SCALA 5G: Un mecanismo de escalamiento automático con aprendizaje por refuerzo profundo para el uso eficiente de recursos en un ambiente NFV

Trabajo de Grado

Juan Sebastian Daza Gaviria
Andrés Mauricio Maca Hurtado

Advisor: PhD. Oscar Mauricio Caicedo Rendón
Co-Advisor: PhD. José Armando Ordoñez Córdoba

Departamento de Telemática
Facultad de Ingeniería Electrónica y Telecomunicaciones
Universidad del Cauca
Popayán, Cauca, 2022

SCALA 5G: Un mecanismo de escalamiento automático con aprendizaje por refuerzo profundo para el uso eficiente de recursos en un ambiente NFV

Juan Sebastian Daza Gaviria
Andrés Mauricio Maca Hurtado

Trabajo de grado presentado a la Facultad de Ingeniería
Electrónica y Telecomunicaciones de la
Universidad del Cauca para obtener el título de:
Ingeniero en Electrónica y Telecomunicaciones

Advisor: PhD. Oscar Mauricio Caicedo Rendón
Co-Advisor: PhD. José Armando Ordoñez Córdoba

Departamento de Telemática
Facultad de Ingeniería Electrónica y Telecomunicaciones
Universidad del Cauca
Popayán, Cauca, 2022

Agradecimientos

En memoria de José Gabriel Daza, quien nos apoyó en todo momento, brindándonos la energía y la motivación durante cada uno de los proyectos que decidimos emprender. Recordamos con anhelo como desde el inicio, apoyaba cada reto que tomabamos y se emocionaba de sobremanera cuando llegaban buenas nuevas, Dios lo tenga en su santa gloria.

A nuestros padres José Daza (Q.E.P.D.), Miryam Gaviria, Mauricio Maca y Lyda Hurtado y familia, quienes nos motivaron a seguir adelante, a persistir y nunca desistir. Sabemos que nunca dudaron en sacrificar su todo, para que tuvieramos las oportunidades que ellos les resulto difícil obtener.

A nuestro líder y director Oscar Caicedo, agradecer toda su paciencia y dedicación, persistió hasta el final y nos guío durante todo el transcurso de nuestro trabajo de grado, exaltamos sus, ya conocidas por todos, excelentes cualidades académicas, técnicas y humanas.

A las compañías HUAWEI, VERITRAN y GLOBANT que confiaron en nuestras habilidades, dándonos la oportunidad de formar parte de ellas y brindaron apoyo total y comprensión durante el proceso de finalización del presente trabajo de grado y a los mentores de estas compañias que nos motivaron a seguir adelante y a crecer profesionalmente.

A nuestros compañeros de carrera, a quienes deseamos salud y prosperidad, nos sentimos orgullosos de compartir aula en la prestigiosa Universidad Del Cauca, éxitos y buena fortuna en esta nuevas fase de nuestras vidas.

Resumen

Las redes actuales y futuras deben cumplir requerimientos de rendimiento para prestar servicios multimedia de alta definición, llamadas sobre internet, y realidad aumentada entre otras. La variación de los servicios mencionados hacen que las redes se modifiquen continuamente respecto al volúmen y a la demanda específica de los usuarios. Para este propósito los operadores de red han implementado técnicas como la Virtualización de Funciones de Red y Redes Definidas por Software. Respecto a la operación de estas redes, el escalamiento surge como una herramienta clave. Sin embargo, este proceso no se puede hacer de manera trivial, debido a los diversos factores incidentes en el comportamiento de las redes. Los mecanismos actuales proporcionan soluciones, satisfaciendo parcialmente la demanda de las redes, pero aún se requieren aportes capaces de optimizar el rendimiento y la adaptabilidad para garantizar los servicios requeridos por los clientes en cada caso de uso. En este documento proponemos una solución mediante un agente de aprendizaje por refuerzo profundo llamada **SCALA 5G**. Este sistema controla el escalamiento de recursos de una manera predictiva lo cual mejora las decisiones de escalamiento haciendolas más estables y precisas respecto a otras tecnologías y algoritmos. SCALA 5G fue probado en una infraestructura 5G con condiciones de operación idénticas a la de una red comercial, escalando servicios de banda ancha multimedia mejorada y se comparó con soluciones previas basadas en los aportes de otros autores, donde mostró una mejora significativa respecto a tecnologías predecesoras siendo hasta un 40 % más preciso, superándolos de igual manera en estabilidad y cumpliendo con los estándares propuestos para la automatización del escalamiento en redes 5G de servicios multimedia.

Índice general

Índice de Figuras

Índice de Tablas

Capítulo 1

Introducción

1.1. Planteamiento del problema

Network Slicing (NS) es esencial para el desarrollo exitoso de las redes 5G [4]. De igual manera, 5G se considera clave para cumplir con diversos requisitos (por ejemplo, cirugía remota, automóviles autónomos, *Internet of Things* (IoT) masivo). NS divide cada grupo de servicios, proporcionando a cada segmento los recursos requeridos, privilegios y funciones sobre una infraestructura física común [5]. La combinación de tecnologías de la nube, *Sofware-Defined Networking* (SDN) y *Network Function Virtualization* (NFV) proporciona las herramientas necesarias para permitir la división de redes [3]. En particular, la infraestructura NFV es una base sólida para propósitos de NS. NFV posibilita componer *Network Functions* (NF) a partir de servicios de red. Cada NF se ejecuta en una máquina virtual y cada *tenant* usa, administra y asigna una *Virtual Network Function* (VNF) de acuerdo con sus propias necesidades [6].

A pesar de las ventajas de NS, quedan algunos problemas abiertos. Uno de los cuales es la complejidad de la escalabilidad de la red. Esta escalabilidad tiene como objetivo proporcionar los recursos de red de manera adecuada para mantener la *Quality of Service* (QoS) y optimizar los *Operational Expenditures* (OPEX) [6]. En 5G NS, las VNF de los *Network Element* (NE) se pueden iniciar o detener en cualquier

momento y componer para crear NS con topologías de red complejas. Los recursos de red (por ejemplo, CPU, memoria y ancho de banda) asignados a cada NS cambian constantemente. En este escenario, la gestión manual de la red completa es inviable. Por esta razón, enfoques recientes han propuesto diversas soluciones basadas en *Artificial Intelligence* (AI) para automatizar el escalamiento de los NS en la red.

Algunos enfoques [7, 8, 9, 10, 11] proponen realizar escalado automático utilizando algoritmos heurísticos. Estos enfoques modelan el estado de la red y toman las medidas de los gastos de recursos, para mediante cálculos matemáticos generar un conjunto de umbrales que definen las políticas de escala. Para reducir la complejidad computacional, estos métodos miden un conjunto reducido de recursos de red, por esta razón, los algoritmos heurísticos son fáciles y simples de implementar. Sin embargo, este tipo de técnica presenta dos limitaciones principales. En primer lugar, los algoritmos heurísticos son reactivos. Es decir, no anticipan una decisión hasta no tener un estímulo concreto, lo que provoca violaciones a los *Service Level Agreement* (SLA). En segundo lugar, estos enfoques representan el estado de la red en un momento dado. Por lo tanto, a medida que cambia la red, los resultados no son útiles y es difícil cambiar los umbrales con frecuencia [12]. En tercer lugar, estos enfoques sufren de rebotes[1] cuando el tráfico de red fluctúa entre umbrales de escalamiento, lo cual se denomina oscilación [13]. Finalmente, en la práctica, modelar redes complejas como un centro de datos o topologías con multiples NS puede convertirse en una tarea tediosa [14].

Para superar las deficiencias mencionadas, algunos enfoques como [13, 14, 15, 16] aplican *Reinforcement Learning* (RL) para escalar NS. Las soluciones basadas en RL han mostrado mayor precisión y menos oscilaciones en comparación con los algoritmos heurísticos. Sin embargo, las soluciones basadas en RL requieren varias iteraciones para aprender del entorno y poder operar en una red. El tiempo que tarda en aprender se llama *Convergence Time* (CT) y la mayoría de los enfoques existentes para el autoescalado no presentan un análisis detallado del CT. Conocer el comportamiento del CT es fundamental para la implementación práctica de RL en entornos de telecomunicaciones. Por otra parte, la mayoría de las soluciones basadas en RL para escalado automático se prueban utilizando escenarios simulados. Por lo

[1] Decisión cíclica no deseada entre un estado y otro consecutivo

anterior los algoritmos de RL no se alinean completamente con los requerimientos de las redes 5G.

En la literatura, existen algunos enfoques [17, 18, 19, 20] que utilizan *Deep Learning* (DL) para realizar el escalado automático. DL usa *Neural Networks* (NN) para encontrar patrones a partir de datos existentes y experiencias pasadas. Estos enfoques tienen como objetivo pronosticar los comportamientos del tráfico, determinando así la cantidad de recursos necesarios para cada NS o VNF. Estos enfoques ofrecen mejoras significativas en comparación con las soluciones basadas en RL mencionadas. Primero, estas soluciones pueden aprender más rápido que las de RL. En segundo lugar, pueden operar en sistemas con un número de variables más grande con un costo menor [21]. DL tambien se caracteriza por encontrar umbrales de escalamiento, pero a diferencia de los algoritmos heurísticos el mismo algoritmo puede auto-ajustarse [12]. No obstante, las soluciones basadas en DL solo se han modelado y probado en entornos simulados. Además, estos enfoques requieren una gran cantidad de datos necesarios para entrenarse.

A partir de lo anterior, los enfoques existentes para el autoescalado comparten las siguientes limitaciones: primero, los enfoques anteriores no consideran la diferencia entre entornos de simulación y la diferencia con la aplicabilidad en escenarios reales [6]. En segundo lugar, ninguno de los trabajos anteriores presenta un análisis completo del tiempo de convergencia, adaptabilidad a diversos escenarios diferentes. Por último, los enfoques no tienen un enfoque completo de escalamiento hacia las nuevas redes como 5G que requieren un tratamiento más crítico y especializado. En este sentido, la pregunta de investigación de este proyecto es:

¿Cómo realizar el escalado automático en un NS sobre un entorno realístico con un tiempo de convergencia competente?

1.2. Objetivos

1.2.1. Objetivo general

- Proponer un mecanismo de auto-escalamiento para el manejo eficiente de recursos en NS.

1.2.2. Objetivos específicos

- Diseñar un mecanismo de escalamiendo basado en DRL para el manejo eficiente de recursos de funciones de red.
- Implementar un prototipo del mecanismo propuesto sobre una infraestructura NFV-Cloud.
- Evaluar la eficiencia del prototipo en términos de adaptabilidad, tiempo de convergencia y precisión.

1.3. Contribuciones

Las contribuciones de este trabajo son las siguientes:

- Un Agente de DRL que permite el auto-escalamiento de recursos de una manera eficiente.
- Un análisis de eficiencia completo del comportamiento del agente DRL que demuestra su capacidad y potencial respecto a otros enfoques.
- Un prototipo estable 5G/NFV sobre la nube que es de fácil despliegue y podrá usarse en investigaciones futuras.

1.4. Estructura

Este documento se divide en 6 capítulos:

- Capítulo 1 presenta la **Introducción** que contiene el planteamiento del problema, objetivos, contribuciones y la estructrura del documento.
- Capítulo 2 muestra el **Marco Teórico** sobre los temas relacionados a nuestra investigación y desarrollo. Los temas principales son redes 5G, NFV, NS, RL y DRL.
- Capítulo 3 presenta los **Trabajos relacionados** que describen las investigaciones previas llevadas a cabo en el área de escalamiento. tomando el tipo de tecnología como criterio para agrupar las soluciones.
- Capítulo 4 presenta **SCALA 5G**, la motivación, arquitectura, y descripción de módulos construídos en el prototipo tanto del algoritmo DRL como de la infraestructura que controla.
- Capítulo 5 presenta la evaluación del prototipo, la configuración del ambiente de pruebas, la parametrización del algoritmo y el **análisis de resultados**.
- Capitulo 6 presenta las **Conclusiones** y el **Trabajo Futuro**. Presentamos el cierre de nuestro trabajo y la proyección que encontramos para futuras investigaciones.

Capítulo 2

Conceptos Fundamentales

Esta sección muestra los conceptos principales relacionados a nuestra propuesta. De esta forma, se inicia con el concepto principal de redes de quinta generación, sus principales caracteristicas e importancia para aplicaciones de red futuras. En segundo lugar, explicamos los conceptos de NFV y su infraestructura. Luego, seguimos con NS donde se resaltarán sus beneficios para las redes. Posteriormente se detallan los conceptos de IA y sus clases mas importantes. Por último se encuentra el concepto de escalamiento y sus tipos principales.

2.1. Redes de quinta generación

La quinta generacion de comunicaciones móviles es el presente y futuro cercano en el campo de acceso de las redes de nueva generación (Next Generation Networks, NGN) [22]. 5G es una tecnología bastante avanzada respecto a sus predecesoras, y trae diversas mejores en las características de la red y en los casos de uso para nuevos servicios. Los *tenants* de red esperan lograr tasas de datos extremadamente altas, latencias despreciables, movilidad de usuario alta, y comunicaciones ultra-confiables con las redes 5G [4].

La Union Internacional de Telecomunicaciones sector radio (ITU-R) definió 3 casos de usos en la Conferencia Internacional de Telecomunicaciones Móviles (IMT)

2020: *enhanced Mobile BroadBand*(eMBB), *Ultra Reliable Low Latency Communications*(URLLC) y *massive Machine Type Communications* (MTC) [4].

- **eMBB**: este caso de uso se enfoca en el cubrimiento de las aplicaciones dirigidas hacia los seres humanos como el acceso multimedia, servicios de comunicación y de trasferencia de datos (e.g. video, audio y contenido generado por el usuario) [23]). Para este propósito se deben soportar grandes cantidades de datos y usuarios. Otros factores como la latencia y la densidad de conexión deben ser tenidos en cuenta pero con menor prioridad [24]. De acuerdo con [25], eMBB requiere una alta tasa de transmisión de datos, al mismo tiempo debe mantener una disponibilidad del 99% valor aceptable para este tipo de aplicación. eMBB al ser un caso de uso para servicios masivos de usuario tiene diversos campos de estudio como ejemplo accesos satelitales [26] o bien optimización de eMBB sobre redes *5G Non-Standalone* (5G NSA) [27].

- **URLLC**: a diferencia de eMBB, este caso de uso cubre aplicaciones de mayor priorización (e.g., sensores industriales, telemedicina, vehiculos autónomos), las cuales requieren latencias ínfimas, movilidad extendida y gran disponibilidad de servicio [24]. *European Telecommunications Standards Institute* (ETSI) y el *3rd Generation Partnership Project* (3GPP) fijaron un tope máximo para latencia de 0,5ms en el plano de datos de usuario tanto en carga como descarga y una confiabilidad de almenos 99,999% [28]. Debido a los estrictos SLA de URLLC se deben priorizar su tráfico como se muestra en [29] y [30] en aplicaciones críticas como V2X [31] o cirugía remota [32].

- **mMTC**: este caso de uso surge principalmente para los dispositivos IoT. Aplicaciones como ciudades inteligentes y servicios con baja dependencia a la latencia de red pueden ser cubiertos. La densidad de dispositivos IoT es elevada y es importante garantizar a cada terminal pueda acceder a la red oportunamente [24]. Por otro lado, existen aplicaciones relacionadas a la automatización de procesos industriales que requieren latencia *End-To-End* (E2E) muy baja [4], tambien necesitan en conjunto el manejo de URLLC para algunas de sus tareas. Un ejemplo típico mMTC es la recolección de medidas de un grán número de

sensores, tambien conocido como *Smart Metering*. Estos dispositivos requieren redes capaces de proveer como mínimo 1Gbps de ancho de banda [33].

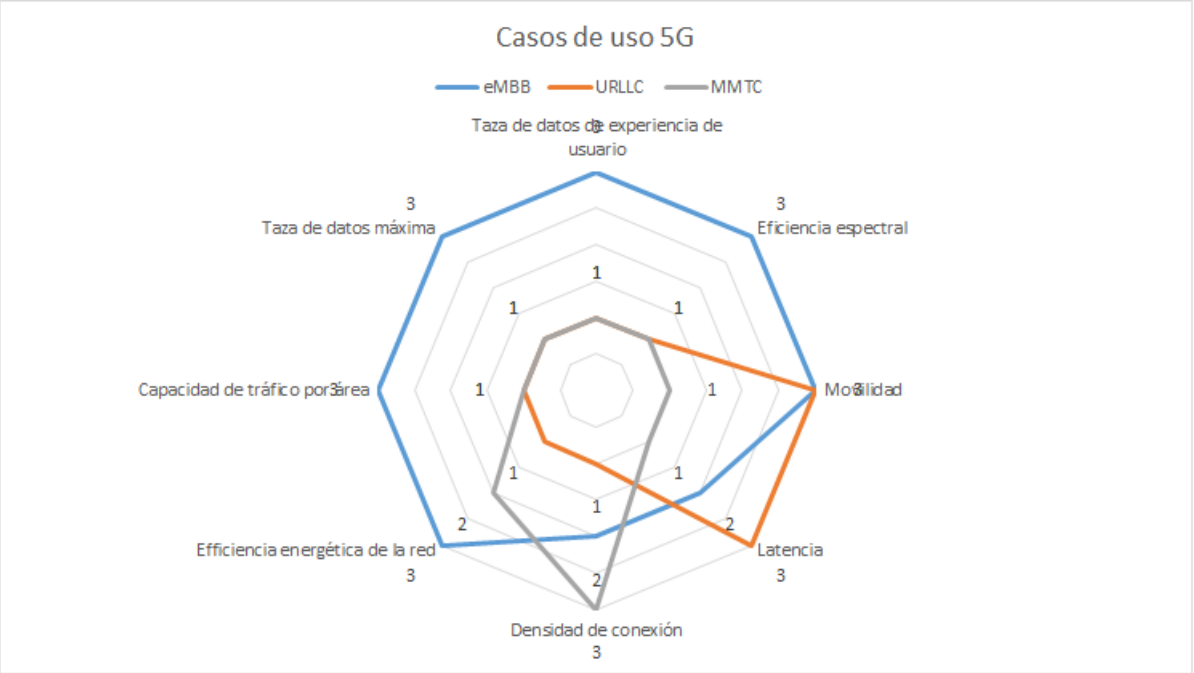


Figura 2.1: Requisitos de 5G

Para facilitar la migración desde LTE, se ha propuesto un modelo de 5G denominado 5G NSA donde se empiezan a establecer accesos 5G pero usando el núcleo LTE-EPC. posteriormente se pasa a una red *5G Standalone* (5G SA), donde el *5G Core* (5GC) reemplace a su predecesor. En la figura 2.2 se muestran las implicaciones de estas dos maneras de proveer servicios de 5G. En la parte izquierda de la figura se encuentra 5G NSA donde los accesos LTE y NR envian el plano de usuario hacia el núcleo EPC, mientras el acceso LTE se encarga de la señalizacion y control. En el lado derecho 5G SA donde LTE-EPC y 5G-NR son desplegados de manera independiente con sus correspondientes planos de control y usuario.

Ahora bien el 3GPP tambien definió rutas para la migración de los operadores de red, con el fin de que los *Capital Expenditures*(CAPEX) sean más comodos para ellos. En la figura 2.3, se muestra con mayor detalle cuales son los caminos que se pueden seguir para hacer un cambio de accesos y núcleo con los menores CAPEX

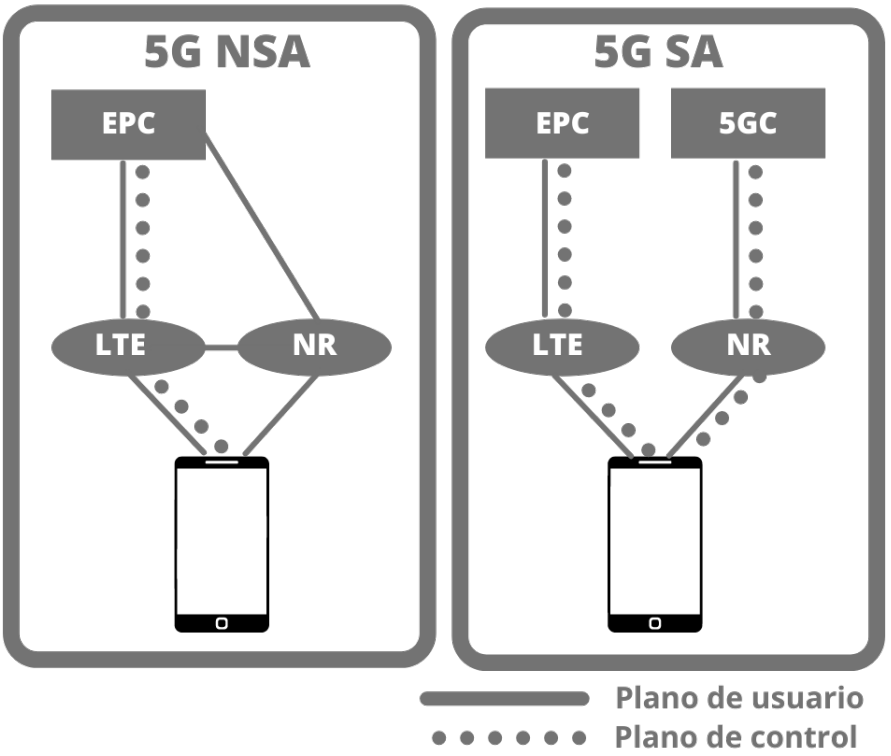


Figura 2.2: 5G *Non-Standalone* & *Standalone*[1]

y OPEX posibles. Las opciones 2 y 4/4a facilitan la transición en el ámbito de la interconexión de tecnologías LTE-5G priorizando el uso del 5G SA, mientras las opciones 3/3a/3x y 7/7a/7x hacen transiciónes con el uso de 5G NSA y la opción 5 es un híbrido menos común que las anteriores.

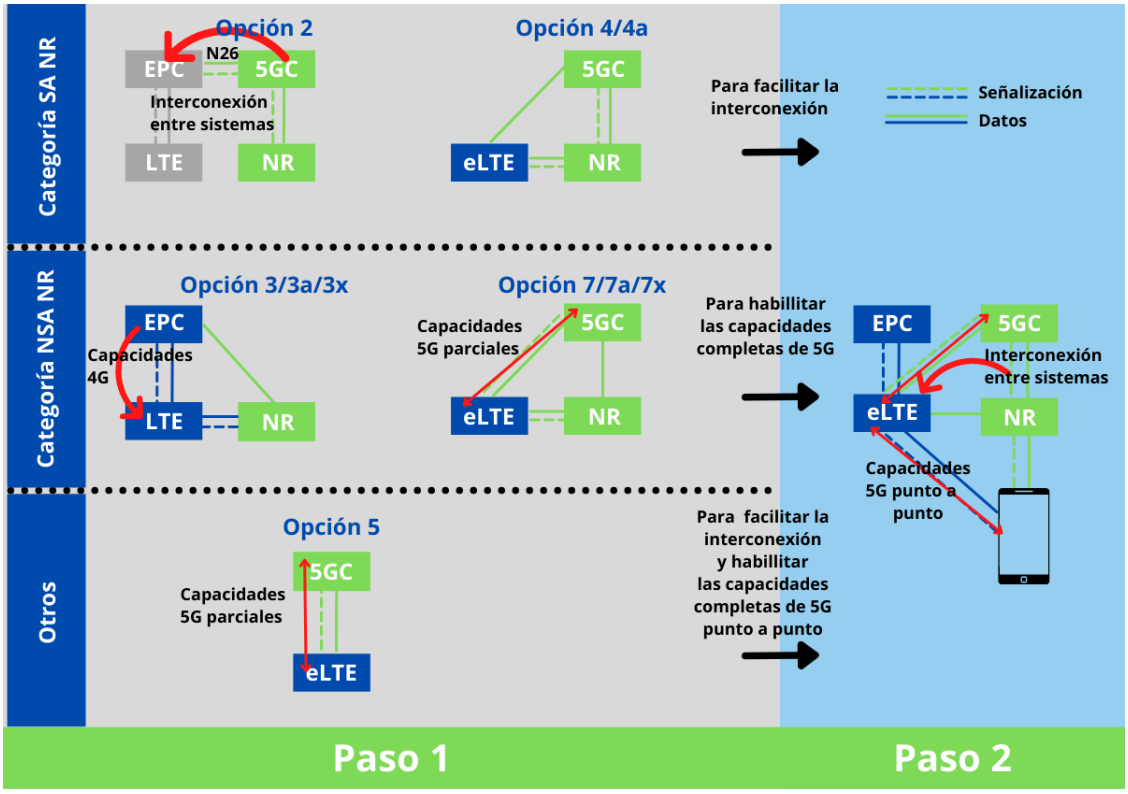


Figura 2.3: Rutas de migración hacia 5G [2]

2.2. Virtualización de funciones de red

La virtualización de funciones de red es una tecnología que se encarga de separar las NF que tradicionalmente eran equipos ligados al hardware y las convierte en aplicaciones que son ejecutables sobre servidores en la nube sobre una infraestructura física compartida [6]. De esta forma, NFV resulta ser altamente beneficiosa para reducir CAPEX y el consumo energético. NFV tambien fomenta la innovación y la personalización de servicios de red creados para cada necesidad [34]. Con la virtualización se garantiza el procesamiento óptimo de las VNF de un NS y la mejora de la eficiencia operacional en la red [24].

La figura 2.4 muestra la arquitectura de NFV y sus tres principales capas: 1) *NFV Infrastructure* (NFVI), 2) VNF y 3) *Management and Orchestration* (MANO).

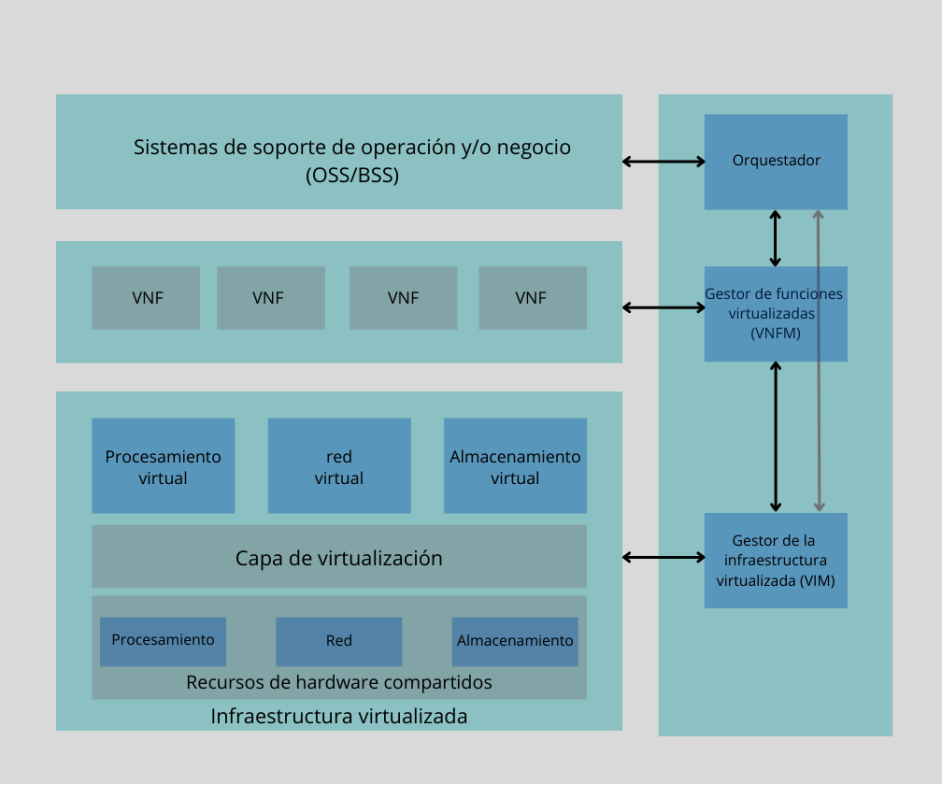


Figura 2.4: Arquitectura de virtualización de funciones de red

- **NFVI**: Esta capa contiene los recursos físicos disponibles, como los nodos de procesamiento, servidores de almacenamiento y conexiones de red. Tambien contiene la infraestructura de virtualización para la generación de *slices* en las capas superiores [35].

- **VNF**: Esta capa ofrece todas las funciones en la red NFV, encargada de alojar las funciones de red sobre la capa de infraestructura establecida en la NFVI. Cada función es definida para el operador de red que la requirió, y es creada con los requerimientos y limitaciones descritos en el diseño de servicio del NS. [35].

- **MANO**: Es la capa trasversal de NFV, es así ya que tiene interacción simultánea tanto con la NFVI como con la capa VNF. Por otra parte, MANO se encarga del control y monitoreo de red. Así, MANO se divide en 3 módulos. El primero es *Virtual Infrastruture Manager* (VIM) que interactúa con la NFVI

manejando sus recursos. El segundo es el *VNF Manager* (VNFM) para controlar la capa VNF. VNFM debe comprobar constantemente el ciclo de vida de las VNF, además de ofrecer un tratamiento para las posibles fallas durante la ejecución. En el nivel superior del MANO está el Orquestador, el cual identifica las tecnologías y funciones apropiadas para el cumplimiento de los requerimientos de NS [35]. Además debe tener en cuenta la cantidad de recursos que debe suministrarle a cada NS haciendo el proceso de escalamiento.

NFV sigue una arquitectura orientada a virtualizar los tipos de recursos físicos, ejecutando múltiples funciones sobre la misma infraestructura física. Cada VNF se asigna a un cierto tipo de recurso virtual, teniendo en cuenta el caso de uso para el cual es requerido y la necesidad específica del operador de red (e.g., un servicio de transmisión de radio requiere mayor estabilidad que servicios web tradicionales).

Por otra parte, las necesidades de reducir la latencia en el plano de usuario y de tener un control de red centralizado hace relevante el uso de tecnologías complementarias a NFV como lo es SDN. En SDN se pueden diferenciar tres capas:

- Capa de aplicación: contiene todos los programas que satisfacen las necesidades y comportamientos deseados sobre la red.
- Capa de control: es el centro de operación de la red, se encarga de controlar todos los elementos de la infraestructura, envía constantemente órdenes para que cada elemento sepa como debe trabajar.
- Capa de infraestructura: es la red y todos sus componentes físicos y lógicos como switches, enrutadores, *firewalls*.

SDN permite que la red pueda cambiar al ritmo que la virtualización requiere lo cual es necesario para que los sistemas 5G puedan desplegarse con mayor flexibilidad.

2.3. Segmentación lógica de red

NS es clave para el funcionamiento efectivo de 5G, hace que elementos y funciones de red sean configurados fácilmente y cubran requerimientos similares [3]. NS permite a una sola red física crear diversas redes lógicas para ser configuradas de acuerdo a los diferentes casos de uso para 5G [4]. Para soportar los *slice* de red, cada una de estas redes lógicas debe estar aisladas entre sí, garantizando la interoperabilidad *End-to-End* (E2E) [36]. Dentro de las redes 5G algunas NF como balanceadores de carga, limitadores de ancho de banda, enrutadores virtuales, pueden desplegarse rápidamente. De igual manera, se pueden conformar funciones de mayor complejidad como VNF de 5GC (e.g., AMF, UPF), o del núcleo vEPC de 4G (e.g., MME, SGW, PGW).

La figura 2.5 muestra que la arquitectura de NS contiene *slices* de acceso (acesso por radio y terrestre), *slices* de *Core Network* (CN), y la parte intermedia la cual es la función de emparejamiento entre la red de acceso y la CN. La función de emparejamiento enruta las comunicaciones desde un *slice* de acceso a un *slice* CN apropiado que está diseñado para proporcionar servicios específicos. Los criterios para definir los *slice* de acceso y los de CN incluyen la necesidad de cumplir con diferentes requisitos de servicios [3]. De acuerdo con [37], algunas tecnologías como SDN [38], *Fog Computing* [39], *Multi-Access Edge Computing* (MEC) [40] y técnicas de virtualización [41] contribuyen a la correcta implementación y aportan a la escalabilidad, seguridad y estabilidad de NS.

2.4. Aprendizaje automático

ML es un área de AI encargada de la representación computacional de los fenómenos presentados en cualquier ambiente a estudiar. Lo anterior con el fin de encontrar patrones capaces de optimizar la toma de decisiones sobre el sistema [42]. Actualmente, ML es un marco de referencia acumulando múltiples contribuciones en las áreas de las telecomunicaciones como *Wireless Sensor Networks* (WSN), clasificación de tráfico de red, predicción de flujo y de movilidad de usuario [43].

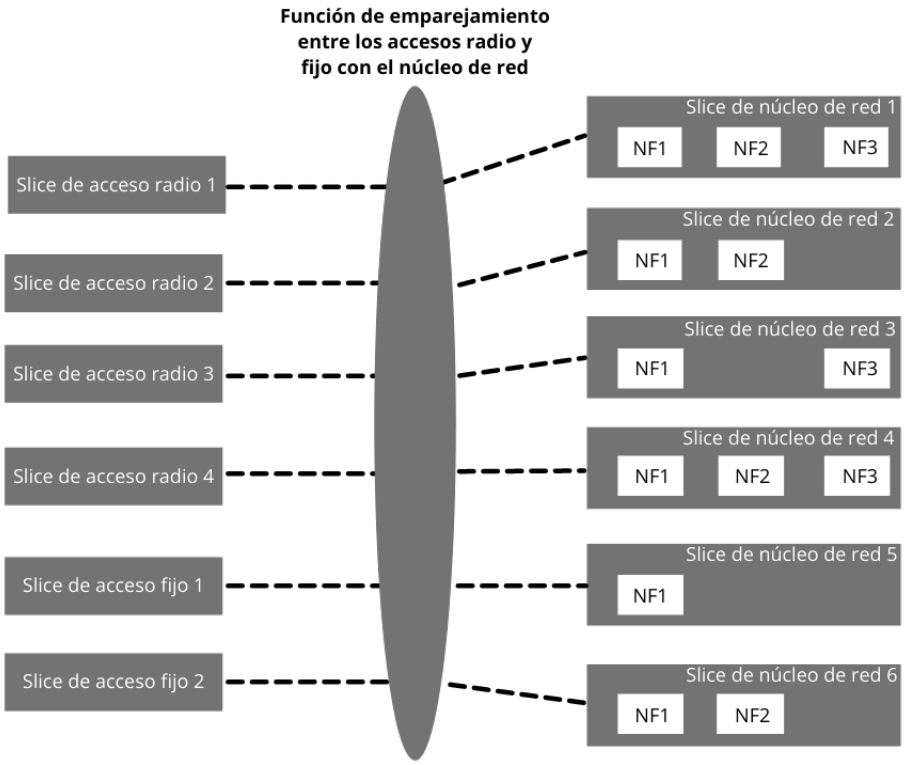


Figura 2.5: Segmentos lógicos de red [3]

Los algoritmos de ML son clasificados de acuerdo a su forma de aprendizaje [42]. Cada grupo tiene distintos propósitos y casos donde pueden ser aplicados. De acuerdo a su forma de entrenamiento y datos de entrada existen los siguientes 5 grupos:

- ***Aprendizaje supervisado*:** *Supervised Learning* (SL) contiene algoritmos adecuados para regresión y clasificación de datos. SL utiliza datos con características etiquetadas, es decir se conoce su proveniencia e importancia en el ambiente. Su finalidad es encontrar relación entre los datos de entrada y de salida [42], comúnmente se intentan buscar relaciones de caracter lineal, cuadrático o polinómico al ser los casos más frecuentes. En nuestra área específica, SL es una herramienta de gran utilidad en casos específicos como

ataques *Distributed Denial of Service* (DDoS), clasificación del tráfico de red y WSN[44]. De acuerdo con [42], generalmente los algoritmos mas comunes de SL son *K-Nearest Neighbors* (KNN) [45], *Artificial Neural Network* (ANN) [46] y *Support Vector Machines* (SVM) [47].

- **Aprendizaje no Supervisado** *Unsupervised Learning* (UL) es un tipo de ML destinado a buscar patrones no detectados previamente con datos no etiquetados. Es decir, se desconoce la proveniencia de los datos. De esta forma, UL es propicio para casos donde los investigadores desconocen la importancia de cada variable. Los algoritmos de UL encuentran correlaciones entre las variables y los datos similares y su aprendizaje esta sujeto a las observaciones desde el momento que inicia su fase de entrenamiento. Algunos algoritmos de UL son *K-Mean Clustering* [48], *Principal Component Analisys* (PCA) [49] y *Self-Organizing Mapping* (SOM) [50].

- **Aprendizaje por refuerzo**: RL es otra sub-area de ML que difiere en gran medida de SL y UL. Para el entrenamiento de RL se debe aprender directamente del entorno. Al principio el ente encargado de la toma de decisiones denominado agente, toma decisiones aleatorias debido a que desconoce totalmente cual es el comportamiento adecuado y a medida que trascurre el tiempo el agente es capaz de mejorar sus elecciones a través de la experiencia adquirida [42, 21]. Por lo anterior, RL resulta ser mejor que otras técnicas de ML en los casos donde se requiere experiencia a largo plazo [44].

  RL normalmente utiliza modelos de estado-acción para saber como se debería comportar en cada posición basándose en las cadenas de Markov, el método mas común que aplica estos principios se llama Q-learning [51].

  Como se mostrará en la siguiente sección, RL tiene rendimientos superiores al tomar decisiones instantáneas respecto a otras técnicas de ML, lo que lo hace adecuado para el control en entornos de red. Además su fácil adaptación y progresión, adapta sus políticas con cada iteración que hace [21]. Los algoritmos de RL puede ser medidos por su tiempo de convergencia, el cual es determinante para saber cuándo el agente estará listo para sus funciones.

- **Aprendizaje profundo**: Es una sub-area de ML que crea modelos de apren-

dizaje con estructuras basadas en capas. DL es capaz de aprender mas rápido que las técnicas anteriores de ML, debido a que usa múltiples capaz de procesamiento que adhieren comportamientos no lineales que aceleran la optimización de la solución. Por su comportamiento orgánico y colaborativo los modelos de DL se denominan redes neuronales (DNN). Unos de los tipos de DNN mas comunes son las redes convolucionales (CNN), las redes recurrentes (RNN), y el aprendizaje por refuerzo profundo (DRL) [43].

- **Aprendizaje por refuerzo profundo**: DRL es una técnica que combina RL y DL. Se caracteriza por ser altamente versatil y tener una respuesta ante cambios efectiva. La estructura de aprendizaje que hereda de RL hace que pueda entrenarse directamente desde el ambiente, mientras que DL cumple las funciones de análisis de los datos que ingresan y adquiere experiencia para mejorar su salida. Este método hace que DRL pueda converger en un tiempo menor a RL y con menos datos que DL. La técnica mas común utilizada para DRL es deep Q-learning, similar al Q-learning de RL, cuyo objetivo principal es optimizar el sistema[21]. DRL tambien se usa para la solución de problemas estadísticos, modelos de predicción de clima y para creación de juegos [52, 53]. El algoritmo Deep Q-Learning itera sobre una NN la cual es alimentada por una función de recompensa y bajo este criterio busca maximizar la recompensa obtenida cada iteración busca que la experiencia obtenida reduzca el error mediante los procesos de optimización y *backpropagation* propios de las NN.

2.5. Escalamiento de funciones de red

El escalamiento es una técnica fundamental para la operación de los ambientes 5G y NFV. Se encarga del suministro adecuado de la cantidad de recursos que debe tener cada VNF o *slice*. El escalamiento debe garantizar que en cada momento de operación los servicios y aplicaciones del ambiente tengan la cantidad exacta para que su QoS no se vea degradado. El escalamiento puede realizarse de diversas formas. Inicialmente para redes pequeñas y previas a 5G como las LTE-EPC se realizaba de manera manual. Sin embargo, con la introducción al mercado de 5G,

los requerimientos y el volumen de usuarios incrementa notablemente desde las redes de campus, hasta las redes troncales y metropolitanas. Por lo que actualmente se busca automatizar este procedimiento.

Actualmente se cuenta con 3 diferentes maneras de realizar escalamiento:

- Vertical: Consiste en modificar la cantidad de recursos (CPU, Memoria, Ancho de Banda) asignados a las instancias ejecutando servicios o aplicaciones. Cuando los recursos son ampliados se denomina *scaling up*, y si se disminuyen *scaling down*.
- Horizontal: Consiste en modificar el número de instancias que soportan un servicio o aplicación. si la operación aumenta las instancias se denomina *scaling out* y si las disminuye *scaling in* [54]. Este es el tipo más utilizado de escalamiento, debido a su baja complejidad computacional ya que los equipos actuales y nubes tienen mayor fluidez para desplegar o terminar instancias que para cambiar recursos de las instancias sobre la marcha. Esta razón fue determinante para la elección de este tipo de escalamiento en el desarrollo de SCALA 5G.
- Elástico: Este tipo sucede cuando se combinan el escalamiento vertical y horizontal. En este escalamiento, aumentan las acciones posibles que se pueden tomar al tener 2 dimensiones. Para esto se requiere que el hardware tenga mayor flexibilidad, incrementando los costos en la infraestructura. En consecuencia, el escalamiento elástico es poco común.

Capítulo 3

Trabajos Relacionados

El escalamiento sobre NS es un tema de gran interés y ampliamente estudiado por los investigadores. Hay diversos trabajos sobre el escalamiento en NS, los cuales estan orientados a tener un gestor de recursos autónomo e inteligente. Los trabajos encontrados dan una visión del panorama actual sobre la la eficiencia en el escalamiento y el cumplimiento de los SLA de los servicios proporcionados, con el fin de mejorar la calidad de servicio de los usuarios.

Esta sección presenta los trabajos relacionados con nuestro trabajo de investigación, divididos en 4 grupos distintos de acuerdo con el tipo de técnica utilizada. Las técnicas encontradas fueron algoritmos heurísticos, RL, DL y DRL. Cada técnica se muestra con un analisis de sus ventajas, desventajas, conclusiones y aspectos reelevantes del escalamiento vertical, horizontal y elástico.

3.1. Algoritmos de auto-escalamiento de funciones de red heurísticos

El trabajo propuesto en [7] muestra una solución de auto-escalado en 5G definida por umbrales. Para esto se usa una red LTE-EPC compuesta por funciones como el *Service Gateway* (SGW) y el *PDN Gateway* (PGW). Este algoritmo se centra

en optimizar el ancho de banda del plano de datos. Sin embargo, este algoritmo solo muestra el escalamiento obtenido con el método propuesto, así es no se tiene certeza de que tan efectivo es respecto a otras soluciones. Por otra parte al ser un algoritmo con umbrales definidos previamente presenta fluctuaciones no deseadas cuando el tráfico de red incrementa y disminuye repetidamente, tomando decisiones de escalamiento repetitivas e innecesarias.

En [8] se muestra un algoritmo similar a [7]. De igual manera, puede escalar funciones de un núcleo LTE-EPC. En este algoritmo se plantean el escalamiento de manera elástica, con lo cual tiene mayor libertad de encontrar estados específicos para cada variación de tráfico presentada. Esta arquitectura tiene definidas regiones de escalamiento de acuerdo a las que toma decisiones sobre lo que debe hacer para mantener las funciones en las condiciones óptimas de servicio. Sin embargo, el uso extendido de umbrales en varias dimensiones dificulta la adaptabilidad, y continua con el problema de las fluctuaciones mencionado.

El siguiente algoritmo *Dynamic Auto Scaling Algorithm* (DASA) [9] es un algoritmo heurístico de escalamiento horizontal. Los umbrales son definidos de acuerdo a un modelo creado basado en encolado *First Come First Served* (FCFS) para atender las peticiones por prioridad de llegada. Así mismo, el algoritmo mide el tiempo que se demora el servicio en responder con lo cual se establecen los limites para tomar las desiciones de escalamiento. La aproximación simula un escenario de redes con tráfico basado en entornos reales. Sin embargo, persiste el uso de umbrales como en los anteriores algoritmos, lo cual dificulta el ajuste y la adaptabilidad en diversos tipos de red, generando afectaciones de servicio e ineficiencia en la gestión de recursos.

[10] propone un algoritmo sobre el *Service Function Chain* (SFC) completo, con un escalado elástico de VNF y de ancho de banda. Esta característica mejora los niveles de granularidad y diferencia el escalado de recursos de procesamiento de los recursos de red, lo cual es una mejora significativa respecto a los trabajos anteriores. Sin embargo, el algoritmo tampoco soluciona la definición estática de los umbrales. Además, fuera del ambiente de simulación el escalado heurístico del ancho de banda puede ocasionar cuellos de botella en la comunicación entre las VNF, desencade-

nando violaciones de SLA y fluctuaciones en horas críticas de tráfico de red.

El trabajo [11] muestra un sistema de escalado para un entorno 5GC. Dicho sistema calcula cuál es el mejor slice seleccionable ya sea de forma manual o automática. Este algoritmo tiene en cuenta las peticiones de usuario entrantes para determinar si los *slice* en ejecución pueden seguir albergando los usuarios o por el contrario, deben ser modificados sus recursos. De igual manera, la propuesta tiene funciones de monitoreo y ciclo de vida de VNF que ayudan al operador de red a verificar si la red se encuentra en sus estados normales de operación. A pesar de todas estas ventajas, el algortimo sigue teniendo base definida por umbrales, por tanto se torna dificil adaptarlo como solución en otros ambientes de red diferentes al entorno de evaluación.

3.2. Algoritmos de auto-escalamiento de funciones de red basados en aprendizaje por refuerzo

El algoritmo mostrado en [13] es un algoritmo de auto-escalado basado en RL. Este algoritmo tiene un espacio de estados M que depende del estado actual, la carga de trabajo y el rendimiento de las VNF, un espacio de acciones A que varía entre un A_{min} y A_{max} de VM desplegadas, una transición entre 2 estados T y una recompensa R que depende de dos variables de monitoreo de las VNF: Rendimiento el cual mide el QoS del slice, y la carga de trabajo que es el cálculo del gasto de recursos. Con esa arquitectura, este algoritmo tiene una mejora significativa respecto a los heurísticos. Los resultados demuestran que este algoritmo puede reducir las violaciones de los SLA hasta en un 50%, y menos probabilidad de ser vulnerable a las fluctuaciones de red respecto a los algoritmos heurísticos, lo que reduce el número de operaciones de escalamiento.

El trabajo [14] presenta el algoritmo Cloud-RL. En este caso, se usa un grupo de agentes RL autónomos que trabajan de manera simultánea y ejecutan tareas en paralelo para tomar decisiones de escalamiento. La técnica anterior utilizada se denomina *Parallel Q-Learning* y optimiza la gestión de los recursos en ambiente

basados en la nube. El espacio de estados de este algoritmo depende del número de peticiones de usuario y del número de VM activas. y un espacio de acciones de 3 para aumentar, disminuir o mantener el número de VM. El Q-Learning en paralelo presenta bajo tiempo de convergencia respecto a RL común debido a la multiplicidad de experiencia que cada agente pueda almacenar. En los resultados se muestra , es una opción precisa para redes geográficamente distribuidas.

En [15] se presenta un algoritmo basado en RL de auto-escalado tiene como propósito de hacer un uso eficiente de los recursos que corresponden a las funciones MME de un núcleo LTE-EPC. El mecanismo se compone de dos algoritmos: Primero un algoritmo basado en *Gaussian Proccesses* (GP) que consta de un modelo de distribución normal de probabilidad que ayuda a anticipar cuales serán los comportamientos futuros del tráfico de usuario. y en segundo lugar un algoritmo de RL tradicional con un espacio de estados por el número actual de instancias y el rendimiento del sistema, y el espacio de acciones que se define como el número de instancias disponibles para soportar las MME. Además el algoritmo recibe la salida del primer algoritmo. De esta manera, la arquitectura usada hace que el algoritmo 1 anticipe el error y haga que el algoritmo 2 tome mejores decisiones. El conjunto de algoritmos muestra mayor rendimiento y precisión respecto a los algortimos heuristicos y al RL común.

[16] propone otro algoritmo de RL similar a [15] previamente mencionada. Este algoritmo tambien usa una combinación de GP pero en este caso se modela a partir del *Mean Response Time* (MRT) para cada VNF vMME en un núcleo LTE-EPC. El espacio de estados de este algoritmo comprende el número de instancias Ademas usa una red neuronal artificial (ANN) para optimizar dichos procesos y organizar la función de Q-Learning. Se observa tambien que no se publicó explicitamente detalles sobre las funciones de recompensa pero se da una idea como se logró este prototipo. La evaluación realizada indica que este trabajo reduce el número de operaciones hasta en 4 veces respecto a un algoritmo heurístico, reduce las violaciones de SLA y no es vulnerable a las fluctuaciones de la red lo que lo cataloga como estable. Sin embargo, el algoritmo no fue probado en un ambiente real y su simulación no fue explicada a detalle en su artículo.

3.3. Algoritmos de auto-escalamiento de funciones de red basados en aprendizaje profundo

El trabajo en [17] muestra una propuesta de escalado automático de acceso 5G SA utilizando RNN como agente de decisiones. En este caso, solo usa una *Long Short-Term Memory* (LSTM), que es otro tipo de red neuronal recurrente. La configuración de LSTM difiere de DNN teniendo en cuenta estados anteriores como información de contexto similar a la forma RL tiene una entrada recursiva que le permite iterar en cada paso. El agente se centra en escalar horizontalmente las VNF de *Access and Mobility Function* (AMF) (similar al MME de LTE-EPC para 5G). Para este caso, las pruebas realizadas comparan la red LSTM con una DNN clásica, donde la red LSTM tiene un 10% más de precisión que DNN, y ambas NN reducen la violación de SLA a través de una reducción de la latencia y menor uso de instancias de AMF. Sin embargo, esta solución solo fue probada en un ambiente de simulación, se propone por lo mismos autores que el algoritmo en trabajos futuros debe probarse en un ambiente real o almenos diferente al actual para determinar el comportamiento del algoritmo con patrones de tráfico diferentes a los de entrenamiento.

En [18] presenta un prototipo que optimiza el algoritmo de [17], Se agrega antes de la red LSTM una CNN con el objetivo de extraer características y encontrar relaciones no lineales. Las redes CNN se utilizan generalmente para procesamiento de imagenes, por lo cual su naturaleza le ayuda a detectar patrones profundos en la red. El proceso anterior reduce notablemente el tiempo de entrenamiento al 50% con respecto a la solución [17] y mantiene las condiciones de SLA y latencia con los mismos parámetros y ambiente de red. Por tanto, esta arquitectura presenta una robustez mayor respecto a tener solo la red LSTM. Aunque, Este algoritmo fue entrenado con datos de un operador reales, su fase de pruebas se hizo en un simulador, por lo que su efectividad en un ambiente real es incierta.

En [19] muestra una solución que compara dos redes neuronales de tipo *Multi-Layer Perceptron* (MLP): un clasificador y un regresor, ambas NN son entrenadas con un conjunto de datos reales que provienen de una empresa de Telecomunicaciones. Ambas NN se enfocan en escalar *User Plane Function* (UPF) para un 5GC. Estos

algoritmos tienen una precisión media del 97% en la evaluación realizada, y reduce notablemente la latencia de la red respecto a algoritmos Heurísticos y RL. Los resultados muestran que ambos algoritmos tiene una gran capacidad de predicción, pero el algoritmo regresor muestra mayor precisión en sus decisiones en el ambiente donde se evaluó. Sin embargo, los algoritmos necesitan una cantidad de datos de entrenamiento amplia para alcanzar su operación estable, factor que es crítico en la adaptabilidad que debe presentar un algoritmo de escalamiento automático.

El trabajo [20] propone un algoritmo de aprendizaje profundo que escala de manera automática las VNFs de una red 5GC utilizando una CNN Tridimensional (3D). La red CNN posee características adecuadas para el procesamiento de imagenes y multimedia, lo cual le permite encontrar patrones espacio-temporales, tomando decisiones mas acertadas que otros algoritmos de predicción. Sin embargo, Este trabajo igual que los previamente mencionados necesita una cantidad de datos considerable para su correcta operación, por otra parte las redes CNN tienen tiempos de entrenamiento extensos respecto a las redes DNN Clásicas como los MLP y las RNN, lo que es crítico cuando las redes cambien su comportamiento o se pretenda adaptar a redes con comportamientos diferentes a su entrenamiento original.

3.4. Algoritmos de auto-escalamiento de funciones de red basados en aprendizaje por refuerzo profundo

El trabajo [55] utiliza DRL para predecir el tráfico entrante y mantener una QoS alta en un sistema de slices de 5G, al igual que en algunos trabajos de RL, tambien se utiliza la regresión por procesos gaussianos (GPR) como técnica para predecir patrones de trafico no estacionarios, los cuales fueron modelados como un sistema de movimiento browniano fraccional (FBm). Por lo que el algoritmo resultante es capaz de predecir de acuerdo a los patrones que se le esten presentando en un instante de tiempo, asumiendo que los patrones se pueden dar en lapsos no periódicos.

Por su parte [56], propone un algoritmo de DRL para Alojamiento de recursos y

bloqueo de tráfico para el cubrimiento de los SLA de los casos de uso principales de 5G. el tráfico entrante se separa en eMBB, URLLC y MTC, dandole a cada uno una respectiva prioridad de acuerdo con los requerimientos específicos. Este trabajo uso una colección de datos sintéticos para simular y entrenar su agente, y agrego una memora de repetición para mejorar las decisiones a partir de comportamientos pasados.

Finalmente [57], usa un algoritmo de DQN mejorado para hacer el control de admisión y alojamiento de recursos simultaneamente, al igual que la propuesta anterior este algoritmo tiene en cuenta los 3 casos de 5G y se puede considerar una de las soluciones mas estables y completas a nivel de escalamiento en 5G debido a que controla la red tanto en el acceso a ella como en los procesos del núcleo.

3.5. Brechas

La tabla 3.1 muestra las brechas de investigación encontradas en la revisión de los trabajos relacionados. Se identificaron algunos factores importantes para el diferenciamiento y la justificación de nuestro algortimo. La tabla asocia los algoritmos de acuerdo a su tecnología, tambien se describe brevemente el propósito de cada uno.

Los primeros trabajos de escalamiento son los basados en algoritmos heurísticos [7, 8, 9, 10, 11] los cuales presentan baja respuesta a cambios y son susceptibles a presentar oscilaciones que lo alejan de los requerimientos de 5G. Los trabajos basados en RL [13, 14, 15, 16] mostraron mayor efectividad y flexibilidad respecto a los heurísticos. Sin embargo, su tiempo de entrenamiento no es apropiado para los casos de uso de 5G. Por otras parte los trabajos [17, 18, 19, 20] son algoritmos basados en DL, y respecto a los algoritmos de RL presentan un entrenamiento que puede hacerse en un tiempo menor, sin embargo se requiere una cantidad de datos abundante para su aprendizaje, sumado a esto ninguno de los trabajos presentados evalua su algoritmo fuera de un ambiente simulado lo que hace incierto su comportamiento en una red de producción.

Se determina entonces que las soluciones encontradas no convergen hacia la opti-

mización del escalamiento en redes 5G eMBB y por tal motivo se encuentra una brecha la cual se pretende disminuir con nuestra propuesta.

Ref	Grupo	Meta	Scaling	Contiene eMBB
[7]	Heuristic	Optimizar BW en DP para funciones LTE	Si	No
[8]		Escalamiento elastico para vLTE-EPC	Si	No
[9]		Escalamiento estadístico para slicing	Si	No
[10]		Escalamiento de BW y NF para VNFs LTE	Si	No
[11]		Escalamiento y LCM para funciones de 5GC	Si	No
[13]	RL	Escalamiento para optimización de slice QoS	Si	No
[14]		Agente en paralelo para gestión de recursos Cloud	Si	No
[15]		Escalamiento de MME LTE-EPC	Si	No
[16]		Escalamiento de VNFs para optimización de MRT	Si	No
[17]	DL	Escalamiento en 5G para funciones AMF	Si	No
[18]		Escalamiento en 5G para funciones AMF mejorado para URLLC	Si	No
[19]		Escalamiento en 5G para funciones UPF	Si	No
[20]		Escalamiento para funciones vLTE con pronóstico y optimización de SLAs	Si	No
[55]	DRL	RA con predicción de tráfico no estacionario para 5G	Parcial	Si
[56]		RA y bloqueo de tráfico para 5G	Parcial	Si
[57]		AC y RA para 5G en los 3 casos de uso	Parcial	Si
Nuestro		Escalamiento para slices 5G eMBB	Si	Si

Cuadro 3.1: Trabajos relacionados

Capítulo 4

SCALA 5G

Este capítulo presenta nuestra propuesta, denominada SCALA 5G, para el escalamiento de recursos en un entorno de 5G eMBB. Las siguientes secciones contienen la motivación, metodología, arquitectura y módulos de SCALA 5G.

4.1. Motivación

5G es una tecnología de referencia actual, y la ruta para continuar con la evolución de las comunicaciones digitales. De acuerdo con [6] existen seis desafíos planteados para la puesta en marcha de 5G: 1) comunicación de interfaces *northbound* y *southbound*, 2) escalabilidad, 3) asinación de recursos, 4) mejora del MANO, 5) aplicación de NS y 6) evaluación y rendimiento de la red.

De los desafíos anteriores, la escalabilidad es uno de los más importantes debido a que los OPEX de la red dependen directamente de la cantidad de recursos consumidos. Diseñar un mecanismo de escalamiento automático eficiente representa un mayor margen de beneficio para los operadores de servicios móviles, con el fin de garantizar QoS y la eficiencia de recursos simultáneamente.

Se espera que tecnologías de AI como DRL asuman totalmente las funciones de escalamiento automático para la próxima generación de redes, teniendo a su favor

diversas ventajas en su funcionamiento. Al automatizar la tarea de escalamiento evitan los errores humanos frecuentes en las redes actuales. DRL simplifica los modelos y acelera el proceso de entrenamiento de sus predecesores RL y DL, haciendo más cortos los tiempos de adaptación a la red [21]. De esta manera, DRL soporta algoritmos competentes para redes comerciales de cualquier tamaño.

La necesidad de un mayor ancho de banda y disponibilidad de red, hace necesario un control automatizado que tome decisiones de escalamiento adecuadas en intérvalos de tiempo que un operador humano o un algoritmo básico no pueden lograr. Todo lo anterior conforma una motivación para que nuevas investigaciones como la presentada en este documento, aporten al conocimiento y lleven al proceso de escalamiento a un mejoramiento continuo.

4.2. Arquitectura

Esta sección presenta la arquitectura de SCALA 5G de 4 módulos (ver figura 4.1): Escalamiento, Monitoreo, Ciclo de Vida y *NFV-Based 5G Infrastructure Module* (N5IM). El módulo de Escalamiento contiene el algoritmo DRL para tomar las decisiones de escalar las VNF. El módulo de Ciclo de Vida controla la infraestructura 5G y ejecuta las decisiones provenientes del módulo de escalamiento. El módulo N5IM es la infraestructura basada en 5G/NFV, la cual es la representación realística[1] de un ambiente 5G eMBB y sobre la cual actúa el algoritmo. El Módulo de monitoreo recopila las variables de QoS como latencia, pérdida de paquetes y estado de las VNF dentro de N5IM. Estos módulos son descritos a continuación:

- **Monitoreo**: Recopila cada tiempo t el uso actual de ancho de banda y la latencia a través de la interfaz S5Mi. Calcula la función de recompensa de acuerdo a las variables que provienen de la red 5G eMBB, para luego enviar la recompensa calculada al módulo de escalamiento.
- **Ciclo de Vida**: Revisa el estado actual de las instancias que soportan las VNF de los servicios 5G. Las decisiones de modificar el número de VNF provienen

[1] Un ambiente no comercial con condiciones de operación reales

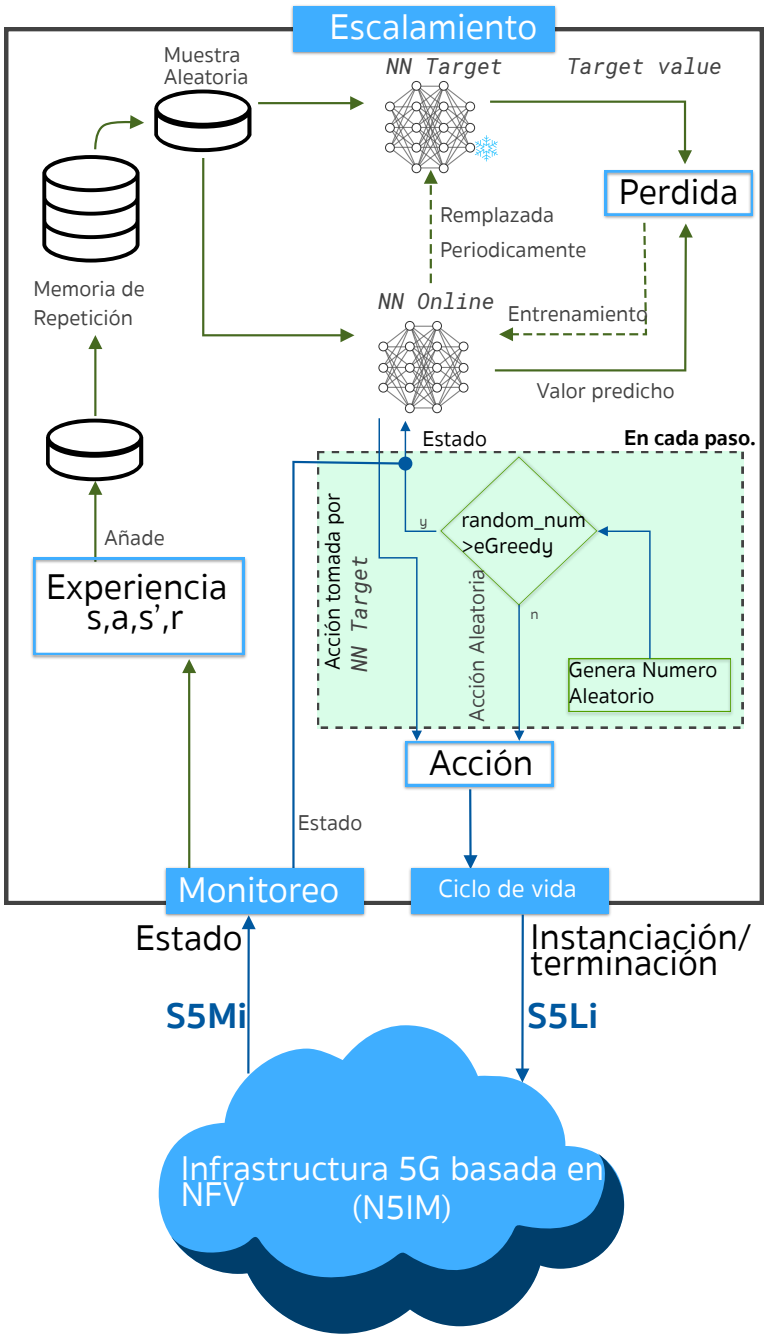


Figura 4.1: Arquitectura SCALA 5G

del módulo de escalamiento. Cuando una orden de escalamiento es recibida el módulo de ciclo de vida, envía la orden a la infraestructura mediante la interfaz

S5Li. Si la decisión es mantener el estado actual este módulo no enviará ningún mensaje a la infraestructura.

- **Escalamiento**: Contiene el algoritmo DRL que toma las decisiones de escalamiento sobre la infraestructura. Al principio, el agente no sabe cómo elegir la acción correcta. Por lo tanto, se esperan decisiones arbitrarias, con el fin de explorar el ambiente. Después de diversos episodios de ensayoy error, el agente converge y su función Deep-Q llega al máximo de experiencia. DQN usa una red neuronal en lugar de una tabla Q estática como la empleada por el RL clásico. Continuamente en cada decisión, el módulo de escalamiento genera una acción y la envía al módulo de ciclo de vida. Posteriormente, consulta con el módulo de monitoreo el estado, la recompensa actual para tomar nuevamente otra decisión y repetir el ciclo.

- **NFVI**: El módulo N5IM aloja el caso de uso de *slicing* para 5G con servicios eMBB estudiado en este trabajo. N5IM implementa servicios del plano de datos de usuario y de tipo transmisión de video de alta definición. Se escogió este tipo de servicio debido a sus altos requerimientos de ancho de banda y latencia, claves en nuestro caso de estudio. La infraestructura suministra la información de los recursos de red a través de la interfaz S5Mi, recibe las órdenes de escalamiento por la interfaz S5LI y ejecuta las órdenes aumentando o disminuyendo las VNF según sea el caso.

4.3. Agente DRL

El agente de DRL creado es útil para aplicaciones de control en tiempo real de diversas áreas. DRL tiene en consideración las características heredadas de RL como los espacios de estados y acciones, la función de recompensa y la política de aprendizaje, desde DL hereda la estructura de red neuronal para la función DQN, el tipo de optimización y las funciones de activación neuronal.

El diseño de nuestra propuesta se centra en el funcionamiento de DRL. Cada vez que llegue una iteración, el agente suma experiencia. La función de recompensa debe

ser precisa a la hora de premiar la labor del agente. La delimitación del espacio de estados y acciones debe ser representativa y de manera simplificada para que el algoritmo converja.

El trabajo de DeepMind [58] es una de las principales aplicaciones recientes de DRL, demostró que el mismo algoritmo es capaz de resolver algunos juegos tipo Arcade de Atari. [59] complementa el trabajo presentado en DeepMind donde un algoritmo DRL optimizado resolvió todos los juegos solamente recibiendo el puntaje como recompensa. Este algoritmo fue tomado como base para nuestra implementación debido a su gran adaptabilidad y precisión en una amplia variedad de aplicaciones sobre redes vehiculares [60], priorización de tráfico 5G [61] y enrutamiento inteligente [62].

Considerando [58] y [59] el agente DRL diseñado contiene 2 redes neuronales denominadas *target* y *online*, además de una memoria de repetición que tambien fue definida previamente en [63]. Esta configuración mejora la estabilidad, y optimiza el proceso de aprendizaje del agente. Las siguientes subsecciones presentan todos los componentes de nuestro algoritmo y su función.

4.3.1. Espacio de estados

El espacio de estados representa todas las posibles posiciones donde el agente observa al ambiente 5G. En nuestra propuesta este estado se obtiene monitoreando todos los recursos disponibles del módulo N5IM. Cada estado está compuesto por la latencia promedio y el número de VNF activas de la siguiente manera:

$$S = S_t\{L, n\} \tag{4.1}$$

Dónde:

L: latencia promedio de las VNF activas

n: número de VNF activas

L actúa como la variable clave e indicadora del QoS actual del servicio. Lo anterior, debido a su alta representatividad y criticidad en los servicios eMBB. Mientras n describe el estado actual de los recursos utilizados. Por ejemplo, si el sistema tiene una latencia de 200ms con 4 VNF activas el estado es representado como $S = S_t\{200, 4\}$.

4.3.2. Espacio de acciones

El espacio de acciones comprende el grupo de todas las decisiones que puede tomar el agente DRL. Las acciones se verán reflejadas sobre la función de plano de usuario 5G UPF, se pueden tomar tres decisiones: Subir, Bajar o Mantener el número de las VNF. El agente inicia tomando estas decisiones de manera aleatoria mientras explora los estados y encuentra relaciones que lo ayuden a aprender y maximizar la recompensa, esta última representa la decisión óptima.

El espacio de acciones tiene ciertos límites en los estados. El número mínimo de VNF activas es 1 (por diseño) o un número máximo N es definido por el administrador de la red.

$$A_t = \{mantener, bajar, subir\} \ \forall \ n \ \epsilon \ (1, N) \tag{4.2}$$

Dónde:

$mantener$: no escalar

$bajar$: reducir 1 instancia

$subir$: aumentar 1 instancia

n: número de VNF activas

N: número máximo de VNF activas

4.3.3. Política de exploración

ϵ-greedy es una política común usada en algoritmos RL y DRL, consiste en organizar una tasa de aprendizaje adecuada para maximizar la recompensa obtenida del entorno, en este caso se obtiene de N5IM. El agente empieza por una fase de exploración total, aquí toma decisiones arbitrarias y busca entender la reacción del sistema. En los agentes de DRL, la NN debe ser entrenada como si fuera una tabla Q de RL básico, con cada iteración el aprendizaje debe ir cayendo hacia 0. En nuestro diseño esta política es de comportamiento lineal. Es decir, la exploración es inversamente proporcional al número de iteraciones trascurridas. La ecuación 4.3 describe este patrón.

$$\epsilon_t = \epsilon_{max} - \frac{\epsilon_{max} - \epsilon_{min}}{T} * t \tag{4.3}$$

Dónde:

T: es el número máximo de episodios de iteración

ϵ_{max}: máximo factor de exploración ó factor inicial

ϵ_{min}: mínimo factor de exploración ó factor final

La ecuación 4.4 muestra los límites del agente sobre la acción que debe tomar. La variable x es un valor aleatorio que cambia en cada iteración t dónde $x \in [0, 1]$. Por lo tanto, el porcentaje de veces que el agente debe seleccionar una acción aleatoria decae a lo largo del número de episodios de entrenamiento.

$$\mathcal{A} = \begin{cases} Q_t(S_t, A_t), & if \quad x > \varepsilon_t \\ aleatorio\, a_t, & Otro\ caso \end{cases} \tag{4.4}$$

4.3.4. Redes neuronales

Nuestro agente DRL usa dos NN para hacer el entrenamiento de manera estable. El propósito de usar esta configuración es no afectar la función de pérdida por el constante cambio de la función DQN en cada iteración. Ambas NN son iguales en estructura, pero realizan tareas complementarias. La NN *online* crea una función de aproximación para valores de acción en el estado actual S_t. Por otra parte la NN *target* proporciona la salida de los valores de acción (Q) para el nuevo estado S_{t+1}. La NN *online* entrena continuamente en cada iteración, mientras que la NN *target* solo actualiza sus pesos cada cierto número de iteraciones. La NN *target* actualiza sus pesos como una copia de los pesos de la NN *online*. La ecuación 4.5 define la ecuación Q utilizada para el cálculo en cada iteración en la NN *online* que depende de la recompensa obtenida sumada al máximo estado siguiente y la ecuación 4.6 de función de pérdida en este caso se define como el valor esperado del cuadrado de la resta de la NN *target* Q y la NN *online* Q^+.

$$Q^+(S_t, a_t) = R_t + \gamma * maxQ(S_{t+1}, A_{t+1}) \tag{4.5}$$

$$L = E[((r + \gamma * maxQ(S_t^{'}, A_t^{'})) - Q(S_t, A_t))^2] \tag{4.6}$$

Las NN *target* y *online* tienen iguales dimensiones en capas de entrada, ocultas y de salida. Todos sus parámetros deben ser iguales para que cada intérvalo de iteraciones C la NN *online* actualice la NN *target* con sus pesos actuales. Al comienzo del aprendizaje, estos pesos son los mismos en ambas NN. La frecuencia de actualización de la NN *target* depende del entorno a controlar. Otra consideración basada en el diseño "Deepmind" es fijar el error entre -1 y 1 para obtener un resultado más estable. Entonces, se minimizan las correlaciones entre los valores de acción y los valores objetivo respecto a las soluciones DRL con una sola NN. La reducción de las correlaciones hace más confiable las actualizaciones en la función Q y que su distribución de datos mejore [59].

4.3.5. Función de recompensa

La recompensa es el valor que se otorga al agente de acuerdo a su desempeño. En nuestro caso, la recompensa se relaciona directamente en términos de latencia del ambiente 5G eMBB. Si el agente toma decisiones que mantengan la latencia dentro de los rangos establecidos la recompensa es máxima. En caso contrario se otorga cero como recompensa al agente.

$$\mathcal{R} = \mathcal{R}_n + \mathcal{R}_{bonus} \tag{4.7}$$

$$\mathcal{R}_n = \begin{cases} 1, & if \quad L_{min} < L < L_{max} \\ 0\,, & Otro\ caso \end{cases} \tag{4.8}$$

$$\mathcal{R}_{bonus} = \begin{cases} 65, & if \quad n > n_{max} - 5 \\ 0\,, & Otro\ caso \end{cases} \tag{4.9}$$

Dónde:

$\mathcal{R}_n$: recompensa por iteración

$\mathcal{R}_{bonus}$: recompensa de bonificación por episodio

L: latencia promedio del servicio

L_{max}: latencia máxima

L_{min}: latencia mínima

n: número de iteraciones alcanzadas por el agente en un episodio

n_{max}: número máximo de iteraciones en un episodio

En la ecuación 4.8, la recompensa $\mathcal{R}_n$ bonifica con 1 punto al agente si este logra mantener la latencia del servicio dentro del SLA establecido el operador de red

en cada iteración. Mientras la ecuación 4.9 $\mathcal{R}_b$ bonifica al agente con 65 si este alcanza almenos 5 iteraciones debajo del máximo en cada episodio. La suma de $\mathcal{R}_n$ y $\mathcal{R}_b$ conforman la recompensa total $\mathcal{R}_n$ en la ecuación 4.7, y fue escogida en pruebas iniciales en las cuales mostró gran precisión, simplicidad y coherencia frente al ambiente 5G eMBB.

4.3.6. Algoritmo de autoescalamiento

En el algoritmo 1 se ejecuta un agente basado en DRL para el autoescalamiento en ambientes 5G eMBB, con el fin de obtener decisiones que cumplan los SLA establecidos para el QoS del servicio. El algoritmo recibe los parametros de entrada: Número de episodios M, parametros de exploración ϵ_{max}, ϵ_{min} y ϵ_{decay}, el factor de descuento γ, el factor de aprendizaje α, la frecuencia de actualización C y el valor de semilla σ. La salida comprende alguna de las 3 opciones incluidas en el espacio de acciones.

El algoritmo 1 inicia con la definición de las NN *online* y *target* (Líneas 1 y 2), dónde ambas redes son dimensionadas (número de capas, número de neuronas y funciones de activación), e inicializadas. Luego empiezan a correr los episodios (Línea 3). Cada vez que inicia un episodio el ambiente se reinicia (Línea 4). De la línea 5 a la 20 se cuenta cada iteración del agente. Al inicio de cada iteración, el agente observa en que estado se encuentra el ambiente (Línea 6). Luego se genera un número aleatorio con el fin de determinar si el agente va a explorar el ambiente o si va a tomar la salida de la NN (Línea 7), si el número generado es menor al ϵ-greedy actual (Línea 8), el agente toma una decisión aleatoria sin importar la lógica de ésta (Línea 9). Si el número es mayor (Línea 11), el agente toma la salida de la NN (Línea 12).

La acción tomada previamente es enviada al módulo de ciclo de vida (Línea 14) para que la ejecute. Luego se obtiene la recompensa desde el módulo de monitoreo con la ecuación 4.7, depende directamente de la latencia (Línea 15). Despues, se almacenan los parámetros actuales (estado, acción, recompensa y estado siguiente) (Línea 16) con el fín de alimentar la memoria del agente.

En la última parte de la iteración, se optimiza la NN *online* de acuerdo al error

respecto a la NN *target* (Línea 17). Esta optimización se realiza con la función de pérdida descrita en la ecuación 4.6. Si la iteración ha cumplido el periodo de C pasos entonces la NN *Target* es actualizada con los valores de la NN *online* (Línea 18). Finalmente el algoritmo pasa a la siguiente iteración (Línea 19).

Algoritmo 1: Auto-escalamiento basado en aprendizaje por refuerzo profundo SCALA 5G

Entrada : Número de episodios: M
Parámetros de exploración: $\epsilon_{max}, \epsilon_{min}, \epsilon_{decay}$
Factor de descuento: γ
Factor de aprendizaje: α
Frecuencia de actualización: C
Valor de semilla: σ

Resultado: Decisión de escalamiento sobre el ambiente

Inicializar NN *online* Q con pesos aleatorios
Inicializar NN *target* Q^* con pesos $\theta^- = \theta$
for *episodio = 1* **to** M **do**
    Reiniciar ambiente
    while *no sea el estado final* **do**
        El agente observa el estado inicial S_i
        probabilidad = random[0,1]
        if *probabilidad* $< \epsilon$ **then**
            $a_t = random([A_t\{\text{subir, bajar, mantener}\})$
        end
        else
            $a_t = NNQ(S_t))$
        end
        Envia accion a_t a ser ejecutada por el módulo de ciclo de vida
        Obtiene la recompensa desde el módulo de monitoreo (ver ecuación 4.7)
        Almancenar transición (s_t, a_t, r_t, s_{t+1})
        Optimización de la NN $Q(S_t)$ con los valores de pérdida $Q^*(S_{t+1})$ ver ecuación 4.6.
        $Q^* = Q$ cada C pasos la NN *target* copia los pesos de la NN *online*
        $s_t \leftarrow s_{t+1}$ Se actualiza el estado actual
    end
end

Capítulo 5

Evaluación

En este capítulo se muestra la definición del ambiente usado, cómo se realizó el entrenamiento del agente y las tecnologías implicadas en este proceso. Se establecen las métricas y pruebas para evaluar al agente y finalmente se realiza un análisis de los resultados obtenidos.

5.1. Ambiente de pruebas

5.1.1. Infraestructura

La infraestructura del ambiente de pruebas corresponde al caso de uso de eMBB, se puede observar el grafo que contiene su estructura lógica en la figura 5.1. El *Access and Mobility Management Function* AMF se encarga de recibir peticiones del equipo de usuario, el *Session Management Function* SMF es responsable de crear y remover las Unidades de protocolo de datos y administrar las sesiones con el UPF y el *User Plane Function* UPF representa la evolución del plano de datos [64].

La figura 5.2 es una variante de la figura 5.1 contiene la organización a nivel físico en el clúster de *Kubernetes* de los *Kubernetes-deployments* para un eMBB compuesto por los VNF: UPF, SMF, AMF, MARTIR y balanceadores de carga. A nivel físico, se

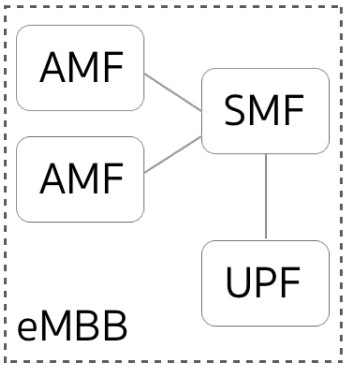


Figura 5.1: Grafo de caso de uso eMBB.

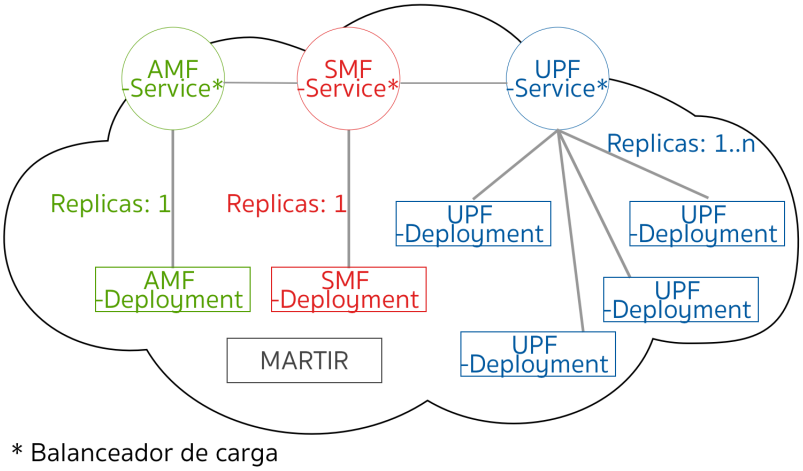


Figura 5.2: Infraestructura de Kubernetes

despliega el clúster compuesto por cuatro nodos. *Kubernetes* instancia cada réplica en el nodo que tenga recursos físicos para correrlo. Cada nodo puede albergar distintos tipos de réplicas. Con la configuración definida en `https://gitlab.com/SCALA5G/Kubernetes-infra` se pueden desplegar hasta 23 replicas distribuidas de la siguiente manera: 1 réplica para SMF, 1 réplica para AMF, de 1 a 20 réplicas para UPF y 1 réplica para la instancia *MARTIR*, encargada de obtener telemetría. Cada nodo es tipo N: 2 CPUs/8 GB de RAM. Este clúster fue desplegado en la nube pública de *Google Cloud*.

Se elige *Kubernetes* como gestor de la infraestructura, ya que permite de manera

simple administrar sus componentes y realizar su despliegue. Por ejemplo, para construir toda la infraestructura que contiene las VNF que componen el servicio eMBB solo se debe ejecutar el comando *kubectl apply -f .*

Cada *Kubernetes-deployment* cuenta con su propio *Kubernetes-service* encargado de tareas como balanceo de cargas. Además, cada *Kubernetes-deployment* corre con el software correspondiente a su VNF, por ejemplo el *"upf-deployment"* ejecuta el software "SCALA 5G upf:latest". El código fuente del software se puede consultar en `https://gitlab.com/SCALA5G/upf`, `https://gitlab.com/SCALA5G/amf` y `https://gitlab.com/SCALA5G/smf`, cada uno de estos componentes de software están escritos en TypeScript [65] y corren un servidor en NodeJS [66]. Se escogió NodeJS debido a su amplio soporte por parte de la comunidad y la facilidad para crear API-REST [67]. TypeScript se escogió porque obliga al uso de variables fuertemente tipadas y de esta manera se pueden depurar errores en tiempo de compilación y no en la ejecución [68]. Por ejemplo, si se tiene una función que acepte una variable de tipo booleano, en JavaScript se puede pasar como argumento de la función una cadena de texto. Por el contrario, TypeScript obliga a pasar un booleano como argumento, si se pasa una cadena de texto, se produce un error de compilación. Además, TypeScript cuenta con soporte oficial de Microsoft y tiene características de Javascript, ya que en tiempo de compilación el código de TypeScript es transformado a JavaScript.

Los balanceadores de cargas utilizan una configuración *round-robin* que es una técnica de balanceo de carga para asignar la misma prioridad a cada servidor o instancia [69]. Por ejemplo, si un balanceador de carga tiene asignado las instancias A, B y C, el balanceador de carga envía la primera petición a la instancia A, la segunda petición a la instancia B y la tercera petición a la instancia C, en este momento todas las instancias atendieron una petición. Cuando llega la cuarta petición, de nuevo la instancia A se encarga de atenderla y así se repite el ciclo.

Del grafo de servicio eMBB, el elemento a escalar es la VNF UPF. *Kubernetes* permite realizar un escalamiento simple por medio de su API, y automáticamente añade o remueve la instancia del UPF y actualiza las IP internas de los balanceadores de carga, para que estos redirijan el tráfico correctamente a la nueva instancia del

UPF.

Nuestro ambiente de pruebas, utiliza la infraestuctura de *Google Cloud Platform* (GCP) para el manejo del hardware que soporta la NFVI [70]. La nube GCP fue escogida debido a su optimización para contenedores y *Kubernetes*, y diversidad de herramientas *DevOps* [71].

5.1.2. Servicio

El ambiente eMBB presentado es un servicio de video de alta definición el cual presenta una duración de 150ms para su descarga y puesta en marcha. Bajo este criterio, un usuario puede acceder vía web a la red móvil y solicitar a la red 5G ser enrutado hacia el servicio de video. La red proporciona el flujo típico basado en las recomendaciones de la ETSI y 3GPP para carga y descarga de datos por medio de la UPF. El diagrama de secuencia de la figura 5.3 está basado específicamente en el 3GPP T23.502 [72] para establecimiento básico de sesión PDU 5G.

En el ambiente de pruebas construido se simplificó el proceso real de un flujo del caso de uso eMBB 5G. A continuacion, la figura 5.3 representa un diagrama de secuencia para el caso de uso eMBB simplificado entre internet, el equipo de usuario y el software de las VNF que componen el eMBB: UPF, AMF y SMF.

En la figura 5.3 en primer lugar el equipo de usuario realiza una petición POST a la dirección del balanceador de carga del AMF `http://amf-service-lb-ip/sessionrequest` en donde envía su MSISDN, este se comunica con el SMF en `http://smf-service-lb-ip/createpdu` reenviando el MSISDN del usuario para crear el *PDU*. El SMF genera la sesión y el identificador de sesión es enviado como respuesta al AMF el cual lo reenvia al equipo de usuario, comunicando que la sesión fue establecida.

En este punto, el equipo de usuario puede realizar la petición GET a la instancia UPF enviando su *sessionid* para realizar el *streaming* de video en `http://upf-service-lb-ip/stream`.

Para implementar el flujo de la figura 5.3, primero se realiza el despliegue de la

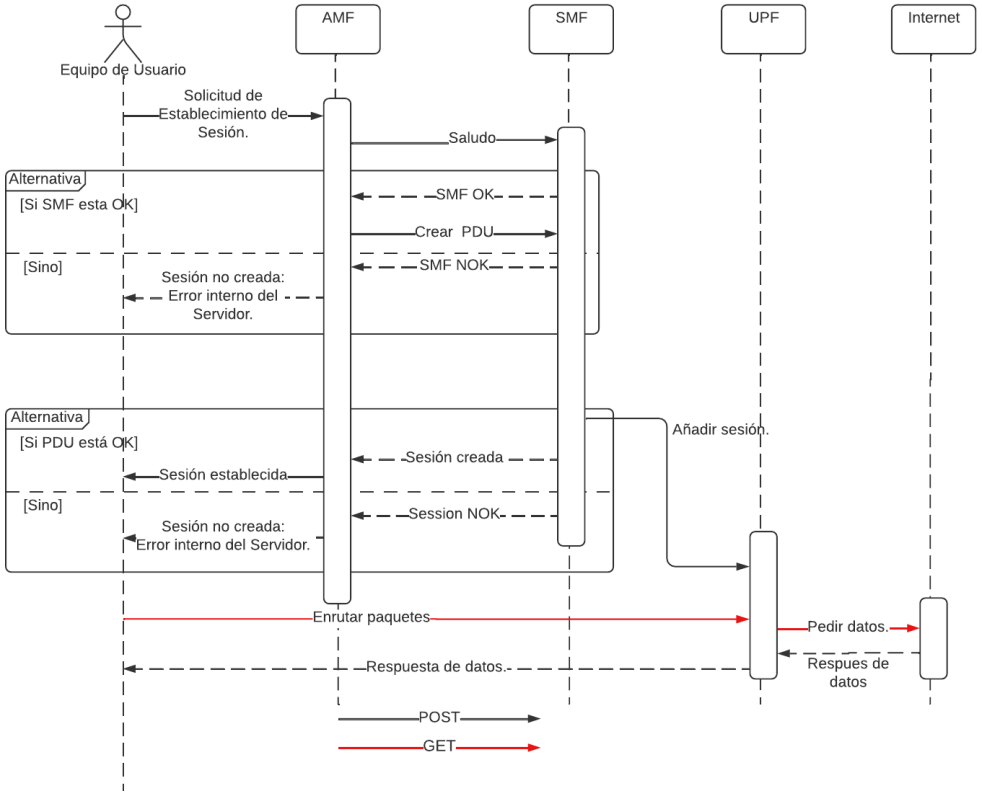


Figura 5.3: Diagrama de secuencia del caso de uso eMBB simplificado, entre la infrastructura y el usuario final

infraestructura de Kubernetes. La infraestructura de la figura 5.2 está definida por código, esto permite desplegar la figura 5.2 con solo el comando *kubectl apply -f .* , cada *Kubernetes-deployment* cuenta con una IP interna, lo que permite la comunicación de red entre las instancias de VNF, sin embargo, están aisladas de Internet. Es por esto que se despliega un balanceador de carga utilizando un *Kubernetes-service*, el cual tiene una IP estática externa que permite la comunicación con internet.

Las peticiones POST/GET mencionadas se realizan a los *kubernetes-services* que actuan como balanceadores de carga, las direcciones ip de upf-service-lb-ip, smf-service-lb-ip y amf-service-lb-ip son direcciones IP externas, para obtenerlas se puede utilizar el comando *kubectl get service –all-namespaces*.

5.1.3. Tráfico de prueba

El tráfico usado para el entrenamiento y evaluación de SCALA 5G es creado de manera sintética, teniendo en cuenta noventa patrones de tráfico comerciales obtenidos de operadores de telecomunicaciones en Colombia para un segmento de red ubicado en los alrededores de la ciudad de Barranquilla. Como muestra de tres de los noventa patrones de tráfico, en la figura 5.5(a) se muestra el patrón de tráfico de plano de usuario correspondiente al 7 de Enero del 2022, la figura 5.5(b) del día 13 de Marzo del 2022 y la figura 5.5(c) es el patrón de tráfico del día 24 de Marzo del 2022 el cual jugó la selección masculina de fútbol colombiano, en el que se puede observar un pico de tráfico al inicio del partido en el DecaMinuto 110, por el contrario los traficos 5.5(b) y 5.5(a) tienen pico de tráfico en el DecaMinuto 120.

Cada patrón de tráfico real fue multiplicado por un factor de ruido generado con una distribución de probabilidad uniforme, este proceso se puede encontrar en `https://gitlab.com/SCALA5G/cartpole/-/blob/graphs/plotting/real_data_plot.py` . Esto se hizo por materia de seguridad y privacidad de la información de los usuarios y la empresa de la que se obtuvieron los datos. Por lo cual no se puede proporcionar mayor detalle.

Los tres patrones de tráfico mencionados tienen un comportamiento similar. No obstante, el proposito de SCALA 5G es generalizar el comportamiento del tráfico para tener un correcto desempeño en la mayor cantidad de escenarios posibles. La ecuación 5.1 puede generar un número infinito de patrones de tráfico diferentes, lo cual es útil para que el agente no correlacione la decisión de salida con un solo patrón de tráfico. Los intervalos para las cinco constantes a, a_1, a_2, p_1 y p_2 se escogieron después de un proceso empírico de experimentación.

$$a_1 \sin \frac{t}{p_1} + a_2 \sin \frac{t}{p_2} + a \tag{5.1}$$

Dónde:

a_1: amplitud $1 \; \forall \; \mathbb{R} \; \epsilon \; (1, 90)$

a_2: amplitud 2 $\forall$ $\mathbb{R}$ ϵ $(1, 90)$

p_1: periodo 1 $\forall$ $\mathbb{R}$ ϵ $(2, 11)$

p_2: periodo 2 $\forall$ $\mathbb{R}$ ϵ $(2, 11)$

t: tiempo $\forall$ $\mathbb{R}$ ϵ $(0, \infty)$

a: amplitud $\forall$ $\mathbb{R}$ ϵ $(1, 180)$

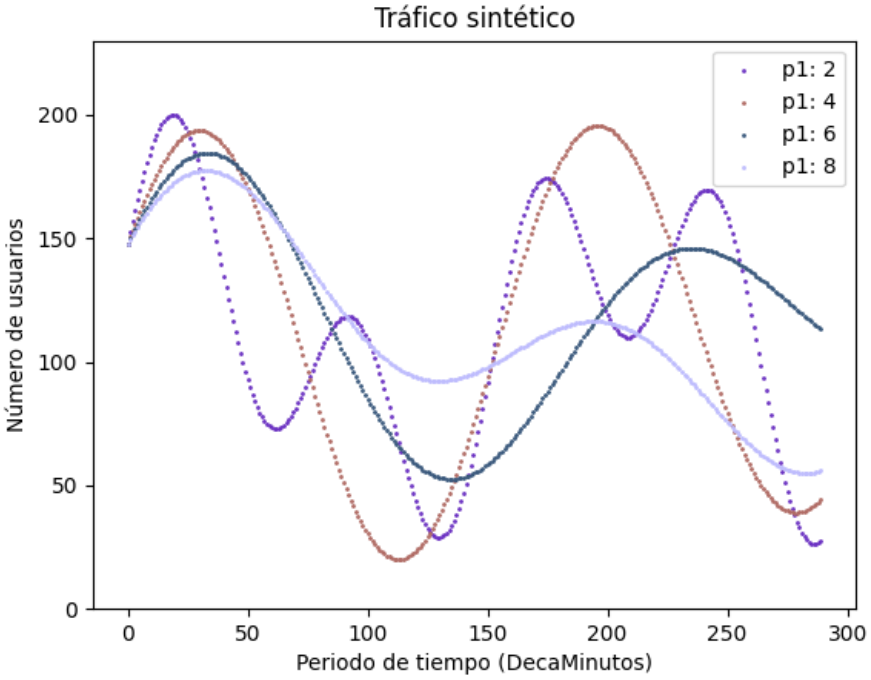
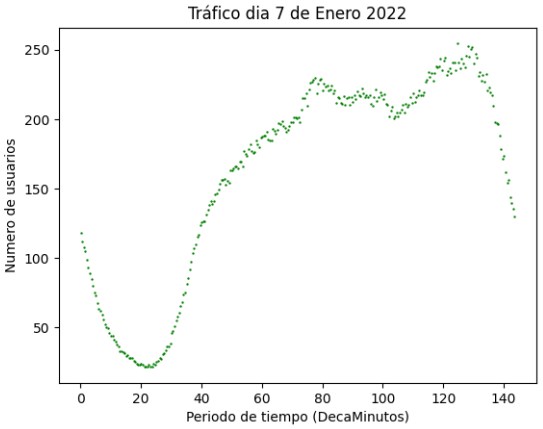


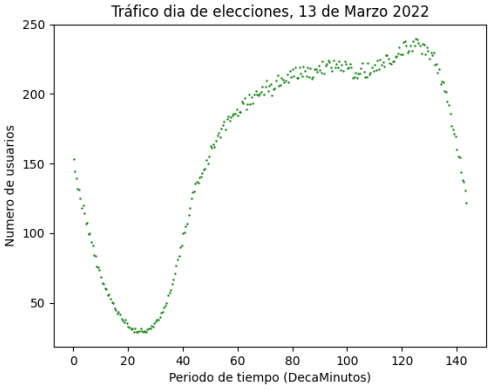
Figura 5.4: Tráfico generado por la ecuación 5.1

Para generar una función que origina un patrón de tráfico las constantes a, a_1, a_2, p_1 y p_2 se producen de manera aleatoria siendo t la variable independendiente.

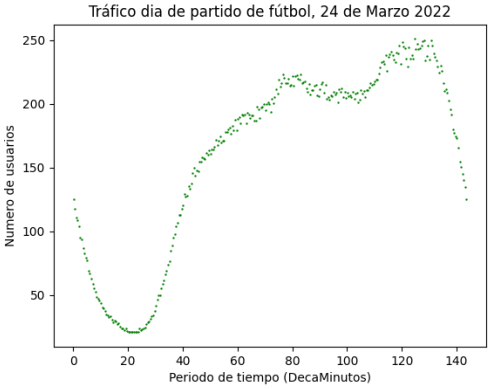
La figura 5.4 presenta cuatro patrones de tráfico sintético (originado de los reales) al utilizar cuatro valores de p_1 diferentes con las constantes $a = 60$, $a_1 = 45$, $a_2 = 45$, $p_2 = 5$ y la variable independiente en el rango $t = 0$–290.



(a) Tráfico real del día 7 de Enero del 2022, para un segmento de red en Barranquilla, Colombia.



(b) Tráfico real del día 13 de Marzo del 2022, para un segmento de red en Barranquilla, Colombia.



(c) Tráfico real del día 24 de Marzo 2022 para un segmento de red en Barranquilla, Colombia.

Figura 5.5: Patrones de tráfico reales

5.1.4. Retardo de una instancia.

El ambiente de pruebas debe ser realístico, en nuestro caso el retardo de la respuesta de una instancia de la infraestructura desplegada en GCP para un número de usuarios determinado es la variable que se modeló en la ecuación 5.2. El retardo o latencia es el tiempo que se tarda desde la solicitud de establecimiento de sesión hasta la respuesta de datos en el diagrama de secuencia 5.3.

$$l = \frac{e^{-1+\frac{u+4}{7.1}} - e^{6-\frac{u}{25}}}{2} + 300 \tag{5.2}$$

Dónde:

l: retardo, en milisegundos

u: número de usuarios por instancia.

El número de usuarios por instancia es calculado con la ecuación 5.3. Como se mencionó en la subsección *Infraestructura*, los balanceadores de carga están configurados en modo *round-robin* y por tanto cada instancia recibe el mismo número de usuarios.

$$u = \frac{u_t}{n_t} \tag{5.3}$$

Dónde:

u: número de usuarios por instancia.

u_t: número de usuarios totales.

n_t: número de instancias totales.

La caracterización del ambiente para obtener la ecuación 5.2 se hizo a partir de la experimentación y los datos de tiempo de retardo de una instancia se tomaron utilizando la herramienta `https://loader.io/` que permite realizar pruebas de estrés

y tráfico a la infraestructura de *Kubernetes* desplegada en GCP. En la figura 5.6 se muestra la relación entre el número de usuarios y el tiempo de retardo de una instancia.

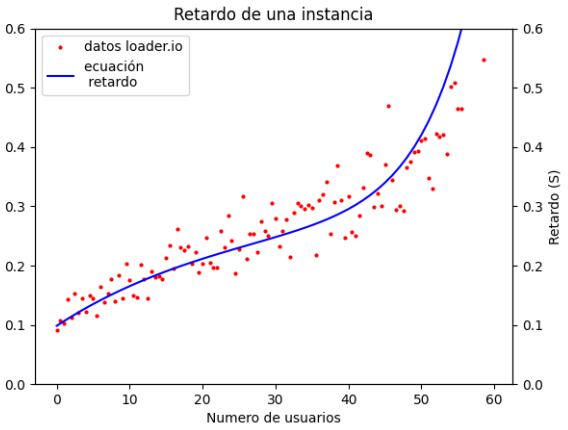


Figura 5.6: Retardo de una instancia respecto al número de usuarios.

5.1.5. Gimnasio de SCALA 5G

Para realizar el entrenamiento de SCALA 5G y la posterior evaluación, se construyó un gimnasio [73], el cual es un ambiente sintético de pruebas que SCALA 5G usa para interactuar y entrenar.

El estado S_t del gimnasio está compuesto por el retardo l y el número total de instancias activas n_t según la ecuación 4.1. El estado S_t es modificado por una acción a_t, el retardo l es calculado utilizando la ecuación 5.2 y depende del número de usuarios totales u_t y el número de instancias totales n_t. SCALA 5G sólo puede modificar el número total de instancias n_t.

La razón por la cual el estado S_t del gimnasio no tiene en cuenta parámetros como la CPU/RAM, es porque GCP y AWS, pueden tener diferentes tipos de arreglos de CPU/RAM para sus nodos, de esta manera SCALA 5G es agnóstico a la plataforma sobre la cual se despliega.

5.1.6. Entrenamiento de SCALA 5G.

SCALA 5G es un agente DQN. Para realizar su entrenamiento se debe producir la interacción entre SCALA 5G y el gimnasio de la subsección 5.1.5.

La recompensa se calcula teniendo en cuenta el SLA según la ecuación 4.7. En cada iteración la función de pérdida calcula la pérdida utilizando los valores de S_t, S_{t+1}, r y a_t que son producto de la interacción entre SCALA 5G y el gimnasio, de esta manera SCALA 5G determina si las acciones que está tomando maximizan la recompensa r. Si las acciones son acertadas los pesos de las redes neuronales de SCALA 5G se modifican ligeramente. Si son erróneas, los pesos sufren una modificación. Para el entrenamiento de SCALA 5G se estableció de manera empírica un SLA de $L_{max} = 320ms$ y $L_{min} = 200ms$, estos valores los puede determinar el usuario administrador teniendo en cuenta la gráfica 5.6 y los requerimientos SLA de su red.

Al iniciar el entrenamiento, SCALA 5G recopila el estado S_t y toma una acción a_t entre: escalar el número de instancias, reducir el numero de instancias ó mantener el numero de instancias, la acción a_t es enviada al gimnasio. El gimnasio produce un estado S_{t+1} y una recompensa r como respuesta, SCALA 5G utiliza S_t, S_{t+1}, r y a_t para aprender. En cada episodio se utilizan patrones de tráfico de 48 horas es decir, 288 pasos. El entrenamiento finaliza cuando la condición de la ecuación 5.4 se cumple, es decir, cuando el promedio de pasos de los últimos 14 episodios es superior a 283. Los datos de entrenamiento son sintéticos y se generan con la ecuación 5.1. Esta ecuación permite generar un patrón de tráfico de diferente longitud, cada patrón de tráfico se divide en intervalos de 10 minutos. Se construyeron dos gimnasios: el gimnasio SCALA 5G y el gimnasio legado. El gimnasio SCALA 5G utiliza patrones de tráfico de 48 horas es decir 288 intervalos y el gimnasio legado de 24 horas es decir de 144 intervalos.

$$p_{av14} > 283 \tag{5.4}$$

Dónde:

p_{av14}: promedio de pasos de los últimos 14 episodios.

Se recuerda que el SLA para el ambiente es $L_{max} = 320ms$ y $L_{min} = 200ms$.

El retardo l puede tomar el rango de valores de 5.5 durante el entrenamiento:

$$L_{min} * 0.85 < l < L_{max} * 1.15 \tag{5.5}$$

Dónde:

l: retardo $\forall$ $\mathbb{R}$ ϵ $(150, 400)$

Un episodio finaliza si SCALA 5G recorre todo el patrón de tráfico, es decir, llega al paso 288 ó si se obtienen retardos que superen los 400 milisegundos o sean inferiores a 150 milisegundos.

Por tanto, si SCALA 5G supera el paso 283 en un episodio significa que escogió las acciones acertadas. Si el episodio termina pronto, se debe a que SCALA 5G tomó decisiones erróneas, por ejemplo, no aumentó el número de servidores ante un máximo de tráfico y esto generó un retardo superior a 400 milisegundos. Terminar el episodio cuando el agente empieza a tomar decisiones erróneas de manera continua permite que el agente no refuerce la toma de acciones equivocadas. Para potenciar la toma continua de decisiones acertadas, se da una recompensa extra de 65 (valor ajustado empiricamente) si SCALA 5G llega al paso 283.

Definir el valor de la recompensa extra o el ajuste de hiperparámetros de 5.1 es una tarea que puede tornarse díficil para las redes neuronales complejas y es un proceso que es realizado por medio de la experimentación [74].

Estrategias similares a terminar el episodio si el agente toma decisiones continuas de manera equivocada, bonificar la toma continua de decisiones acertadas y utilizar un gimnasio como ambiente de entrenamiento son aplicadas en problemas de DRL [75] [76] ampliamente conocidos como *cart-pole* o *climb-mountain*. Los gimnasios son utilizados para entrenar algoritmos en ambientes DRL, el gimnasio utilizado se extiende de la libreria *OpenAI Gym* de Elon Musk [77].

	SCALA 5G	Agente Legado
Número de Neuronas	128	64
Recompensa de Bonificación	65	30
Memory Replay	Si	No
NN Target	Si	No
Número de episodios	Dinámico	1000
Pasos por episodios	288	144
Learning Rate	0.007	0.007
Dimensión de entrada a la red Neuronal	2	2
Dimensión de salida a la red Neuronal	3	3
Frecuencia de actualización de NN Target	100	-
Factor γ	0.9	0.9

Cuadro 5.1: Parámetros de SCALA 5G y el Agente Legado

Para la evaluación del algoritmo SCALA 5G se usa como referencia un algoritmo de DRL clásico, denominado Agente Legado. Este agente interactúa con el gimnasio legado para su entrenamiento, las características del Agente Legado y SCALA 5G se pueden encontrar en el cuadro 5.1.

Para el entrenamiento del algoritmo se usan 2 máquinas locales con procesadores Intel Core i7-6700HQ e Intel Core i7-10700K. El Agente DQN fue escrito en Python 3.9 [78], para construir las redes neuronales fue usada la libreria PyTorch [79]. El código fuente del algoritmo construido está disponible en el repositorio `https://gitlab.com/SCALA5G/cartpole`.

5.1.7. Métricas

La definición de métricas es importante para garantizar el cumplimiento de los objetivos planteados en el proyecto. Así, SCALA 5G se evalúa en términos de tiempo de convergencia, precisión y adaptabilidad.

- **Tiempo de convergencia**: determina el tiempo desde que el algoritmo em-

pieza hasta que finaliza su aprendizaje, se evalúa con la métrica que lleva su mismo nombre.

- **Precisión**: este aspecto respalda la correcta operación de SCALA 5G y esta asociada a la métrica de **satisfacción de nivel de servicio**.
- **Adaptabilidad**: la evaluación de este aspecto determina que tan viable es desplegar SCALA 5G en diferentes proveedores de nube. Las métricas que evaluan este aspecto son **satisfacción de nivel de servicio** y **cobertura de pruebas unitarias**

Las métricas mencionadas son descritas a continuación:

5.1.7.1. Tiempo de convergencia

El tiempo de convergencia *Convergence Time* (CT) mide el número de episodios que toma el agente DRL para llegar a la fase final de aprendizaje. Cada episodio tiene un tiempo de ejecución t_i que depende del entorno de ejecución (procesador, memoria, lenguaje de programación).

$$CT = \sum_{i=0}^{i_{ct}} 1 = i_{ct} \tag{5.6}$$

$$CT(t) = \sum_{i=0}^{i_{ct}} t_i \tag{5.7}$$

Dónde:

i_{ct}: número de episodios para alcanzar la convergencia del agente

t_i: tiempo transcurrido en un episodio i

La ecuación 5.6 muestra el tiempo de convergencia calculado en número de episodios y la ecuación 5.7 en función del tiempo. Como objetivo se establece que SCALA 5G

debe converger en menos de 5000 episodios y en términos de tiempo se considera que converge de manera rápida si la duración total es menor a 10 minutos. Agentes que resuelven problemas similares como *cart-pole* convergen en la misma orden de magnitud de tiempo y número de episodios [75] [76].

5.1.7.2. Satisfacción de nivel de servicio

La segunda métrica es $\%SLA$ se refiere al porcentaje de peticiones de usuario atendidas dentro del SLA establecido.

$$\%SLA = \frac{R_+}{R_{total}} = \frac{\sum_{i=0}^{n} R+_i}{\sum_{i=0}^{n} Rtotal_i} \tag{5.8}$$

Dónde:

R_+: número de peticiones de usuario atendidas dentro del SLA.

R_{total}: número de peticiones de usuario atendidas satisfactoriamente.

Un SLA de 1.0 es el mejor escenario posible y ocurre al atender todas las peticiones de usuario dentro del SLA establecido. Por ejemplo si se realizan 100 peticiones en total y las 100 peticiones son atendidas en el SLA. Por el contrario, un SLA de 0.0 es el peor escenario posible, en este caso, ninguna petición es atendida en el SLA establecido. Para nuestro agente, el mantener la latencia entre los rangos L_{max} y L_{min} asignados por el operador lo hace cumplir el ratio de satisfacción.

5.1.7.3. Cobertura de pruebas unitarias

La tercera métrica utilizada para evaluar nuestro algoritmo es la prueba de cubrimiento *coverage* de pruebas unitarias [80]. El *coverage* es una medida de cuantas lineas de código son ejecutadas cuando las pruebas unitarias[1] están corriendo. Como ejem-

[1]método de prueba de una fracción de código para determinar si cumple con el propósito que fue creada

plo de cobertura, si un archivo está compuesto por 100 líneas de código, y las pruebas unitarias interactúan con 90 líneas, se tendrá una cobertura del 90%. Por tanto, la mayor parte del código de ese archivo estará protegido por pruebas unitarias. Si en el futuro un colaborador introduce un error en el archivo, las pruebas unitarias fallarán y los desarrolladores podrán saber que el código tiene un comportamiento inesperado y corregirlo.

El *coverage* permite garantizar la calidad del código de programación realizado para que en trabajos futuros académicos y comerciales lo puedan replicar, modificar o usar módulos específicos de la solución con facilidad. Las pruebas unitarias documentan la intención original del código y aseguran que futuras modificaciones del mismo no dañen el sistema. De acuerdo con [81] una medición buena de cubrimiento debería estar por encima del 70%, y se recomienda pasar de 80% para *software* de propósito comercial.

$$\%C = \frac{C_{+}}{C_{total}} \tag{5.9}$$

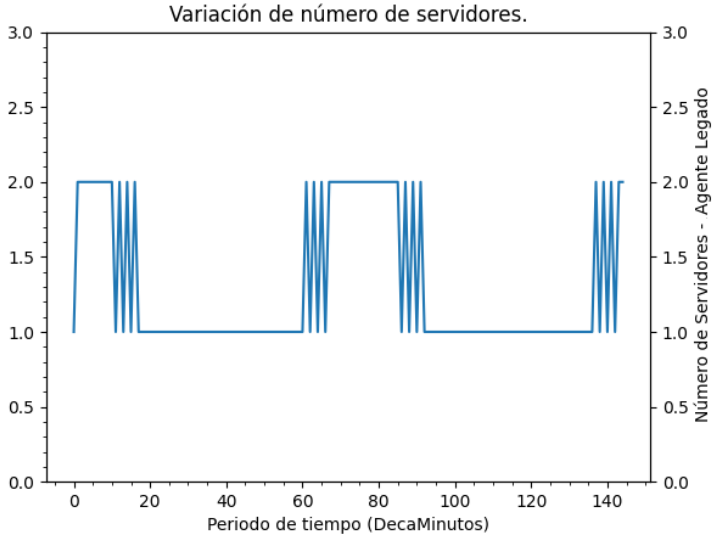
Dónde:

C_{+}: número de líneas de código que se ejecutan cuando las pruebas unitarias corren.

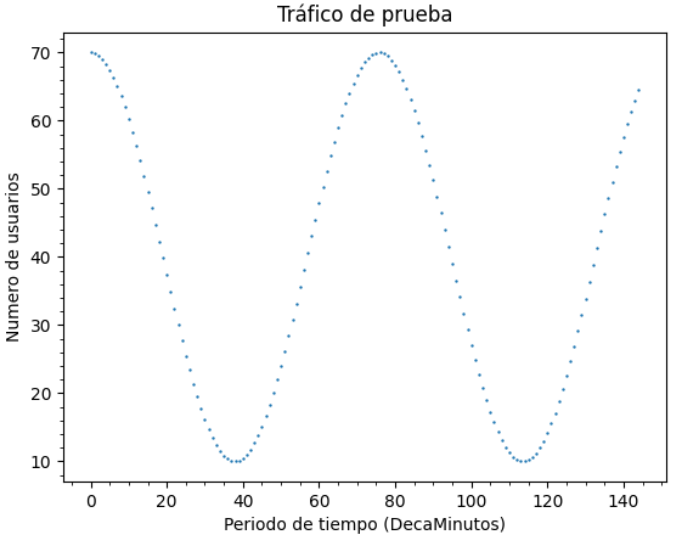
C_{total}: número de líneas de código totales.

5.1.8. Resultados iniciales: Agente Legado

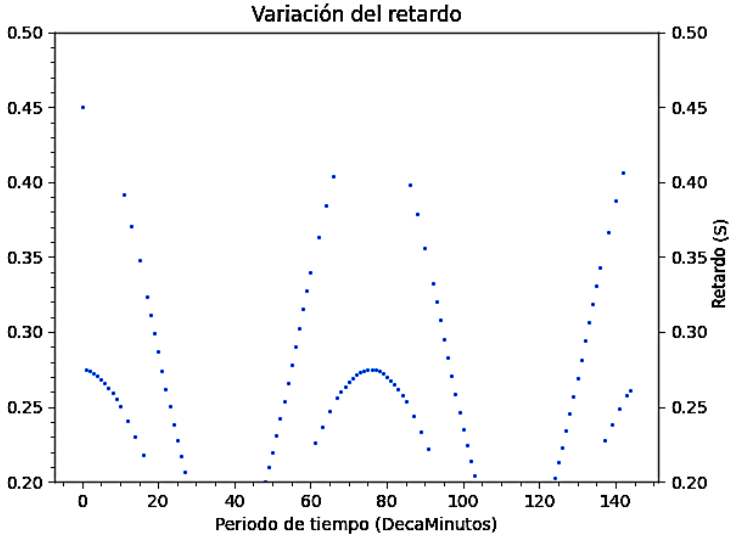
En la figura 5.7(a) se puede observar la variación de servidores para el tráfico de la figura 5.7(b), las acciones son tomadas por el Agente Legado. El patrón de tráfico sintético utilizado para realizar esta evaluación es generado con la ecuación 5.1. Los hiperparámetros utilizados para este agente son presentados en la tabla 5.1.



(a) Variación del número de servidores Agente Legado.



(b) Tráfico de prueba Agente Legado.



(c) Retardo Agente Legado.

Figura 5.7: Agente Legado

En la figura 5.7(a) se pueden observar rebotes causados en la cercanía a los límites de toma de decisión. Se considera rebote cuando el agente escala el número de servidores y en los próximos 3 decaminutos o menos reduce el número de servidores o viceversa. Además, en la figura 5.7(c) se puede observar como continuamente se sobrepasan los intervalos del SLA. En el decaminuto 66 por ejemplo, se observa como el retardo es de 400 ms, entre los decaminutos 60 y 70 vemos como el retardo se sobrepone, esto es por los rebotes que se produjeron. Pese a tener margen de mejora, el Agente Legado es capaz de tomar acciones de manera autónoma, uno de los objetivos de este trabajo.

En la siguiente sección, el Agente Legado se compara con SCALA 5G, que es la versión final del Agente presentado en este trabajo de grado. Se describen a detalle las mejoras realizadas al Agente Legado para obtener SCALA 5G.

5.2. Resultados y Análisis: Agente SCALA 5G

Para la realización de este análisis, se establecen como métricas las previamente mencionadas en la sección 5.1.7. SCALA 5G opera con los hiperparámetros de la tabla 5.1. Estos valores fueron obtenidos en base a otras soluciones DRL de problemas como *cart-pole* y *climb-mountain* y modificadas cuidadosamente de manera empírica para alcanzar los valores finales.

5.2.1. Tiempo de convergencia

Se considera el número de pasos por episodio como métrica de convergencia porque el gimnasio termina un episodio si el Agente con el que interactúa selecciona acciones equivocadas de manera consecutiva. Entonces, un número elevado de pasos por episodio representa que el agente está tomando decisiones acertadas. SCALA 5G tiene un número variable de episodios, el entrenamiento finaliza cuando la condición de la ecuación 5.4 se cumple. El Agente Legado tiene un número fijo de 1000 episodios de entrenamiento. La tabla 5.2 y la figura 5.8 presentan la convergencia de SCALA 5G y el Agente Legado. En la figura 5.8 para cada episodio, se asigna el

valor de 1.0 si el agente recorre todos los pasos posibles, 288 pasos para SCALA 5G y -1.0 si el agente no lo logra.

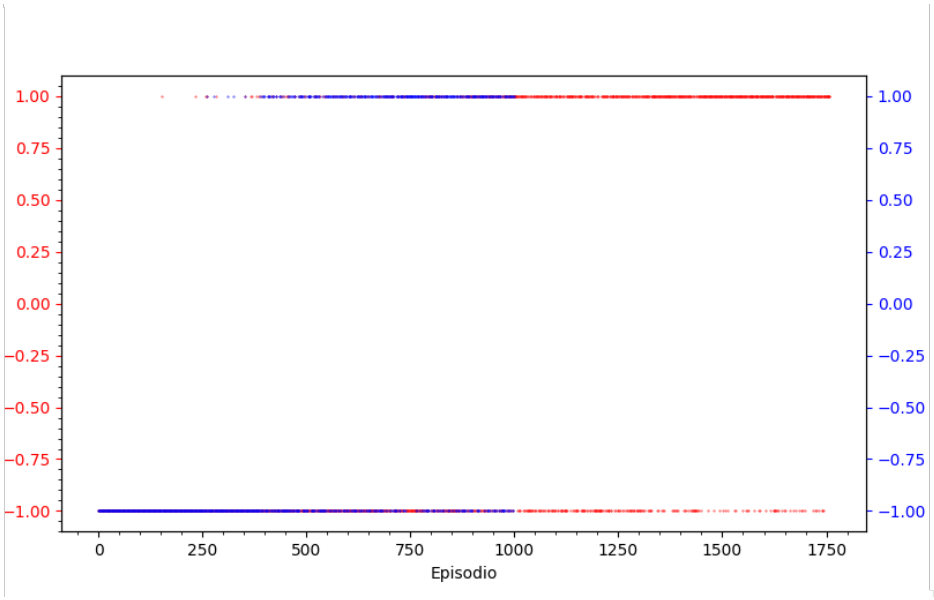


Figura 5.8: Convergencia SCALA 5G y Agente Legado.

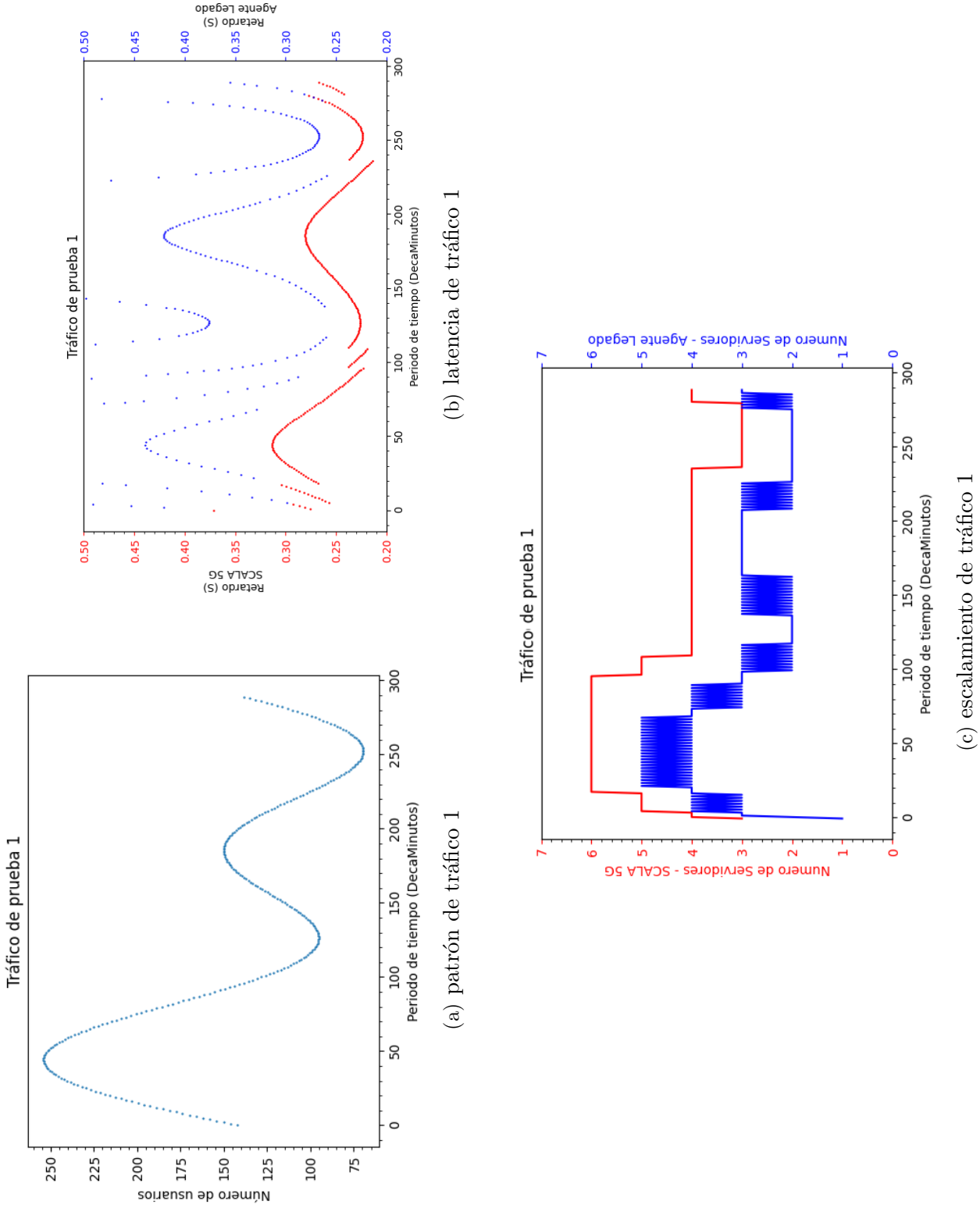
	SCALA 5G	Agente Legado
Episodios.	1740	1000
Pasos posibles	288	144
Tiempo de convergencia	3 minutos y 27 segundos	1 minuto y 2 segundos

Cuadro 5.2: Convergencia SCALA 5G y Agente Legado.

SCALA 5G utiliza más tiempo que el Agente Legado durante su entrenamiento esto se debe a que SCALA 5G tiene un número de episodios dínamico. No obstante, SCALA 5G tiene un mejor comportamiento en todos los patrones de tráfico probados (ver sección 5.2.2). Por tanto, el tiempo adicional que SCALA 5G tarda en entrenar se traduce en mejores resultados. Resaltar que SCALA 5G converge de manera rápida ya que tarda menos de 5 minutos, siendo menor al límite establecido en la sección 5.1.7.1.

5.2.2. Satisfacción de nivel de servicio

Para evaluar el desempeño de SCALA 5G en términos de cumplimiento del SLA, se realizó en primera instancia la comparación con un algoritmo DRL básico: el Agente Legado. Para esta comparación, se probaron cuatro diferentes funciones de tráfico generadas por la ecuación 5.1.

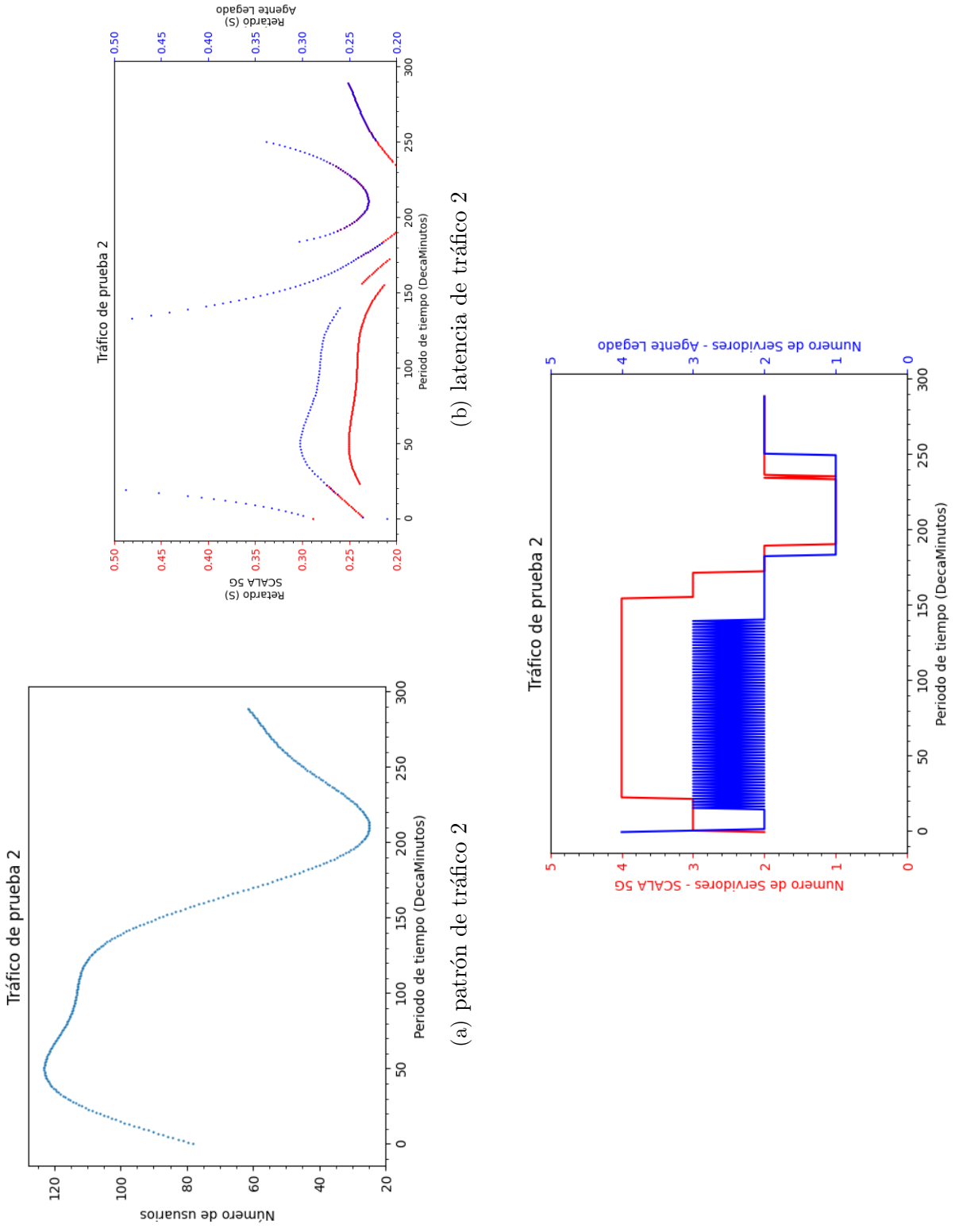


(a) patrón de tráfico 1

(b) latencia de tráfico 1

(c) escalamiento de tráfico 1

Figura 5.9: Escenario 1

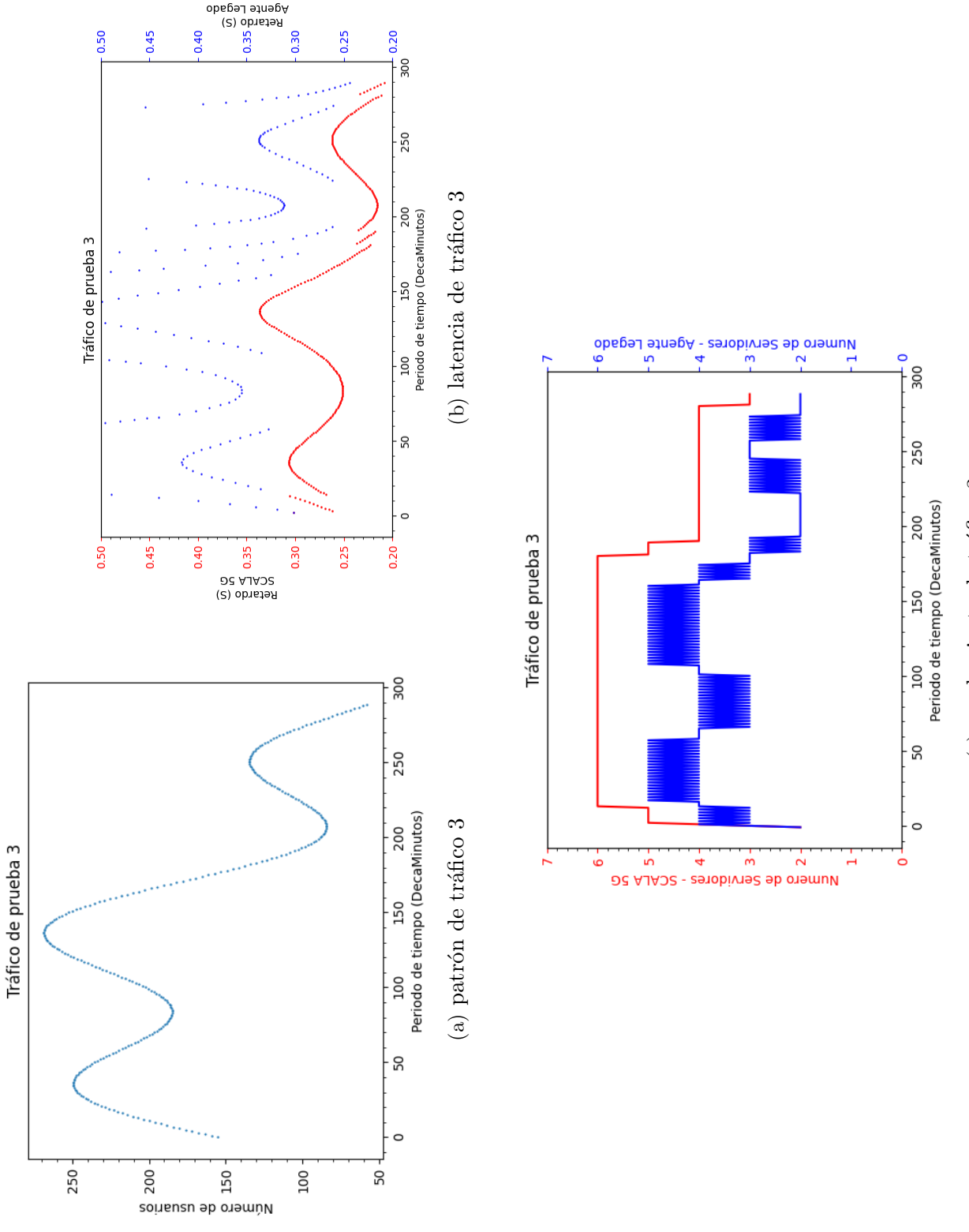


(a) patrón de tráfico 2

(b) latencia de tráfico 2

(c) escalamiento de tráfico 2

Figura 5.10: Escenario 2

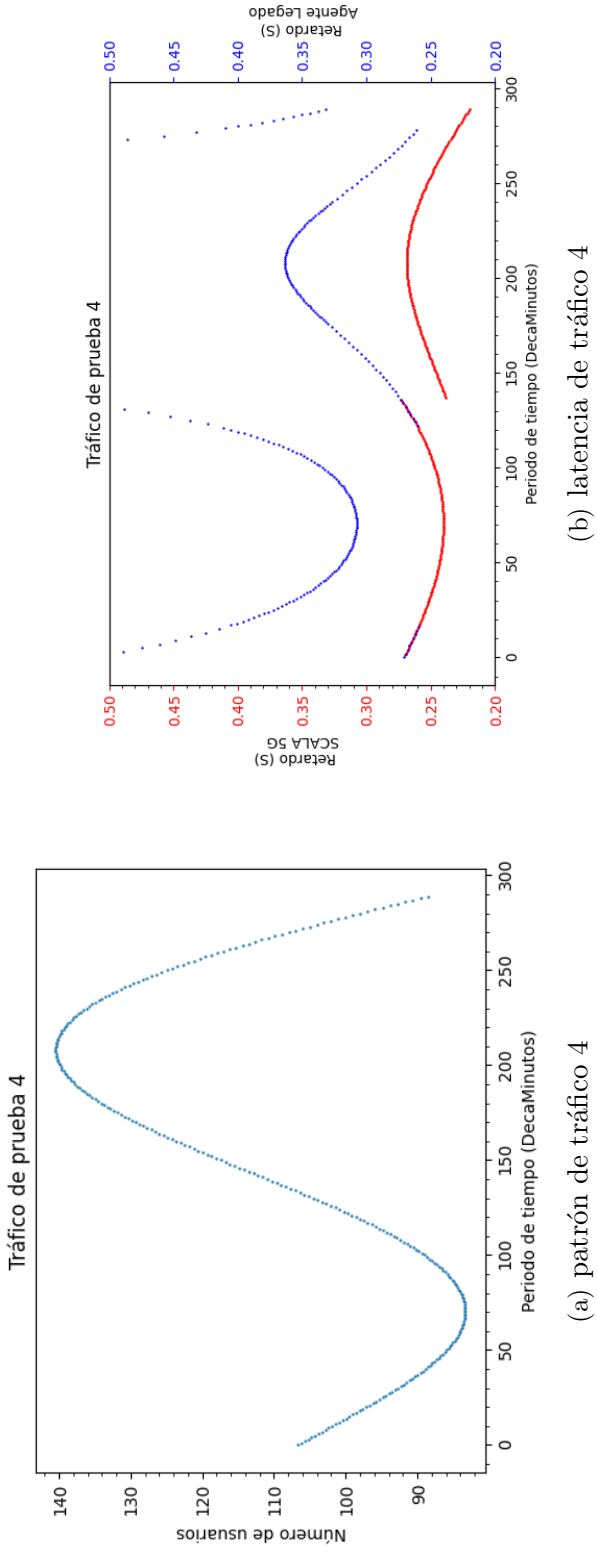


(a) patrón de tráfico 3

(b) latencia de tráfico 3

(c) escalamiento de tráfico 3

Figura 5.11: Escenario 3



(a) patrón de tráfico 4

(b) latencia de tráfico 4

(c) escalamiento de tráfico 4

Figura 5.12: Escenario 4

Las gráficas 5.9, 5.10, 5.11 y 5.12 son los resultados de SCALA 5G comparado con el Agente Legado. Cada figura contiene 3 gráficos, el subgráfico (a) muestra el patrón de tráfico de la red, el subgráfico (b) muestra la latencia promedio del servicio de 5G eMBB, y el subgráfico (c) muestra el número de VNF que despliega cada agente en un momento determinado y las modificaciones[2] que este hace sobre el entorno, todas las gráficas tienen el eje horizontal de 0 a 300 decaminutos, cada decaminuto será denominado intérvalo.

En la figura 5.9, se observa un patrón de tráfico con 2 picos de 255 y 150 usuarios. Los resultados de latencia muestran a SCALA 5G manteniendo valores dentro del rango establecido, mientras el Agente Legado tiene dificultad y sus decisiones erráticas elevan la latencia a niveles no deseados por encima de los 500ms. En el gráfico de VNF desplegadas, SCALA 5G modificó el número VNF 7 veces sin tener problemas de rebote, mientras que el Agente Legado superó las 20 modificaciones, teniendo rebotes (acciones de aumentar y disminuir el número de instancias de forma repetitiva) en todas las decisiones que tomó.

En la figura 5.10, el tráfico tiene un máximo de 130 usuarios y solamente un pico. Se observa que SCALA 5G logra mantener la latencia por debajo de los 300ms, mientras el Agente Legado supera los 500ms. Respecto al escalamiento de las VNF, SCALA 5G presenta 8 modificaciones y 1 punto mínimo de rebote en el intervalo 235, mientras el Agente Legado supera de nuevo las 20 modificaciones y tiene 1 punto de rebote crítico[3] entre los intervalos 10 y 140.

En la figura 5.11, el patrón de tráfico presenta 3 picos de tráfico en los intervalos 35, 140 y 250, teniendo 250, 270 y 130 usuarios simultáneos respectivamente. En este escenario SCALA 5G se comporta de la mejor manera posible, debido a que los primeros 2 picos de tráfico superan la capacidad máxima de 6 VNF establecida que causa que SCALA 5G no pueda seguir escalando por encima de este valor, se establece un número máximo de instancias disponibles para que el agente experimente el caso de uso donde no puede seguir escalando aunque el patrón de tráfico lo requiera, el número 6 es escogido de manera arbitraria. Por su parte, el Agente Legado no logra valores de latencia aceptables. En cuanto al número de modifica-

[2] cambio en el número de VNF

[3] cuando el agente ejecuta repetidamente escalar hacia afuera y hacia adentro 10 o más veces

ciones SCALA 5G realizó 6 sin rebotes, y el Agente Legado superó las 30 con rebotes frecuentes y extendidos a lo largo de todo el episodio. Además, no escaló al máximo en los picos de tráfico, factor crítico para evitar la saturación del servicio.

El último escenario se muestra en la figura 5.12, contiene patrón de tráfico con comportamiento sinusoidal típico, con un pico de 140 usuarios alrededor del intervalo 210. Respecto a la latencia, SCALA 5G logra mantener valores inferiores a 290ms y el Agente Legado sube hasta los 500ms. Respecto al escalamiento, SCALA 5G realizó 2 modificaciones a la red sin presentar rebote, mientras el Agente Legado superó las 20 y tuvo rebotes durante todo el episodio.

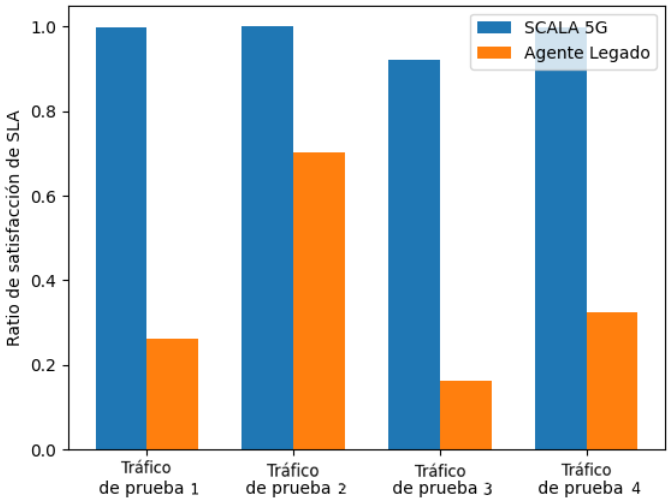


Figura 5.13: Satisfacción de SLA SCALA 5G vs Agente Legado

Después de las evaluaciones sobre los 4 escenarios anteriores, en la figura 5.13 se observa la satisfacción de nivel de servicio de SCALA 5G con respecto al Agente Legado. Se observa entonces que SCALA 5G es una mejor solución que el Agente Legado logrando escalar satisfactoriamente las instancias de servidores sin presentar rebotes en ningún patrón de tráfico probado.

Los cambios claves que mejoraron al Agente Legado hasta obtener SCALA 5G fueron:

- SCALA 5G fue entrenado con un patrón de tráfico de 48 horas, teniendo 288

intervalos en total, el doble que el Agente Legado.

- SCALA 5G no tiene un número fijo de episodios, el entrenamiento termina si SCALA 5G llega al menos al intervalo 283 durante 14 episodios consecutivos, esto asegura tener los pesos de las redes neuronales en un punto donde se resuelve el problema de manera satisfactoria. El Agente Legado entrena durante 1000 episodios de manera fija y sin tener en cuenta otra variable como el desempeño promedio en ese punto.
- SCALA 5G utiliza elementos como *memory replay* y *target network* obteniendo las ventajas mencionadas previamente en la subsección 4.3.4.
- SCALA 5G da el doble de recompensa que el Agente Legado, cuando SCALA 5G llega al paso 283 del tráfico para el episodio.
- SCALA 5G posee 128 neuronas y el Agente Legado, 64 neuronas.

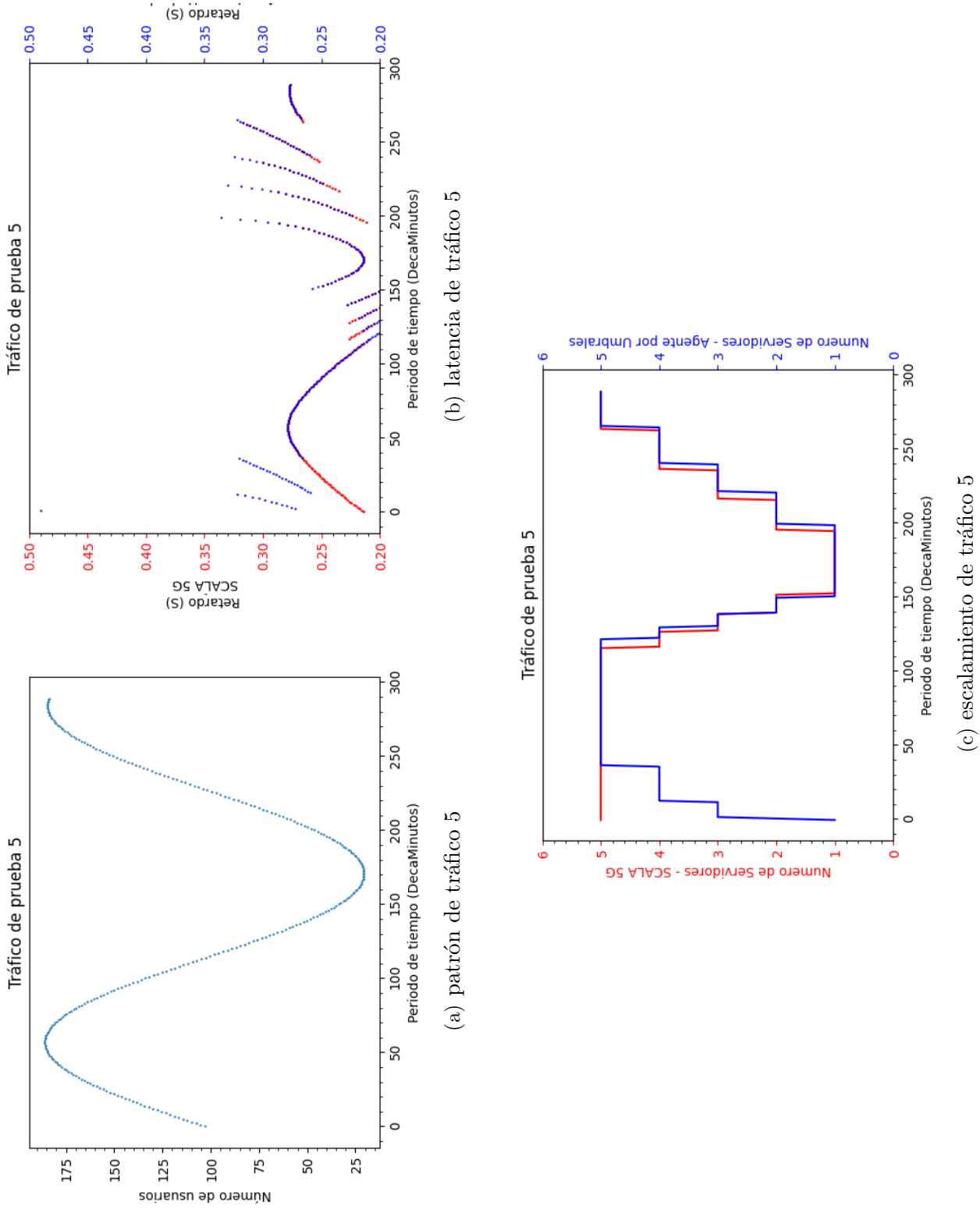
Para finalizar se comparó SCALA 5G con un algoritmo de escalamiento por umbrales, en este caso se generaron 2 episodios.

En la figura 5.14 se muestra el primer escenario de SCALA 5G comparado con el algoritmo por umbrales. El patrón de tráfico tiene un comportamiento senoidal con 2 picos iguales de 190 usuarios en los intervalos 50 y 280. Respecto a la latencia, ambos algoritmos son capaces de mantener valores inferiores a 350ms. Sin embargo, SCALA 5G reacciona de manera anticipada cuando se requiere una decisión de escalamiento. En relación al número de VNF, SCALA 5G realizó 8 modificaciones y el algoritmo por umbrales 11.

La figura 5.15 muestra el último escenario de pruebas recreado. El patrón de tráfico se comporta de manera decreciente, tiene 2 picos con 275 y 175 usuarios en los intervalos 50 y 240 respectivamente. La latencia muestra que ambos algoritmos conservaron los valores inferiores a 350ms, tienen un comportamiento idéntico salvo en pequeños intérvalos. Respecto al escalamiento, ambos algoritmos realizaron 4 modificaciones sin presentar rebotes.

De estas pruebas se obtuvo la satisfacción de nivel de servicio, en la figura 5.16 se observa que ambos agentes tuvieron porcentajes sobresalientes de satisfacción. Sin

embargo, SCALA 5G supera al algoritmo por umbrales un 2% en el escenario 5 y 1% en el escenario 6. SCALA 5G realiza un mejor trabajo en los patrones de tráfico analizados, para el escenario 5, cumple en un 98% mientras que el agente por umbrales con un 96%, en el tráfico de escenario 6, SCALA 5G tiene un 87% y el agente por umbrales un 86%.

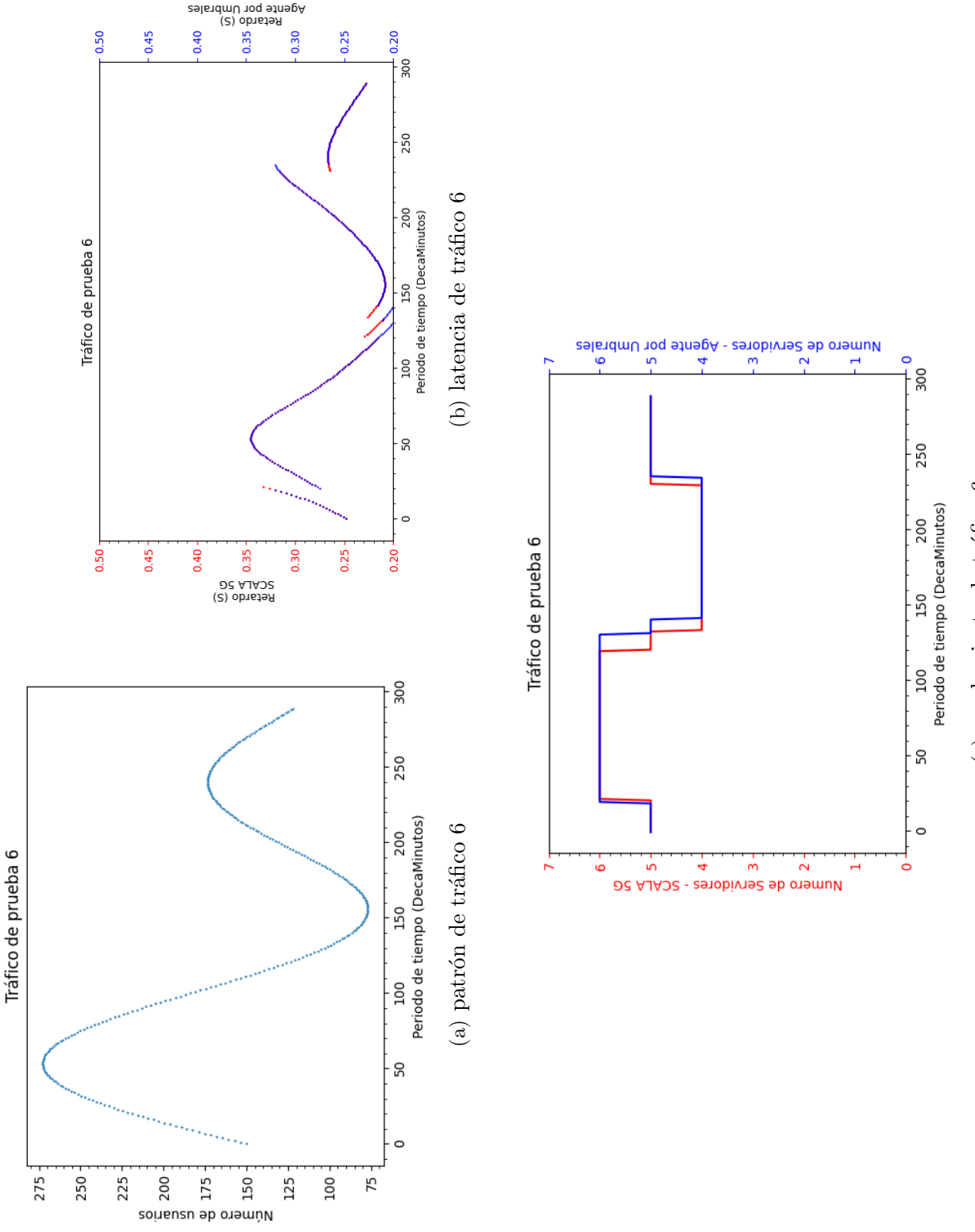


(a) patrón de tráfico 5

(b) latencia de tráfico 5

(c) escalamiento de tráfico 5

Figura 5.14: Escenario 5



(a) patrón de tráfico 6

(b) latencia de tráfico 6

(c) escalamiento de tráfico 6

Figura 5.15: Escenario 6

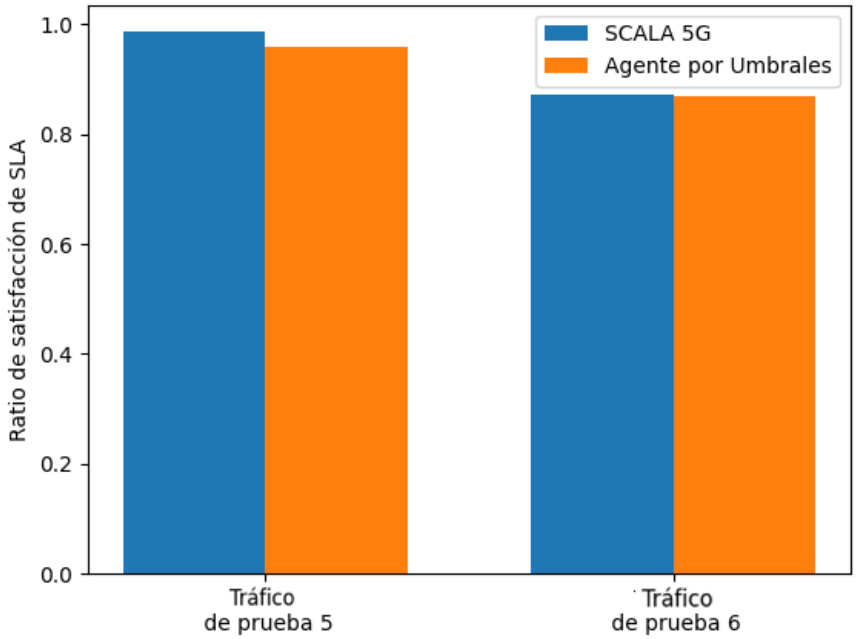


Figura 5.16: Satisfacción de SLA de SCALA 5G vs Agente por umbrales

Finalmente, se comprueba que SCALA 5G toma mejores decisiones que el Agente Legado y el Agente por umbrales en todos los patrones de tráfico analizados. El Agente por umbrales tiene una buena satisfacción de SLA y logró un desempeño similar a SCALA 5G. Sin embargo, el Agente por umbrales no es capaz de anticipar el comportamiento del tráfico de red y solo basa su criterio de escalamiento en revisar si las variables traspasan los límites estáticos predefinidos. Por lo que SCALA 5G proporciona mejores acciones y adaptación a diferentes escenarios.

Nombre de Archivo	Líneas Totales	Líneas no testeadas	Cobertura
dqn_5g_environment/VideoStreamingEnvironment.py	110	6	95 %
dqn_5g_environment/__init__.py	0	0	100 %
dqn_agent/__init__.py	0	0	100 %
dqn_agent/dqn_agent.py	56	2	96 %
dqn_improvers/ExperienceReplay.py	17	0	100 %
dqn_improvers/__init__.py	0	0	100 %
dqn_training/__init__.py	0	0	100 %
dqn_training/train_video_server_dqn.py	89	2	98 %
math_helpers/__init__.py	0	0	100 %
math_helpers/epsilon_decay.py	4	0	100 %
neural_network_/NeuralNetwork.py	13	0	100 %
neural_network_/__init__.py	0	0	100 %
plotting/__init__.py b	0	0	100 %
tune_parameters.py	9	0	100 %
TOTAL	298	10	97 %

Cuadro 5.3: Reporte de cobertura

5.2.3. Cobertura de pruebas unitarias

Por otra parte la medición de la métrica de *coverage* se hizo de manera directa sobre el código utilizando pruebas unitarias. Para obtener la medida de *coverage* se escribieron 23 pruebas, el reporte es generado utilizando el comando:

coverage run –source=./ -m unittest discover -s tests/ && coverage html

Los resultados se encuentran en el cuadro 5.3, se tiene un total de 97% del código cubierto por al menos una prueba unitaria, cumpliendo con el objetivo de garantizar la fiabilidad del código escrito.

Capítulo 6

Conclusiones y trabajos futuros

En este capítulo presentamos la respuesta a la pregunta **¿Cómo realizar el escalado automático en un NS sobre un entorno realístico con un tiempo de convergencia competente?** y sus respectivos trabajos futuros.

6.1. Conclusiones

Para responder a la pregunta principal de este trabajo se deben hacer las siguientes consideraciones:

- El algoritmo SCALA 5G demostró una supremacia sobre el Agente Legado y el Agente por umbrales sobre los ambientes de evaluación. Se marca una gran diferencia en los patrones de tráfico de mayor complejidad. El cumplimiento del SLA bajo las condiciones descritas anteriormente es del 97%, asegurando un correcto desempeño en ambientes con diversos patrones de tráfico.
- El uso de variables de QoS (en nuestro caso la latencia promedio de usuario) hace que el algoritmo sea más representativo respecto a los algoritmos que solo tienen en cuenta variables de recursos y abarque mayores caso de uso, siendo agnóstico a la infraestructura donde se despliegue.

- La infraestructura usada para la evaluación de nuestro algoritmo garantiza un rápido despliegue y operación, asegurando que el ambiente representa de manera realística la operación de un ambiente 5G.
- El tiempo de convergencia es bajo y hace que SCALA 5G pueda aprender de manera rápida nuevos patrones de tráfico. Se infiere que la elección del retardo y el número de usuarios como componentes de un estado S_t y S_{t+1} proporciona la suficiente información para que SCALA 5G determine si está tomando decisiones acertadas y que por ende converja.

De esta forma se logró un algoritmo de escalamiento automático, que actúa de una manera eficiente y logra mantener servicios y aplicaciones propias de 5G eMBB en condiciones de operación óptimas.

6.2. Trabajo futuro

Finalmente posterior al desarrollo de nuestro trabajo de grado, encontramos los siguientes ítems sobre los temas y desarrollos que se pueden realizar:

- Ampliar el entorno para que el algoritmo sea capaz de cubrir URLLC y MMTC añadiendo prioridades entre los casos de uso con el fin de que cada uno cumpla con las regulaciones establecidas por el 3GPP y la ETSI.
- Entrenar y probar el algoritmo sobre una red de servicios 5G comercial. Donde tambien se requiera un control automático de escalamiento.
- Continuar el desarrollo de nuevas técnicas y algoritmos teniendo en cuenta la evolución de las tecnologías y el inminente diseño y despliegue de redes de sexta generación (6G).

Bibliografía

[1] O. Bondarenko, D. Ageyev, and O. Mohammed, "Optimization model for 5g network planning," in *2019 IEEE 15th International Conference on the Experience of Designing and Application of CAD Systems (CADSM)*, 2019.

[2] G. Liu, Y. Huang, Z. Chen, L. Liu, Q. Wang, and N. Li, "5g deployment: Standalone vs. non-standalone from the operator perspective," *IEEE Communications Magazine*, vol. 58, no. 11, pp. 83–89, 2020.

[3] "Network Slicing for 5g Networks & Services," *5G Americas*, Nov. 2016.

[4] D. Kim and S. Kim, "Network slicing as enablers for 5g services: state of the art and challenges for mobile industry," *Telecommunication Systems*, vol. 71, pp. 517–527, July 2019.

[5] A. Sanchoyerto, R. Solozabal, B. Blanco, E. Jimeno, E. Aldecoa, E. Basurto, and F. Liberal, "Orchestration of Mission-Critical Services over an NFV Architecture," in *Artificial Intelligence Applications and Innovations*, (Cham), pp. 70–77, Springer International Publishing, 2019.

[6] V. Nguyen, A. Brunstrom, K. Grinnemo, and J. Taheri, "SDN/NFV-Based Mobile Packet Core Network Architectures: A Survey," *IEEE Communications Surveys and Tutorials*, vol. 19, pp. 1567–1602, Apr. 2017.

[7] T. Buyakar, A. Rangisetti, A. Franklin, and B. Tamma, "Auto scaling of data plane VNFs in 5g networks," in *2017 13th International Conference on Network and Service Management (CNSM)*, pp. 1–4, Nov. 2017.

[8] F. Botina and K. Tobar, "Scalability Analysis of LTE-EPC in an NFV Environment," 2018.

[9] Y. Ren, T. Phung-Duc, J. Chen, and Z. Yu, "Dynamic Auto Scaling Algorithm (DASA) for 5g Mobile Networks," in *2016 IEEE Global Communications Conference (GLOBECOM)*, pp. 1–6, Dec. 2016.

[10] A. Nadjaran, J. Son, Q. Chi, and R. Buyya, "ElasticSFC: Auto-scaling techniques for elastic service function chaining in network functions virtualization-based clouds," *Journal of Systems and Software*, vol. 152, pp. 108–119, June 2019.

[11] M. Singh, S. Vittal, and A. Franklin, "Serens: Self regulating network slicing in 5g for efficient resource utilization," *2020 IEEE 3rd 5G World Forum*, 2020.

[12] S. Waidande, "A Literature Survey on Scaling Approaches for VNF in NFV Monitoring," *International Research Journal of Engineering and Technology (IRJET)*, vol. 5, pp. 1528–1531, Dec. 2018.

[13] P. Tang, F. Li, W. Zhou, W. Hu, and L. Yang, "Efficient Auto-Scaling Approach in the Telco Cloud Using Self-Learning Algorithm," in *2015 IEEE Global Communications Conference (GLOBECOM)*, pp. 1–6, Dec. 2015.

[14] E. Barrett, E. Howley, and J. Duggan, "Applying reinforcement learning towards automating resource allocation and application scalability in the cloud and Experience," *Concurrency and computation practice and experience*, vol. 25, pp. 1656–1674, May 2012.

[15] C. Arteaga, F. Rissoi, and O. Caicedo, "An adaptive scaling mechanism for managing performance variations in network functions virtualization: A case study in an NFV-based EPC," in *2017 13th International Conference on Network and Service Management (CNSM)*, pp. 1–7, Nov. 2017.

[16] G. Yunjie, H. Yuxiang, D. Yuehang, and X. Jichao, "Joint Optimization of Resource Allocation and Service Performance in vEPC Using Reinforcement Learning," *2019 IEEE 4th International Conference on Cloud Computing and Big Data Analysis (ICCCBDA)*, Apr. 2019.

[17] I. Alawe, Y. Hadjadj-Aoul, A. Ksentini, P. Bertin, C. Viho, and D. Darche, "Smart scaling of the 5g core network: An rnn-based approach," *2018 IEEE Global Communications Conference*, pp. 1–6, Dec. 2018.

[18] I. Alawe, Y. Hadjadj-Aoul, A. Ksentinit, P. Bertin, C. Viho, and D. Darche, "An Efficient and Lightweight Load Forecasting for Proactive Scaling in 5g Mobile Networks," *2018 IEEE Conference on Standards for Communications and Networking (CSCN)*, Oct. 2018.

[19] T. Subramanya, D. Harutyunyan, and R. Riggio, "Machine learning-driven service function chain placement and scaling in MEC-enabled 5g networks," *Computer Networks*, vol. 166, Nov. 2019.

[20] D. Bega, M. Gramaglia, M. Fiore, A. Banchs, and X. Costa-Perez, "DeepCog: Cognitive Network Management in Sliced 5g Networks with Deep Learning," in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, pp. 280–288, Apr. 2019.

[21] Z. Xiong, Y. Zhang, D. Niyato, R. Deng, P. Wang, and L. Wang, "Deep Reinforcement Learning for Mobile 5g and Beyond: Fundamentals, Applications, and Challenges," *IEEE Vehicular Technology Magazine*, vol. 14, pp. 44–52, June 2019.

[22] S. Borkar and H. Pande, "Application of 5g next generation network to internet of things," *2016 International Conference on Internet of Things and Applications (IOTA)*, Jan. 2016.

[23] "5g; Service requirements for next generation new services and markets," *(3GPP TS 22.261 version 15.5.0 Release 15)*, p. 53, July 2018.

[24] "IMT Vision – Framework and overall objectives of the future development of IMT for 2020 and beyond," p. 21, Sept. 2015.

[25] P. Popovski, K. Trillingsgaard, O. Simeone, and G. Durisi, "5g wireless network slicing for embb, urllc, and mmtc: A communication-theoretic view," *IEEE Access*, vol. 6, pp. 55765–55779, Sept. 2018.

[26] T. de Cola and I. Bisio, "Qos optimisation of embb services in converged 5g-satellite networks," *IEEE Transactions on Vehicular Technology*, 2020.

[27] M. Erel-Özçevik and B. Canberk, "Road to 5g reduced-latency: A software defined handover model for embb services," *IEEE Transactions on Vehicular Technology*, 2019.

[28] "Study on scenarios and requirements for next generation access technologies," *3GPP TR 38.913 version 14.3.0 Release 14*, Oct. 2017.

[29] M. Alsenwi, N. Tran, M. Bennis, C. Hong, S. Pandey, and A. Bairagi, "Intelligent resource slicing for embb and urllc coexistence in 5g and beyond: A deep reinforcement learning based approach," *IEEE Transactions on Wireless Communications*, July 2021.

[30] M. Almekhlafi, M. Arfaoui, M. Elhattab, C. Assi, and G. A, "Joint scheduling of embb and urllc services in ris-aided downlink cellular networks," *International Conference on Computer Communications and Networks*, July 2021.

[31] C. Campolo, A. Molinaro, A. Iera, R. Fontes, and C. Rothenberg, "Towards 5g network slicing for the v2x ecosystem," *IEEE Conference on Network Softwarization and Workshops (NetSoft)*, June 2018.

[32] W. Tian, M. Fan, C. Zeng, Y. Liu, D. He, and Q. Zhang, "Telerobotic spinal surgery based on 5g network," *Neurospine*, Mar. 2020.

[33] C. Bockelmann, N. Pratas, H. Nikopour, K. Au, T. Svensson, C. Stefanovic, P. Popovski, and A. Dekorsy, "Massive machine-type communications in 5g: physical and mac-layer solutions," *IEEE Communications Magazine*, vol. 54, pp. 59–65, Sept. 2016.

[34] B. Han, V. Gopalakrishnan, L. Ji, and S. Lee, "Network function virtualization: Challenges and opportunities for innovations," *IEEE Communications Magazine*, vol. 53, pp. 90–97, Feb. 2015.

[35] X. Foukas, G. Patounas, A. Elmokashfi, and M. Mahesh, "Network Slicing in 5g: Survey and Challenges," *IEEE Communications Magazine*, vol. 55, pp. 94–100, May 2017.

[36] D. Alotaibi, "Survey on network slice isolation in 5g networks: Fundamental challenges," *Procedia Computer Science*, vol. 182, pp. 38–45, 2021.

[37] F. Debbabi, R. Jmal, and F. Chaari, "5g network slicing: Fundamental concepts, architectures, algorithmics, projects practices, and open issues," *Concurrency and Computation: Practice and Experience*, Oct. 2021.

[38] R. Jmal and L. Chaari, "Ieee/acs 14th international conference on computer systems and applications (aiccsa)," *ETSI GS MEC 002 V2.1.1*, Oct. 2017.

[39] F. Rahman, S. Newaz, T. Au, W. Suhaili, M. M, and G. Lee, "Entruve: Energy and trust-aware virtual machine allocation in vehicle fog computing for catering applications in 5g," *Future Generation Computer System*, Jan. 2022.

[40] ETSI, "Multi-access edge computing (mec); phase 2: Use cases and requirements," *ETSI GS MEC 002 V2.1.1*, Oct. 2018.

[41] A. Younge, R. Henschel, J. Brown, G. von Laszewski, J. Qiu, and G. Fox, "Analysis of virtualization technologies for high performance computing environments," *IEEE 4th International Conference on Cloud Computing*, July 2011.

[42] D. Rafique and L. Velasco, "Machine learning for network automation: overview, architecture, and applications," *IEEE/OSA Journal of Optical Communications and Networking*, vol. 10, pp. D126–D143, Oct. 2018.

[43] Z. Fadlullah, F. Tang, B. Mao, N. Kato, O. Akashi, T. Inoue, and K. Mizutani, "State-of-the-Art Deep Learning: Evolving Machine Intelligence Toward Tomorrow's Intelligent Network Traffic Control Systems," *IEEE Communications Surveys and Tutorials*, vol. 19, pp. 2432–2455, May 2017.

[44] S. Ayoubi, N. Limam, M. Salahuddin, N. Shahriar, R. Boutaba, F. Estrada-Solano, and O. Caicedo, "Machine Learning for Cognitive Network Management," *IEEE Communications Magazine*, vol. 56, pp. 158–165, Jan. 2018.

[45] J. Laaksonen and E. Oja, "Classification with learning k-nearest neighbors," in *Proceedings of International Conference on Neural Networks (ICNN'96)*, vol. 3, pp. 1480–1483 vol.3, 1996.

[46] B. YEGNANARAYANA, *Artificial Neural Networks*. 2009.

[47] S. Suthaharan, *Support Vector Machine*, pp. 207–235. Boston, MA: Springer US, 2016.

[48] R. Kumari, Sheetanshu, M. K. Singh, R. Jha, and N. Singh, "Anomaly detection in network traffic using k-mean clustering," in *2016 3rd International Conference on Recent Advances in Information Technology (RAIT)*, pp. 387–393, 2016.

[49] H. Abdi and L. J. Williams, "Principal component analysis," *WIREs Computational Statistics*, vol. 2, no. 4, pp. 433–459, 2010.

[50] T. Kohonen, "The self-organizing map," *Proceedings of the IEEE*, vol. 78, no. 9, pp. 1464–1480, 1990.

[51] C. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, pp. 279–292, May 1992.

[52] D. Bega, M. Gramaglia, A. Banchs, V. Sciancalepore, K. Samdanis, and X. Costa-Perez, "Optimising 5g infrastructure markets: The business of network slicing," *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*, May 2017.

[53] P. Caballero, A. Banchs, G. de Veciana, X. Costa-Pérez, and A. Azcorra, "Network slicing for guaranteed rate services: Admission control and resource allocation games," *IEEE Transactions on Wireless Communications*, vol. 17, Aug. 2018.

[54] I. Alawe, Y. Hadjadj-Aoul, A. Ksentini, and D. Darche, "On the scalability of 5g core network: the amf case," *2018 15th IEEE Annual Consumer Communications Networking Conference (CCNC)*, Jan. 2018.

[55] K. Qu, W. Zhuang, X. Shen, X. Li, and J. Rao, "Dynamic resource scaling for vnf over nonstationary traffic: A learning approach," *IEEE Transactions on Cognitive Communications and Networking*, 2020.

[56] A. Aslan, G. Balce, and C. Toker, "Dynamic resource management in next generation networks with dense user traffic," *IEEE International Black Sea Conference on Communications and Networking*, 2020.

[57] W. Villota, O. Caicedo, and N. Saldanha, "Admission control for 5g network slicing based on (deep) reinforcement learning," 2020.

[58] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *DeepMind*, 2013.

[59] V. Mnih, K. Kavukcuoglu, D. Silver, A. Rusu, J. Veness, M. Bellemare, A. Graves, M. Riedmiller, A. Fidjeland, G. Ostroviski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, 2015.

[60] B. R. Kiran, I. Sobh, V. Talpaert, P. Mannion, A. A. A. Sallab, S. Yogamani, and P. Pérez, "Deep reinforcement learning for autonomous driving: A survey," *IEEE Transactions on Intelligent Transportation Systems*, 2021.

[61] J. Li and X. Zhang, "Deep reinforcement learning-based joint scheduling of embb and urllc in 5g networks," *IEEE Wireless Communications Letters*, vol. 9, no. 9, pp. 1543–1546, 2020.

[62] D. M. Casas-Velasco, O. M. C. Rendon, and N. L. S. da Fonseca, "Intelligent routing based on reinforcement learning for software-defined networking," *IEEE Transactions on Network and Service Management*, vol. 18, no. 1, pp. 870–881, 2021.

[63] L.-j. Lin, "Reinforcement learning for robots using neural networks," *Carnegie Mellon*, 1993.

[64] G. Brown, "Service-based architecture for 5g core networks," *A Heavy Reading white paper produced for Huawei Technologies Co. Ltd. Online: https://www.huawei.com/en/press-events/news/2017/11/HeavyReading-WhitePaper-5G-Core-Network*, vol. 1, pp. 1–12, 2017.

[65] Microsoft, "Typescript https://www.typescriptlang.org/," oct 2012.

[66] R. Dahl, "Nodejs https://nodejs.org/es/," may 2009.

[67] M. Satheesh, B. J. D'mello, and J. Krol, *Web development with MongoDB and NodeJs*. Packt Publishing Ltd, 2015.

[68] O. L. Madsen, B. Magnusson, and B. Mølier-Pedersen, "Strong typing of object-oriented languages revisited," *ACM SIGPLAN Notices*, vol. 25, no. 10, pp. 140–150, 1990.

[69] S. Kaur, K. Kumar, J. Singh, and N. S. Ghumman, "Round-robin based load balancing in software defined networking," in *2015 2nd international conference on computing for sustainable global development (INDIACom)*, pp. 2136–2139, IEEE, 2015.

[70] E. Bisong, *Building Machine Learning and Deep Learning Models on Google Cloud Platform*. Canada: Apress, 2019.

[71] J. Shah and D. Dubaria, "Building modern clouds: Using docker, kubernetes amp; google cloud platform," in *2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC)*, pp. 0184–0189, 2019.

[72] 3GPP, "Procedures for the 5g system," *ETSI TS 123 502*, 2018.

[73] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.

[74] M. M. A. Monshi, J. Poon, V. Chung, and F. M. Monshi, "Covidxraynet: optimizing data augmentation and cnn hyperparameters for improved covid-19 detection from cxr," *Computers in biology and medicine*, vol. 133, p. 104375, 2021.

[75] S. Kumar, "Balancing a cartpole system with reinforcement learning–a tutorial," *arXiv preprint arXiv:2006.04938*, 2020.

[76] Í. M. Miranda, M. Ladeira, and C. de Castro Aranha, "A comparison study between deep learning and genetic programming application in cart pole balancing problem," in *2018 IEEE Congress on Evolutionary Computation (CEC)*, pp. 1–7, IEEE, 2018.

[77] E. Musk, "Openai gym `https://gym.openai.com/`," dec 2015.

[78] G. van Rossum, "Python 3.9 `https://www.python.org/`," oct 2020.

[79] A. Paszke, S. Gross, S. Chintala, and G. Chanan, "Pytorch `https://pytorch.org/`," sep 2016.

[80] P. Runeson, "A survey of unit testing practices," *IEEE Software*, vol. 23, no. 4, pp. 22–29, 2006.

[81] C. Zhang, Y. Yan, H. Zhou, Y. Yao, K. Wu, T. Su, W. Miao, and G. Pu, "Smartunit: Empirical evaluations for automated unit testing of embedded software in industry," in *2018 IEEE/ACM 40th International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP)*, pp. 296–305, 2018.

SCALA 5G: Un mecanismo de escalamiento automático con aprendizaje por refuerzo profundo para el uso eficiente de recursos en un ambiente NFV

Anexos

Trabajo de Grado

Juan Sebastian Daza Gaviria

Andrés Mauricio Maca Hurtado

Advisor: PhD. Oscar Mauricio Caicedo Rendón
Co-Advisor: PhD. José Armando Ordoñez Córdoba

Departamento de Telemática
Facultad de Ingeniería Electrónica y Telecomunicaciones
Universidad del Cauca
Popayán, Cauca, 2022

ANEXO A

El Anexo A presenta el enlace que redirecciona al grupo de GitLab con el código fuente del proyecto:

https://gitlab.com/scala5g

El Agente SCALA 5G se encuentra en:

https://gitlab.com/scala5g/cartpole

La infraestructura de kubernetes:

https://gitlab.com/scala5g/kubernetes-infra

Las VNFS:

https://gitlab.com/scala5g/amf

https://gitlab.com/scala5g/smf

https://gitlab.com/scala5g/umf

ANEXO B

El Anexo B presenta artículo de investigación desde la página siguiente se muestra el artículo **SCALA 5G: Efficient scaling on Network Slicing using Deep Reinforcement Learning** complemento de nuestro trabajo de grado y que muestra los resultados del mismo.

SCALA 5G: Efficient scaling on Network Slicing using Deep Reinforcement Learning

Juan S. Daza, Andres M. Maca, Oscar Mauricio Caicedo Rendon, Armando Ordoñez

***Abstract*—Scaling is a key enabler to support 5G/NFV slicing since it guarantees adequate resource provisioning as network slices require it. Scaling is not a trivial process as it requires dynamic reconfigurations according to the changing requirements. An inadequate scaling leads to an increased Operational Expenditures (OPEX) and poor user experience. Considerable effort has been devoted during recent years to the development of automated scaling methods. These approaches base on Heuristic, Reinforcement Learning (RL), and Deep Reinforcement Learning (DRL). However, previous solutions are not applicable on Enhanced Mobile Broadband (eMBB) specific requirements because these are not targeting eMBB data plane. Here, we introduce SCALA 5G, a clever agent to scale 5G eMBB slices. SCALA 5G defines a novel DRL algorithm that scales network slices resources accomplishing eMBB data plane requirements. The agent guarantees an optimal balance between OPEX and Quality of Service (QoS). SCALA 5G was tested in a 5G eMBB realistic prototype environment. Results show an increase of 2% accuracy more concerning other scaling research solutions like a threshold agent. Which is a good performance in scaling scope.**

***Index Terms*—Scaling, Deep Reinforcement Learning, NFV, Quality of Service,**

I. Introduction

NETWORK SLICING is crucial for current and future 5G use cases as eMBB, Ultra Low Latency Communications (URLLC) and massive Machine Type Communications (mMTC) [1]. scalability is a process into management area that provides the network resources properly to increase QoS and optimize the OPEX. Network tenants consider Software-Defined Networking (SDN), Network Functions Virtualization (NFV), and cloud solutions to operate their 5G slices [2] to make a proper scaling. However, tenants cannot scale their network slices efficiently without adequate techniques. Capable scaling methods generate a noteworthy reduction OPEX. Thereby, scalability for slicing has become one of the main interest areas for the 5G tenants [3]. eMBB needs a robust solution to support broad multimedia aplications such as video streaming, Aumented Reality (AR), Virtual Reality (VR) among others.

Some efforts [3]–[6], designed basic threshold-based algorithms for scale virtualized 5G core functions as Access and Mobility Management Function (AMF), User Plane Function (UPF). When a target variable overpasses a threshold, the algorithm returns a decision about scale an application resource (function, slice, service). However, these algorithms present a poor response to changes. So when the network fluctuates, heuristics loss accuracy and stability. On this way, these algorithms may not accomplish requirements for eMBB use case.

Few algorithms as [7] have used RL technique to scale 5G slices for individual or 5G Core (5GC) functions. This algorithm is more accurate than heuristics, taking decisions over different network scenarios in diverse core network applications. However, Convergence Time (CT) is not enough fast to manage 5G networks and it is not scalable for cases that need a lot of states as eMBB [8]. Finally, other researches use DRL models, [9]–[11] to scale 5GC network slices. DRL Forecasting enhances the accuracy of decisions and controls a 5G network with fewer issues than previous algorithms as heuristic and RL ones. However, none DRL algorithms focus specifically in eMBB scope. Thereby, neither algorithm has been tested in realistic 5G eMBB environments, which does not guarantee covering new services and features with the adequate performance [12].

Having in account the previous issues of each algorithm group, still not find a completely efficient way to scale 5G eMBB slices with the appropriate requirement accomplishment. Our solution aim in search a new efficient solution to optimize scaling in 5G eMBB slicing networks in order to accomplish tenant requirements. On this way, SCALA 5G defines a DRL agent with a Deep Q-Network (DQN) structure, which is a novel DRL method proposed by DeepMind [13], Our DRL search in a fast way patterns that identify typical behaviors in the network, the agent take a decision and learn about the better way to scale the resources. In that vein, scaling errors are negligible, the training time is less than other techniques, and the system keeps QoS and reduce OPEX. SCALA 5G was tested in a realistic environment of 5GC for scale virtual functions that contains eMBB applications. Results show a significant enhancement respect to the some previous mentioned methods and also respect to Kubernetes commercial solution. Our algorithm gets a 98% of accuracy, also it converge on less than 2000 episodes, what is a proficient performance and it opens the door to continue researching about the scaling topic.

The contributions of this paper are:

- A novel scaling algorithm based on DQN with replay memory, target and online network [14] to scale 5G eMBB slices efficiently.
- A prototype of the proposed algorithm for the whole NSL granularity type.
- A scaling comparison between a no-target-network, no-memory-replay DQN algorithm, threshold algorithm and SCALA 5G-DQN given the same eMBB environment and conditions.

O. M. Caicedo is with the Department of Telematics, Universidad del Cauca, Popayán, Colombia. e-mail: omcaicedo@unicauca.edu.co.

The remainder of this paper is organized as follows. Section II describes the related work. Section III details about SCALA 5G and the proposed architecture, Section IV introduces the DRL algorithm implemented. Section 5 presents the SCALA 5G present the complete prototype and the testing over the cloud. Section 6 presents conclusions and future work.

II. Related Work

This section shows a broad view about 5G network slicing scaling techniques. Table I presents all the related works found in the literature. We consider 4 different characteristics in table: the scaling type that describes the technology used to make the solution, the scaling item specifically if they are scaling eMBB use case or not , and the metrics used to evaluate each work.

Some works used heuristic techniques as in [3] scales dataplane functions for Evolved Packet Core (EPC). [4] enables scaling for 5GC functions. [5] scale 5G Virtual Network Function (VNF) functions using Holt-Winters model, and [6] for RAN functions. These works have good performance in some small and test environments. However, Heuristics techniques lack of flexibility and stability when the network have several states. So, in a 5G environment heuristics techniques may violate the Service Level Agreement (SLA) requirements for a 5g use case. For eMBB purpose the sudden changes and fluctuations make that heuristic loose stability.

Only one work used RL as technique. In [7], and the remaining approaches employed DRL. As in [9] to scale 5G SDN/NFV general functions. [10] and [11] that focus on the three 5G use cases. Highlighting [11] that covers access and core functions focus on allocation resources, having into account the 5G use cases. These have several benefits with respect to heuristic solutions. RL and DRL approaches can identify complex patterns that assure adaptability and efficient management. However, RL has limits respect to environments with high number of states. Meanwhile DRL approaches has not focus on cover specifically an scaling eMBB environment, where high volume of traffic and new services that requires high bandwidth and QoS real scenario can maintain the same performance.

SCALA 5G overcomes the previous issues about using a based-NFV architecture and the DRL benefits. In this way, our solution reduces SLA violations and training time, maintaining the accuracy in decisions. Also we can deploy and scale high requirement services that belong to 5G eMBB context. In the next section, we present the system architecture, components, software, and platforms that support this approach.

III. SCALA 5G

This section presents SCALA 5G and its principal features about the design implemented. First, we show the components of the architecture and the technologies within the prototype as cloud platforms, Neural Network (NN) structures, and the interfaces to enable in 5G-NFV functions. After, we show the algorithm designed to scale the services required for efficient network slicing.

As depicted in Figure 1 the architecture has four modules: *Scaling module*, *Monitoring module*, *Lifecycle module* and *NFV-based 5G infrastructure module (N5IM)*. The *Scaling module* take the scaling decision and host the DQN algorithm so *Lifecycle module* communicates the scaling decision to the *N5IM* and *Monitoring module* overseers state of *N5IM* and the impact of the scaling decisions. The *N5IM* accommodate the Network Slice (NSL) of 5G.

A. Components

In this subsection, we reveal Scala 5G architecture components.

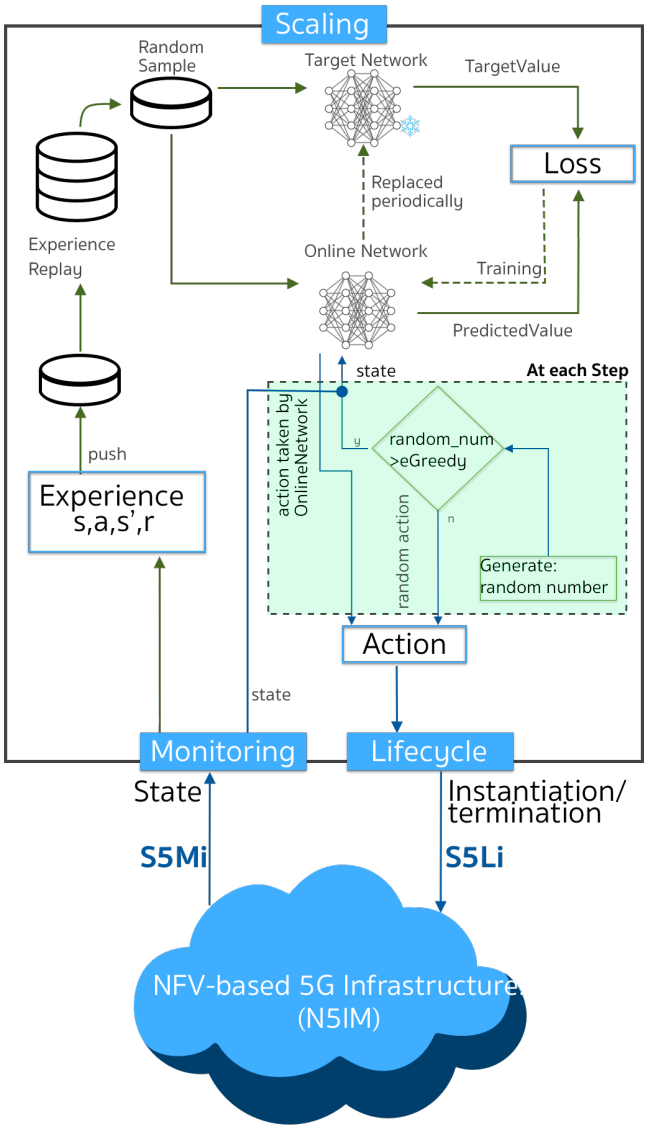


Figure 1: SCALA 5G architecture

- *Monitoring module* contains a Reward Service that calculates how well the last action performs. How good is an action is determined by the reward function, the explanation is on the next section. Monitoring Module needs the new state from N5IM. SCALA-5G Monitoring interface (S5Mi) is used for send state information from N5IM to Monitoring Module.

Table I: Related Work

Ref	Description	Scaling type	eMBB-scaling	Granularity	Metrics
[3]	Scaling proposal based on a 5G NSA based architecture for scale LTE functions over slices using LXC technology.	Heuristic	No	VNF	Throughput
[4]	A framework that performs monitoring, Analytics and Selection of the Network Slices. in 5GC environment.	Heuristic	No	Slice	User rate, Request TTL, CPU, Memory
[5]	Adaptive scaling VNF over 5G networks, using a VNF adaptive scaling strategy based on Holt Winters prediction results to determine the number of VNFs and resources to conserve resources and reduce network slice deployment costs.	Heuristic	No	VNF	Throughput
[6]	A service selection, monitoring and NS deployment using resource orchestration for 5G Radio Access Network (RAN) functions.	Heuristic	No	Slice	CPU, Scaling interval
[7]	a VNF flexible deployment scheme based on RL to scale 5G NFV functions with a non-convex linear mathematical optimization that aim at optimize energy consumption.	RL	No	VNF	Throughput
[9]	A change-point-driven traffic parameter learning and resource demand prediction scheme for VNF scaling problem on 5G networks.	DRL	No	Slice	VNF migration cost, Resource SLA
[10]	Up/down scaling of 5G instances for data request of users with a DRL agent that also accept/reject user incoming traffic according with availability and priority policies.	DRL	No	Slice	Throughput, Blocked traffic
[11]	Algorithm for the performance of admission control and resource allocation for NSLRs of eMBB, URLLC, and MIoT in the 5G core network.	DRL	No	Slice	CPU, Throughput
	SCALA 5G	DRL	Yes	VNF	CPU, Bandwidth, SLA violations

- *Lifecycle module* communicates the scaling decision(action) taken by the scaling module to NFV-based 5G infrastructure. SCALA-5G Lifecycle interface (S5Li) sends scaling information from Lifecycle module to Monitoring Module.

- The *Scaling module* contains a DQN that decides which action take between a set of scale-up, scale-down, or keep. In the beginning, the agent doesn't know how to choose the right action. Thereby, decisions are expected. After several steps, the agent converges the Q approximation function. DQN uses a Q approximation function, instead of a Q table. Along each step, the scaling module generates an action and send it to the Lifecycle module, and it takes a state from the Monitoring module. DQN has an Experience Replay and Target Network to stabilize the training, and these parts will be explained in next subsection.

- N5IM accommodates the NSL for 5G eMBB uses case. N5IM deploys the eMBB as a NSL. We use the whole NSL scaling granularity.

B. Use Case

eMBB is one of the most common use cases on 5G [11]. This scenario focus on cover human-centric use cases such as multimedia access, services, and data (e.g. video, music, and user-generated content [15]). Each NSL graph has an AMF, Session Management Function (SMF) for interacting with the decoupled data plane and session management, and UPF for the data plane.

Figure 2a depicts scaling granularity: element of NSL, Scala 5G scales UPF from 1 to num_max where num_max is the maximum number of instances for a given NSL

Figure 2b depicts scaling granularity: whole NSL , SCALA 5G scales the whole NSL from 1 to num_max where num_max is the maximum number of instances for a given NSL

In the following subsections, we explain detailed about state and action spaces, reward function, experience replay, target and online NNs, e-greedy method, and other processes indeed for SCALA 5G.

C. Deep Reinforcement Learning Agent

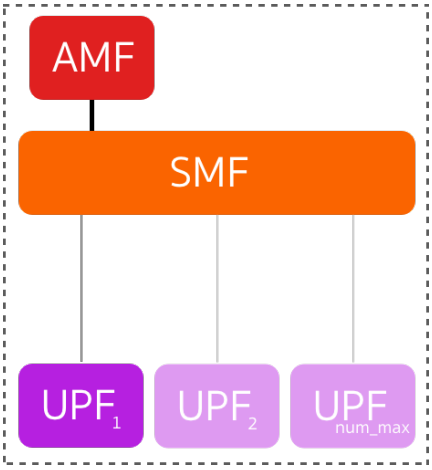
1) The Reward Function: The reward is the value that is given to the agent according to their performance. In our case, the reward is directly related in terms of the latency of the 5G eMBB environment. If the agent makes decisions that keep the latency within the established ranges, the reward is maximum. Otherwise, zero is awarded as a reward to the agent.

$$\mathcal{R} = \mathcal{R}_n + \mathcal{R}_{bonus} \tag{1}$$

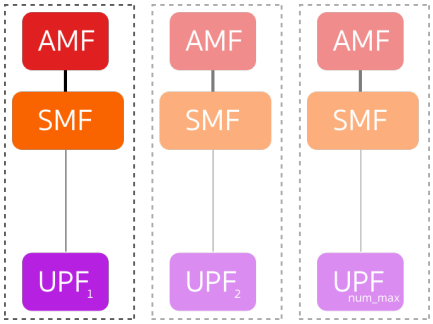
$$\mathcal{R}_n = \begin{cases} 1, & if \quad L_{min} < L < L_{max} \\ 0, & Other\ case \end{cases} \tag{2}$$

$$\mathcal{R}_{bonus} = \begin{cases} 65, & if \quad n > n_{max} - 5 \\ 0, & Other\ case \end{cases} \tag{3}$$

Where:



(a) Scaling granularity: element of NSL



(b) Scaling granularity: whole NSL

Figure 2: Scaling granularity types

$\mathcal{R}_n$: reward per iteration
$\mathcal{R}_{bonus}$: reward bonus per episode.
L: average delay of the service
L_{max}: maximum delay
L_{min}: minimum delay
n: number of steps of an episode.
n_{max}: maximum number of steps per episode.

In the equation 2, the reward $\mathcal{R}_n$ bonus the agent with 1 point if it manages to maintain the latency of the service within the SLA established by the network operator in each iteration. While the equation 3 $\mathcal{R}_b$ rewards the agent with 65 if it reaches at least 5 iterations below the maximum in each episode. The sum of $\mathcal{R}_n$ and $\mathcal{R}_b$ make up the total reward $\mathcal{R}_n$ in the equation 1, and was chosen in initial tests in which showed great precision, simplicity and coherence for the 5G eMBB environment.

2) State Space (S): The state-space represents all the possible positions where the agent observes the 5G environment. In our proposal, this state is obtained by monitoring all the available resources of the N5IM module. Each state is composed of the average latency and the number of active VNFs as follows:

$$S = S_t\{L, n\} \tag{4}$$

Where:

L: average delay of the active VNF.
n: number of active VNF.

L acts as the key variable and indicator of the current QoS of the service. The foregoing, due to its high representativeness and criticality in eMBB services. While n describes the current state of the resources used. For example, if the system has a latency of 200ms with 4 VNFs active, the state is represented as $S = S_t\{200, 4\}$.

3) Action Space (A): The action space comprises the set of all decisions that the DRL agent can make. The actions will be reflected on the 5G UPF user plane function, three decisions can be made: Increase, Decrease or Maintain the number of VNFs. The agent starts by making these decisions randomly while exploring the states and finding relationships that help it learn and maximize the reward, the latter representing the optimal decision.

The space of actions has certain limits in the states. The minimum number of active VNFs is 1 (by design) or a maximum number N is defined by the network administrator.

$$A_t = \{maintain, decrease, increase\} \; \forall \; n \; \epsilon \; (1, N) \tag{5}$$

Where:

$mantener$: no scaling
$bajar$: decrease 1 instance
$subir$: increase 1 instance
n: active VNF number
N: total number of VNF

4) Decay E-greedy Policy: ϵ-greedy is a common policy used in RL and DRL algorithms to schedule a proper learning rate in order to maximize the reward obtained from the N5IM environment. The agent begins with a phase of total exploration taking arbitrary actions and looking for response in the system behavior, the Deep Q-learning must train the NN on the same way if it were a Q-Table, with each new state, the exploration must decay, and the decisions are based on the acquired experience, for our case this policy is linear-formed what means that the schedule decay is inversely proportional to the number of performed iterations. Equation 6 describes this pattern.

$$\epsilon_t = \epsilon_{max} - \frac{\epsilon_{max} - \epsilon_{min}}{T} * t \tag{6}$$

Where T is the maximum number of episodes of sampling, and ϵ_{max}, ϵ_{min} are the extremes between the beginning and ending of agent exploration.

Equation 7 shows the agent limits over the action that it must take. Variable x is a random value that change in each step t where $x \in [0, 1]$. Thereby, the percentage of times that agent must select a random action decays along the number of episodes of training.

$$\mathcal{A} = \begin{cases} DQ_t(S_t, A_t), & if \quad x > \varepsilon_t \\ random\, a_t, & otherwise \end{cases} \tag{7}$$

5) Target and Online Neural networks: Our DRL agent uses two NNs for making the training stable, the goal is loss function is not affected by the modifications to the Q function at current step. Both of them make diverse and complementary

tasks. These NNs are called Online and Target. Online NN creates an approximation function for action-values (Q) in the current state $S = s_t$. On the other hand, Target NN gives the output action-values (Q) for the new state $S = s_{t+1}$. Online NN trains continuously in each step, while Target NN is frozen and only updates its weights periodically. Update means that Target NN sets its weight values as a copy of Online NN weights. Equations 8 and 9 define the next state and loss function for this technique.

$$Q^{+}(S_t, a_t) = R_t + \gamma * maxQ(S_{t+1}, A_{t+1}) \tag{8}$$

$$L = E[((r + \gamma * maxQ(S_t^{'}, A_t^{'})) - Q(S_t, A_t))^2] \tag{9}$$

Both NNs are designed in the same way and share the same structure, input, hidden, and output layers. All their parameters must be equal due to Online NN updates to target NN with its current weights. At the beginning of the learning, these weights are the same in both NNs. The frequency of update target NN depends on the environment to control. Another consideration based on the "Deepmind" design is to clamp the error between -1 and 1 to get a more stable result. So, using the whole previous approach minimize the correlations between action-values and the target-values respect to single NN solutions which present a high-correlation between these variables. Because small updates in Q may represent significant change in policy and data distribution [14].

6) Experience replay: Our agent use it to reduce the high correlation in input data to DQN. This technique batch past data with a specific size and state variables S_t, A_t, R_t, S_{t+1}. DQN uses the batches of data as an input for the NNs, which reduces high correlations between diverse data shreds. For the replay method, the agent selects a random sample from the past data. Then, training becomes more smooth and stable [16]. An optimal training requires that batch size be an intermediate value regarding the total number of learning steps. Otherwise, it will cause under or over replay issues [17]. Thereby if the agent gets in under replay, the experience is useless. On the other hand, if it gets in over replay, agent learning will be unstable.

D. DRL Scale Process Algorithm

This scale process is all the algorithm used for the selection of actions and state changes in SCALA 5G system, the scaling algorithm have several ways to configure it. For our case we focus on two main parameters types. First, we found the learning rate α, this one must be near to 0 for ensure a proper learning curve. Second we set the ϵ-greedy for exploration policy, linear-decay-schedule is one of better ways for complement the learning rate, Algorithm 1, based on Google's DeepMind DQN Algorithm [14] introduces a pseudo-code about the high-level processes into the agent.

Algorithm 1 starts with online and target NN (Lines 1 - 2), where both NN are sized (numbers of layers, number of neurons y activation functions), and NN are initialized. then, episodes begins to run (Line 3). Each time the a new episode begins also enviroment is reseted (Line 4). From Line 5 to 20

Algorithm 1: Auto-scaling based on deep reinforcement learning SCALA 5G

Input : Episode number: M
Exploration parameters: $\epsilon_{max}, \epsilon_{min}$, ϵ_{decay}
discount factor: γ
learning rate: α
update frecuency: C
seed value: σ

Result : Scaling decision over the environment

```
1  Initialize online NN Q with random weights
2  Initialize target NN Q* with weights θ⁻ = θ
3  for episode = 1 to M do
4      Reset environment
5      while not be final state do
6          Agent watch initial state S_i
7          probability = random[0,1]
8          if probability < ε then
9              a_t = random([A_t{out, in, keep})
10         end
11         else
12             a_t = NNQ(S_t))
13         end
14         Send action a_t and life cycle module execute it
15         Get the reward from monitoring module (see
             equation 1)
16         Store transition (s_t, a_t, r_t, s_{t+1})
17         NN optimize Q(S_t) with loss values Q*(S_{t+1})
             see equation 9.
18         Q* = Q each C steps target NN copy the
             weights from online NN
19         s_t ← s_{t+1} updates current state
20     end
21 end
```

contains the iteration step process. At the beginning of each step, agent watches which state is the enviroment in (Line 6). Then, a random number is generated to determine if the agent will explore the environment, else if agent will take the output of NN (Line 7), if random number is less than current ϵ-greedy (Line 8), agent take a random action and no matter the action coherence (Line 9). If random number is more than current ϵ-greedy (Line 11), agent takes the NN output (Line 12).

Action taken previously is sent to lifecycle module (Line 14) to run. then, agent gets the reward from monitoring module with equation 1, it depends directly from latency (Line 15). After, agent stores current parameters (state, action, reward y next state) (Line 16) to feed the agent memory.

In the last part of the step process, agent optimizes online NN according to error respect to target NN (Line 17). This optimization is executed with the loss function (see equation 9). If agent have reached C steps, agent updates target NN with online nn values (Line 18). Finally, algorithm forward to the next step (Line 19).

IV. Environment

This chapter shows the definition of the environment used, how the agent training was performed and the technologies involved in this process. The metrics and tests to evaluate the agent are established and finally an analysis of the results obtained is carried out.

A. Infrastructure

The infrastructure of the test environment corresponds to the eMBB use case, the graph that contains its logical structure is in the picture 3. The *Access and Mobility Management Function* AMF is responsible for receiving requests from the user equipment, the *Session Management Function* SMF is responsible for creating and removing Data Protocol Units and managing sessions with the UPF and the *User Plane Function* UPF represents the evolution of the [18] data plane.

Figure 4 is a variant of the figure 3 and contains the organization at the physical level in the *Kubernetes* cluster of the *Kubernetes-deployments* for an eMBB composed of the VNFs: UPF, SMF, AMF, MARTIR and load balancers. At the physical level, the cluster composed of four nodes is deployed. *Kubernetes* instantiates each replica on the node that has the physical resources to run it. Each node can host different types of replicas. With the configuration defined in *https://gitlab.com/SCALA5G/Kubernetes-infra*, up to 23 replicas can be deployed as follows: 1 replica for SMF, 1 replica for AMF, from 1 to 20 replicas for UPF and 1 replica for the *MARTIR* instance, in charge of obtaining telemetry. Each node is type N: 2 CPUs/8 GB of RAM. This cluster was deployed in the *Google Cloud* public cloud.

Load balancers use a *round-robin* configuration which is a load-balancing technique to assign the same priority to each server or instance [19]. From the eMBB service graph, the element to be scaled is the VNF UPF. *Kubernetes* allows simple scaling through its API and updates the internal IPs of the actives VNFs on the cluster so the load balancers can forward the traffic to the new instances.

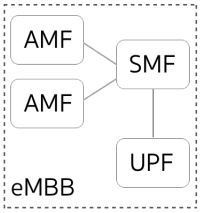


Figure 3: Graph for eMBB use case.

B. Service

The eMBB environment presented is a high definition video service which has a duration of 150ms for the downloading and start-up. Under this criterion, a user can access the mobile network via the web and request the 5G network to be routed to the video service. The network provides typical flow based on ETSI and 3GPP recommendations for data upload and download via the UPF. In the test environment, the real process of a 5G eMBB use case flow was simplified.

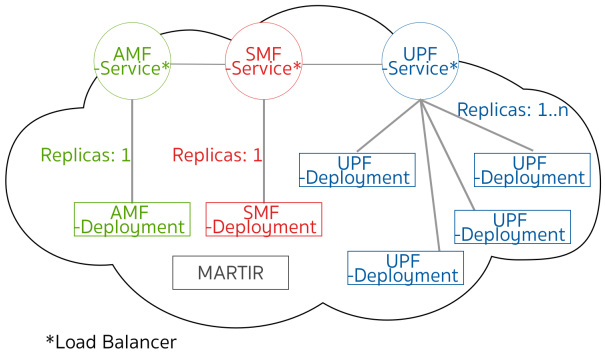


Figure 4: Kubernetes Infrastructure

C. Test traffic

The traffic used for the training and evaluation of SCALA 5G is created synthetically, taking into account ninety commercial traffic patterns obtained from telecommunications operators in Colombia for a network segment located in the surroundings of the city of Barranquilla.

Each real traffic pattern was multiplied by a noise factor generated by a uniform probability distribution. This was done for security and privacy of user information and the company from which the data was obtained. Therefore, no further details can be provided.

The purpose of SCALA 5G is to generalize the behavior of the traffic to have a correct performance in the greatest number of possible scenarios. The equation 10 can generate an infinite number of different traffic patterns, which is useful for the agent to not correlate the output decision with a single traffic pattern. The intervals for the five constants a, a_1, a_2, p_1 and p_2 were chosen after a process of empirical experimentation.

$$a_1 \sin \frac{t}{p_1} + a_2 \sin \frac{t}{p_2} + a \tag{10}$$

Where:

a_1: amplitude 1 $\forall\ \mathbb{R}\ \epsilon\ (1, 90)$
a_2: amplitude 2 $\forall\ \mathbb{R}\ \epsilon\ (1, 90)$
p_1: period 1 $\forall\ \mathbb{R}\ \epsilon\ (2, 11)$
p_2: period 2 $\forall\ \mathbb{R}\ \epsilon\ (2, 11)$
t: time $\forall\ \mathbb{R}\ \epsilon\ (0, \infty)$
a: amplitude $\forall\ \mathbb{R}\ \epsilon\ (1, 180)$

Figure 5 shows four synthetic traffic patterns (derived from the real ones) by using four different values of p_1 with the constants $a = 60$, $a_1 = 45$, $a_2 = 45$, $p_2 = 5$ and the independent variable in the range $t =$ 0–290.

D. Delay per instance.

The test environment must be realistic, in our case the response delay of an instance of the infrastructure deployed in GCP for a given number of users is the variable that was modeled in the equation 11. Delay or latency is the time it takes from the session establishment request to the data response.

$$l = \frac{e^{-1+\frac{u+4}{7.1}} - e^{6-\frac{u}{25}}}{2} + 300 \tag{11}$$

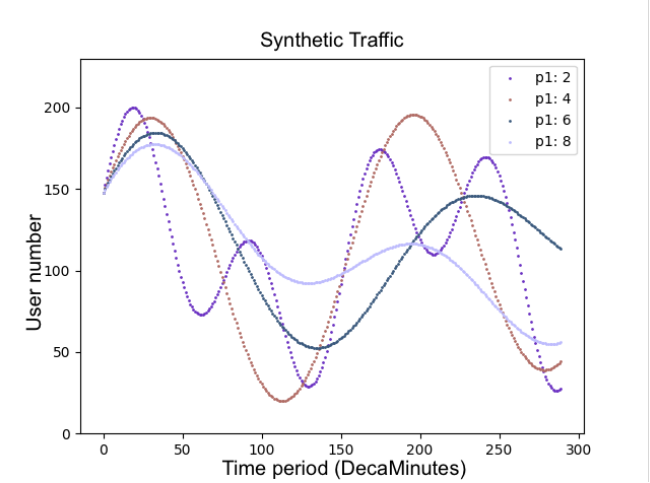


Figure 5: Traffic generated by the equation 10

Where:

l: delay, milliseconds
u: users per instance.

The characterization of the environment to obtain the equation 11 was made from experimentation and the delay time data of an instance was taken using the tool *https://loader.io/* that allows to perform stress and traffic tests on the *Kubernetes* infrastructure deployed on GCP. The figure 6 shows the relationship between the number of users and the delay time of an instance.

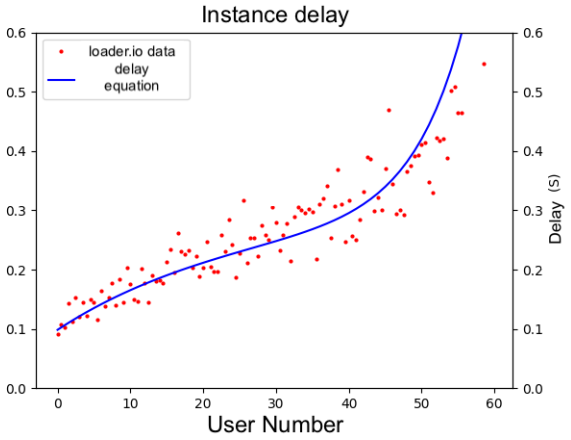


Figure 6: Instance delay per user number.

E. SCALA 5G Gym

To conduct SCALA 5G training and subsequent evaluation, a gym [20] was built, which is a synthetic testing environment that SCALA 5G uses to interact and train.

The state S_t of the gym is composed of the delay l and the total number of active instances n_t according to the equation 4. The state S_t is modified by an action a_t, the delay l is calculated using the equation 11 and depends on the number of total users u_t and the number of total instances n_t. SCALA 5G can only modify the total number of instances n_t.

The reason behind the gym's S_t state doesn't take into consideration parameters like CPU/RAM, is because GCP and AWS may have different types of CPU/RAM setups for their nodes, so SCALA 5G is agnostic to the platform on which it is deployed.

F. SCALA 5G training.

SCALA 5G is a DQN agent. To carry out its training, the interaction between SCALA 5G and the gym of the IV-E subsection must occur. The reward is calculated taking into account the SLA according to the equation 1. In each iteration, the loss function calculates the loss using the values of S_t, S_{t+1}, r and a_t that are the product of the interaction between SCALA 5G and the gym, in this way SCALA 5G figures out if the actions it is making maximize the reward r. If the actions are correct, the weights of the SCALA 5G neural networks are slightly modified. If they are wrong, the weights are modified. For SCALA 5G training, an SLA of $L_{max} = 320ms$ and $L_{min} = 200ms$ was set empirically, these values can be determined by the administrator user taking into account the graph 6 and the SLA requirements of his network.

At the beginning of the training, SCALA 5G collects the state S_t and takes an action a_t between: scaling the number of instances, reducing the number of instances or keeping the number of instances, the action a_t is sent to the gym. The gym produces a status S_{t+1} and a reward r as a response, SCALA 5G uses S_t, S_{t+1}, r and a_t to learn. In each episode, 48-hour traffic patterns are used, that is, 288 steps. The training ends when the condition of the equation 12 is met, that is, when the average number of steps of the last 14 episodes is greater than 283. The training data is synthetic and is generated with the equation 10. This equation allows to generate a traffic pattern of different length, each traffic pattern is divided into 10 minute intervals. Two gyms were built: the SCALA 5G gym and the legacy gym. The SCALA 5G gym uses 48-hour traffic patterns, that is, 288 intervals, and the legacy gym uses 24-hour traffic patterns, that is, 144 intervals.

$$p_{av14} > 283 \tag{12}$$

Where:

p_{av14}: average number of steps on the last 14 episodes.

The delay l can take the range of values of 13 during training:

$$L_{min} * 0.85 < l < L_{max} * 1.15 \tag{13}$$

Where:

l: delay $\forall\ \mathbb{R}\ \epsilon\ (150, 400)$

An episode ends if SCALA 5G goes through the entire traffic pattern, in other words when it reaches step 288 or if delay obtained from the gym that exceed 400 milliseconds or are less than 150 milliseconds.

Therefore, if SCALA 5G passes to step 283 in an episode, it means that it chose the right actions. If the episode ends early, it is because SCALA 5G made the wrong decisions, for example, it did not increase the number of servers when there

	SCALA 5G	Legacy Agent
Neurons Number	128	64
Bonus Reward	65	30
Memory Replay	Si	No
NN Target	Si	No
Episode Number	Dynamic	1000
Steps per Episode	288	144
Learning Rate	0.007	0.007
Input dimension for the Neuronal Network	2	2
Output dimension for the Neuronal Network	3	3
Update frequency NN Target	100	-
Factor γ	0.9	0.9

Table II: SCALA 5G and Legacy Agent parameters

is a peak on the traffic and this resulted in a delay greater than 400 milliseconds. Ending the episode when the agent starts to continually make wrong decisions allows the agent not to reinforce taking wrong actions. To enhance continuous sound decision-making, an extra reward of 65 (empirically adjusted value) is given if SCALA 5G reaches step 283.

Gyms are used to train algorithms in DRL environments, the gym used extends from Elon Musk's *OpenAI Gym* [21] library.

For benchmarking the SCALA 5G algorithm, a classic DRL algorithm, called Legacy Agent, is used as a reference. This agent interacts with the legacy gym for training, the characteristics of the Legacy Agent and SCALA 5G can be found in the II table.

G. Metrics

The definition of the right metrics is important to ensure compliance with the objectives set out in the project. Thus, SCALA 5G is evaluated in terms of convergence time, accuracy and adaptability.

- **Convergence time**: determines the time from when the algorithm starts until the end of its learning, it is evaluated with the metric that bears the same name.
- **Accuracy**: This aspect supports the correct operation of SCALA 5G and is associated with the **service level satisfaction** metric.
- **Adaptability**: The evaluation of this aspect determines how viable it is to deploy SCALA 5G in different cloud providers. The metrics that evaluate this aspect are **service level satisfaction** and **unit test coverage**

1) Convergence time: The convergence time *Convergence Time* (CT) measures the number of episodes it takes for the DRL agent to reach the final learning phase. Each episode has a run time t_i that depends on the execution environment (processor, memory, programming language).

$$CT = \sum_{i=0}^{i_{ct}} 1 = i_{ct} \tag{14}$$

$$CT(t) = \sum_{i=0}^{i_{ct}} t_i \tag{15}$$

Where:

i_{ct}: number of episodes to converge
t_i: time elapsed for an episode i

The equation 14 shows the convergence time calculated in number of episodes and the equation 15 as a function of time. As an objective, it is established that SCALA 5G must converge in less than 5000 episodes, and in terms of time, it is considered that it converges quickly if the total duration is less than 10 minutes. Agents that solve similar problems as *cart-pole* converge in the same order of magnitude of time and number of episodes [22] [23].

2) Service level agreement: .

The second metric in $\%SLA$ refers to the percentage of user requests served within the established SLA.

$$\%SLA = \frac{R_+}{R_{total}} = \frac{\sum_{i=0}^{n} R+_i}{\sum_{i=0}^{n} Rtotal_i} \tag{16}$$

Where:

R_+: number of request handled inside the SLA.
R_{total}: total number of request handled.

H. Unit test Coverage

The third metric used to evaluate our algorithm is the coverage test *coverage* of unit tests. The *coverage* is a measure of how many lines of code are executed when the unit tests[1] are running. As an example of coverage, if a file is made up of 100 lines of code, and the unit tests interact with 90 lines, it will have a coverage of 90%. Therefore, most of the code in that file will be protected by unit tests. If a contributor introduces a bug in the file in the future, the unit tests will fail and the developers will be able to tell that the code is behaving unexpectedly and fix it.

The *coverage* allows to guarantee the quality of the programming code made so that on future academic and commercial works they can easily replicate, modify or use specific modules of the solution. Unit tests document the original intent of the code and ensure that future code modifications do not break the system. According to [24] a good coverage measurement should be above 70%, and it is recommended to go above 80% for commercial *software*.

$$\%C = \frac{C_+}{C_{total}} \tag{17}$$

Where:

C_+: number of lines of code that are executed during unit testing execution.
C_{total}: total number of lines of code.

V. SCALA 5G: Analysis and results

.

To perform this analysis, the previously mentioned in the IV-G section are established as metrics. SCALA 5G operates with the hyperparameters of the table II. These values were obtained based on other DRL well-known set of problems such as *cart-pole* and *climb-mountain* and carefully modified empirically to reach the final values.

[1]method of testing a fraction of code to determine if it serves its intended purpose

	SCALA 5G	Legacy Agent
Episodes	1740	1000
Possibles steps	288	144
Convergence time	3 minutes and 27 seconds	1 minute y 2 seconds

Table III: SCALA 5G and Legacy Agent convergence.

A. *Convergence time*

The number of steps per episode is considered a convergence metric because the gym ends an episode if the Agent which interacts with it selects wrong actions consecutively. So, a high number of steps per episode represents that the agent is making the right decisions. SCALA 5G has a variable number of episodes, the training ends when the condition of the equation 12 is fulfilled. The Legacy Agent has a set number of 1,000 training episodes. The table III and the figure 7 present the convergence of SCALA 5G and the Legacy Agent. In the figure 7 for each episode, a value of 1.0 is assigned if the agent goes through all possible steps, 288 steps for SCALA 5G, and -1.0 if the agent fails.

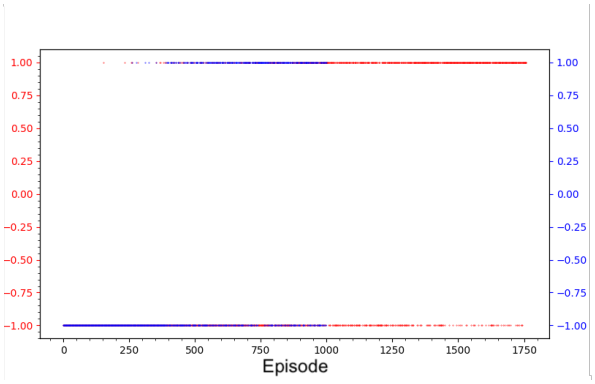


Figure 7: SCALA 5G and Legacy Agent convergence.

SCALA 5G uses more time than the Legacy Agent during his training this is because SCALA 5G has a dynamic number of episodes. However, SCALA 5G performs better in all traffic patterns tested (see section V-B). Therefore, the additional time that SCALA 5G takes to train translates into better results. It should be noted that SCALA 5G converges quickly since it takes less than 5 minutes, which is less than the limit established in the section IV-G1.

B. *Service Level Agreement*

To test the performance of SCALA 5G in terms of SLA compliance, the comparison with a basic DRL algorithm: the Legacy Agent. For this comparison, four different traffic functions generated by the equation 10 were tested.

Figure 10 shows a comparison of how SCALA 5G and the Legacy Agent handle the number of servers given the traffic of 9, SCALA 5G modified the number of VNFs 7 times without having rebound problems, while the Legacy Agent exceeded 20 modifications, having bounces (actions of increasing and decreasing the number of instances in a repetitive manner) in all the decisions that the Legacy Agent takes.

Figure 8 shows the level of service satisfaction of SCALA 5G with respect to the Legacy Agent.

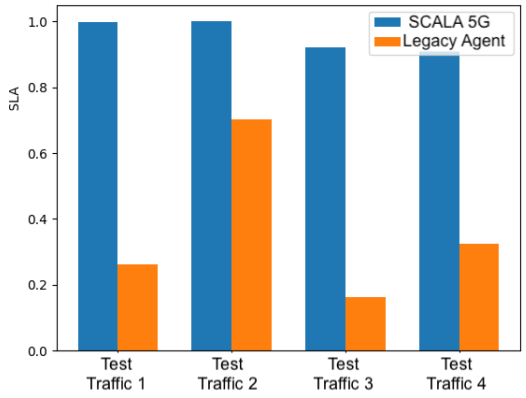


Figure 8: SLA SCALA 5G vs Legacy Agent

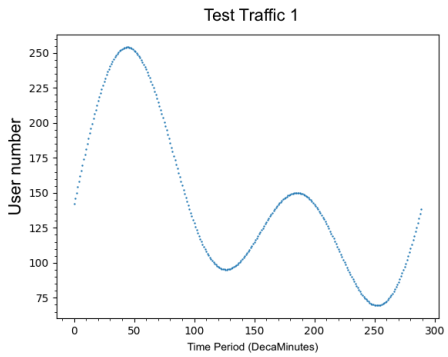


Figure 9: Test traffic 1

- SCALA 5G was trained with a 48-hour traffic pattern, having 288 slots in total, double than of the Legacy Agent.
- SCALA 5G does not have a fixed number of episodes, the training ends if SCALA 5G reaches at least interval 283 for 14 consecutive episodes, this ensures that the weights of the neural networks are at a point where the problem is solved satisfactorily. The Legacy Agent trains for 1000 episodes in a fixed manner and without taking into account another variable such as the average performance at that point.
- SCALA 5G uses elements like *memory replay* and *target network* obtaining the advantages previously mentioned in the subsection III-C6.
- SCALA 5G gives twice the reward of the Legacy Agent, when SCALA 5G reaches step 283 of the traffic for the episode.
- SCALA 5G has 128 neurons and the Legacy Agent has 64 neurons.

Finally, SCALA 5G was compared with a threshold scaling algorithm, in this case 2 episodes were generated.

Figure 12 shows the first scenario of SCALA 5G compared to the threshold algorithm. The traffic pattern of the figure 11 has a sinusoidal behavior with 2 equal peaks of 190 users at intervals 50 and 280. SCALA 5G reacts early when a scaling decision is required and can anticipate the right

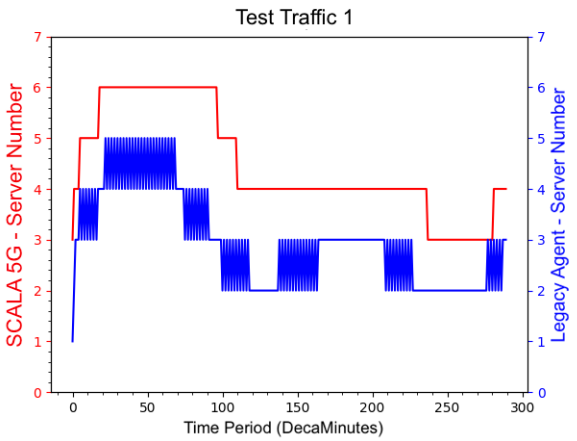


Figure 10: Servers comparison for test traffic 1, SCALA 5G v. Legacy Agent

decision. In relation to the number of VNF, SCALA 5G made 8 modifications and the threshold algorithm 11.

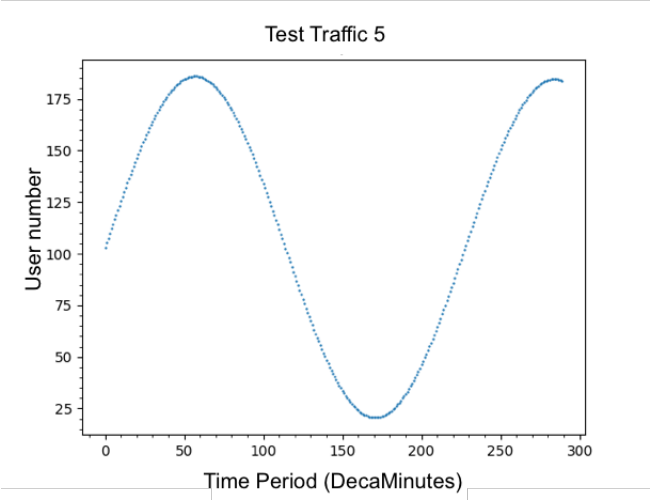


Figure 11: Test traffic 5

Figure 13 shows that both agents had outstanding percentages of satisfaction. However, SCALA 5G outperforms the algorithm by thresholds by 2% in scenario 5 and 1% in scenario 6. SCALA 5G does a better job in the analyzed traffic patterns, for scenario 5, it meets 98% while the threshold agent is 96%, in scenario 6 traffic, SCALA 5G has 87% and the threshold agent is 86%.

C. Unit test Coverage

coverage metric have a direct way to measure over the code, using unit test libraries. So, to get the coverage mean we build 23 test, report is executed using the command *coverage run –source=./ -m unittest discover -s tests/ && coverage html*

table IV shows coverage report of SCALA 5G, agent has a 97% of covered code with almost 1 unit test. This accomplish with the objective to guarantee reliability in written code.

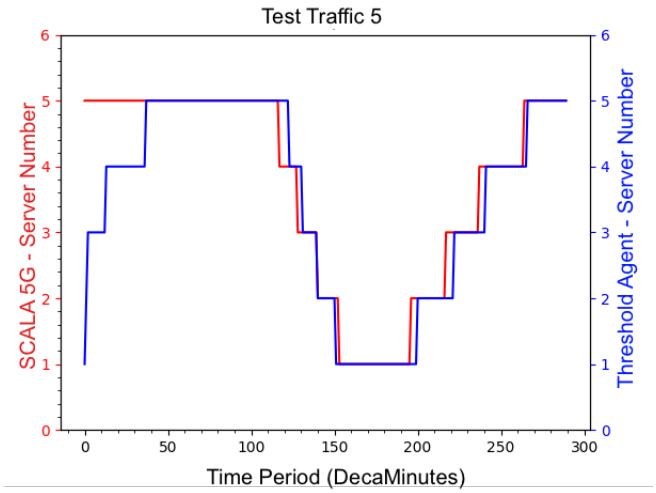


Figure 12: Servers comparison for test traffic 5, SCALA 5G v. threshold agent

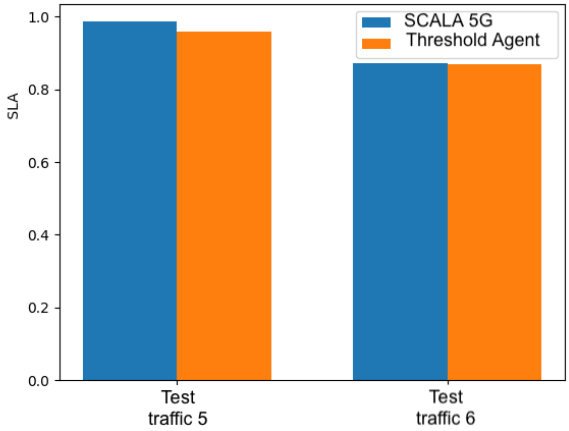


Figure 13: SLA SCALA 5G vs Threshold Agent

VI. Conclusions and Future Work

This paper has introduced SCALA 5G, an DRL-based auto-scaling mechanism to 5G eMBB network slicing. SCALA 5G algorithm demonstrated supremacy over Legacy Agent and Threshold Agent over evaluation environments. It makes a big difference in the more complex traffic patterns. Compliance with the SLA under the conditions described above is 97%, ensuring correct performance in environments with different traffic patterns. Furthermore, convergence time is low and SCALA 5G can quickly learn new traffic patterns. So, the choice of the delay and the number of users as components of a state S_t and S_{t+1} provides enough information for SCALA 5G to determine if it is making correct decisions and therefore converge.

For future work, we plan continue with new contributions and enhancements to get a complete solutions that includes URLLC and mMTC cases of 5G and improve SCALA 5G algorithm to become a commercial solution.

Filename	Coverage
dqn_5g_environment/VideoStreamingEnvironment.py	95 %
dqn_5g_environment/__init__.py	100 %
dqn_agent/__init__.py	100 %
dqn_agent/dqn_agent.py	96 %
dqn_improvers/ExperienceReplay.py	100 %
dqn_improvers/__init__.py	100 %
dqn_training/__init__.py	100 %
dqn_training/train_video_server_dqn.py	98 %
math_helpers/__init__.py	100 %
math_helpers/epsilon_decay.py	100 %
neural_network_/NeuralNetwork.py	100 %
neural_network_/__init__.py	100 %
plotting/__init__.py b	100 %
tune_parameters.py	100 %
TOTAL	97 %

Table IV: Coverage report

REFERENCES

[1] J. Kim, S. Kim, and H. Lim, "Reinforcement learning based resource management for network slicing," *Applied Sciences*, 2019.

[2] C. Bouras, A. Kollia, and A. Papazois, "Sdn nfv in 5g: Advancements and challenges," *2017 20th Conference on Innovations in Clouds, Internet and Networks (ICIN)*, Mar. 2017.

[3] T. Buyakar, A. Rangisetti, A. Franklin, and B. Tamma, "Auto scaling of data plane VNFs in 5g networks," in *2017 13th International Conference on Network and Service Management (CNSM)*, Nov. 2017, pp. 1–4.

[4] M. Singh, S. Vittal, and A. Franklin, "Serens: Self regulating network slicing in 5g for efficient resource utilization," *2020 IEEE 3rd 5G World Forum*, 2020.

[5] J. Zhou, W. Zhao, and S. Chen, "Dynamic network slice scaling assisted by prediction in 5g network," *IEEE Access*, 2020.

[6] N. Salhab, S. El Falou, R. Rahim, S. El Ayoubi, and L. Rami, "Optimization of the implementation of network slicing in 5g ran," *IEEE Middle East and North Africa Communications Conference*, 2020.

[7] J. Yao and M. Chen, "A flexible deployment scheme for virtual network function based on reinforcement learning," *IEEE 6th International Conference on Computer and Communications*, 2020.

[8] Z. Xiong, Y. Zhang, D. Niyato, R. Deng, P. Wang, and L. Wang, "Deep Reinforcement Learning for Mobile 5g and Beyond: Fundamentals, Applications, and Challenges," *IEEE Vehicular Technology Magazine*, vol. 14, no. 2, pp. 44–52, Jun. 2019.

[9] K. Qu, W. Zhuang, X. Shen, X. Li, and J. Rao, "Dynamic resource scaling for vnf over nonstationary traffic: A learning approach," *IEEE Transactions on Cognitive Communications and Networking*, 2020.

[10] A. Aslan, G. Balce, and C. Toker, "Dynamic resource management in next generation networks with dense user traffic," *IEEE International Black Sea Conference on Communications and Networking*, 2020.

[11] W. Villota, O. Caicedo, and N. Saldanha, "Admission control for 5g network slicing based on (deep) reinforcement learning," 2020.

[12] V. Nguyen, A. Brunstrom, K. Grinnemo, and J. Taheri, "SDN/NFV-Based Mobile Packet Core Network Architectures: A Survey," *IEEE Communications Surveys and Tutorials*, vol. 19, no. 3, pp. 1567–1602, Apr. 2017.

[13] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *DeepMind*, 2013.

[14] V. Mnih, K. Kavukcuoglu, D. Silver, A. Rusu, J. Veness, M. Bellemare, A. Graves, M. Riedmiller, A. Fidjeland, G. Ostroviski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, 2015.

[15] "5g; Service requirements for next generation new services and markets," *(3GPP TS 22.261 version 15.5.0 Release 15)*, p. 53, Jul. 2018.

[16] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *NIPS Deep Learning Workshop 2013*, Dec. 2013.

[17] L.-j. Lin, "Reinforcement learning for robots using neural networks," *Carnegie Mellon*, 1993.

[18] G. Brown, "Service-based architecture for 5g core networks," *A Heavy Reading white paper produced for Huawei Technologies Co. Ltd. Online: https://www.huawei.com/en/press-events/news/2017/11/HeavyReading-WhitePaper-5G-Core-Network*, vol. 1, pp. 1–12, 2017.

[19] S. Kaur, K. Kumar, J. Singh, and N. S. Ghumman, "Round-robin based load balancing in software defined networking," in *2015 2nd international conference on computing for sustainable global development (INDIACom)*. IEEE, 2015, pp. 2136–2139.

[20] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.

[21] E. Musk, "Openai gym https://gym.openai.com/," dec 2015.

[22] S. Kumar, "Balancing a cartpole system with reinforcement learning–a tutorial," *arXiv preprint arXiv:2006.04938*, 2020.

[23] Í. M. Miranda, M. Ladeira, and C. de Castro Aranha, "A comparison study between deep learning and genetic programming application in cart pole balancing problem," in *2018 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 2018, pp. 1–7.

[24] C. Zhang, Y. Yan, H. Zhou, Y. Yao, K. Wu, T. Su, W. Miao, and G. Pu, "Smartunit: Empirical evaluations for automated unit testing of embedded software in industry," in *2018 IEEE/ACM 40th International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP)*, 2018, pp. 296–305.