickly deteriorates and provides worse results. Keep in mind that this results are not testing any sort of generalization and are just attempts at CFR-RL.

When we were conceded access to the Frontera supercomputer, we started the training of longer runs on different topologies, the following graph will present the first results of training in different topologies. The reward was defined slightly different to be scaled between -1 (ECMP) and 1 (TOP-K), this approach was later

avoided due to rounding issues.

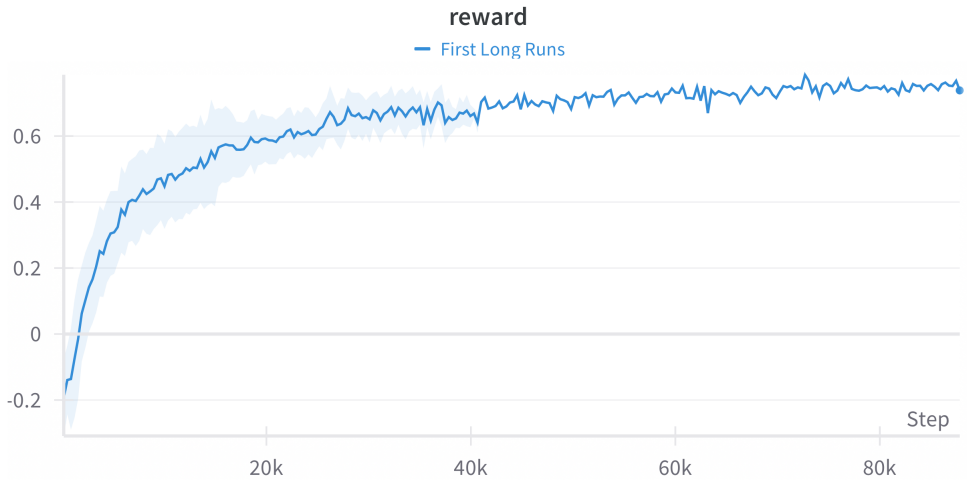

Figure B.3: Training in different topologies

The obtained results were somewhat positive, but we were not able to obtain a model capable of surpassing 0.8 reward.

After obtaining this results, we tested the Actor-Critic approach with multiple topologies.

The results are significantly worse than the results obtained in the REINFORCE algorithm, for this reason we decided to discard the AC algorithm and focus on REINFORCE. In hindsight, we should have explored more this approach as it could have brought some capabilities not present in a different approach.

We proceeded to explore having a different number of traffic matrices per topology, in the following graph you will see the results of each approach, having 500, 1000, 3000 and 4000. Previous tests were performed with 2000, so it was ignored in this experiment.

We continued by comparing the different architectures presented previously in this thesis, the MLP architecture, the transformer architecture and the Conv architecture.

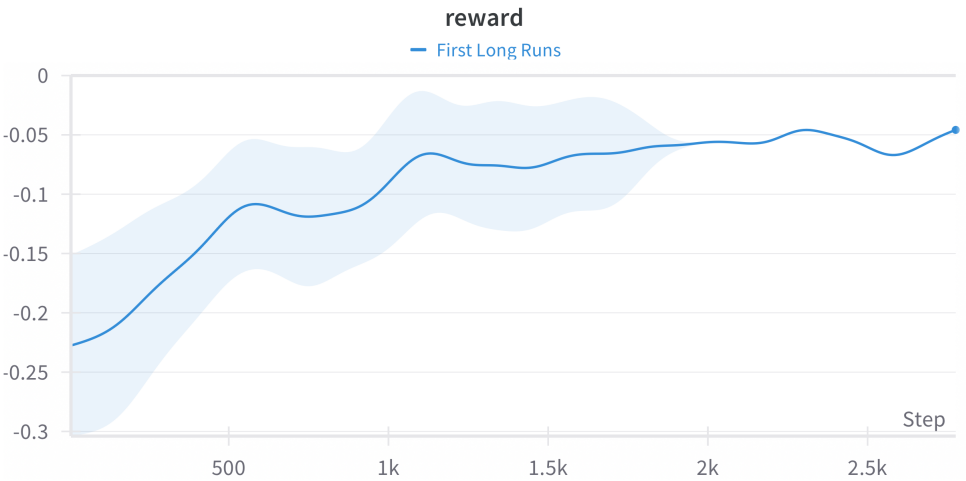

Figure B.4: AC in different topologies

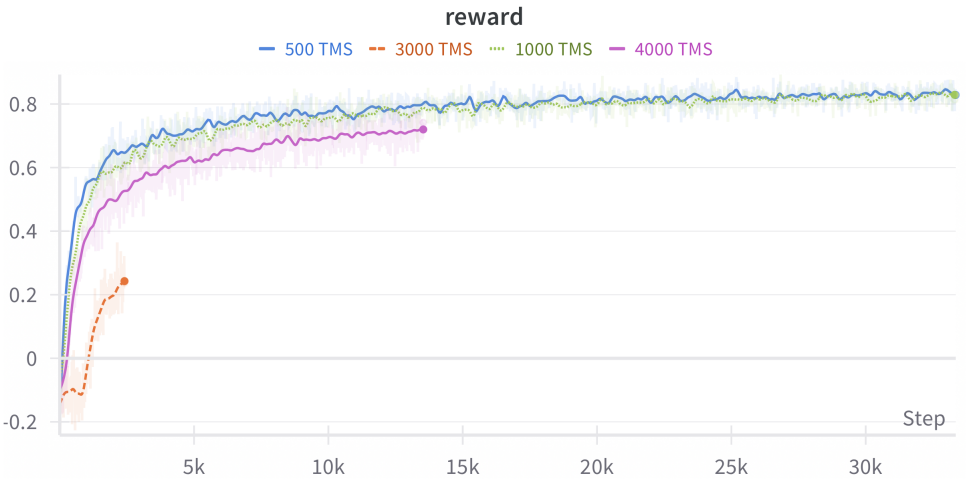

Figure B.5: Different amount of TMs per agent

All the models until this point were trained on the old reward, this is the case for it being limited at 0.8, we discovered a rounding error in our reward function that caused this the reward to be lower than expected in some cases. After solving this issue we continued with the Convolutional model and obtained the results found in the Results section.

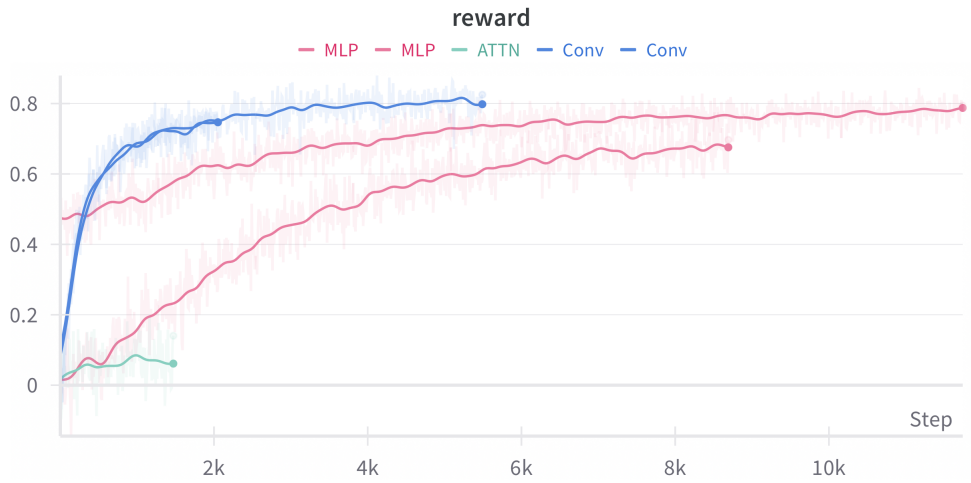

Figure B.6: Different Architecture Comparison