INSTITUT POLYTECHNIQUE DE PARIS

NNT : 2026IPPAT023

Thèse de doctorat

Methods for Sizing and Deploying Virtualised Mobile Networks: Application to 5G/6G, Optimisation for Smart Electrical Grids

Thèse de doctorat de l'Institut Polytechnique de Paris
préparée à Télécom Paris

École doctorale n°626 École doctorale de l'Institut Polytechnique de Paris (ED IP Paris)
Spécialité de doctorat : Mathématiques et Informatique

Thèse présentée et soutenue à Palaiseau, le 15/06/2026, par

ROSY ALTAWIL

Composition du Jury :

Laurent Decreusefond Professeur Institut Polytechnique de Paris, Télécom Paris	Président/Examinateur
Thi-Mai-Trang Nguyen Professeure des universités Université Sorbonne Paris Nord	Rapporteure
Xavier Lagrange Professeur IMT Atlantique	Rapporteur
Nicolas Bihannic Ingénieur Orange	Examinateur
Philippe Martins Professeur Institut Polytechnique de Paris, Télécom Paris	Directeur de thèse
Sawsan Al Zahr Docteure Institut Polytechnique de Paris, Télécom Paris	Co-encadrante de thèse
Wilfried Frelin Ingénieur EDF	Invité

Dedicated to Teta Mehabe and Sawsan El Zahr
For being there at the start,
though you left too soon to see the end.
You are deeply missed and forever honoured in these pages.

Acknowledgements

I would like to express my deepest and most sincere gratitude to my thesis supervisor, Prof. Philippe Martins. He taught me how to navigate the many technical and personal challenges we faced over the last three years. His mentorship has been a cornerstone of my growth, and I feel privileged to have learned so much under his supervision.

I also honour the memory of my late co-supervisor, Dr. Sawsan El Zahr, whose early vision laid the foundation for this work, her absence is deeply felt. My thanks also go to Prof. Laurent Decreusefond for his crucial insights during the initial stages of this work.

I would also like to thank the reviewers, Prof. Thi-Mai-Trang Nguyen and Prof. Xavier Lagrange, for accepting the task of reviewing this manuscript and for the time they will dedicate to evaluating this research. I look forward to their expertise and constructive feedback.

I am equally grateful to my industrial supervisor, Vincent Audebert, for providing the opportunity and environment to conduct this research. I also am grateful for the whole EDF R&D Saclay team, especially to Lucas Delolme, whose hard work and support were invaluable throughout these three years. I also extend my gratitude to the EDF lab in Chatou, specifically Hichem and Joselin, for their warm welcome and all their help.

This work would not have been possible without the generous support of Rohde & Schwarz and Ericsson, who provided access to their specialised equipment and technical expertise to support the setup.

To my parents, thank you for paving the way so your children could accomplish what you could not; this milestone is yours as much as mine. To my brothers and role models, Chady and Tony: I truly would not be here without you. You are the reason I could pursue this PhD, and I am eternally grateful for your sacrifices and strength. I also thank my sisters-in-law, Sarah and Pierette—with a special thought for Pierette in her own doctoral journey—and my nephews, Raphael and Marcus, for always giving me a much-needed reason to smile.

To Melissa, my sister, thank you for being around through the countless ups and downs of the last three years.

Finally, my deepest thanks go to Adrien. Thank you for your constant company and emotional support, especially during the lowest of times. Your presence made the finish line possible.

Contents

Résumé étendu **10**

Publications **24**

List of Figures **27**

List of Tables **28**

1 Introduction **29**
1.1 Context and Motivation 29
1.2 Problematics and Objectives 30
1.3 Structure of the Thesis 31

2 Simulation Model for Resource Allocation **33**
2.1 Introduction 33
2.1.1 System Notations 33
2.1.2 5G Network Slicing 33
2.1.3 5G System Architecture and Radio Interface 35
A. Control Plane 37
B. User Plane 39
2.1.4 Network Slicing Implementation at gNB level 40
2.1.5 Mathematical Background 41
A. Queuing Theory 41
B. Markov Process and Markov Decision Process . . 42
C. Optimization and Bellman's Equation 43
2.2 State of the Art 44
2.3 System Model 45
2.4 Simulation and results 49
2.4.1 Simulation methodology 49
2.4.2 Simulation algorithm 49
2.4.3 Results and interpretation 50
2.5 Conclusion 53
2.5.1 Model Limitations 54

3 Slice Classification Model **55**
3.1 Introduction 55
3.1.1 System Notations 55
3.1.2 5G QoS and Low Layer Identifiers 55
3.1.3 Machine Learning 57

3.1.4 LGBM Algorithm . . . 58
A. Optimization via Gradient Boosting . . . 58
B. Leaf-wise Growth Strategy . . . 59
C. Exclusive Feature Bundling (EFB) . . . 59
D. Gradient-based One-Side Sampling (GOSS) . . . 60
3.1.5 Evaluation Metrics . . . 60
3.2 State of the Art . . . 63
3.2.1 Classification with information access from all layers . . . 63
3.2.2 Classification using only information from low layers . . . 63
3.3 System Model . . . 65
3.3.1 Input Dataset Generation . . . 65
A. Dataset Generation Using a Rohde & Schwarz Mobile and Network . . . 65
B. Dataset Generation Using an Amarisoft UE Simulator and an Ericsson Network . . . 67
3.3.2 Model and Training . . . 68
A. LGBM Model Algorithm . . . 71
3.3.3 Results and Interpretation . . . 77
A. Results Using the Rohde & Schwarz Mobile and Network Dataset . . . 78
B. Results Using the Rohde & Schwarz Mobile and Network Dataset without the HARQ Identifier . . . 80
C. Results Using then Amarisoft UE Simulator and Ericsson Network Dataset . . . 83
3.4 Conclusion . . . 87

4 Deep Reinforcement Learning for Resource Allocation Model 88
4.1 Introduction . . . 88
4.1.1 System Notations . . . 91
4.1.2 Deep Reinforcement Learning . . . 91
4.1.3 Deep Q-Network Algorithm . . . 92
A. Optimization Using the Bellman Equation . . . 93
B. Experience Replay . . . 93
C. Target Network Stability . . . 94
4.1.4 Performance and Stability Metrics . . . 94
4.2 State of the Art . . . 95
4.3 System Model . . . 97
4.3.1 DQN Model Algorithm . . . 99
4.4 Simulation and Results . . . 104
4.5 Conclusion . . . 109

5 Conclusion 110

A Algorithm Presented in Chapter 2 112

B Algorithms Presented in Chapter 3 113

C Algorithms Presented in Chapter 4 118

Abstract

The modernisation of industrial smart grids, such as those operated by EDF, necessitates highly reliable communication for Protection, Automation, and Control (PAC) systems. Although 5G network slicing enables support for mission-critical messages such as Ultra-Reliable Low Latency Communications (URLLC), industrial clients lack visibility into the operator's core network, making it challenging to guarantee the needed 10^{-5} URLLC packet loss probability. Additionally, traditional resource allocation models rely on theoretical Poisson's law traffic assumptions, which do not accurately represent the burstiness of real-world traffic. This difference results in a resource deficit approximately 30% of the time.

To address these limitations and solve the visibility gap, a Machine Learning (ML) classification model was developed. Due to restricted access to real-life network traces, a Light Gradient-Boosting Machine (LGBM) model was implemented to classify network traffic into URLLC and eMBB slices using data exclusively from the lower PHY and MAC layers from an experimentally generated traffic. Achieving a 97.6% classification accuracy shows that industrial clients can independently monitor their traffic without requiring operator-level KPIs.

Furthermore, building on this realistic classification, a dynamic resource allocation model was introduced using Deep Reinforcement Learning (DRL). A Deep Q-Network (DQN) agent was trained to interact continuously with the network environment. By operating within a continuous state space defined by URLLC load, eMBB load, and normalised cell capacity, the agent effectively balances the exploration-exploitation trade-off. This allows for the dynamic allocation of the optimal number of guard channels required to minimise URLLC packet loss while maintaining fairness toward the eMBB buffer.

By bringing together theoretical queuing limits, ML traffic classification, and DQN, this work delivers a complete solution. This thesis gives industrial clients the ability to take control of their 5G network slices, allowing them to autonomously predict, dimension, and adjust their resources instantaneously.

Résumé étendu

Introduction

Depuis la découverte et la mise en service initiale de l'électricité à la fin du XIXe siècle, la demande et les besoins en matière de réseau électrique n'ont cessé d'augmenter, poussant les pays du monde entier à accroître leur production pour répondre à ces besoins croissants.

Récemment, l'intégration croissante des énergies renouvelables dans le réseau électrique a entraîné le passage d'un modèle centralisé, qui reposait sur des sources d'énergie prévisibles (telles que le charbon et les combustibles fossiles), à un modèle décentralisé, qui s'appuie sur des sources renouvelables intermittentes (telles que l'éolien et le solaire). Cette nouvelle instabilité et variabilité de la production exigent un flux d'informations plus important que les anciennes infrastructures ne peuvent gérer. Le vieillissement des infrastructures, l'intégration des énergies renouvelables et l'augmentation des besoins en communication bidirectionnelle entre les gestionnaires de réseau de distribution et les clients entraînent une demande croissante de systèmes de distribution d'électricité plus efficaces, fiables et durables. L'évolution des systèmes traditionnels vers des réseaux intelligents constitue un moyen de relever ces défis.

Un réseau intelligent est un réseau électrique connecté qui permet un échange d'informations bidirectionnel, permettant ainsi aux consommateurs d'avoir davantage de contrôle sur leur consommation [**?**].

Au cœur du réseau intelligent se trouvent les systèmes Protection, Automation, and Control (PAC); ils constituent un élément important du réseau électrique et sont conçus pour garantir la protection des actifs, la fiabilité et la stabilité, pour améliorer principalement l'efficacité opérationnelle du réseau intelligent [**?**].

Pour garantir la stabilité requise, les systèmes PAC nécessitent des connexions robustes sur l'ensemble du réseau électrique. Bien que la fibre optique offre des performances optimales, son déploiement sur une infrastructure aussi vaste est très coûteux et prend beaucoup de temps, ce qui pousse les fournisseurs de réseaux électriques à rechercher une solution sans fil. Cependant, les technologies plus anciennes comme la 4G ne peuvent pas garantir la faible latence nécessaire pour ces messages critiques. Avec l'introduction du slicing en 5G, en particulier la tranche Ultra-Reliable Low Latency Communications (URLLC), la latence requise peut être atteinte tout en bénéficiant de la flexibilité d'une connexion 5G sans fil.

Même si le slicing offre une solution prometteuse pour la surveillance et la protection du réseau électrique, son déploiement nécessite une coordination complexe entre le DSO et le client, ce qui peut entraîner un conflit d'intérêts majeur.

L'opérateur cherchera à optimiser l'efficacité de son infrastructure, tandis que le fournisseur d'électricité privilégiera des exigences strictes en matière de QoS afin de garantir une faible probabilité globale de perte. Cela est particulièrement vrai dans le cas de messages critiques, tels que le trafic URLLC qui exige une latence et une probabilité de perte très faibles, ce qui peut s'avérer exigeant et coûteux à mettre en œuvre.

De plus, le fournisseur d'électricité, en tant que client et non en tant qu'opérateur, dispose d'une visibilité et d'un contrôle très limités sur la politique d'allocation des ressources au-delà de la couche physique. Pour protéger son réseau, le client doit négocier ces politiques avec l'opérateur avant le déploiement du réseau.

Cependant, il est difficile de négocier correctement les besoins et les exigences du réseau électrique intelligent.

Tout d'abord, étant donné que le slicing n'est pas encore largement déployé, un modèle analytique est nécessaire pour anticiper le trafic et ses besoins. Depuis des décennies, les processus de Poisson sont la référence en matière de simulation du trafic réseau. Cependant, même si un tel modèle peut fournir une première vision nécessaire des besoins attendus, il ne parvient toujours pas à saisir la nature par intermittence et imprévisible du trafic industriel réel. Cela soulève la nécessité de comparer les lois de Poisson au trafic réel, afin de déterminer si un processus de Poisson est optimiste, pessimiste ou suffisamment précis.

Deuxièmement, le client est confronté à un manque de visibilité sur le trafic, comme seul l'opérateur a accès au réseau central et aux identifiants de paquets de haut niveau, tels que l'ID du slice. Le client n'a accès qu'aux couches PHY et MAC inférieures, avec des informations très limitées. Cette vision restreinte rend difficile pour le client de distinguer son propre trafic de celui de l'opérateur afin de vérifier si la politique d'allocation des ressources convenue est bien mise en œuvre.

Par conséquent, l'objectif de cette thèse est de fournir au client EDF un outil de dimensionnement qui résout ces problèmes. Le premier objectif de ce travail est de développer une simulation théorique du processus de Poisson comme première base de référence pour les besoins en allocation de ressources. Ensuite, un modèle d'apprentissage automatique capable de classer le trafic entre les différentes tranches est développé afin de combler le manque de visibilité et de permettre au client de quantifier le trafic transitant. Ce trafic classé est ensuite utilisé pour calculer la politique d'allocation optimale des ressources et être comparé à la simulation théorique du processus de Poisson.

Dimensionnement des ressources pour les tranches URLLC

L'arrivée de la 5G a permis le déploiement du "Network Slicing", qui permet à plusieurs réseaux virtuels de coexister sur une infrastructure physique partagée. Chaque tranche peut être configurée en fonction des exigences d'une catégorie de service spécifique. Parmi les trois principales slices définies par le 3GPP, eMBB se concentre sur les applications à haut débit, tandis que URLLC cible les services critiques nécessitant des garanties strictes en matière de latence et de fiabilité. Les applications de protection, d'automatisation et de contrôle des

réseaux intelligents appartiennent à cette dernière catégorie et nécessitent donc des liaisons de communication hautement fiables.

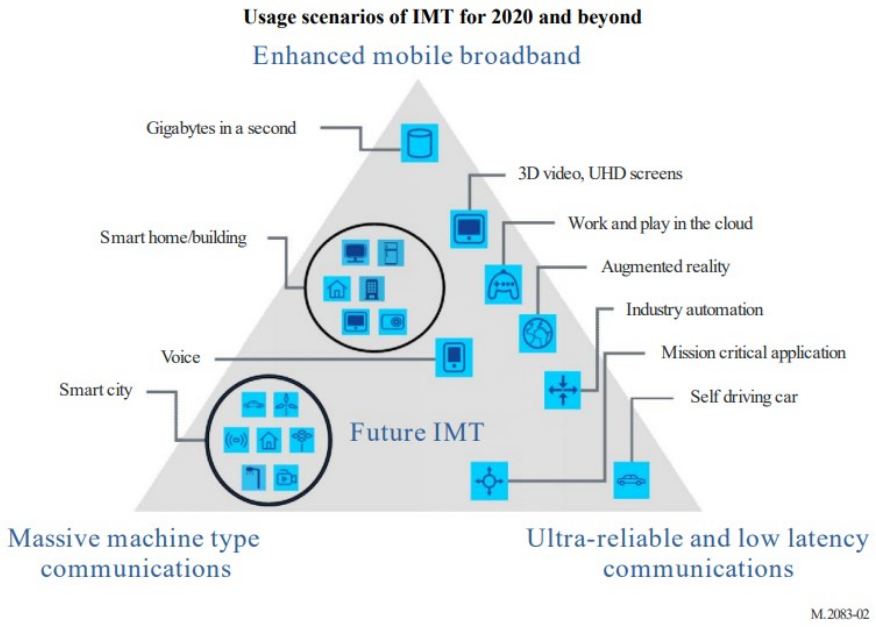


Figure 1: Les tranches 5G et leurs principales applications.

La plupart des études existantes sur l'allocation des ressources et le découpage du réseau sont élaborées du point de vue de l'opérateur mobile. Leur objectif est généralement de maximiser l'utilisation des ressources, l'efficacité spectrale ou les revenus de l'opérateur. Cependant, les acteurs industriels tels que EDF sont généralement des clients d'un opérateur mobile plutôt que des propriétaires de l'infrastructure de communication. Par conséquent, leurs besoins diffèrent considérablement. Du point de vue du client, l'objectif n'est pas d'optimiser l'ensemble du réseau, mais plutôt de déterminer la quantité de ressources radio nécessaires pour garantir un niveau souhaité de QoS avant de négocier un contrat de service avec un opérateur.

Pour relever ce défi, un modèle de dimensionnement des ressources centré sur le client et basé sur la théorie des files d'attente a été développé. Le système considéré consiste en une cellule 5G prenant en charge à la fois le trafic eMBB et URLLC. Les arrivées de paquets sont modélisées comme des processus de Poisson indépendants avec des taux d'arrivée λ_U et λ_e, tandis que les temps de service suivent des distributions exponentielles avec des taux de service μ_U et μ_e. La cellule contient au total S RBs disponibles.

L'approche proposée repose sur le concept de canaux de garde. Un nombre G de RB est réservé au trafic URLLC afin de garantir sa transmission même dans des conditions de charge eMBB élevée. Les paquets URLLC bénéficient d'un accès prioritaire aux ressources disponibles, tandis que les paquets eMBB peuvent être temporairement envoyés dans une file d'attente lorsque les ressources partagées sont saturées.

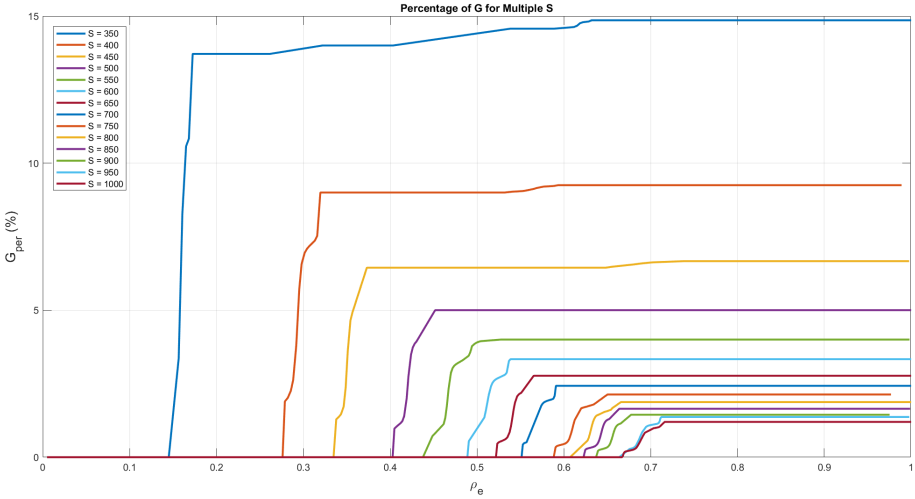
L'état du système est représenté par le vecteur (x_1, x_2, x_3), où x_1 désigne le nombre de paquets URLLC en cours de traitement, x_2 le nombre de paquets eMBB en cours de traitement, et x_3 le nombre de paquets eMBB mis en file d'attente. Étant donné que les transitions futures dépendent uniquement de l'état

actuel et non des états précédents, le système peut être modélisé comme un MP. Le générateur infinitésimal est dérivé des taux d'arrivée et de service des deux classes de trafic.

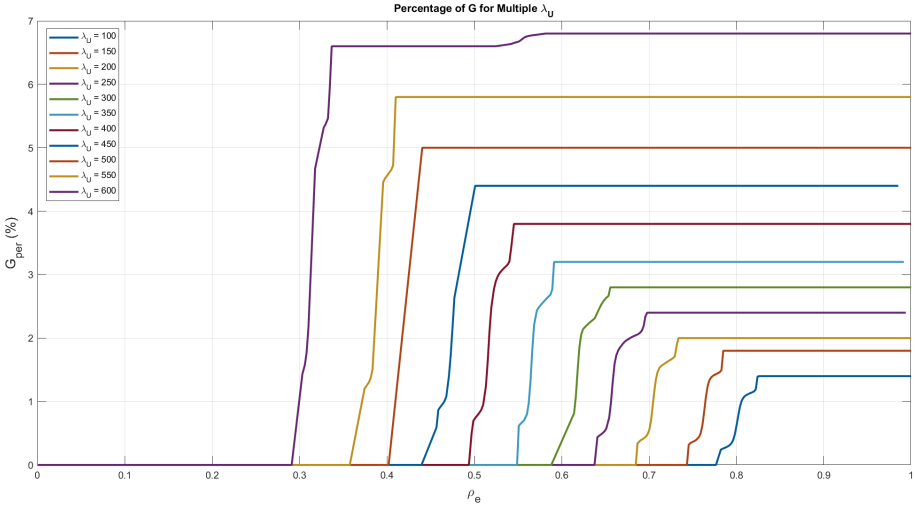
L'objectif du modèle est de déterminer le nombre minimal de canaux de garde requis pour satisfaire une probabilité d'erreur de transmission cible ϵ, fixée à 10^{-5} dans ce travail.

Pour évaluer le modèle, un cadre de simulation a été développé. Pour chaque configuration de trafic, l'algorithme augmente de manière incrémentielle le nombre de canaux de garde jusqu'à ce que la contrainte de fiabilité URLLC souhaitée soit atteinte. Afin de faciliter les comparaisons entre des systèmes de tailles différentes, le nombre de ressources réservées est normalisé selon

$$G_{per} = \frac{G}{S} \times 100. \tag{1}$$



(a) Pourcentage de canaux de garde dédiés requis par rapport à la charge du trafic eMBB pour différentes valeurs de S.

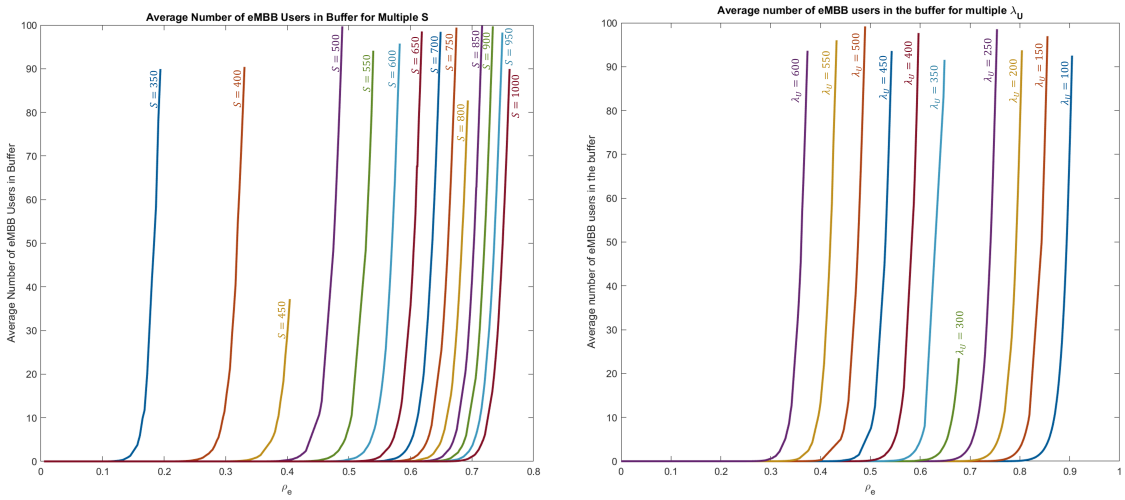


(b) Pourcentage de canaux de garde dédiés requis par rapport à la charge du trafic eMBB pour différentes valeurs de λ_U.

Figure 2: Pourcentage de canaux de garde dédiés requis par rapport à la charge du trafic eMBB.

Les résultats révèlent trois régions de fonctionnement distinctes. Pour des valeurs faibles de la charge eMBB ρ_e, très peu de canaux de garde sont nécessaires car des ressources suffisantes restent disponibles pour les deux classes de trafic. À mesure que ρ_e augmente, une zone de transition apparaît dans laquelle la valeur requise de G_{per} croît progressivement pour maintenir la probabilité d'erreur de transmission URLLC souhaitée. Enfin, une fois que le système approche de la saturation, le pourcentage de ressources réservées atteint un plateau.

Les résultats montrent également que le nombre total de RB disponibles a un impact plus important sur le niveau de réservation requis que l'intensité du trafic URLLC elle-même. Les systèmes de plus grande envergure nécessitent proportionnellement moins de ressources dédiées pour atteindre le même objectif de fiabilité, tandis qu'une augmentation de λ_U entraîne une hausse plus modérée du nombre de canaux de garde.



(a) Nombre moyen de paquets eMBB mis en mémoire tampon pour différentes valeurs de S.

(b) Nombre moyen de paquets eMBB mis en mémoire tampon pour différentes valeurs de λ_U.

Figure 3: Évolution du nombre moyen de paquets eMBB mis en mémoire tampon en fonction de la charge de trafic eMBB.

Les figures 3a et 3b présentent l'évolution du nombre moyen de paquets eMBB mis en file d'attente en fonction de la charge de trafic eMBB ρ_e. Dans les deux cas, la longueur moyenne de la file d'attente reste proche de zéro pour des charges de trafic faibles et augmente rapidement dès que le système approche de la saturation. Pour des valeurs plus faibles de S, la congestion apparaît plus tôt en raison de la quantité limitée de ressources radio disponibles. De même, l'augmentation du taux d'arrivée URLLC λ_U réduit les ressources disponibles pour le trafic eMBB, ce qui entraîne une augmentation plus rapide de l'occupation de la file d'attente. Ces résultats mettent en évidence le compromis introduit par le mécanisme de canal de garde: si la réservation de ressources améliore la fiabilité du trafic URLLC et garantit la probabilité d'erreur de transmission cible, elle augmente également le temps d'attente subi par le trafic eMBB dans des conditions de charge élevée.

Cette première contribution fournit un outil de dimensionnement pratique permettant aux acteurs industriels tels que EDF d'estimer les ressources nécessaires

pour garantir un niveau souhaité de QoS avant de négocier avec un opérateur mobile.

Néanmoins, plusieurs limites subsistent. Le modèle repose sur des hypothèses simplifiées, notamment des arrivées de Poisson. De plus, la stratégie d'allocation reste statique et ne peut s'adapter aux variations du trafic au fil du temps. Enfin, le modèle suppose que le type de trafic est parfaitement connu, alors que les utilisateurs industriels n'ont généralement pas accès direct aux informations de slicing. Ces limites motivent les approches de classification et d'apprentissage par renforcement présentées dans les chapitres suivants.

Classification des tranches à l'aide des informations PHY/MAC

Le modèle de dimensionnement des ressources présenté au chapitre 2 suppose que le type de trafic est connu. Dans la pratique, cependant, les acteurs industriels tels que EDF n'ont généralement pas accès aux informations de slicing gérées par l'opérateur mobile. Par conséquent, la capacité à déterminer si un paquet appartient à une tranche URLLC ou eMBB en utilisant uniquement des informations observables de couche inférieure devient une exigence clé.

Des études existantes ont montré que la classification du trafic peut atteindre d'excellentes performances lorsque des informations de couche supérieure, telles que les indicateurs QoS, les KPI ou les identifiants de slice, sont disponibles. Cependant, ces informations sont rarement accessibles du point de vue du client. Alors que plusieurs travaux ont étudié la classification des services à l'aide de traces de couche inférieure, le problème de la distinction entre les slices URLLC et eMBB en utilisant uniquement des informations PHY et MAC reste largement inexploré. Ce chapitre propose un classificateur basé sur le LGBM, entraîné et testé sur des ensembles de données expérimentaux 5G SA.

Deux ensembles de données expérimentaux ont été pris en compte. Le premier a été généré à l'aide d'un émulateur de réseau Rohde & Schwarz et d'un smartphone COTS dans un environnement contrôlé. Le second a été généré dans un laboratoire de R&D EDF à l'aide d'un simulateur Amarisoft UE connecté à un réseau Ericsson gNB. Dans les deux cas, la configuration radio utilisait une fréquence porteuse de 3891,6 MHz dans la bande n77, une largeur de bande de 100 MHz, 273 PRBs et un espacement entre sous-porteuses de 30 kHz. Seules les informations des couches PHY et MAC ont été conservées.

L'ensemble de données généré avec R&S contient 2 964 000 paquets URLLC et 2 105 000 paquets eMBB, tandis que l'ensemble de données Amarisoft/Ericsson contient 1 811 648 paquets, dont 1 448 320 paquets URLLC et 363 328 paquets eMBB. Le trafic URLLC a été généré à l'aide de petits paquets arrivant de manière quasi périodique, tandis que le trafic eMBB a été généré à l'aide de profils de trafic plus volumineux et plus variables, tels que des téléchargements de fichiers et des sessions de flood TCP. L'ensemble de données a été divisé en 70% pour l'apprentissage et 30% pour les tests.

Le classificateur sélectionné est LGBM, en raison de son bon compromis entre précision, temps d'apprentissage et complexité computationnelle. Le modèle a été entraîné sur des indicateurs PHY/MAC bruts tels que `tb_len`, `prbs`, `mcs`,

`snr`, `harq` et `total_len`. De plus, des caractéristiques temporelles artificielles ont été calculées à l'aide de fenêtres glissantes de 5ms. Ces caractéristiques comprennent le temps entre les paquets, le temps jusqu'au paquet suivant, les moyennes glissantes et les écarts-types glissants pour chaque indicateur. Cela permet au modèle de saisir le comportement temporel du trafic plutôt que de se fier uniquement à des valeurs de paquets isolées.

Les premières expériences menées sur l'ensemble des données R&S ont atteint une précision proche de 95%. Cependant, l'analyse d'importance des caractéristiques a révélé que le modèle s'appuyait presque exclusivement sur des caractéristiques liées à HARQ. Cela indiquait que le classificateur n'apprenait pas une signature de trafic robuste, mais exploitait plutôt un biais introduit par le montage expérimental. Étant donné que la configuration R&S impliquait un seul utilisateur actif et ne reproduisait pas une planification multi-utilisateurs réaliste, HARQ agissait comme un identifiant de tranche implicite.

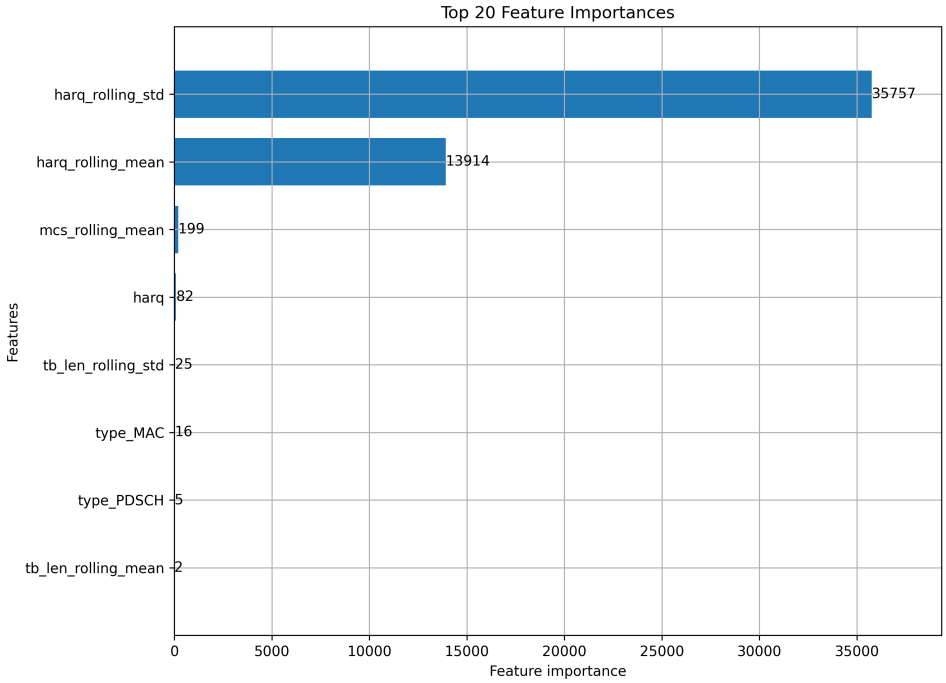


Figure 4: Importance des caractéristiques du classificateur LGBM sur l'ensemble de données Rohde & Schwarz.

Pour vérifier cette interprétation, toutes les caractéristiques liées à HARQ ont été supprimées et le classificateur a été réentraîné. La précision a chuté de manière significative, atteignant 53%, un niveau proche de la classification aléatoire. Cela a confirmé que les performances initiales n'étaient pas suffisamment généralisables et a motivé l'utilisation de l'ensemble de données Amarisoft/Ericsson, qui reflète mieux les conditions réelles de planification 5G SA.

Sur l'ensemble des données Amarisoft/Ericsson, le modèle s'est appuyé sur des indicateurs de trafic plus significatifs. Les caractéristiques les plus importantes étaient `tb_len`, `total_len` et leurs statistiques glissantes. Ces indicateurs sont cohérents avec le comportement attendu des deux tranches: le trafic URLLC est composé de paquets petits et réguliers, tandis que le trafic eMBB est plus volumineux, plus irrégulier et plus variable dans le temps.

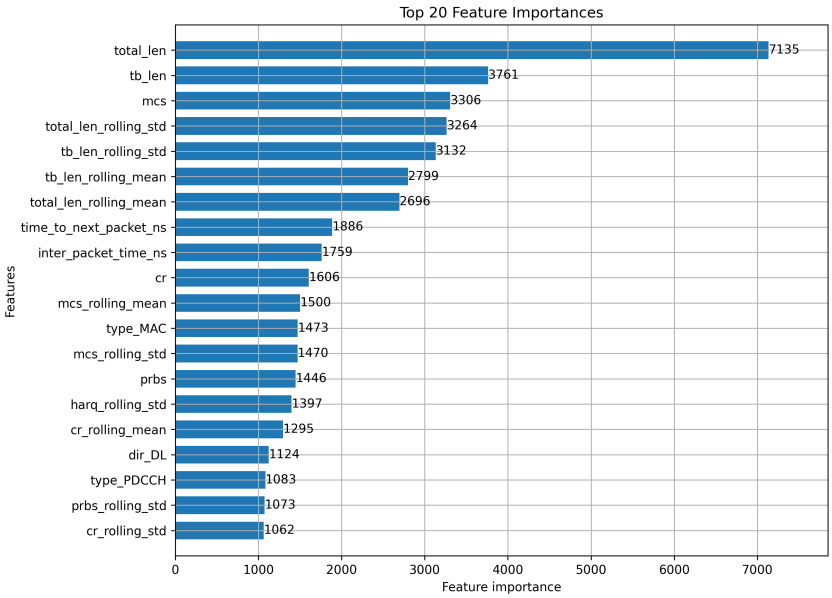


Figure 5: Importance des caractéristiques du classificateur LGBM sur l'ensemble de données Amarisoft/Ericsson.

Les résultats finaux de la classification obtenus sur l'ensemble de données Amarisoft/Ericsson sont présentés dans la figure 6. Le classificateur a atteint une précision globale de 97,6%. La slice URLLC a obtenu une précision de 0,99, un rappel de 0,97 et un F1-score de 0,98. La classe eMBB a obtenu une précision de 0,90, un rappel de 0,98 et un F1-score de 0,94.

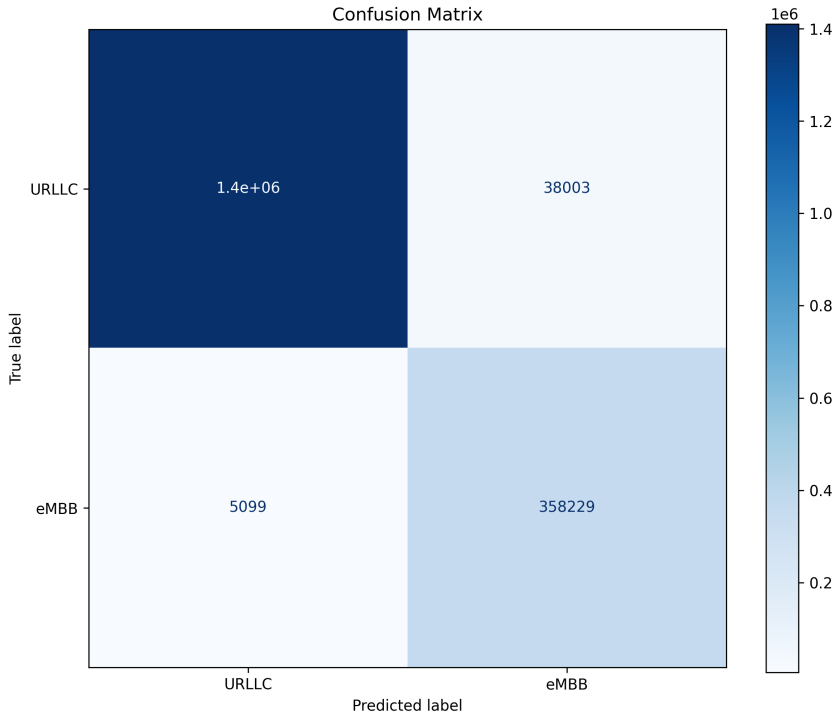


Figure 6: Matrice de confusion du classificateur LGBM sur l'ensemble de données Amarisoft/Ericsson.

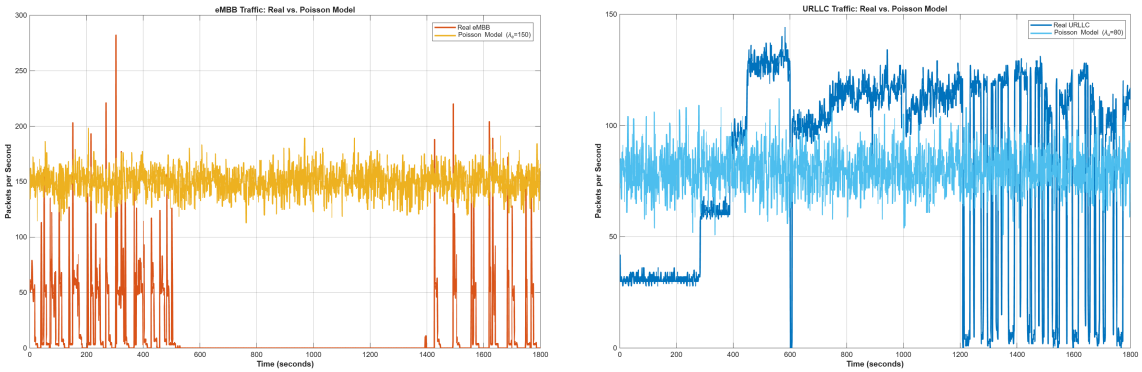
Ces résultats montrent qu'une classification précise des tranches est possible en utilisant uniquement les informations des couches PHY et MAC. Il s'agit d'un résultat important du point de vue du client, car il démontre qu'un utilisateur industriel peut déduire la nature du trafic sans s'appuyer sur des informations de couches supérieures ou sur des données propriétaires de l'opérateur.

Cette deuxième contribution complète donc le modèle de dimensionnement présenté au Chapitre 2. Le premier modèle détermine le nombre de RB à réserver pour le trafic URLLC, tandis que le classificateur développé dans ce chapitre permet d'identifier le type de trafic à l'aide de traces observables des couches basses. Cependant, l'allocation des ressources reste statique. Le chapitre suivant présente donc une approche d'apprentissage par renforcement dans laquelle un réseau Q profond adapte dynamiquement le nombre de canaux de garde en fonction de l'état observé du réseau.

Allocation dynamique des ressources à l'aide de réseaux Q profonds

Le modèle analytique proposé au Chapitre 2 fournit une méthode pratique pour estimer le nombre de canaux de garde nécessaires pour respecter une probabilité cible de perte de paquets URLLC. Cependant, cette approche repose sur des hypothèses simplificatrices, notamment des arrivées de type Poisson et des temps de service exponentiels. Si ces hypothèses facilitent le traitement analytique, elles ne reflètent pas fidèlement le trafic réseau réel.

Les traces expérimentales générées au Chapitre 3 ont révélé que les trafics URLLC et eMBB présentent tous deux des comportements en rafales et très variables qui diffèrent considérablement des processus de Poisson idéaux. Par conséquent, une stratégie d'allocation statique des ressources peut soit sous-allouer les ressources, entraînant des violations du SLA, soit sur-allouer les ressources, réduisant ainsi la capacité disponible pour le trafic eMBB.



(a) Trafic eMBB (b) Trafic URLLC

Figure 7: Comparaison entre le trafic de Poisson et le trafic généré expérimentalement.

Ces observations motivent le développement d'un mécanisme d'allocation dynamique des ressources capable d'adapter le nombre de canaux de garde

réservés en fonction des conditions actuelles du réseau. Pour répondre à ce problème, un agent Deep Q-Network (DQN) a été développé.

L'objectif de l'agent est de déterminer le nombre optimal de canaux de garde G nécessaires pour garantir une probabilité de perte de paquets URLLC inférieure au seuil cible de 10^{-5} tout en minimisant l'impact sur le trafic eMBB.

L'espace d'états de l'agent est composé de trois variables :

- Charge URLLC ρ_U
- Charge eMBB ρ_e
- Capacité normalisée du système $\frac{S}{S_{max}}$

L'espace d'action comprend 21 actions possibles correspondant à des niveaux de réservation de ressources allant de 0% à 100% de la capacité disponible, par incréments de 5%.

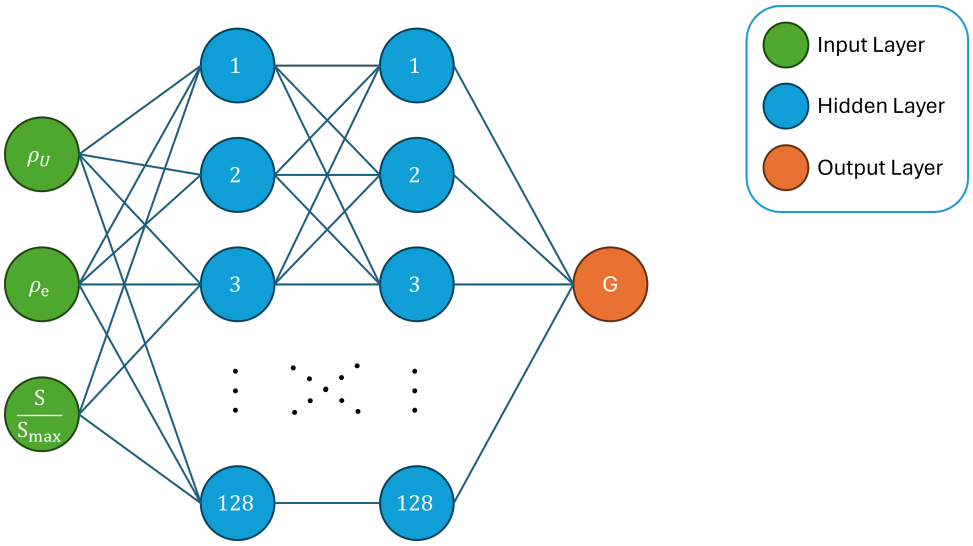
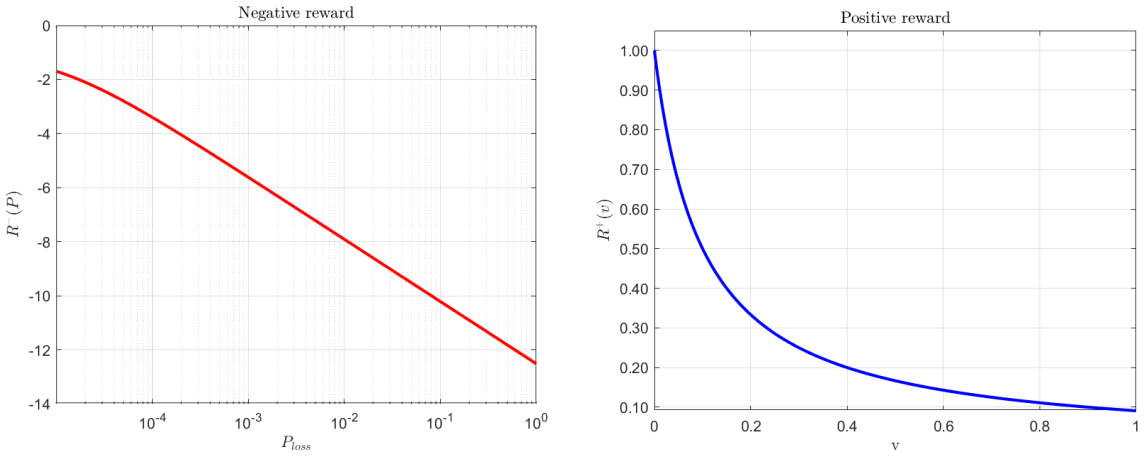


Figure 8: Architecture du modèle DQN proposé.

Le réseau neuronal est composé de deux couches cachées contenant chacune 128 neurones et utilise des fonctions d'activation ReLU. La couche de sortie estime la valeur attendue à long terme associée à chaque stratégie d'allocation possible du canal de garde.

Une fonction de récompense a été conçue pour donner la priorité au respect de l'exigence de fiabilité URLLC. Les configurations qui dépassent le seuil de perte de paquets reçoivent une pénalité logarithmique, tandis que les configurations viables sont récompensées en fonction de leur proximité avec le nombre minimum de canaux de garde requis pour satisfaire le SLA. Cette structure de récompense encourage l'agent à trouver des solutions économes en ressources tout en maintenant des performances de communication ultra-fiables.



(a) Partie négative (b) Partie positive

Figure 9: Fonction de récompense du modèle

L'agent a été entraîné à l'aide de la relecture d'expérience et d'un réseau cible afin d'améliorer la stabilité pendant l'apprentissage. L'entraînement a été réalisé sur plusieurs configurations de réseau couvrant différentes charges de trafic et capacités du système.

Le comportement de convergence du modèle est présenté dans la figure 10.

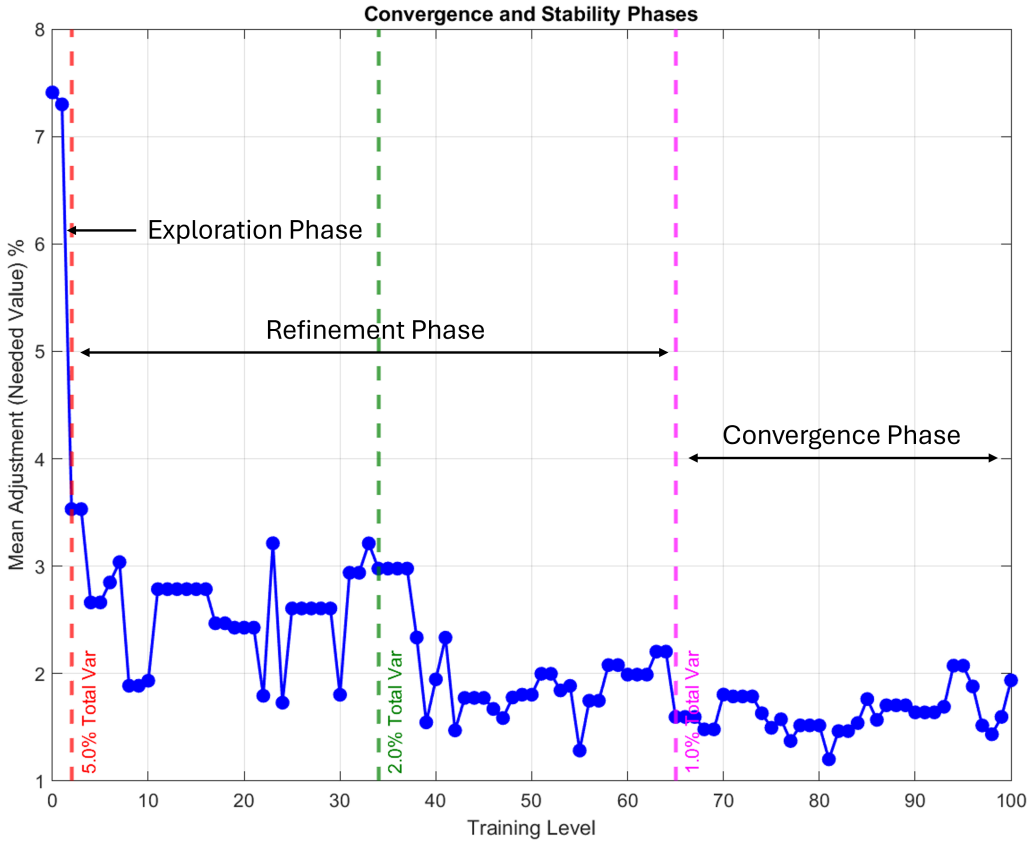


Figure 10: Analyse de la convergence et de la stabilité de l'agent DQN.

Les résultats montrent une convergence rapide au cours des premières itérations d'apprentissage. Le modèle atteint un seuil de stabilité de 5% après seulement

deux étapes d'apprentissage et continue d'affiner sa politique jusqu'à ce qu'une convergence stable soit atteinte vers l'étape d'apprentissage 65. Au-delà de ce point, l'écart entre les décisions de l'agent et l'optimum théorique reste inférieur à 1%.

L'agent entraîné a ensuite été évalué pour plusieurs largeurs de bande de canal correspondant à différentes valeurs du nombre total de PRBs S disponibles.

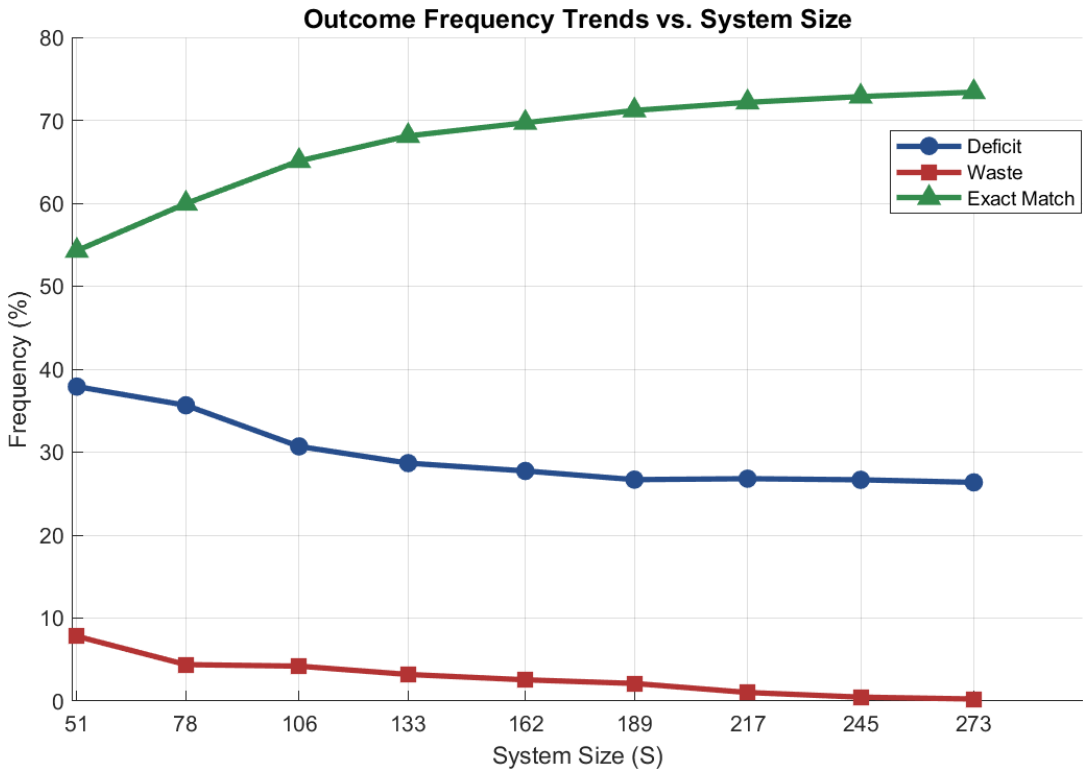


Figure 11: Comparaison entre l'allocation théorique et l'allocation de ressources basée sur le DQN.

Les résultats indiquent que le modèle analytique développé au chapitre 2 fournit l'allocation correcte dans environ 67% des cas. Cependant, il sous-estime le nombre requis de canaux de garde dans environ 30% des scénarios évalués, ce qui entraîne des probabilités de perte de paquets URLLC supérieures au seuil cible. Une surallocation ne se produit que dans une petite fraction des cas.

Ces observations montrent que l'allocation statique des ressources ne permet pas de rendre pleinement compte de la variabilité des conditions de trafic réelles et justifient le recours à une approche dynamique fondée sur l'apprentissage.

Afin de quantifier davantage l'écart entre les besoins théoriques et observés en ressources, des modèles de régression de loi de puissance ont été dérivés des résultats du DQN.

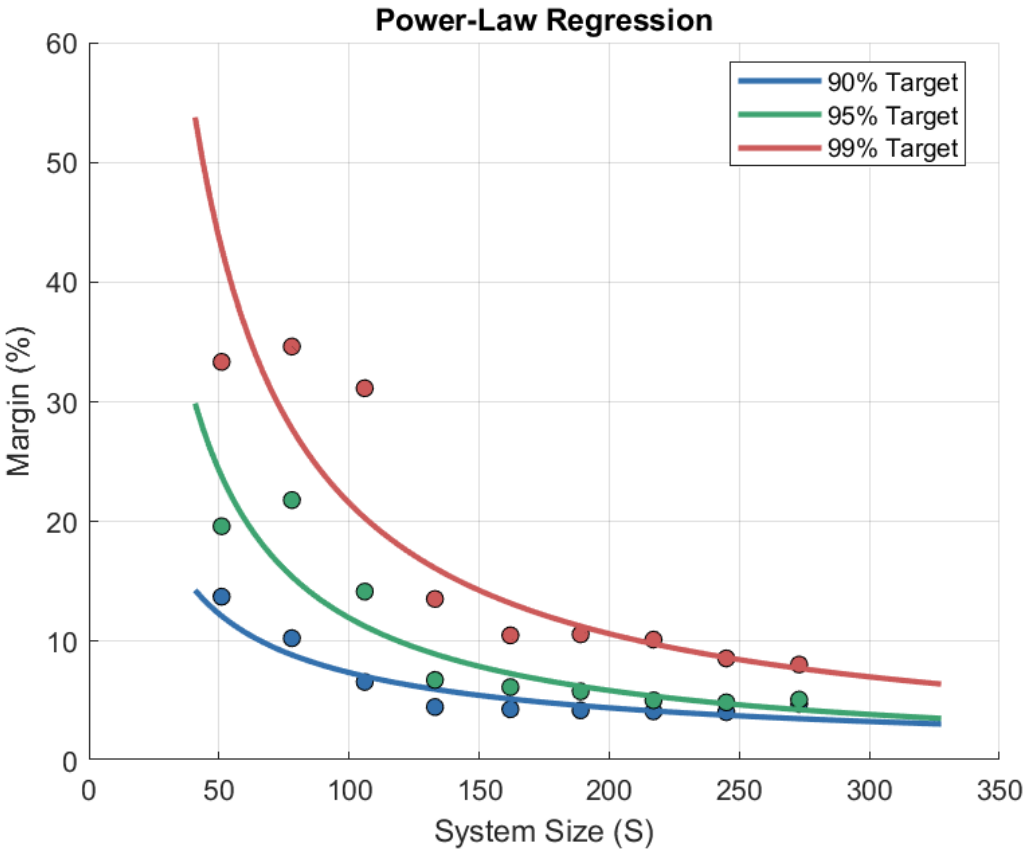


Figure 12: Modèles de régression de loi de puissance dérivés des résultats du DQN.

Les régressions obtenues fournissent des facteurs de correction permettant au modèle analytique d'atteindre des taux de réussite cibles de 90%, 95% et 99%. Pour l'objectif de fiabilité de 90%, la régression atteint une valeur R^2 de 0,90 et une faible erreur de prédiction, indiquant que les fluctuations modérées du trafic peuvent être efficacement approximées. Cependant, l'erreur de prédiction augmente considérablement pour l'objectif de 99%, ce qui reflète la difficulté de saisir les pics de trafic rares et extrêmes à l'aide d'un modèle de correction statique.

Cette troisième contribution démontre qu'une stratégie d'allocation des ressources basée sur DQN peut adapter avec succès les réservations de canaux de garde aux conditions changeantes du réseau tout en respectant les contraintes strictes de fiabilité URLLC. Combinée au cadre de dimensionnement développé au Chapitre 2 et au modèle de classification des tranches PHY/MAC présenté au Chapitre 3, elle forme une méthodologie de bout en bout complète permettant aux acteurs industriels de dimensionner, d'identifier et de gérer dynamiquement les tranches de réseau 5G sans avoir besoin d'accéder aux informations propriétaires des opérateurs.

Conclusion

Cette thèse s'est concentrée sur le principal défi que représente l'allocation des ressources du réseau 5G, en particulier entre le trafic URLLC et le trafic eMBB.

Pour les clients industriels, tels que EDF, la modernisation des réseaux intelligents nécessite des systèmes Protection, Automation, and Control (PAC) hautement fiables. Cependant, l'accès limité aux couches centrales du réseau empêche le client de garantir la probabilité de perte URLLC requise. De plus, l'approche poissonienne traditionnelle de l'allocation des ressources a tendance à être très optimiste en sous-allouant les ressources. Pour combler cette lacune, cette thèse a d'abord développé un modèle de classification du trafic avec un accès limité aux couches inférieures du réseau, en utilisant un algorithme d'apprentissage automatique Light Gradient-Boosting Machine (LGBM) entraîné sur des caractéristiques extraites de données au niveau des paquets afin de distinguer les modèles de trafic URLLC et eMBB. Le deuxième modèle s'est concentré sur l'allocation de canaux de garde où un agent d'apprentissage par renforcement (plus précisément, l'apprentissage Q) a été entraîné dans un environnement simulé en conditions réelles afin de déterminer de manière dynamique le nombre optimal de canaux de garde pour URLLC. Les principaux résultats ont mis en évidence que le classificateur LGBM atteignait une grande précision dans la séparation des types de trafic sans données de la couche centrale, tandis que l'agent d'apprentissage par renforcement identifiait des stratégies d'allocation qui répondaient mieux aux exigences de perte URLLC par rapport à l'approche statique basée sur le modèle de Poisson.

Cela a conduit à la nécessité de générer un ensemble de données pouvant être utilisé pour l'entraînement et le test du modèle présenté au chapitre 3. Un modèle LGBM a été développé afin de classer les paquets réseau entre les tranches URLLC et eMBB en utilisant des données extraites uniquement des couches PHY et MAC d'un réseau 5G SA. Le modèle a atteint une précision de classification de 97,6%, démontrant que les clients industriels peuvent surveiller efficacement leur trafic sans avoir accès à tous les KPI.

Le chapitre 4 présente un agent DQN qui utilise l'ensemble de données classifiées du chapitre 3 afin de s'entraîner et de générer le nombre optimal de canaux de garde nécessaires. En naviguant dans un espace d'états défini par la charge URLLC, la charge eMBB et la capacité normalisée, l'agent a appris à équilibrer le dilemme exploration-exploitation afin d'ajuster dynamiquement les canaux de garde. Les résultats de ce modèle ont montré que l'approche théorique est optimiste environ 30% du temps, et ne peut donc pas être fiable. De plus, des régressions de loi de puissance ont été générées pour des taux de réussite de 90%, 95% et 99%, ce qui permet au client d'utiliser simplement la formule de Poisson et la régression afin de calculer instantanément le nombre de canaux de garde réellement nécessaires.

Cette thèse ouvre la voie à de multiples perspectives d'avenir. Premièrement, les modèles présentés peuvent être généralisés pour accepter tous les types de slices disponibles et pas seulement URLLC et eMBB, comme par exemple mMTC. Deuxièmement, le modèle d'allocation dynamique des ressources pourrait être étendu d'un environnement à cellule unique à une architecture multicellulaire. Enfin, un modèle maître peut être développé pour intégrer les trois modèles présentés dans cette thèse, en vue d'un déploiement sur un banc d'essai physique à grande échelle de réseau intelligent.

Publications

This thesis allowed for the publication of the following works:

- R. Altawil, S. Al Zahr, V. Audebert, L. Decreusefond and P. Martins, ”A Simulation on Resource Allocation Policies: URLLC Packets Loss and eMBB Packets Waiting Time”, 2025 Global Information Infrastructure and Networking Symposium (GIIS), Dubai, United Arab Emirates, 2025, pp. 1-5, doi: 10.1109/GIIS64151.2025.10922079.
- R. Altawil, L. Delolme, V. Audebert, and P. Martins ”LightGBM for Slice Recognition at 5G PHY and MAC Layers”, Engineering Proceedings 122, 2025, no. 1:24. doi: 10.3390/engproc2026122024
- R. Altawil, L. Delolme, V. Audebert, and P. Martins ”LightGBM for Slice Recognition at 5G PHY and MAC Layers”, IEEE 6th International Conference on Communications, Information, Electronic and Energy Systems, 2025.

List of Figures

1 Les tranches 5G et leurs principales applications. 12
2 Pourcentage de canaux de garde dédiés requis par rapport à la charge du trafic eMBB. 13
3 Évolution du nombre moyen de paquets eMBB mis en mémoire tampon en fonction de la charge de trafic eMBB. 14
4 Importance des caractéristiques du classificateur LGBM sur l'ensemble de données Rohde & Schwarz. 16
5 Importance des caractéristiques du classificateur LGBM sur l'ensemble de données Amarisoft/Ericsson. 17
6 Matrice de confusion du classificateur LGBM sur l'ensemble de données Amarisoft/Ericsson. 17
7 Comparaison entre le trafic de Poisson et le trafic généré expérimentalement. 18
8 Architecture du modèle DQN proposé. 19
9 Fonction de récompense du modèle 20
10 Analyse de la convergence et de la stabilité de l'agent DQN. . . . 20
11 Comparaison entre l'allocation théorique et l'allocation de ressources basée sur le DQN. 21
12 Modèles de régression de loi de puissance dérivés des résultats du DQN. 22

1.1 Global Electricity Production by Source [**?**] 29
1.2 Pipeline of the Work in the Thesis 31

2.1 5G Slices and Their Uses [**?**] 34
2.2 Difference between SA 5G and NSA 5G 35
2.3 An example of 5G Standalone (SA) network architecture 36
2.4 5G Control Plane Stack 37
2.5 5G User Plane Stack 39
2.6 Illustration of a 5G Cell Supporting Heterogeneous eMBB and URLLC Services 40
2.7 Bandwidth parts in 5G 41
2.8 Frequency Resource Sharing Between URLLC and eMBB Traffics 46
2.9 URLLC and eMBB Flows and Service 46
2.10 The states and actions of the system and their respective rates . . 47
2.11 Percentage of Dedicated Guard Channels Needed with Respect to the Load of the eMBB Traffic for Different Values of S 50
2.12 Average Number of eMBB Packets in the Buffer with Respect to the Load of the eMBB Traffic for Different Values of S 51

2.13 Percentage of Dedicated Guard Channels Needed with Respect to the Load of the eMBB Traffic for Different Values of λ_U 51
2.14 Average Number of eMBB Packets in the Buffer with Respect to the Load of the eMBB Traffic for Different Values of λ_U 52
2.15 Variation of G_{per} in the Steady Stage for Different Values of S and λ_U 53

3.1 An Example of Underfitting, Overfitting, and a Robust Fit in ML [**?**] 58
3.2 Chart Showing LGBM's Leafsplit Strategy 59
3.3 Different ROC Curves for Different Classifier Model Performances 62
3.4 An Exemple of the ROC, AUC, and Chance Line of a Model 62
3.5 Rohde & Schwarz Hardware Setup 65
3.6 Rohde & Schwarz Interface 66
3.7 Example of an eMBB and URLLC Trace 69
3.8 Example of a Rolling Window 71
3.9 Light Gradient-Boosting Machine model pipeline 71
3.10 Algorithm 2's Pipeline 72
3.11 Algorithm 3's Pipeline 73
3.12 Algorithm 4's Pipeline 75
3.13 Algorithm 5's Pipeline 76
3.14 Confusion Matrix of the Model for the R&S Dataset 78
3.15 ROC Curves of the Model for the R&S Dataset 79
3.16 Bar Graph of the LightGBM Model Key Features by Importance for the R&S Dataset 80
3.17 Bar Graph of the LightGBM Model Key Features by Importance for the R&S Dataset without the HARQ Identifier 81
3.18 Confusion Matrix of the Model for the R&S Dataset without the HARQ Identifier 82
3.19 ROC Curves of the Model for the R&S Dataset without the HARQ Identifier 83
3.20 Bar Graph of the LightGBM Model Key Features by Importance for the Amarisoft/Ericsson Dataset 84
3.21 Confusion Matrix of the Model for the Amarisoft/Ericsson Dataset . 85
3.22 ROC Curves of the Model for the Amarisoft/Ericsson Dataset . . . 86

4.1 Comparaison between Poisson traffic and experimentally generated eMBB traffic ($\lambda_e = 150$) for various time windows 89
4.2 Comparaison between Poisson traffic and experimentally generated URLLC traffic ($\lambda_e = 80$) for various time windows 90
4.3 Architecture of the DQN used for dynamic guard channel allocation 97
4.4 Negative Section of the Reward Function 98
4.5 Positive Section of the Reward Function 99
4.6 DQN Model's pipeline 99
4.7 Algorithm 6's pipeline 100
4.8 Algorithm 7's pipeline 101
4.9 Algorithm 8's pipeline 102
4.10 Algorithm 9's pipeline 103
4.11 Convergence and Stability Analysis of DQN Resource Allocation Model 105

4.12 Results of the DQN Resource Allocation Model for Different Values of S 107
4.13 Power Regressions for 90%, 95% and 99% Success Targets . . . 108

List of Tables

2.1 System notations used in Chapter 2 and their descriptions and units 33
2.2 Summary comparison table for the articles mentioned in the state of the art of Chapter 2 45

3.1 System Notations Used in Chapter 3 and their Descriptions and Units 56
3.2 Sample 5QI Values List [**?**] 56
3.3 PHY and MAC Layers and their Channels 57
3.4 Summary Comparison of Representative Works on 5G Traffic Classification 64
3.5 Radio Parameters for Dataset Generation 66
3.6 Simulation Parameters for Dataset Generation Using Rohde & . . 66
3.7 Simulation Traffic Profiles Used for Dataset Generation 67
3.8 Radio Parameters for Dataset Generation 67
3.9 Simulation Parameters for Dataset Generation 68
3.10 Simulation Traffic Profiles Used for Dataset Generation 68
3.11 LGBM Hyperparameters 68
3.12 List of Features Found in the Traces 69
3.13 List of Engineered Features for the Traces 70
3.14 Classification Report of the LightGBM Traffic Classifier for the R&S Dataset 78
3.15 Classification Report of the LightGBM Traffic Classifier for the R&S Dataset without the HARQ Identifier 82
3.16 Classification Report of the LightGBM Traffic Classifier 85

4.1 Summary of Notations and Parameters used in the Proposed Model 91
4.2 Summary Comparison of Representative Works on 5G Resource Allocation 96
4.3 DQN Agent and Environment Hyperparameters 105
4.4 Training Convergence Levels by Stability Threshold 106
4.5 Different Channel Bandwidth Values and Physical Resource Blocks (S) for a Sub-Carrier Spacing of 30kHz 106
4.6 The Results of the Regressions for the Different Target Levels . . . 108

Chapter 1

Introduction

1.1 Context and Motivation

Since the discovery and initial deployment of electricity in the late 19th century, the demand and need for the electrical grid kept rising, pushing countries worldwide to increase production to meet these growing demands, as can be shown in Figure 1.1.

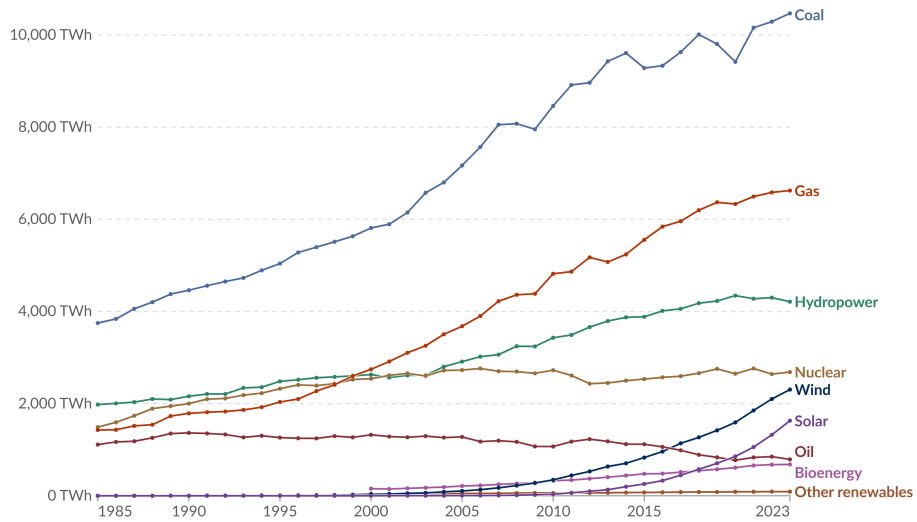


Figure 1.1: Global Electricity Production by Source [**?**]

Recently, the growing introduction of renewable energy to the electrical grid has shifted from a centralised model that relied on predictable power sources (such as coal and fuel) to a decentralised model that relies on intermittent renewable sources (such as wind and solar). This new instability and variability in production calls for a more demanding information flow that the older infrastructure cannot handle. This ageing infrastructure, the integration of renewable energy, and the surge of a need for two-way communication between the Distribution System Operator (DSO)s and the clients lead to an increasing demand for more efficient, reliable, and sustainable electricity distribution systems. The development of traditional systems into smart grids is a way to answer the challenges faced. A Smart Grid is a connected electrical grid that enables two-way

information exchange, enabling consumers to have more control over their consumption [**?**].

At the core of the Smart Grid are Protection, Automation, and Control (PAC) systems, they're an important component of the electrical grid, made to guarantee asset protection, reliability, and stability, mainly improving the operational efficiency of the smart grid [**?**].

To guarantee the needed stability, PAC systems require robust connections over the entire electrical grid. Although fibre optics offers an optimal performance, its deployment over such a vast infrastructure is very costly and time-consuming, causing the electrical grid provider to seek a wireless solution. However, older technologies like 4G can not guarantee the needed low latency for such critical messages. With the introduction of slicing in 5G, especially the Ultra-Reliable Low Latency Communications (URLLC) slice, the needed latency can be reached in addition to the flexibility of a wireless 5G connection.

1.2 Problematics and Objectives

Even though slicing offers a promising solution for the monitoring and protection of the electrical grid, its deployment demands a complicated coordination between the DSO and the client, which can lead to a great conflict of interest. The operator will seek to maximise the efficiency of their infrastructure without inconveniencing their other clients, while the electricity provider prioritises strict QoS requirements to guarantee a low global loss probability. This is true especially in the case of mission-critical messages like a URLLC traffic that requires a very low latency and loss probability, which can be demanding and costly to deploy.

Moreover, the electricity provider, being a client and not an operator, has very limited visibility and control over the resource distribution policy beyond the physical layer. To protect its network, the client needs to negotiate the policies of resource allocation with the operator before the deployment of the network.

However, properly negotiating the needs and requirements of the smart grid network shows difficulty.

First, given that 5G network slicing isn't widely deployed yet, an analytical model is needed in order to anticipate the traffic and its needs. Over the decades, Poisson processes have been the go-to for network traffic simulation. However, even though such a model can provide a necessary first vision into the expected needs, it still fails to capture the bursty and unpredictable nature of real-life industrial traffic. This raises the need to compare Poisson laws to real-life traffic, in order to determine whether a Poisson process is optimistic, pessimistic, or sufficiently accurate.

Second, the client is confronted with a lack of visibility on the traffic since only the operator has access to the core network and to high-level packet identifiers, such as the slice ID. The client's only access is the lower PHY and MAC layers with very limited information. This narrow view creates a difficulty for the client to classify the traffic between its own and the operator's in order to verify if the agreed-upon resource allocation policy is put in place.

Therefore, the goal of this thesis is to provide the client EDF with a dimensioning tool that solves these issues. The first aim of this work is to develop a

theoretical Poisson process simulation as the first baseline for the resource allocation needs. Next, a machine learning model that can classify the traffic between the different slices is developed to bridge the visibility gap and allow the client to quantify the passing traffic. This classified traffic is then to be used to calculate the optimal resource allocation policy and to be compared to the theoretical Poisson process simulation.

1.3 Structure of the Thesis

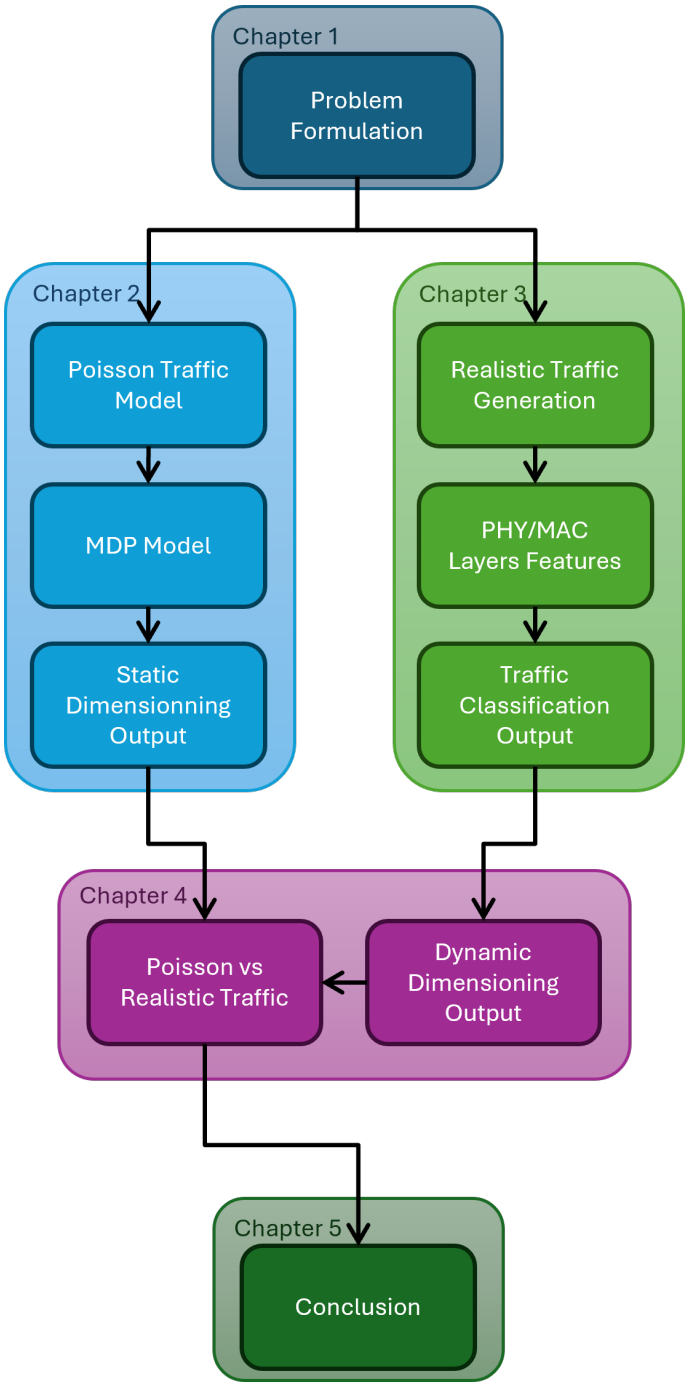


Figure 1.2: Pipeline of the Work in the Thesis

The remainder of this manuscript is organised into four main chapters that address the different components of the pipeline seen in Figure 1.2.

- Chapter 1 - Introduction: This first chapter presents the context and motivation for the work in this thesis, followed by the core research problem formulation and the structure of the thesis.
- Chapter 2 - Simulation Model for Resource Allocation: This chapter is the theoretical baseline of the research by developing a static dimensioning simulation. It models slicing between eMBB and URLLC traffic using queuing theory and an Markov Decision Process. In this chapter, the concept of guard channels G is introduced to guarantee a URLLC loss probability of 10^{-5}. This simulation is the baseline to which the experimental model will be compared in Chapter 4.
- Chapter 3 - Slice Classification Model: This chapter moves the work to a more realistic approach by using generated traffic instead of a Poisson process. It then resolves the lack of visibility by presenting a Machine Learning (ML) model that classifies the traffic between the slices using only the information found in the PHY and MAC layers. This classification is an essential prerequisite to the dynamic resource allocation done in Chapter 4.
- Chapter 4 - Deep Reinforcement Learning for Resource Allocation: This chapter tackles the same problem as Chapter 2 but in a realistic experimental environment. Using the classified generated traffic from Chapter 3, this chapter introduces a deep reinforcement learning agent trained on generated traffic that mimics reality. This agent dynamically learns and chooses the optimal number of guard channels G to reserve for URLLC traffic. It then compares it to the results found in Chapter 2 to show that a Poisson process tends to be optimistic in its calculations.
- Chapter 5 - Conclusion: This final chapter recaps the work presented in the thesis and draws the conclusions regarding client-centred resource allocation. It also presents the possible future works that can be derived from this manuscript.

Chapter 2

Simulation Model for Resource Allocation

2.1 Introduction

2.1.1 System Notations

Table 2.1 groups the mathematical notations and system parameters used in this chapter.

Table 2.1: System notations used in Chapter 2 and their descriptions and units

Notation	Description	Unit
ρ_U	Load of URLLC Clients	-
ρ_e	Load of eMBB Clients	-
λ_U	URLLC Clients Arrival Rate	packets/s
λ_e	eMBB Clients Arrival Rate	packets/s
μ_e	eMBB Clients Service Rate	packets/s
μ_U	URLLC Clients Service Rate	packets/s
P_{error}	URLLC Transmission Error Probability	-
ϵ	Desired URLLC Transmission Error Probability	-
S	Total Number of Resources in a Cell	RBs
G	Number of RBs reserved for URLLC Clients Use	RBs
x_1	Number of URLLC packets in service	packets
x_2	Number of eMBB packets in service	packets
x_3	Number of eMBB packets in the buffer	packets

2.1.2 5G Network Slicing

Network slicing, as defined by 3GPP, is a new approach in which virtual networks with customised resources and configurations are created within the physical network. They are designed for specific user types by focusing on the most impor-

tant parameters for each group. This approach facilitates more efficient use of network resources and supports the development of specialised services tailored to different requirements [**?**].

In [**?**], it was shown that high-reliability wireless telecommunication means are good candidates for the operators to pilot and monitor their equipment. 5G has introduced many improvements compared to its predecessor in terms of speed, capacity, flexibility, and latency [**?**]. One of the key innovations driving this improvement is network slicing, a concept essential for optimising and tailoring the network to meet the different demands of the diverse types of users.

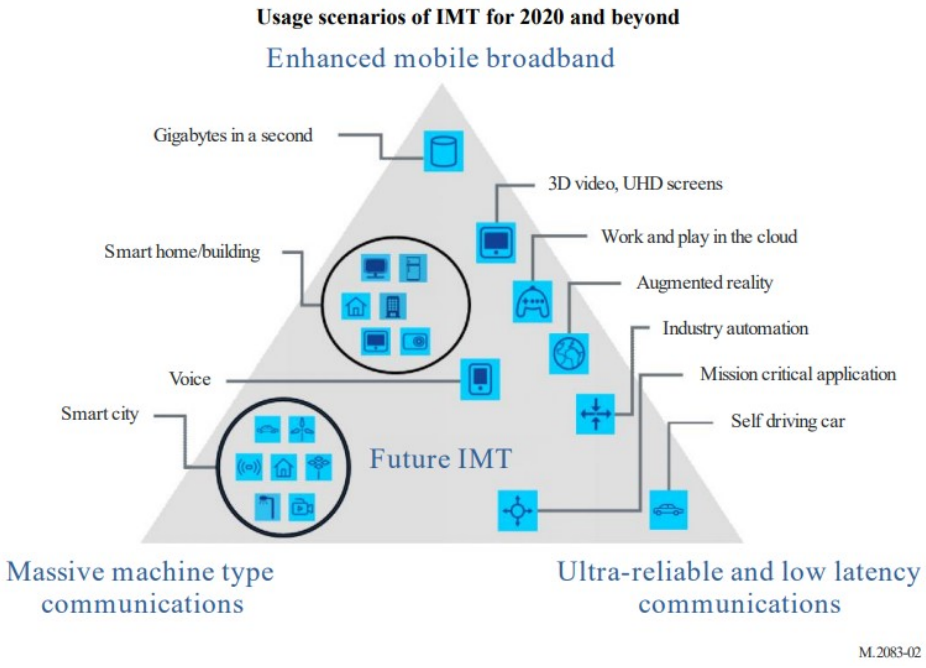


Figure 2.1: 5G Slices and Their Uses [**?**]

As can be seen in Figure 2.1 the main slices in 5G slicing are:

- Ultra-Reliable Low Latency Communications (URLLC): The URLLC slice is intended for applications that demand highly reliable and low-latency communications. [**?**]. It ensures that data is transmitted with minimal delay and errors. It is used primarily for IoT applications, focusing on mission-critical applications and smart grids. Examples include autonomous vehicles, industrial automation, and smart grids, where even a small delay or error can have severe consequences.

- Enhanced Mobile Broadband (eMBB): The eMBB slice is designed to deliver high data rates and enhanced network performance in environments that demand substantial bandwidth, such as crowded urban areas or large-scale events. It exceeds 4G in terms of data rate capacity, traffic and connection density, user mobility, and deployment flexibility [**?**]. Typical applications include video streaming, Augmented Reality (AR), Virtual Reality (VR), smartphones, and wearables, all of which place high traffic demands on networks.

- Massive Machine-Type Communication (mMTC): The mMTC slice supports massive amounts of small devices that need a low data rate and low energy consumption [**?**]. These devices are often sensors in the IoT network that do not need constant, high-speed data transmission but require energy efficiency and a reliable connection. Examples include smart cities, environmental monitoring, and agricultural sensors, where massive amounts of devices need to be connected and transmit small amounts of data intermittently.

This great diversity of the use cases demands an efficient and intelligent resource allocation to guarantee the required QoS to each type of user.

2.1.3 5G System Architecture and Radio Interface

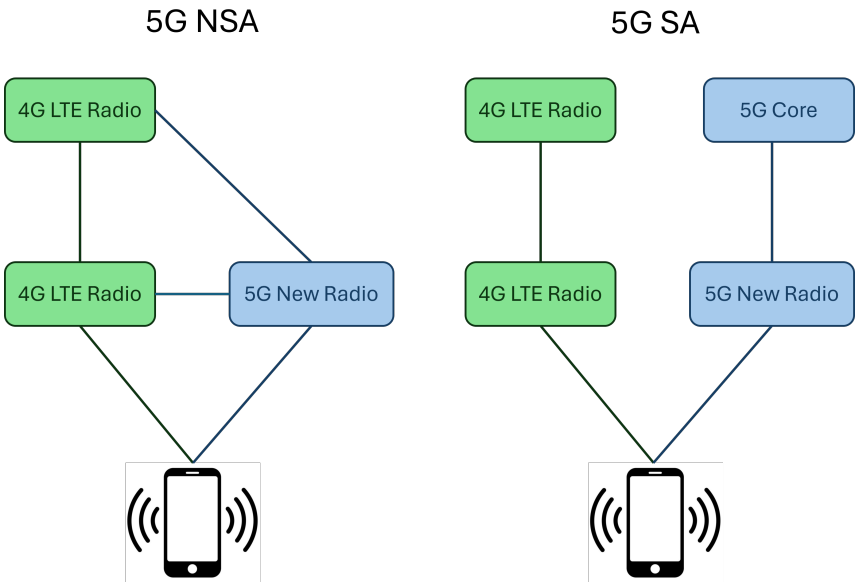


Figure 2.2: Difference between SA 5G and NSA 5G

5G networks can be deployed following two different architectures, either Standalone (SA) or Non-Standalone (NSA). 5G NSA depends on a 4G Core network by adding a 5G gNB besides a pre-existing 4G eNB, as can be seen in Figure 2.2. On the other hand, 5G SA does not depend on a 4G network; instead, it uses its own independent 5G access and core networks.

The deployment of 5G SA networks allowed for the passage from a rigid hardware setup to a more flexible software defined Service-Oriented Architecture (SOA). Figure 2.3 above illustrates this architecture and its components, such as the New Generation Radio Access Network (NG-RAN) and the 5G Core (5GC).

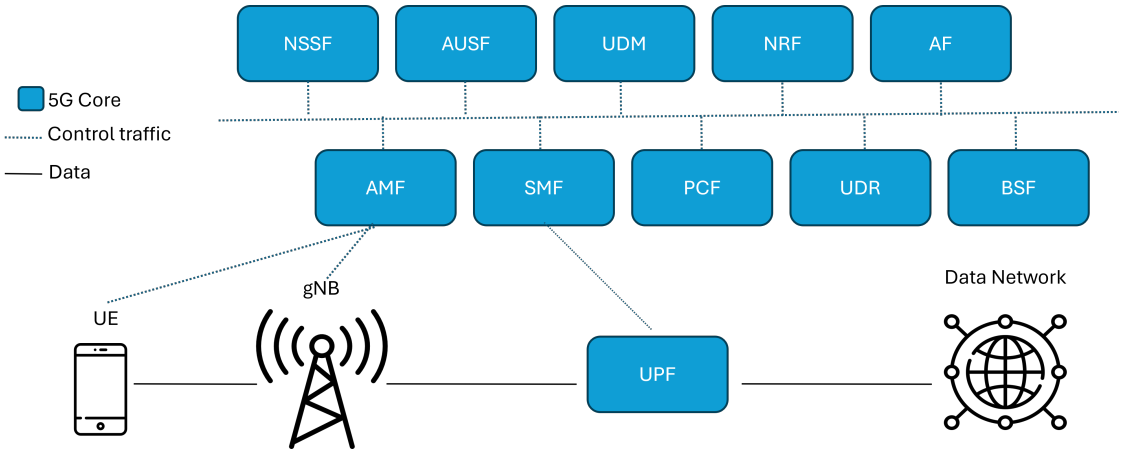


Figure 2.3: An example of 5G Standalone (SA) network architecture

The NG-RAN acts as the bridge that connects the user to the 5G Core. In the case of 5G SA architecture, it is composed of the gNB only, as opposed to NSA architecture, where it groups the gNB and the eNB. The main functions of the NG-RAN include and are not limited to radio transmission and reception, resource allocation, scheduling, etc.

The primary components of the radio access level are:

- User Equipment (UE): Equipment used by an end-user, such as a mobile cellphone, IoT device, smart TV, etc. It is either the source of the message or the recipient.

- gNodeB (gNB): Base station in 5G New Radio, replacing eNodeB (eNB) in Long Term Evolution (LTE). It is the bridge that connects the UE to the 5GC in SA architecture by receiving and transmitting the radio signal and allocating the available resources.

As can be seen in figure 2.3, in 5G architecture, the 5GC handles the control of the network and data routing. It is also responsible for the creation of the different slices and the association of the UEs and service to the corresponding slices. The 5GC finally connects the user with the targeted network.

The 5GC connects the user to the data network. It is composed of multiple functions such as [**?**]:

- User Plane Function (UPF): ”A user plane function that routes the actual data to the desired network by packet routing and forwarding, or enforcing QoS requirement on the user plane. The UPF also does packet inspection on the radio level.”
- Access and Mobility Management Function (AMF): ”A control plane function that manages the access and mobility in a 5G network. Some of its main features are registration, connection, and reachability management. In addition, it provides transport for Session Management (SM) messages.”
- Session Management Function (SMF): ”A control plane function responsible for session and QoS management, and the selection and control of the UPF. The SMF chooses the appropriate UPF and instructs it on how to route the traffic.”

- Policy Control Function (PCF): ”It mainly supports a unified policy framework, providing policy rules to the control plane functions.”
- Network Repository Function (NRF): ”It allows for the different network functions in the system to find each other in order to communicate dynamically.”
- Unified Data Management (UDM): ”It mainly generates the authentication credentials and manages user identification and subscription data.”
- Authentication Server Function (AUSF): ”It supports the authentification for 3GPP access, and for a disaster roaming service.”
- Application Function (AF): ”It provides services such as application influence on traffic routing, accessing the network exposure function, and interacting with the policy framework for policy control.”
- Unified Data Repository (UDR): ”It handles the storage and retrieval of subscription and policy data, in addition to structured data for exposure, application data, and NF group ID.”
- Network Slice Selection Function (NSSF): ”It selects the set of network slice instances serve which UE.”
- Binding Support Function (BSF): ”Makes sure that all the sessions from the same user are bound to the same PCF.”

To allow for a connection between the architectural components shown in 2.3, the radio interface depends on two main protocol stacks: the Control Plane (CP) and User Plane (UP).

A. Control Plane

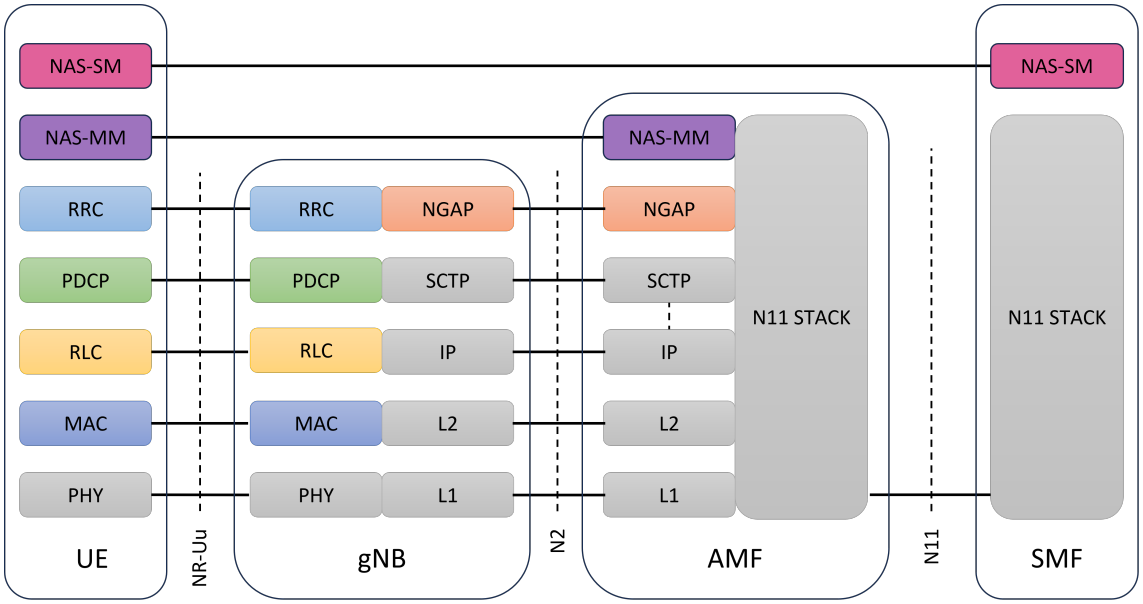


Figure 2.4: 5G Control Plane Stack

The control plane is the set of procedures and functional entities that make it possible for a given user to access all services provided by a mobile network. Its main functional entities and protocols are depicted in Figure 2.4.

The components of the CP are:

- Protocols and layers in UE-gNB interface [**?**], [**?**]:

  - Non-Access Stratum Session Management (NAS-SM): ”Supports the handling of the PDU session management procedures to support its establishment, modification, and release.”

  - Non-Access Stratum Mobility Management (NAS-MM): ”Supports the registration and connection management functionality, and user plane connection activation and deactivation.”

  - Radio Resource Control (RRC): ”Supports the broadcast of the system information, QoS management functions, and mobility functions.”

  - Packet Data Convergence Protocol (PDCP): ”Supports the transfer of the data, header compression and decompression, ciphering, and integrity protection.”

  - Radio Link Control (RLC): ”Supports the transfer of upper layers PDUs, error correction, protocol error detection, and duplicate detection.”

  - Medium Access Control (MAC): ”Supports the mapping between logical and transport channels, error correction, and priority handling.”

  - Physical (PHY): ”Offers data transport services to higher layers, supports modulation, coding, antenna mapping, and multi-antenna processing.”

- Protocols and layers in the core network [**?**]:

  - New Generation Application Protocol (NGAP): ”Supports application layer signalling between NG-RAN and 5GC.”

  - Stream Control Transmission Protocol (SCTP): ”Ensures a reliable transport of signalling messages.”

- Interfaces and reference points [**?**], [**?**]:

  - NR-Uu: ”Radio interface between the UE and the gNB. It supports both the user and control plane protocol stacks.”

  - N2: ”Reference point between the gNB and the AMF. It is used for the control plane signalling and for the management of the UE context and mobility.”

  - N11: ”Reference point between AMF and SMF. It supports the forwarding of NAS-SM messages and PDU session management.”

B. User Plane

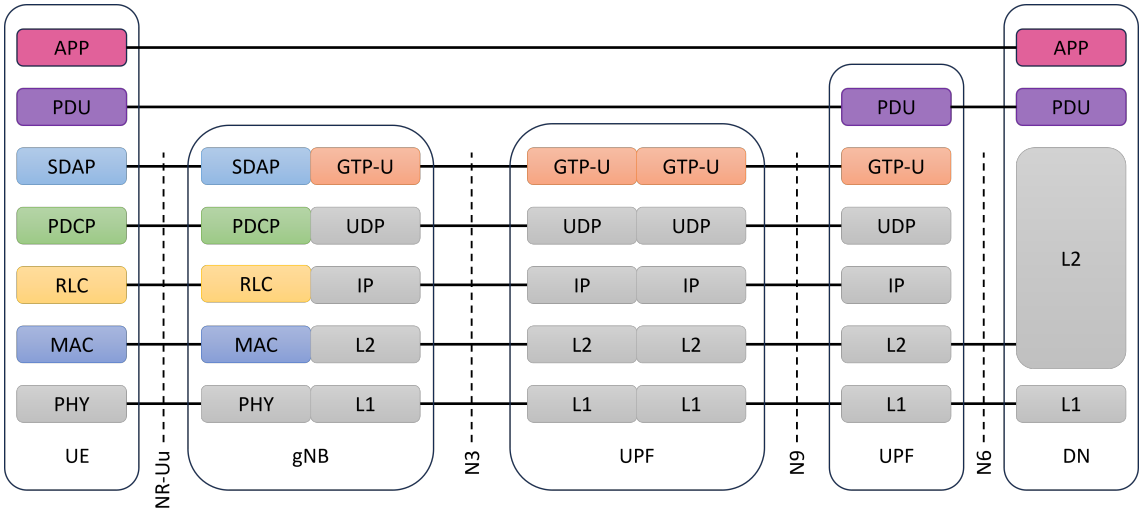


Figure 2.5: 5G User Plane Stack

The UP, illustrated in figure 2.5, constitutes the architectural component responsible for transporting user traffic between the UE and the data network, while ensuring that QoS requirements are satisfied.

The components of the UP are:

- Protocols and layers in UE-gNB interface [**?**]:
  - Service Data Adaptation Protocol (SDAP): ”Supports the mapping between a QoS flow and a data radio bearer, and marks the QoS flow ID in both downlink and uplink packets.”
  - PDCP, RLC, MAC, and PHY: Present in the user plane and perform functions similar to those described in the control plane as presented in section A..
- Protocols and layers in the transport stack [**?**]:
  - GPRS Tunnelling Protocol User (GTP-U): ”Tunnels user data between the NG-RAN node and the UPF, allowing the carrying of user data packets and signalling information.”
  - User Datagram Protocol (UDP): ”Used for the transport of GTP-U packets, utilizing checksums for error detection.”
- Upper layers and nodes [**?**]:
  - Packet Data Unit (PDU) Layer: ”Provides a connectivity service between the UE and the Data Network supported by the UPF.”
  - Application Layer: ”Services that communicate with the application in the UE operating over the PDU session connectivity service.”
  - User Plane Function (UPF): ”Supports packet routing and forwarding, packet inspection, and QoS handling for the user plane.”

- Interfaces and reference points [**?**]:
  - N3: ”Reference point between the gNB and the UPF. It transports User Plane data for a specific PDU session. It also uses the GTP-U protocol to transport data between the RAN and the core network.”
  - N6: ”Reference point between the UPF and the network. It provides the egress/ingress point for user traffic in order to reach external services.”
  - N9: ”Reference point between two UPFs. It uses the GTP-U to carry UP data when multiple UPFs are involved in a PDU session.”

2.1.4 Network Slicing Implementation at gNB level

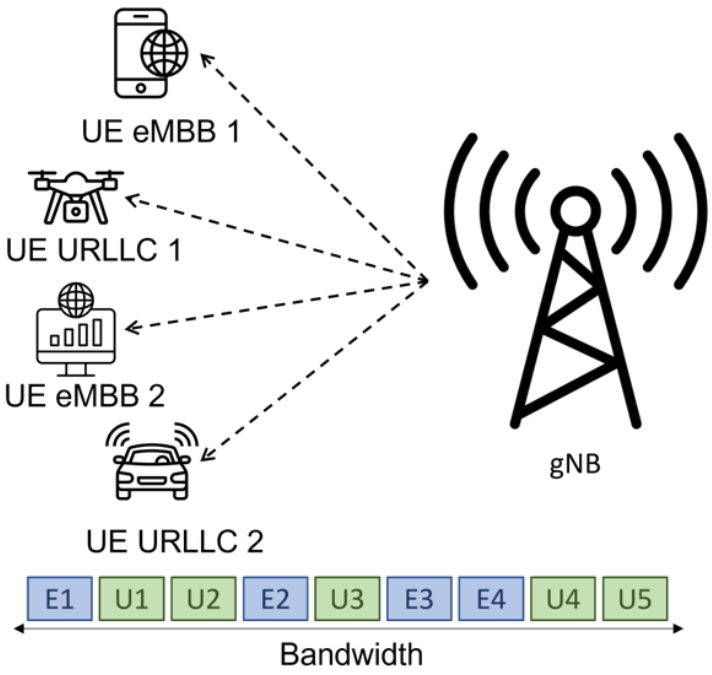


Figure 2.6: Illustration of a 5G Cell Supporting Heterogeneous eMBB and URLLC Services

Network slicing, as presented in section 2.1.2, is an End-to-End (E2E) concept that goes from the 5GC functions to the NG-RAN physical radio resources. While some control plane functions handle slice selection, one of the roles of the gNB is the physical implementation of that choice. To adapt to the diverse needs of the different slices on a common carrier, the notion of Bandwidth Part (BWP) was established in 5G NR. Unlike 4G, this allows for multiple sub-carrier spacings to coexist by allocating each slice a different sub-carrier frequency. This allows the system to separate the slices while still keeping the different traffics on the same network, as can be seen in figure 2.7.

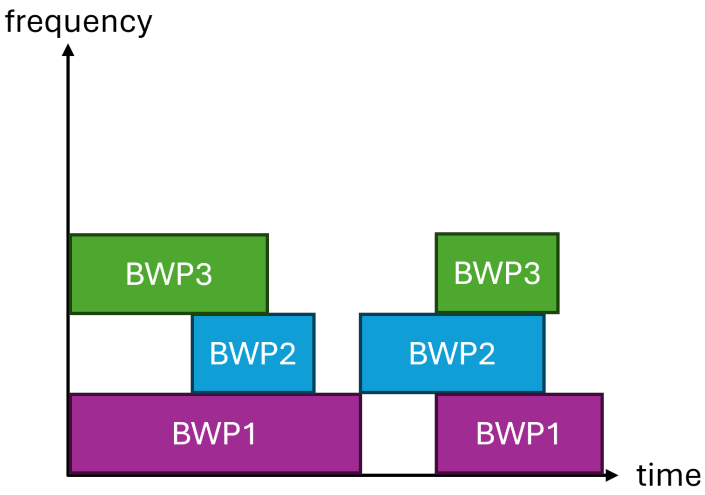


Figure 2.7: Bandwidth parts in 5G

The gNB then has to choose the number of RBs that each slice will get, which is either done statically or dynamically [**?**].

- Static resource allocation: Also called hard slicing. A specific number of resources is strictly reserved for each slice. It guarantees the required reliability, especially for time-critical traffic, by removing the scheduling latency that would have been associated with dynamic resource allocation. However, when the traffic load is low, these RBs are not used, which leads to a waste of resources.

- Dynamic resource allocation: Also called soft slicing. The RBs are allocated in real-time based on the current demand. It provides a more personalised and optimised allocation by guaranteeing that no RB stays under-utilised. However, it can be more complex and costly to implement, and it can also be risky for URLLC traffic since it adds a processing time for the gNB scheduler that needs to make rapid decisions at every transmission interval. A common way to deploy soft slicing is by puncturing an ongoing eMBB traffic with the URLLC transmissions when needed [**?**].

Queuing theory is used to mathematically represent the resource allocation, which will be presented in section A..

2.1.5 Mathematical Background

A. Queuing Theory

Queuing theory is a mathematical theory that provides tools to assess the performance of the system. It allows to mathematically quantify the server occupation and rate of transmission and loss, which then allows to draw conclusions on the delay, the needed throughput, and the average loss rate [**?**].

”Kendall notation is used to represent the different models of queues.

Let $A\backslash B\backslash C\backslash D\backslash E$ be a queue where:

1. A: The law of the arrival process (M for Markov, GI for General Independent, etc)
2. B: The law of the service distribution (M for Markov, GI for General Independent, etc)
3. C: The total number of available servers
4. D: The total number of available resources in the system (servers + buffer)
5. E: The queue's discipline (FIFO for First In First Out, LIFO for Last In First Out, Processor Sharing, Earliest Deadline First, etc.)

"

B. Markov Process and Markov Decision Process

"A Markov Process (MP) is a stochastic process considered memory-less" [**?**]. In other words, the evolution of the system's future state depends solely on its present state, without regard to any previous states or the sequence of steps taken to reach it.

"That is, if $(X(t), t \in \mathbb{R}^+)$ is defined in $\mathbb{R}^+$ with countable values in the space of states *E* , it is a MP if and only if":

$$\begin{aligned} &P\big(X(t_{i+1}) = y \mid X(t_i) = x, \{X(t_{i-1}), t_{i-1} < t_i\}\big) \\ &= P\big(X(t_{i+1}) = y \mid X(t_i) = x\big), \quad \forall\, t_n > 0,\ \forall\, x, y \in E. \end{aligned} \tag{2.1}$$

This implies that the probability of the system being at state y at moment t_{i+1}, while knowing that at the moment t_i it is at the state x, is the same no matter the state of the system at the previous moment t_{i-1}.

The infinitesimal generator is a matrix used to represent the rates of probability of passage from one state to all the possible next states in a Markov Process.

"Let X be a homogeneous MP on E. Its infinitesimal generator is the matrix A such that:

$$A(x, x') = \begin{cases} A(x, x) = -q(x, x) \forall x \in E, \\ A(x, x') = q(x, x) q(x, x') \forall x, x' \in E \text{ and } x \neq x' \end{cases} \tag{2.2}$$

With $q(x, x')$ being the transition probability from state x to state x'."

A Markov Decision Process (MDP) extends a Markov Process by introducing the concepts of an agent, policies, and a reward function. In this framework, the agent interacts with the environment by following a policy to select subsequent actions and states. The agent receives feedback for each decision in the form of a reward, which may be positive or negative. The application of MDP is particularly prominent in network modelling, especially within queuing theory, because the system state depends solely on the current actions of users and equipment rather than on historical states.

The main components of a MDP form a tuple (S, A, P, R, γ):

- S: All the possible states of the system

- A: The set of actions available to the agent may depend on the current state s. Therefore, $A(s) \subseteq A$.
- P: The probability of transitioning to the next state s' from the current state s. It may depend on both the present state s and the current action a, but not on any previous states or actions. This probability is denoted as $P(s' \mid s, a)$. For any given state and action, the sum of all transition probabilities equals 1.

$$\sum_{s' \in \mathcal{S}} P(s' \mid s, a) = 1 \tag{2.3}$$

- R: The reward function represents the reward assigned to an agent after it takes an action a and reaches a state s. This function quantifies the agent's performance up to that point. The reward, denoted as $R(s, a)$, can be either positive or negative.
- γ: Discount factor, $\gamma \in [0, 1]$, represents the weight of immediate reward versus future rewards. The closer the discount factor is to 0, the more short-sighted the agent will be. A discount factor of 1 might cause the system to oscillate and never converge.

It is important to note that the probability of passage P can be replaced with a transition rate matrix $Q(s' \mid s, a)$ showing the rates of passing from a state s to a state s' using action a, also called an infinitesimal generator.

C. Optimization and Bellman's Equation

The agent's main goal is to maximize the total reward it gets over time, not just the immediate reward at each step.

To do that, a policy has to be defined that tells the system which action to choose for every state. The policy is represented by $\pi(a \mid s)$. To know which action and state to choose, the system returns to the value function:

$$V^{\pi}(s) = \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t R(S_t, A_t) \,\middle|\, S_0 = s\right] \tag{2.4}$$

$V^{\pi}(s)$ gives the expected total reward the system will reach when starting from state s and following policy π.

To find the best policy $\pi*$, the Bellman equation should be solved. The Bellman equation shows that the value of a state is the current reward plus the value of the next state affected by the discount factor:

$$V^*(s) = \max_{a \in A}\left\{R(s, a) + \gamma \sum_{s' \in S} P(s' \mid s, a) V^*(s')\right\} \tag{2.5}$$

In other words, the Bellman equation determines the optimal value for state s by selecting the action a that maximises the sum of the immediate reward and the expected future value across all possible subsequent states s'.

This equation is usually very complex to solve, hence the usage of Reinforcement Learning (RL) to approximate it through trial and error by searching for the optimal route to take.

2.2 State of the Art

The introduction of 5G enables network slicing to serve as a primary solution for meeting the diverse requirements of various users and their services. In particular, high bandwidth (eMBB) and low latency (URLLC), which have very different resource requirements [**?**]. Slicing then allows for the creation of multiple virtual slices, each dedicated to a type of traffic.

The problem of slicing dimensioning between the eMBB and URLLC slices has been studied in many works, and many different approaches were considered. Notably, Ege Engin et al. [**?**] uses the Patient and Intelligent PreEmpter (PIPE) algorithm, which dynamically shares resources between eMBB and URLLC, prioritising eMBB throughput while respecting URLLC latency requirements by temporarily puncturing the eMBB traffic as needed for the passage of the URLLC packets. This method is similar to the one used in [**?**], where the eMBB codewords are punctured to allow the dynamic allocation of the incoming URLLC traffic.

Another approach can be found in [**?**] and [**?**], where the reliability of the URLLC slice cannot be guaranteed, as the main focus is on the optimisation of the eMBB traffic allocation. Tenteng Ma et al. [**?**] use another approach to the same problem with the goal of maximising spectral efficiency within a two-slice system by using mixed-integer programming.

All the aforementioned models, and most -if not all- the published models, focus on dynamic resource allocation in 5G between eMBB and URLLC traffic and not a static approach.

Moreover, most of the existing work is developed from the point of view of the DSOs and not of the client of said DSOs. These works mainly focus on the optimal way for the operators to manage and allocate the radio resources to help maximise their use and minimise the cost [**?**], [**?**]. Hence, the resource allocation models developed tend to be dynamic and positioned from the point of view of the operator and not the client. This reveals a gap in the research concerning the needs and demands of the client who will be using the slices provided. However, industrial entities such as EDF, being an energy provider, often are not the operator of their own network, it is instead a client of a DSO. The research from the point of view of the client reveals different needs and requirements than the already established needs of the operator. An industrial client, such as EDF, is not looking to optimise the global 5G network. The main concern is to guarantee that the slice that they are paying for will meet their QoS and performance requirements for a fair price. Yet, there is a clear lack of studies and tools to help the client be prepared before the negotiations with the operator. This work aims to bridge this gap by providing the client with the information needed prior to any discussion or negotiation with the operator.

The articles mentioned were classified in the table 2.2.

In summary, while a lot of work is being developed on operator-centric resource allocation, a gap can be found in client-motivated research, this work's goal is to fill that gap. In order to do that, a system model using guard channels is introduced in section 2.3

Ref	Focus	Approach	Perspective
[**?**]	Resource allocation between eMBB and URLLC	Punctures eMBB traffic with URLLC packets	Operator-centric
[**?**]	Resource allocation between eMBB and URLLC	Punctures eMBB code-words to allow URLLC traffic passage	Operator-centric
[**?**]	Optimization of eMBB resource allocation	Prioritizes eMBB; does not guarantee URLLC reliability	Operator-centric
[**?**]	Optimization of eMBB resource allocation	Prioritizes eMBB; does not guarantee URLLC reliability	Operator-centric
[**?**]	Maximisation of spectral efficiency in slicing	Focuses on eMBB-URLLC joint optimization	Operator-centric
[**?**]	Optimal radio resource allocation	Maximises operator profit and efficiency	Operator-centric
[**?**]	Optimal radio resource allocation	Maximises operator profit and efficiency	Operator-centric
This work	**Optimal QoS for industrial clients**	**Provides tool to evaluate slicing needs before negotiation with DSO**	**Client-centric**

Table 2.2: Summary comparison table for the articles mentioned in the state of the art of Chapter 2

2.3 System Model

This Section introduces a new queuing theory model for 5G cells that dynamically allocates radio resources between eMBB and URLLC slices. eMBB and URLLC will use two different numerologies. Usually, in outdoor deployments, eMBB transmission will use a 30 KHz sub-carrier spacing, while URLLC will use a 60 KHz one, as can be seen in Figure 2.8. Each User Equipment (UE) is supposed to use a single slice at a time.

As can be seen in Figure 2.9, URLLC and eMBB incoming traffics are modelled by a Poisson Process with respective arrival rates λ_U and λ_e.

URLLC and eMBB service laws will be modelled by exponential random variables of parameters μ_e and μ_U. For the sake of modelling simplicity, numerology will be taken into account in the service rate of our queuing model. The system will be able to process URLLC packets at twice the rate of eMBB packets.

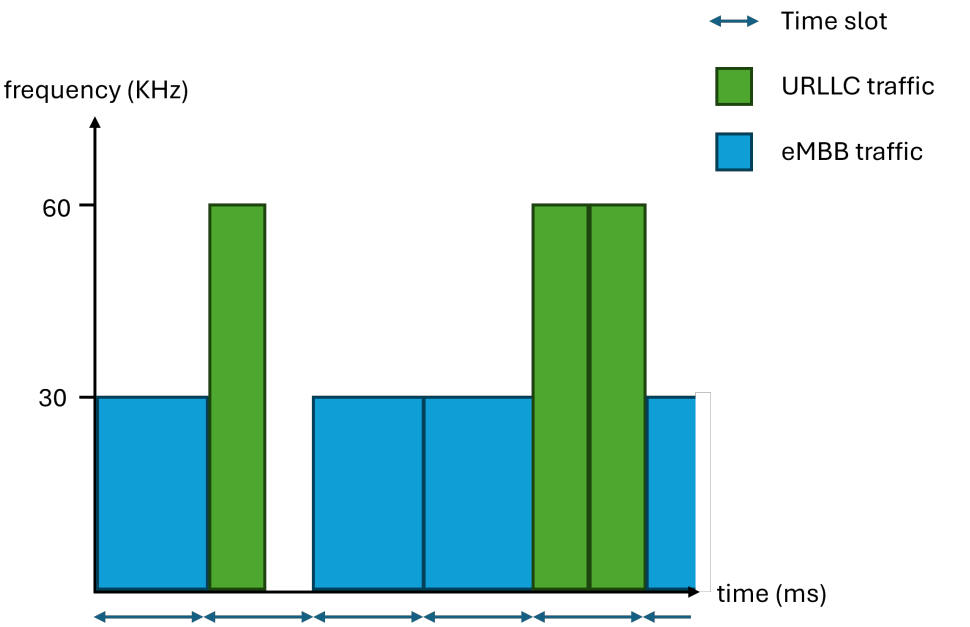


Figure 2.8: Frequency Resource Sharing Between URLLC and eMBB Traffics

The total number of RBs in the system is S. Given that the URLLC packets are time-sensitive and the eMBB packets are data-heavy, a number G of guard channels are dedicated to the URLLC flow. When a URLLC packet arrives, it is serviced if there is at least one RB available, otherwise, it is lost. When an eMBB packet arrives, it is serviced if there are at least $G + 1$ RB available, otherwise, it is sent to an infinite buffer. After the transmission of a URLLC packet, another eMBB packet can take its place, if there are at least $G + 1$ RB available.

"In queuing theory, the system load is defined as the ratio of the incoming traffic to the maximum available service capacity" [**?**]. The URLLC load ρ_U is calculated against the total system capacity available for URLLC traffic. Since in this model, URLLC traffic is given priority and access to all the available resources S. The total capacity for URLLC traffic is then $S\mu_U$, which gives the URLLC load formula:

$$\rho_U = \frac{\lambda_U}{S\mu_U} \tag{2.6}$$

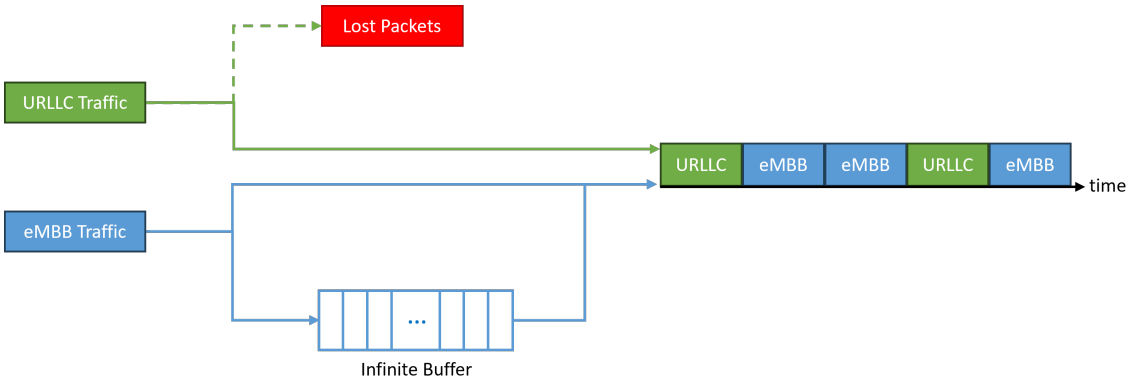


Figure 2.9: URLLC and eMBB Flows and Service

However, by reserving a specific number of guard channels G for URLLC traffic, eMBB traffic is constrained in capacity. In result eMBB traffic has access only to $S - G$ RBs, leading to a total capacity for eMBB traffic of $(S - G)\mu_e$. This leads to the eMBB load formula:

$$\rho_e = \frac{\lambda_e}{(S - G)\mu_e} \tag{2.7}$$

The system state is defined by the vector (x_1, x_2, x_3), where x_1 denotes the number of URLLC packets in service, x_2 denotes the number of eMBB packets in service, and x_3 denotes the number of buffered eMBB packets awaiting transmission.

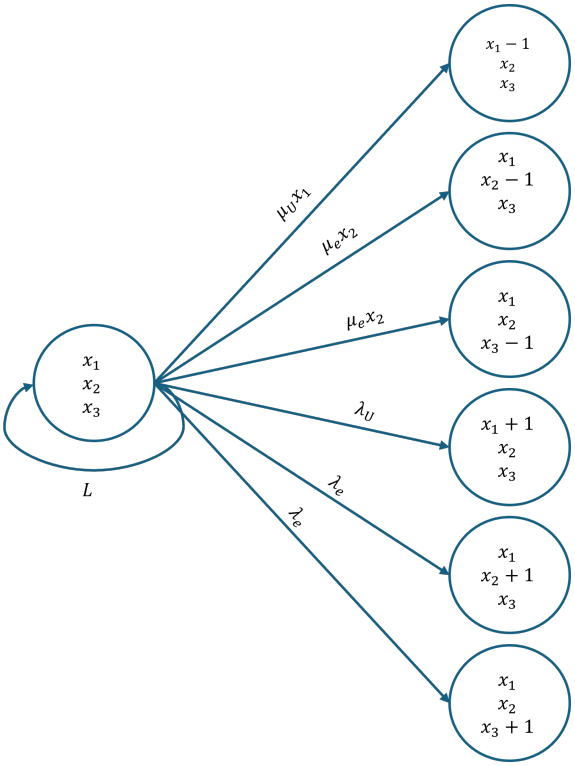


Figure 2.10: The states and actions of the system and their respective rates

The possible actions given state (x_1, x_2, x_3) are six and are divided between transmission and reception of packets.

The first possible action is the transmission of an URLLC packet from the service, if the number of URLLC packets in service (x_1) is not null. Another possibility is the transmission of an eMBB packet from the buffer (x_3), if both the eMBB service line (x_2) and the buffer (x_3) are not empty, and the total number of packets being serviced is less than or equal to the mutualised RBs $(S - G)$. Otherwise, if there is at least one eMBB packet in service, and the buffer is empty, the URLLC packet in service can be transmitted.

The first possible action for reception is, if the total number of packets in service $(x_1 + x_2)$ is less than the total number of resource blocks (S), an URLLC packet can be received into service. However, if the total number of packets in service $(x_1 + x_2)$ is less than the number of mutualised RBs $(S - G)$, an eMBB packet can be received into the eMBB service line (x_3). Otherwise, if the total

number of the packets in service $(x_1 + x_2)$ is greater than or equal to the number of mutualised RBs, an eMBB packet can be received into the buffer (x_3).

The possible transitions from the state (x_1, x_2, x_3) discussed above are summarized as follows:

- $(x_1 - 1, x_2, x_3)$ if $x_1 > 0$, with a rate of $\mu_U x_1$
- If $x_2 > 0$
  - $(x_1, x_2, x_3 - 1)$ if $x_3 > 0$ and $x_1 + x_2 \leq S - G$, with a rate of $\mu_e x_2$
  - $(x_1, x_2 - 1, x_3)$ otherwise, with a rate of $\mu_e x_2$
- $(x_1 + 1, x_2, x_3)$ if $x_1 + x_2 < S$, with a rate of λ_U
- $(x_1, x_2 + 1, x_3)$ if $x_1 + x_2 < S - G$, with a rate of λ_e
- $(x_1, x_2, x_3 + 1)$ if $x_1 + x_2 \geq S - G$, with a rate of λ_e

The states and actions of the system and their respective rates are represented in figure 2.10 with $L = -(\mu_U x_1.1_{\{x_1>0\}} + \mu_e x_2.1_{\{x_2>0\}} + \lambda_U.1_{\{x_1+x_2<S\}} + \lambda_e)$.

This model follows a MP since the transitions and their rates are unaffected by the earlier states. They are only affected by the current states and the arrival and transmission rates.

The infinitesimal generator A is then calculated as:

$$
\begin{array}{c}
 \\
(x_1,x_2,x_3) \\
(x_1-1,x_2,x_3) \\
(x_1,x_2-1,x_3) \\
(x_1,x_2,x_3-1) \\
(x_1+1,x_2,x_3) \\
(x_1,x_2+1,x_3) \\
(x_1,x_2,x_3+1)
\end{array}
\begin{array}{c}
\begin{array}{ccccccc}
(x_1,x_2,x_3) & (x_1-1,x_2,x_3) & (x_1,x_2-1,x_3) & (x_1,x_2,x_3-1) & (x_1+1,x_2,x_3) & (x_1,x_2+1,x_3) & (x_1,x_2,x_3+1)
\end{array} \\
\begin{pmatrix}
L & T_U & T_{e1} & T_{e2} & R_U & R_{e1} & R_{e2} \\
R_U & L & 0 & 0 & 0 & 0 & 0 \\
R_{e1} & 0 & L & 0 & 0 & 0 & 0 \\
R_{e2} & 0 & 0 & L & 0 & 0 & 0 \\
T_U & 0 & 0 & 0 & L & 0 & 0 \\
T_{e1} & 0 & 0 & 0 & 0 & L & 0 \\
T_{e2} & 0 & 0 & 0 & 0 & 0 & L
\end{pmatrix}
\end{array}
\tag{2.8}
$$

where:

$$
\begin{aligned}
L &= -\sum_{j \neq i} q_{ij} = -(T_U + T_{e1} + T_{e2} + R_U + R_{e1} + R_{e2}) \\
T_U &= \mu_U x_1.1_{\{x_1>0\}} \\
T_{e1} &= \mu_e x_2.1_{\{x_2>0\}} \\
T_{e2} &= \mu_e x_2.1_{\{x_2,x_3>0;x_1+x_2\leq S-G\}} \\
R_U &= \lambda_U.1_{\{x_1+x_2<S\}} \\
R_{e1} &= \lambda_e.1_{\{x_1+x_2<S-G\}} \\
R_{e2} &= \lambda_e.1_{\{x_1+x_2\geq S-G\}}
\end{aligned}
$$

The goal of this process is to find the smallest optimal number of RBs G to reserve for the URLLC traffic in order to guarantee the error probability P_{error} stays below or equal to the desired value ϵ set to 10^{-5} in this experience.

$$
G = \min\{g \in \mathbb{N} \mid P_{error}(S, G, \mu_e, \mu_U, \lambda_e, \lambda_U) \leq \epsilon\} \tag{2.9}
$$

Since the process follows a deterministic search, it evaluates the reward at every action and state, getting a negative reward (-1) if the desired error probability is not reached and a neutral reward (0) if the desired error probability is reached.

$$R(s,a) = \begin{cases} 0 & \text{if } P_{error}(S, G, \mu_e, \mu_U, \lambda_e, \lambda_U) \leq \epsilon \\ -1 & \text{if } P_{error}(S, G, \mu_e, \mu_U, \lambda_e, \lambda_U) > \epsilon \end{cases} \tag{2.10}$$

Given that the process works by incrementing the value of g and evaluating, the optimal number of RBs to reserve G is reached at the transition point where the reward shifts from a penalty (-1) to a neutral reward (0).

Having defined the theoretical and mathematical aspects of the problem, the following section details the simulation used to evaluate the model's performance and calculate the optimal number of guard channels to reserve.

2.4 Simulation and results

2.4.1 Simulation methodology

In this section, URLLC packets must be delivered with a transmission error probability $P_{error} = 10^{-5}$. The different simulation parameters are described in Table 2.1. Two simulation scenarios have been developed to quantify the effect of all system parameters on G dimensioning.

The first simulation scenario, depicted in Figure 2.11, shows the required number G of guard resource blocks versus the load of eMBB clients ρ_e. The curves in Figure 2.11 are obtained for different values of S. The URLLC client's arrival rate is fixed to $\lambda_U = 500$.

The second set of simulations, depicted in Figure 2.13, depicts the number G of guard resource blocks versus the load of eMBB clients ρ_e. The curves of Figure 2.13 are obtained for different values λ_U. The total number of resources is fixed to $S = 500$. The different curves in Figures 2.11 and 2.13 are obtained by averaging G on 20000 trajectories of 20000 transitions each. To avoid scale effects, G was normalized according to the following formula

$$G_{per} = \frac{G}{S} \times 100 \tag{2.11}$$

To execute these scenarios and quantify G for each case, a simulation algorithm was developed.

2.4.2 Simulation algorithm

The simulation algorithm operates as follows. For each triplet of ρ_e, ρ_U and S, the number of G guard resource blocks is computed to ensure the transmission error probability remains below P_{error} for URLLC clients. It means that the algorithm starts by generating trajectories for a given triplet and starts from $G = 0$. The simulation algorithm increases G by steps of one until the transmission error probability requirements are fulfilled for the given simulation. The transitions between different states are generated according to the program given in Algorithm

1, which can be found in Appendix A. The full code for the simulation can be found on the public Github [**?**].

Algorithm 1 operates in four main steps. First, it enumerates all the possible transitions from state (x_1, x_2, x_3), whether it be packet arrival or transmission, by checking the system constraints such as the availability of the RBs and the state of the buffer. Second, the system computes the transition rate $\mathcal{R}$ by summing the rates of all the identified possible transitions. Third, it calculates the holding time, which is the time before the next event, from an exponential distribution with parameter $\mathcal{R}$. Finally, the next state (x'_1, x'_2, x'_3) is randomly selected from the pre-identified possible transitions, each transition with a probability of selection proportionate to the ratio of its individual rate to the global rate $\mathcal{R}$. This weighted selection mechanism allows for the transitions with a higher rate to have a higher chance of occurrence, in accordance with MDP.

After running the algorithm, the results show the relationship between the traffic loads, system resources, and the optimal required number of guard channels G. These results will be presented and discussed in section 2.4.3.

2.4.3 Results and interpretation

Figure 2.11 plots the evolution of G_{per} versus the load of eMBB clients ρ_e for different values of S. Figure 2.12 shows the average number of eMBB packets in the buffer for different values of S.

The analysis of Figure 2.11 indicates that regardless of the value of S, G_{per} evolves in a three-stage process. During the first stage, for very low values of eMBB clients load ρ_e, G_{per} is nearly null. As eMBB traffic is very low, no guard RBs are required to grant URLLC clients transmission. As the load of eMBB clients increases, a second stage appears. In this region, the curves are strictly increasing until reaching a plateau.

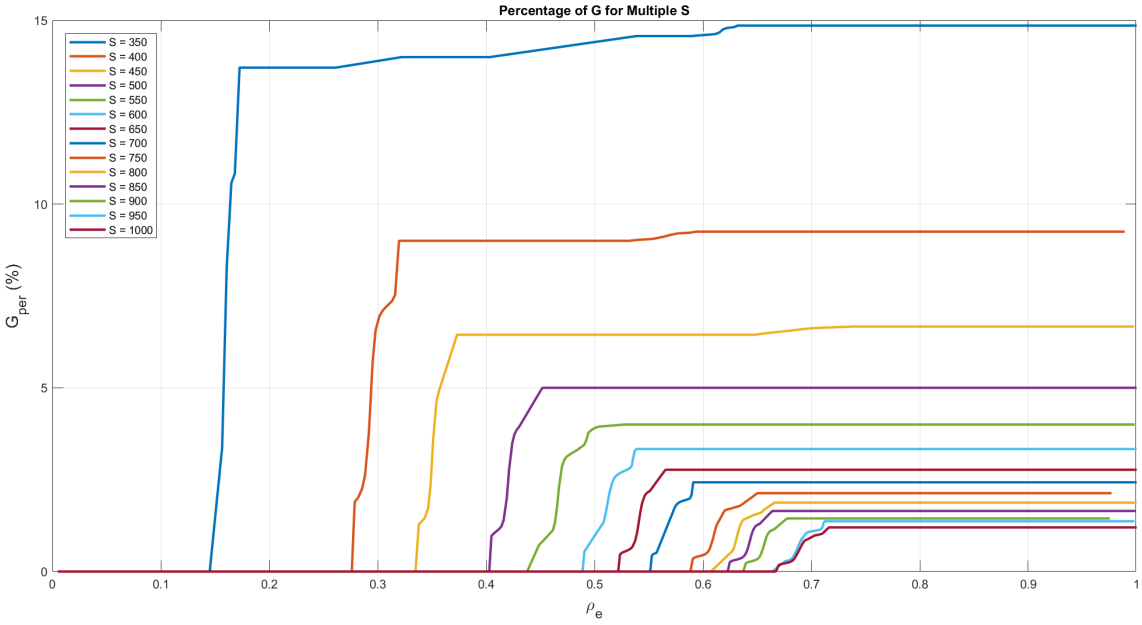


Figure 2.11: Percentage of Dedicated Guard Channels Needed with Respect to the Load of the eMBB Traffic for Different Values of S

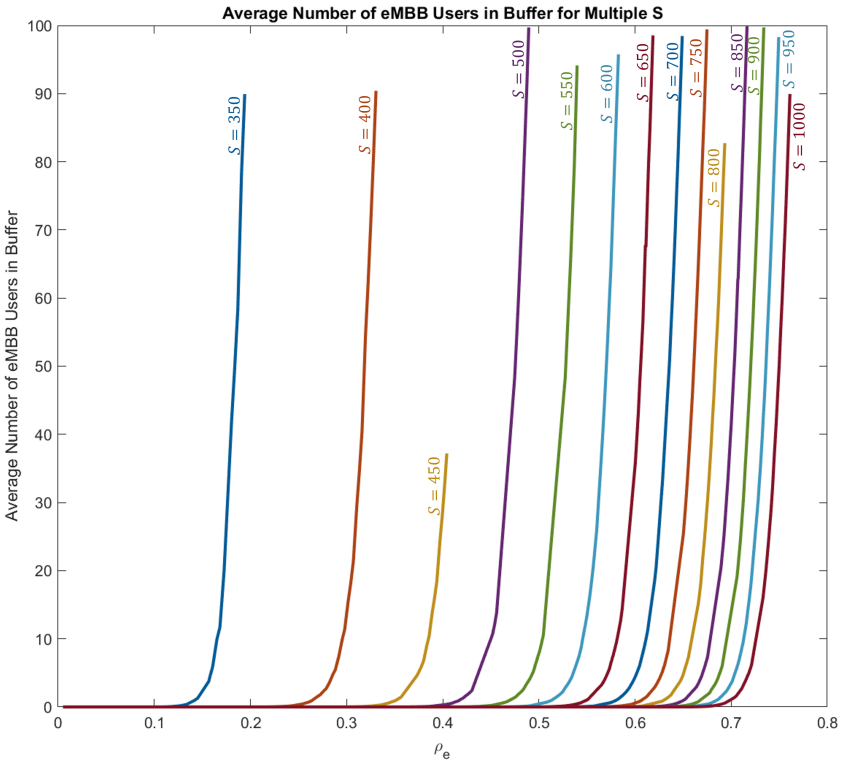


Figure 2.12: Average Number of eMBB Packets in the Buffer with Respect to the Load of the eMBB Traffic for Different Values of S

The G guard RBs grant that URLLC clients are transmitted by abiding by the error probability requirement P_{error}. As ρ_e increases, the number of eMBB clients buffered increases but remains finite.

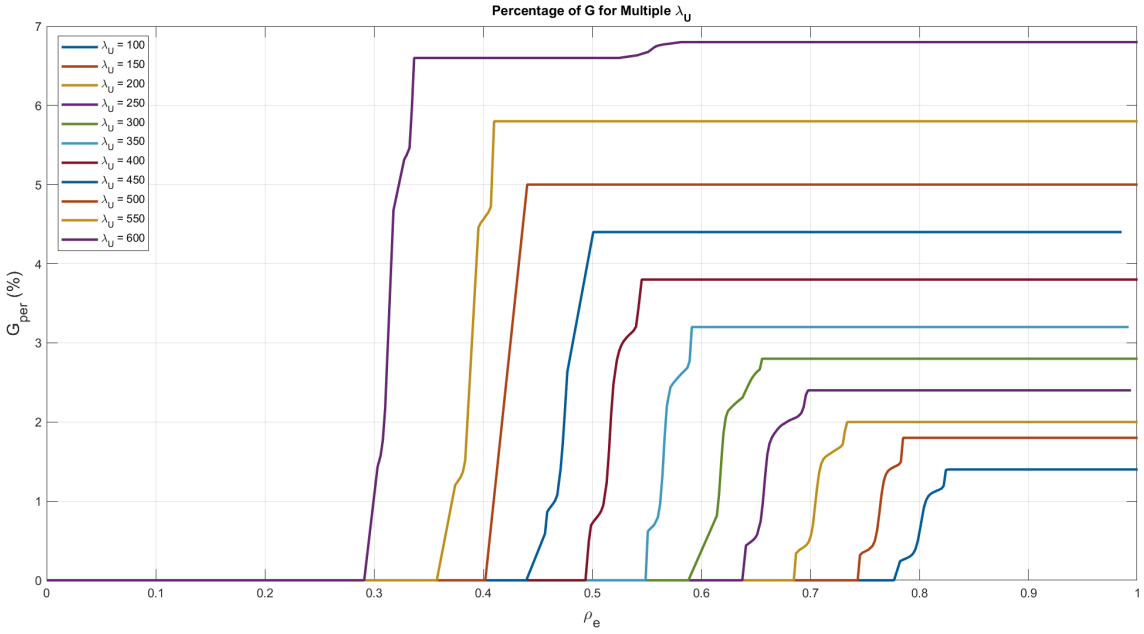


Figure 2.13: Percentage of Dedicated Guard Channels Needed with Respect to the Load of the eMBB Traffic for Different Values of λ_U

The system is stable. In the third and last stage, the system becomes unstable. G_{per} is nearly constant when ρ_e increases and the average number of

buffered clients tends to infinity. G grants that URLLC clients are served by respecting the error probability requirement P_{error}. From the point of view of URLLC clients, the system behaves as a $M/M/G/G$ queue.

It can also be noted that for systems with a very small total number of resource blocks S, the second region of G_{per} starts for smaller values of ρ_e and requires a higher proportion of guard channels (e.g. for $S = 350$, $G_{per} = 16\%$ in the third region). As S increases, the second region starts for higher values of ρ_e and a smaller proportion of G_{per} guard channels is required to grant the URLLC clients with transmission error probability P_{error} (e.g. for $S > 550$, $G < 5\%$, and for $S > 800$, $G < 2\%$).

Figure 2.13 plots the evolution of G_{per} versus the load of eMBB clients ρ_e for different values of URLLC clients' load λ_U and Figure 2.14 shows the average number of eMBB packets in the buffer for different values of λ_U.

The curves in Figure 2.13 have the same general appearance as the ones in Figure 2.11. We can observe the same three regions of the evolution of G_{per} as those identified in Figure 2.11. Furthermore, as URLLC traffic load ρ_U increases, the second region starts from smaller values of ρ_e. G_{per} reaches a plateau for smaller values of ρ_e as λ_U increases.

The curves in Figures 2.12 and 2.14 show the average number of eMBB packets in the buffer in respect to ρ_e for multiple S and λ_U (respectively). The curves in both graphs follow the same pattern, starting at $G = 0$ and having a rapid increase in values when the system over-saturates. In Figure 2.12, the lower the value of S, the sooner the growth is reached. As for Figure 2.14, the lower the value of λ_U, the later the oversaturation is reached.

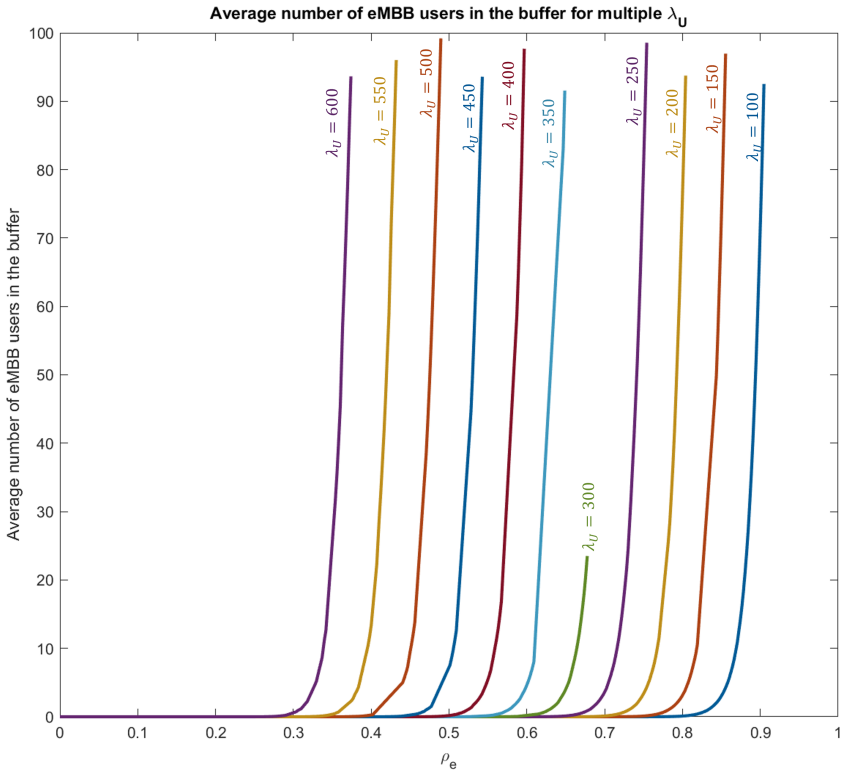


Figure 2.14: Average Number of eMBB Packets in the Buffer with Respect to the Load of the eMBB Traffic for Different Values of λ_U

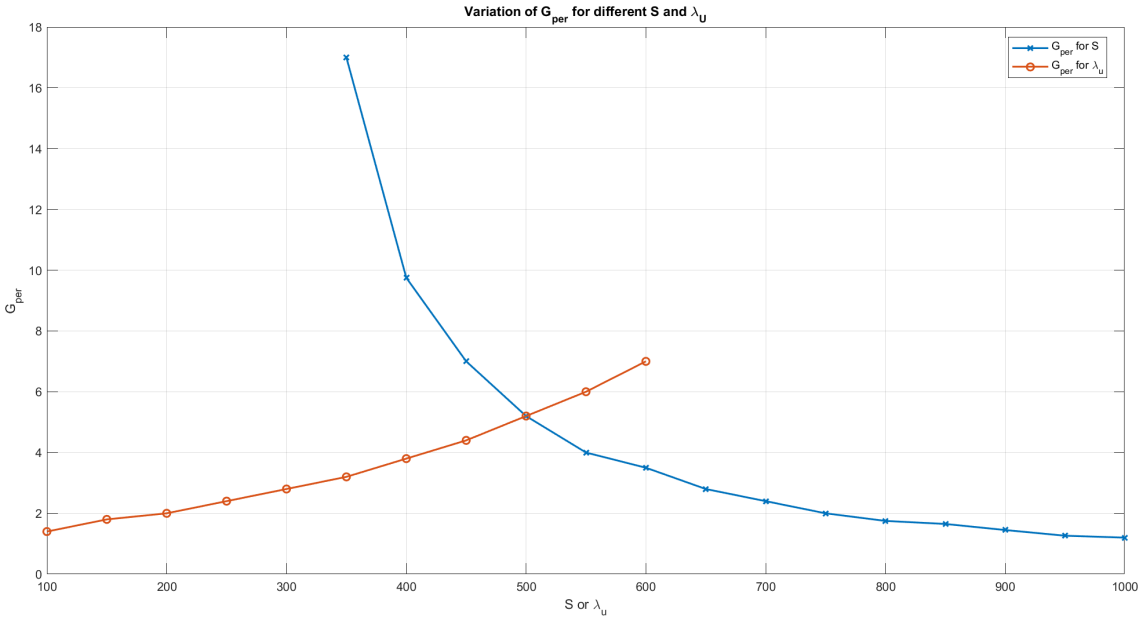


Figure 2.15: Variation of G_{per} in the Steady Stage for Different Values of S and λ_U

It can also be observed from Figures 2.11 and 2.13 that the total number of RBs S has a greater impact on G_{per} than the load of URLLC clients ρ_u. G_{per} in 2.11 decreases rapidly as S increases. As can be seen in Figure 2.15, the value of G_{per} for the different values of S starts at a maximum (16%) and decreases rapidly for higher values of S.

In Figure 2.13, G_{per} decreases as the load of URLLC clients decreases, but that evolution is smoother than in Figure 2.11. G_{per} starts at a minimum (1.8%) and increases slightly as the value of λ_U increases.

It can be concluded that the variation of λ_U and S has opposite effects on G_{per} and that the effect of λ_U is close to being linear while the effect of S is far from it.

The results presented in this section show the impact of the system parameters on resource allocation, leading to the conclusions presented in section 2.5.

2.5 Conclusion

This chapter simulated and studied the effect of the variation of different parameters on the optimal number of RBs to be dedicated to the URLLC traffic. From the point of view of EDF, which relies on third-party MNOs, the ideal situation would be to implement the number of RBs reached in the third region in advance, guaranteeing the maximum loss that they aim for in all possible situations. Such a demand might be exaggerated to ask for from the operator, but it gives EDF an idea of the RBs needed to guarantee a satisfactory maximum loss of URLLC packets, which allows to determine the expectations of the QoS.

Despite the insights provided by this theoretical model, it is important to acknowledge its limitations and the need for further work.

2.5.1 Model Limitations

The approach in this chapter presents certain limitations. From an operator's point of view, eMBB traffic will experience congestion and higher delays. The assumption of an infinite buffer implies potential for a waste of resources for a higher delay in eMBB packets transmission. As a result, a more dynamic scheme is required to adapt G according to eMBB and URLLC traffic evolution. Moreover, this approach assumes that both eMBB and URLLC traffic follow a Poisson process, which does not accurately represent real-world network conditions. Although the inherent randomness of the Poisson process facilitates more tractable simulations, actual network traffic often exhibits less predictable patterns and random bursts of packets.

Finally, as the client, EDF does not have access to any network information beyond the radio level, especially the type of slice the packet is transmitted on. Hence, the next model is a machine learning model that is trained to classify the traffic between eMBB and URLLC. The next model then calculates the optimal number of RBs needed for the URLLC traffic G in Chapter 4.

Chapter 3

Slice Classification Model

3.1 Introduction

Chapter 2 introduced a queuing-theory model that returns the optimal number of resource blocks to be reserved to URLLC traffic to ensure the chosen loss probability. The simulation was tested on a traffic that follows Poisson's law. Given that real-life traffic does not follow Poisson's law, the present chapter introduces a LightGBM model for slice recognition, trained and tested on experimental radio traffic. This model serves as a prerequisite to the dynamic resource allocation machine learning model explored in Chapter 4. The work in this chapter shows that accurate slice recognition can be reached with only access to the lower layers. The LightGBM classifier trained on an experimental 5G SA setup can reach a 97.6% accuracy for some configurations (100MHz bandwidth), outperforming other state-of-the-art low-layer models.

3.1.1 System Notations

Table 3.1 groups the mathematical notations and system parameters used in this chapter.

3.1.2 5G QoS and Low Layer Identifiers

This section groups the key concepts needed to understand the proposed classification model presented in this chapter. It introduces 5G QoS framework, PHY and MAC layers and their channels, and the metrics used to evaluate the model's accuracy and performance.

In 5G, QoS classes are called the 5G QoS Identifier (5QI). Slicing allows for the application of personalised QoS standards for each slice, applying a set of parameters described by the chosen 5QI value. Examples of 5QI values are listed in Table 3.2.

Table 3.1: System Notations Used in Chapter 3 and their Descriptions and Units

Notation	Description	Unit
n	Number of Samples in the Dataset	packets
y_i	Truth Value for the i-th Observation	-
$F_{t-1}(x_i)$	Cumulative Prediction Score from Iterations 1 to $t-1$	-
$h_t(x_i)$	New Decision Tree Added in the Current Step	-
L	Loss Function	-
Ω	Regularization Term	-
TP	True Positive	packets
FP	False Positive	packets
TN	True Negative	packets
FN	False Negative	packets
A	Global Model Accuracy	-
P	Model Precision	-
R	Model Recall	-
$F1$	F1 Score	-
AUC	Area Under the ROC Curve	-

Table 3.2: Sample 5QI Values List [**?**]

5QI	Default Slice	Delay	Error Rate	Priority[1]	Example Services
1	eMBB	100 ms	10^{-2}	20	Conversational Voice
2	eMBB	150 ms	10^{-3}	40	Conversational Video
65	URLLC	75 ms	10^{-2}	7	Mission-Critical Push-to-Talk Voice (MCPTT-V)
82	URLLC	10 ms	10^{-4}	19	Automation
85	URLLC	5 ms	10^{-5}	21	Electricity Distribution

[1] Lower priority value corresponds to a higher priority level.

The 5QI assigned to each slice affects the data provided by the low-level layers of the 5G radio stack (PHY and MAC).

Bidirectional traffic was considered for the experiments. Multiple channels exist in the physical and MAC layers, as can be seen in Table 3.3. Each channel transports either control information or data, both with added indicators for each packet. PDSCH and PUSCH channels transport user data and demodulation information; in addition, they have indicators such as harq, tb_len, and prbs, respectively, the HARQ process ID, the length of the transmitted block, and the number of PRBs allocated for that packet. PDCCH and PUCCH channels transport control information regarding the network connection. The MAC layer maps logical and transport channels together to allow traffic of a logical channel to be transmitted over a transport channel. Transport channels are, in turn, mapped on physical channels by the physical layer.

Table 3.3: PHY and MAC Layers and their Channels

Layer & Direction	Channel	Traffic Content
PHY / DL	PDSCH	User data and UE demodulation information
	PDCCH	Control information
PHY / UL	PUSCH	User data and RAN demodulation information
	PUCCH	Control information
MAC	-	Maps logical and physical channels together

3.1.3 Machine Learning

The term Machine Learning (ML) was first popularised by Arthur Samuel in 1959 as the ”field of study that gives computers the ability to learn without being explicitly programmed” [**?**]. It is a field of AI that enables systems to learn and improve from experience without explicit programming of every step and action.

In 1997, Tom Mitchell provided the formal definition: ”A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E” [**?**].

In traditional programming, the programmer defines the rules for the model to follow, such as a list of *if* and *else*. This requires the programmer to know the logic and outcome of the conditions beforehand. This can be found in standard network engineering, such as the simulation presented in Chapter 2. However, in ML, the model requires only the data and the desired outcome, instead of a set of rules to follow. The model's job can be to figure out the optimal road to reach the desired outcome, or to figure out the rules that connect the data to the output. This broadens the possibilities by resolving problems that are too complex and non-linear for a human to solve using simple *if* and *else* rules.

However, not any dataset fed through a ML model will yield satisfactory results. A fundamental principle in ML is Garbage In, Garbage Out (GIGO), meaning that no matter how complex the model developed is, if the dataset is of low quality, so will the outcome be. This goes to show that ML algorithms do not learn from abstract concepts, they learn from measurable features of the experiment being observed. The ultimate goal of the model at the end is to be able to accurately predict the output for new unseen data, to do that, it has to capture the right trend of the seen data first. The two main errors that can affect the accuracy of the trend prediction in the model are the bias error and the variance error. The bias error is when the model makes wrong assumptions while learning, a high bias leads to the model missing the relations between the features and the output. A model with a high bias error is called underfit. On the other hand, the variance error is when the model is way too sensitive to very small changes in the dataset, a high variance error can lead to the model considering noise in the dataset. A model with a high variance error is called overfit. This leads to the bias-variance tradeoff, which represents the relation between the model's complexity and its accuracy. The more accurate the model, the lower the bias, the more complex the model, the higher the variance. An example of underfitting and overfitting can be seen in figure 3.1.

Bagging, or Bootstrap Aggregating, is a way to limit variance and help prevent

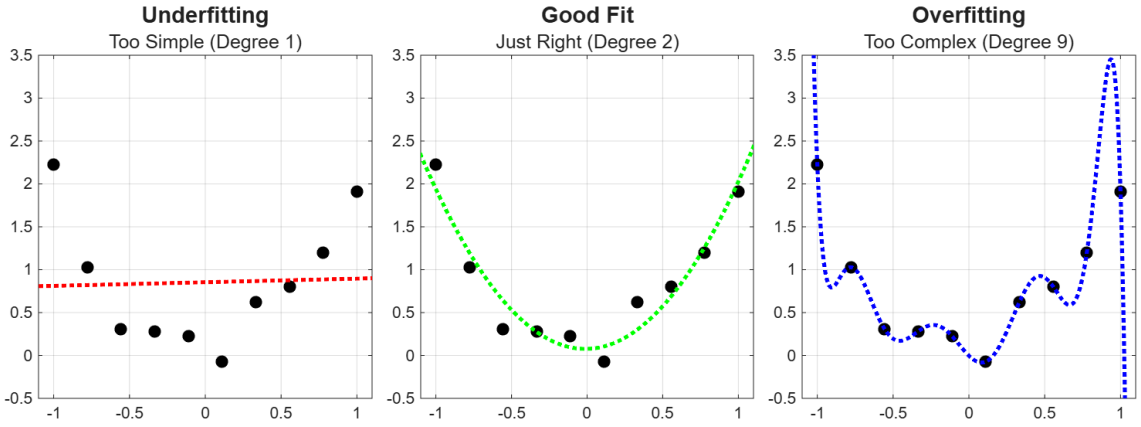


Figure 3.1: An Example of Underfitting, Overfitting, and a Robust Fit in ML [**?**]

overfitting in a model by simultaneously training different base models on separate random subsets of the training dataset [**?**].

The types of ML can be divided into four main categories [**?**]:

- Supervised Learning: The algorithm learns the tendency of the data using a set of examples of input-output pairs. The model is trained with a correctly labelled data set to learn and then tested with the remaining unlabeled data.
- Unsupervised Learning: The algorithm learns patterns from the unlabeled data and acts on them without guidance. The algorithm groups the data by similarity and patterns.
- Semi-Supervised Learning: The algorithm uses a small labelled dataset in order to learn the rules, and then applies them to the rest of the unlabeled data in order to categorise it.
- Reinforcement Learning: This type of model introduces the concept of an agent that continuously learns from the environment while maximising a reward and exploring the possible actions and states.

3.1.4 LGBM Algorithm

”Light Gradient-Boosting Machine (LGBM) is a supervised learning model and an open-source gradient-boosting algorithm developed by Microsoft” [**?**].

A. Optimization via Gradient Boosting

Unlike the value function $V^{\pi}(s)$ in MDP presented in section B., which works by optimising over an infinite horizon, LGBM optimises the objective function O by gradually adding weak learners at each level [**?**]. The model minimises:

$$O^{(t)} = \sum_{i=1}^{n} L(y_i, F_{t-1}(x_i) + h_t(x_i)) + \Omega(h_t) \tag{3.1}$$

L denotes the loss function, which quantifies the difference between the predicted and actual values. The regularisation term Ω penalises model complexity

to mitigate overfitting, as discussed in Section 3.1.3, by encouraging a balance between accurately fitting the training data and maintaining generalisability to unseen data.

The rest of the terms in this formula represent:

- n: Total number of samples in the dataset.
- y_i: The true value of the i-th observation.
- $F_{t-1}(x_i)$: The cumulative prediction score of the model from iterations 1 to $t-1$.
- $h_t(x_i)$: The new decision tree added in the current step, its goal is to minimise the residual error (the gap between y_i and F_{t-1}).

In addition, the learning rate is a hyperparameter with a value between 0 and 1, set by the programmer in order to control the model's step size while developing the decision tree. The lower the learning rate, the slower the training process, but the model has a better convergence [**?**].

B. Leaf-wise Growth Strategy

Gradient Boosting Decision Tree (GBDT) is a ML approach that develops a predictive model by incrementing steps in a decision tree. Most GBDT models grow trees level-wise, which can be inefficient and time-consuming [**?**].

LGBM's leafsplit strategy, as can be seen in Fig. 3.2, allows for better accuracy by choosing the leaf with the highest gain to split at every level, as can be seen in Figure 3.2. It does that by using a pruning mechanism that abandons the paths with low potential. This approach increases efficiency by reducing the total number of splits required to achieve satisfactory accuracy.

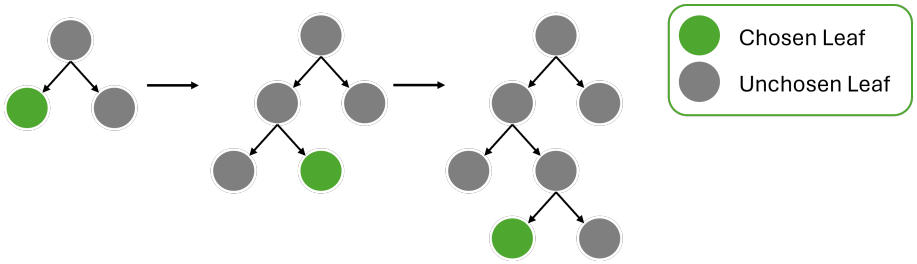


Figure 3.2: Chart Showing LGBM's Leafsplit Strategy

C. Exclusive Feature Bundling (EFB)

Traditional GBDT encounters significant challenges when optimising high-dimensional datasets because training complexity increases proportionally with both the number of instances and features. LGBM uses Exclusive Feature Bundling (EFB) to reduce this complexity. Often in large-scale applications, the features are sparse and mutually exclusive, meaning that they usually do not take non-zero values at the same time. EFB allows to identify these sparse features and then to group them into a single feature that preserves the original information. The

creation of these bundles enables EFB to reduce computational complexity from $O(data \times features)$ to $O(data \times bundles)$, where the number of bundles is substantially smaller than the number of features. This reduction in complexity enables the model to achieve faster training speeds and lower memory consumption while maintaining accuracy.

D. Gradient-based One-Side Sampling (GOSS)

Traditional GBDT models scan all data instances to calculate the gain, resulting in increased resource consumption as the number of samples n increases. To address this issue, LGBM introduces GOSS, which optimises the process by recognising that not every data instance contributes equally to the computation of information gain.

To implement that, the model uses the following mechanisms:

- Gradient as Error Proxy: The gradient of each instance is considered as a measure of training error. The instances with the higher gradient are the ones that the model hasn't learned well yet, hence contributing more to the information gain.
- Selective Down-Sampling: The algorithm targets the instances with lower gradient with random sampling in order to reduce the data size processed during training.
- Information Gain Compensation: The algorithm adds a weight multiplier to the instances with a small gradient in order to ensure a global view of the data tendencies.

The way the GOSS samples the dataset allows the algorithm to achieve a higher training speed than traditional GBDT models while maintaining an accuracy close to the one reached when the whole dataset is studied.

3.1.5 Evaluation Metrics

To analyse the results of such a model, the precision, recall, and f1-score are computed from the following values [**?**]:

- TP: Number of True Positives
- FP: Number of False Positives
- TN: Number of True Negatives
- FN: Number of False Negatives

The metrics used throughout the chapter to quantify the model's performance are as follows.

The precision shows the accuracy of a positive prediction. It gives the ratio between the number of packets that the model has correctly labelled positive and

the total number of packets labelled positive. It is computed using the following formula:

$$P = \frac{TP}{TP + FP}$$

The recall shows the ability of the model to find positive instances. It shows from all the positive values how many the model correctly identified. The recall is computed using the following formula:

$$R = \frac{TP}{TP + FN}$$

As for the F1-score, it is a global metric used to evaluate the model in both precision and recall. It is computed using the following formula:

$$F1 = 2 \times \frac{P \times R}{P + R}$$

Finally, the global accuracy of the model is computed from the following formula:

$$A = \frac{TP + TN}{TP + FP + TN + FN}$$

Another evaluation metric is the Receiver Operating Characteristic (ROC) curve, which demonstrates the model's ability to distinguish between two classes. It plots the True Positive Rate ($TPR = \frac{TP}{TP+FN}$) against the False Positive Rate ($FPR = \frac{FP}{FP+TN}$) at various thresholds. The threshold τ is a value between 0 and 1 selected to impose a decision constraint on the model. Typically, the model outputs a probability $P(class)$ rather than a discrete class label. If $P(class) \geq \tau$, the output is classified as positive, otherwise, it is classified as negative. The ROC curve represents the relationship between TPR and FPR across varying threshold values.

- For $\tau = 1$, the model is extremely strict. Nothing is classified as positive, which means that $TPR = FPR = 0$. This is the bottom-left corner of the ROC curve as can be seen in Figure 3.4.
- As τ decreases, the model becomes less strict. It starts classifying more TP instances, but it also mistakenly includes FP instances.
- For $\tau = 0$ the model is extremely reckless. Everything is classified as positive, which means that $TPR = FPR = 1$. This is the top-right corner of the ROC curve as can be seen in Figure 3.4.

A receiver operating characteristic (ROC) curve that approaches the top left corner indicates superior classification performance, as can be seen in Figure 3.3.

A way to quantify this performance is the Area Under Curve (AUC) indicator, which is the proportion of the area that is under the ROC curve, ranging from 0 to 1, as can be seen in Figure 3.4.

- $AUC = 0$ Perfectly incorrect classification (always wrong)
- $AUC < 0.5$ Bad classification (model performs worse than random)

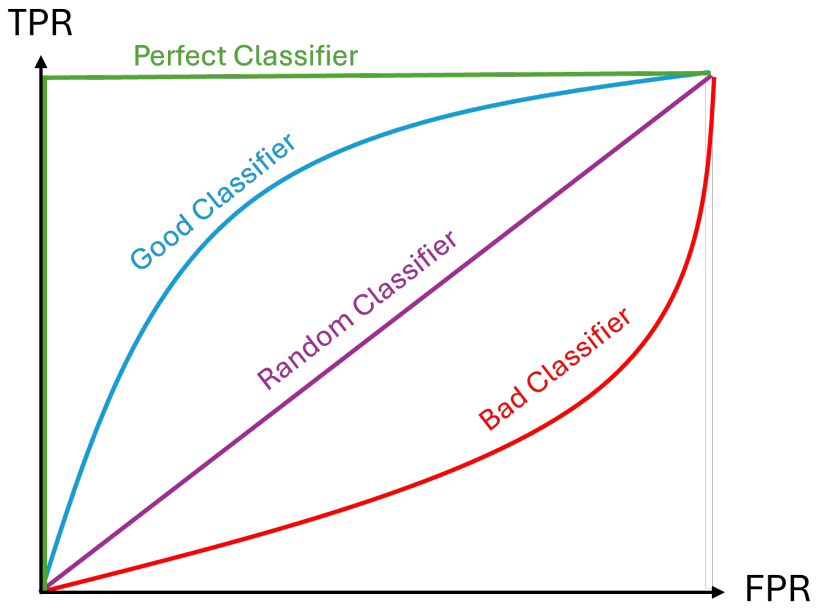


Figure 3.3: Different ROC Curves for Different Classifier Model Performances

- $AUC = 0.5$ Random classification
- $AUC > 0.5$ Good classification (model performs better than random)
- $AUC = 1$ Perfect classification (always right)

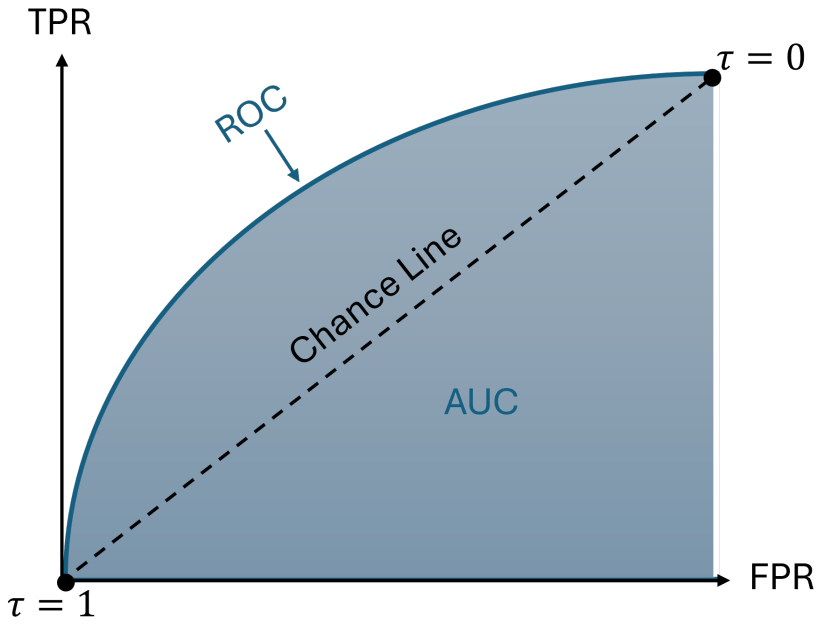


Figure 3.4: An Exemple of the ROC, AUC, and Chance Line of a Model

The following section presents and discusses the related works on slice and service classification in the state-of-the-art.

3.2 State of the Art

With the introduction of slicing, traffic classification has become crucial to guarantee optimal QoS standards for the different groups of users. The literature shows that traffic classification can be approached following two main axes: either classifying by type of service or by slice type. The related work can be grouped into two families: those that classify the traffic with information access from all layers, and those that do it only using the lower layers.

3.2.1 Classification with information access from all layers

A first approach to slice identification described in [**?**] relies on multiple Key Performance Indicators (KPI) features such as Guaranteed Bitrate (GBR), roaming, and slice type. Several supervised machine learning models were tested and compared, with a coarse decision tree giving a perfect precision of 100%. A similar approach can be found in [**?**] where a Deep Learning (DL) model called DeepSlice is introduced. It is able to identify the network slice even in the case of network failure using high-layer KPIs. The DeepSlice model returned a 91% overall accuracy. Another approach can be found in [**?**], where an Explainable Machine Learning (XAI) framework was introduced that classifies slices using high-level QoS metrics. This approach allowed for an accuracy of over 99%. However, these three models' results rely on information provided from higher layers, which are rarely accessible in practice.

3.2.2 Classification using only information from low layers

Works, such as [**?**] and [**?**], focus on an eMBB traffic flow classification by extracting information from the physical (PHY) and MAC layers of the generated traces. Their LightGBM models returned 95% accuracy for [**?**] (5G NSA) and 89% accuracy for [**?**] (5G SA). In [**?**], three machine learning models were tested to classify different applications in an NSA environment having access only to the PHY layer. Their Long Short-Term Memory (LSTM) model returned up to 41% accuracy, the k-Nearest Neighbours (k-NN) model returned up to 50.65% accuracy, and the best results came from the Random Forest (RF) model that returned up to 87.73% accuracy.

On the one hand, the literature shows that traffic service classification has already been greatly investigated. Results have shown excellent accuracy when depending on high layers, yet showing challenges while relying only on the low layers. On the other hand, traffic slice classification is less discussed, with some works returning near-perfect results when it relies on KPIs obtained from high layers. Approaches using only low layers are yet to be developed. This shows a clear gap in the literature: the possibility to classify eMBB and URLLC slices with only access to the PHY and MAC layers. This is the contribution of this work, where a LightGBM model is trained and tested on the PHY and MAC layers of a mixed URLLC and eMBB traffic dataset.

Table 3.4 provides a short classification of the previously stated works.

Ref	Focus	Approach	ML Model	System - Layer	Acc.
[**?**]	Slice Classif.	Supervised ML using high-layer KPI (GBR, roaming, slice ID)	Decision Trees, SVM, k-NN	SA - High Layers	100%
[**?**]	Slice Classif.	DeepSlice model using device/network KPIs (QCI, delay, loss, UE type)	Random Forest, CNN (DLNN)	SA - High Layers	91%
[**?**]	Slice Classif.	Explainable Machine Learning (XAI) using high-level QoS (jitter, packet loss)	Random Forest, Decision trees	SA - High Layers	99%
[**?**]	Service Classif.	LightGBM on PHY-layer records for encrypted flows	LightGBM	NSA - PHY	95%
[**?**]	Service Classif.	Refined LightGBM on standalone traffic	LightGBM	SA - PHY/MAC	89%
[**?**]	Service Classif.	Classified different applications in NSA traffic	LSTM, k-NN, Random Forest	NSA - PHY	87.7%
This Work	**Slice Classif.**	**LightGBM classification restricted to PHY/MAC traces**	**LightGBM**	**SA - PHY/MAC**	97.6%

Table 3.4: Summary Comparison of Representative Works on 5G Traffic Classification

3.3 System Model

3.3.1 Input Dataset Generation

The purpose of this section is to generate a dataset that is representative of a real-life SA 5G traffic, with a mixed eMBB and URLLC traffic. Unlike the Poisson-based traffic adapted in Chapter 2, this dataset was obtained from a 5G SA test network that reproduces real-life radio conditions.

A. Dataset Generation Using a Rohde & Schwarz Mobile and Network

The first approach for the dataset generation was with a Rohde & Schwarz network emulator and a Commercial Off-The-Shelf (COTS) smartphone equipped with a Rohde & Schwarz application for traffic logging. The experiment was conducted in the Faraday Shield of Télécom ParisTech to avoid any outside disturbance. The mobile was, in its turn, put in a smaller Faraday Shield to guarantee its connection to the right network, as can be seen in Figure 3.5.



Figure 3.5: Rohde & Schwarz Hardware Setup

The network emulator was controlled through an interface that can be seen in Figure 3.6. A 5G radio cell was configured operating in the n77 band with a frequency of 3.7 GHz. The chosen bandwidth was 100MHz, and the subcarrier spacing was 30 kHz.

The radio parameters used can be seen in Table 3.5.

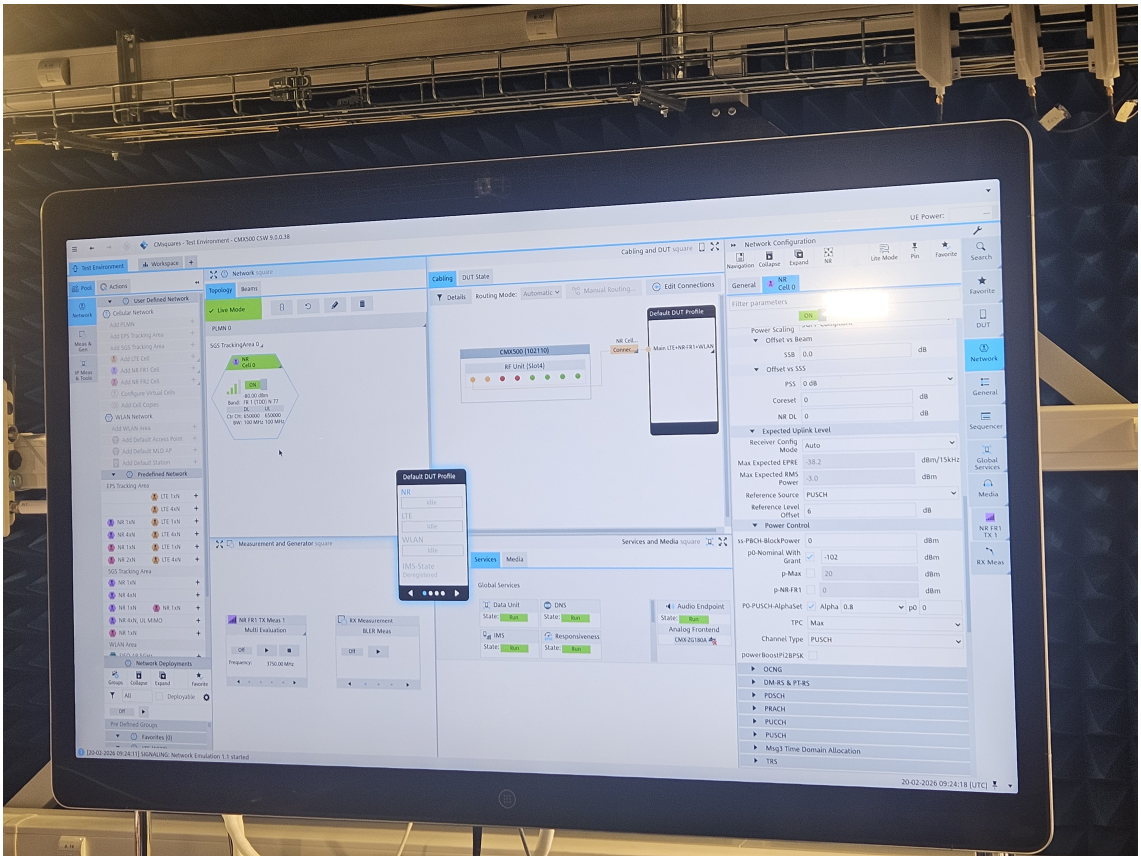


Figure 3.6: Rohde & Schwarz Interface

Table 3.5: Radio Parameters for Dataset Generation

Parameter	**Value**
Carrier frequency	3891.6 MHz (NR ARFCN 659424)
System bandwidth	100 MHz
Number of PRBs	273
Subcarrier Spacing	30 kHz

The dataset generated included a total of 5,069,00 packets (2,105,000 URLLC; 2,964,000 eMBB); only the PHY and MAC layers were logged into the dataset. The data was later sanitised to fit a dataframe suitable for ML training, as can be seen in Table 3.6

Table 3.6: Simulation Parameters for Dataset Generation Using Rohde &

Parameter	**Value**
Slices configured	URLLC, eMBB
URLLC traffic	Small TBs (100b), ∼1s inter-arrival time
eMBB traffic	Large TBs (1000b), bursty inter-arrival time (TCP session)
Layers kept	PHY and MAC only
Total dataset size	2,964,000 URLLC; 2,105,000 eMBB
Train/Test split	70/30

To represent URLLC traffic, a mix of ICMP and UDP small-sized packets was

used. For eMBB, more demanding applications were used, such as Iperf and large file downloads with curl. See Table 3.7 for more details.

Table 3.7: Simulation Traffic Profiles Used for Dataset Generation

Test Number	URLLC Traffic Type	eMBB Traffic Type
1	1x Ping	None
2	None	1x Youtube
3	None	1x Ping

B. Dataset Generation Using an Amarisoft UE Simulator and an Ericsson Network

The other approach used to generate the traffic needed to train and test the model was obtained from a 5G SA network operated in an EDF R&D laboratory. An Amarisoft UE simulator and an Ericsson gNB and core network were used to set up UEs that use either an URLLC or eMBB slice following the 5QI classes.

The radio parameters used in the dataset generation were grouped in Table 3.8.

Table 3.8: Radio Parameters for Dataset Generation

Parameter	Value
Carrier frequency	3891.6 MHz (NR ARFCN 659424)
System bandwidth	100 MHz
Number of PRBs	273
Subcarrier Spacing	30 kHz

The chosen frequency, 3891.6 MHz, falls in the 5G NR band n77 (3.3-4.2 GHz), which is a widely deployed 5G band due to its availability. The maximum channel bandwidth defined for n77 is 100 MHz as specified in [**?**], which is what was used in this experiment, as it provides enough resources to study both URLLC and eMBB presence. The number of PRB depends on the bandwidth (100 MHz) and subcarrier spacing (30 kHz), which is why 273 PRBs were chosen to respect the standard NR resources grid.

As for the traffic model chosen, URLLC packet size is very small since it is mission-critical traffic, with a very short inter-arrival time, almost periodic, in addition to some sudden bursts of traffic. As for the eMBB traffic, packets tend to be large with a very variable inter-arrival time, with traffic arriving almost exclusively in bursts. In addition, URLLC traffic requires a lower number of RBs than eMBB traffic.

A total of 1,811,648 packets (1,448,320 URLLC; 363,328 eMBB) were generated; only the PHY and MAC layers were logged into the dataset. The data was later sanitised to fit a dataframe suitable for ML training, as can be seen in Table 3.9.

Table 3.9: Simulation Parameters for Dataset Generation

Parameter	Value
Slices configured	URLLC, eMBB
URLLC traffic	Small TBs (100b), ∼1s inter-arrival time
eMBB traffic	Large TBs (1000b), variable inter-arrival time
Layers kept	PHY and MAC only
Total dataset size	1,448,320 URLLC; 363,328 eMBB
Train/Test split	70/30

To represent URLLC traffic, a mix of ICMP and UDP small-sized packets was used. For eMBB, more demanding applications were used, such as Iperf and large file downloads with curl. See Table 3.10 for more details.

Table 3.10: Simulation Traffic Profiles Used for Dataset Generation

Test Number	URLLC Traffic Type	eMBB Traffic Type
1	1× Ping	None
2	1× Small UDP	None
3	None	3× File Download (1GB, 500MB, 100MB)
4	None	1× TCP flood (Iperf3)
5	1× Ping	2× File Download (500MB, 200MB)
6	1× Ping	1× TCP flood (Iperf3)

3.3.2 Model and Training

After the construction of the aforementioned dataset, this section details the ML model that was developed to characterise the traffic between URLLC and eMBB with low-layer access. This section explains the model selection, the choice and implementation of the engineered features, and the training and testing process.

As was mentioned in Section 3.2, the LightGBM model was already tested on a similar problem in [**?**] and [**?**], for service recognition in eMBB traffic, and it returned very promising results. It returned an accuracy of 89% in [**?**] and 95% in [**?**]. Hence, it was chosen as a start for this slice recognition model, and the results it yielded justified the lack of need for another model.

Table 3.11: LGBM Hyperparameters

Hyperparameter	Value
num_leaves	31
max_depth	-1
learning_rate	0.01
n_estimators	100
subsample	1
colsample_bytree	1
min_child_samples	20
objective	binary (URLLC vs eMBB)
threshold	0.5

In addition, it was also chosen for its great compromise between training time, complexity, and accuracy. The LGBM model can achieve great accuracy in a small training time thanks to optimised built-in mechanisms.

A Gradient Boosting Decision Tree (GBDT) was used with a bagging data sample strategy and a learning rate of 0.01. Since the model is used to classify between URLLC or eMBB traffic, the objective is binary.

The model's hyperparameters can be found in Table 3.11.

Training of the model started by feeding it the complete, unmodified, low-level traces. Figure 3.7 is an example of one eMBB log and one URLLC log.

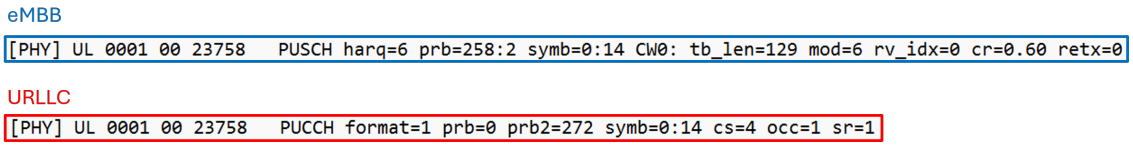


Figure 3.7: Example of an eMBB and URLLC Trace

The full list of these parameters can be found in Table 3.12.

Table 3.12: List of Features Found in the Traces

Name	**Description**
harq	HARQ process ID
prbs	Number of PRBs allocated
tb_len	Transport block length (bits)
mod	Modulation order
rv_idx	Redundancy version index
cr	Code rate
retx	Retransmission count
crc_ok	CRC success indicator
snr	Signal-to-noise ratio
mcs	Modulation and coding scheme index
total_len	MAC packet length (bytes)
dir_DL	Dowlink direction
dir_UL	Uplink direction
type_PDCCH	Physical Downlink Control Channel (PDCCH) message
type_PDSCH	Physical Downlink Shared Channel (PDSCH) message
type_PUCCH	Physical Uplink Control Channel (PUCCH) message
type_PUSCH	Physical Uplink Shared Channel (PUSCH) message
type_MAC	Medium Access Control (MAC) layer PDU

However, the model struggled to distinguish between small-sized eMBB packets and URLLC traffic. This is why some contextual packet features were engineered, like ”Time to next packet” and ”Rolling packet size mean”.

These features were built working in $5ms$ windows around each packet to gather a sufficient amount of data while keeping data size and model I/O latency in a reasonable range (for possible Real Time applications) as can be seen in Figure 3.8. By computing the values of the features over a rolling window, the model is able to better detect the packet tendencies and the evolution of their values.

The full list of these engineered features is available in Table 3.13.

Table 3.13: List of Engineered Features for the Traces

Name	**Description**
inter_packet_time_ns	Time elapsed since previous packet arrival (ns)
time_to_next_packet_ns	Time until next packet arrival (ns)
harq_rolling_mean	Mean HARQ process ID
harq_rolling_std	Standard deviation of HARQ process ID
prbs_rolling_mean	Mean number of PRBs allocated
prbs_rolling_std	Standard deviation of PRBs allocated
tb_len_rolling_mean	Mean transport block length (bits)
tb_len_rolling_std	Standard deviation of transport block length (bits)
mod_rolling_mean	Mean modulation order
mod_rolling_std	Standard deviation of modulation order
rv_idx_rolling_mean	Mean redundancy version index
rv_idx_rolling_std	Standard deviation of redundancy version index
cr_rolling_mean	Mean code rate
cr_rolling_std	Standard deviation of code rate
retx_rolling_mean	Mean retransmission count
retx_rolling_std	Standard deviation of retransmission count
crc_ok_rolling_mean	Mean CRC success indicator
crc_ok_rolling_std	Standard deviation of CRC success indicator
snr_rolling_mean	Mean signal-to-noise ratio
snr_rolling_std	Standard deviation of signal-to-noise ratio
mcs_rolling_mean	Mean modulation and coding scheme (MCS) index
mcs_rolling_std	Standard deviation of MCS index
format_rolling_mean	Mean PUCCH format indicator
format_rolling_std	Standard deviation of PUCCH format indicator
total_len_rolling_mean	Mean MAC packet length (bytes)
total_len_rolling_std	Standard deviation of MAC packet length (bytes)

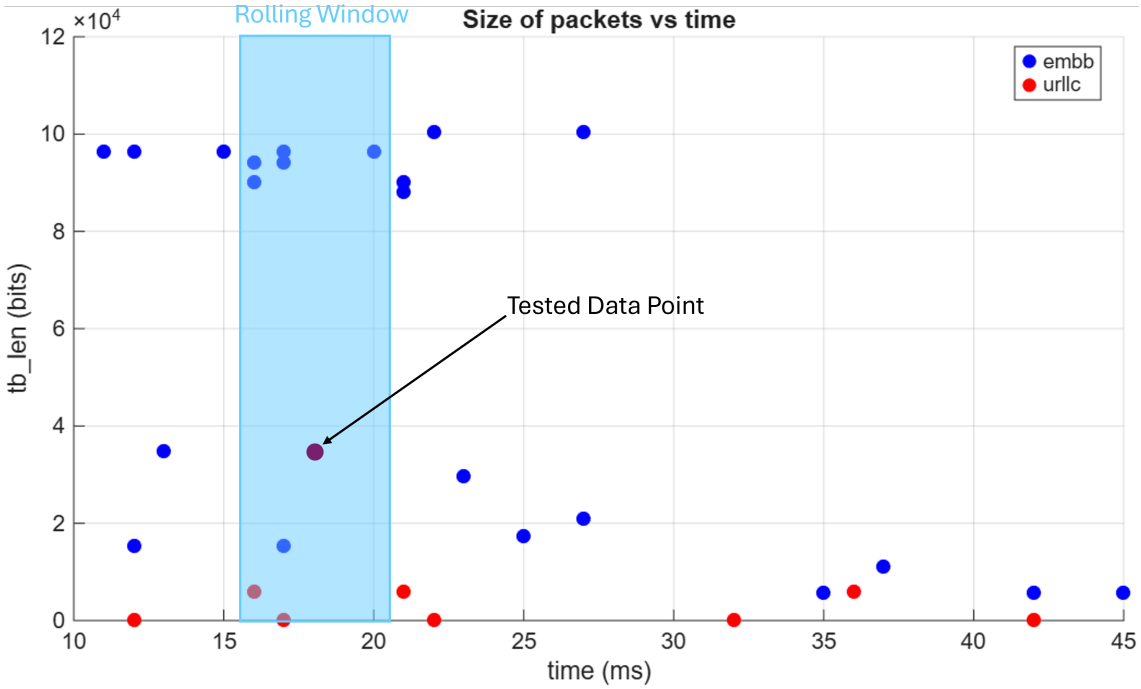


Figure 3.8: Example of a Rolling Window

A. LGBM Model Algorithm

The LGBM model is coded over multiple modules as can be seen in Figure 3.9. The full code for the model can be found on the public GitHub [**?**].

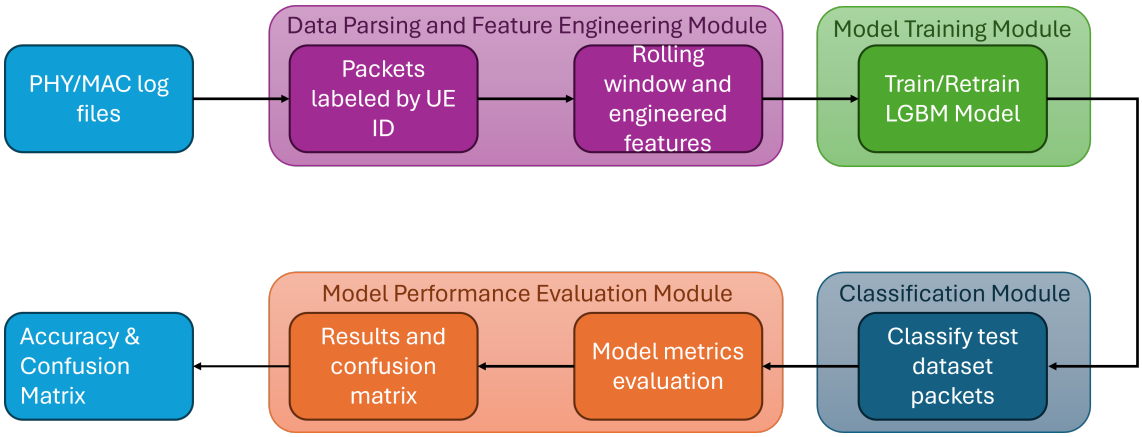


Figure 3.9: Light Gradient-Boosting Machine model pipeline

The first module is shown in Algorithm 2 in Appendix B. In this module, the algorithm works in three phases to produce the feature matrix X from the raw log file F and a chosen rolling window size W. As can be seen in Figure 3.10, this first module parses the raw log file F in order to filter only the low-layer information. The algorithm then extracts the features using regular expressions (Regex) and sorts them chronologically. Temporal features are then created following the rolling window size W. The standard and engineered features are placed into a matrix X, with the missing values filled with zeros, and the ID and timestamps removed to generate the final matrix.

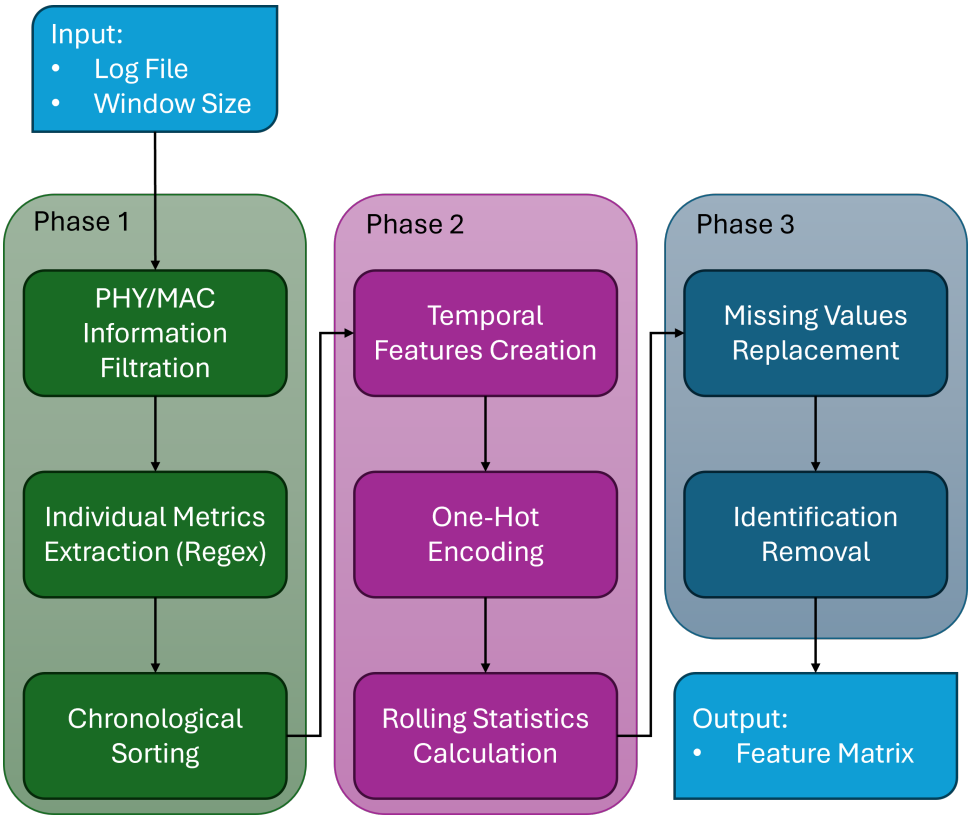


Figure 3.10: Algorithm 2's Pipeline

Algorithm 2 is divided into three phases:

- Phase 1 - Log Parsing: In this first phase, the algorithm goes through all the lines in the raw log file F and filters only the low-layer information (PHY and MAC). Line by line, the timestamp t and the user ID u are extracted for each packet. The individual metrics from every packet, such as HARQ, TB_len, SNR, etc., are then extracted into a list m using regular expressions. Next, a packet object p is created containing the tuple $\{t, u, m\}$, which is then added to the list P_{raw}. Finally, the algorithm sorts the list P_{raw} by timestamp t to ensure that it is in chronological order for the creation of the temporal features in the next step.

- Phase 2 - Feature Engineering: In this second phase, the algorithm creates the temporal features Δt_{prev} and Δt_{next} in order to calculate the inter-arrival time between the current, previous, and next packets. Then, the non-numerical data, such as the packet type or direction (UL/DL), present in the raw packet list P_{raw} is converted into a numeric vector using One-Hot Encoding, allowing for the ML algorithm to process it. "One-Hot Encoding is a process used in ML in order to convert strings into a binary format that the algorithm can understand" [**?**]. With all the data set up, the algorithm gets to the core of the module: the rolling features production. A rolling window S_i of size W is created containing a subset of packets from the list P_{raw}. The algorithm then loops through all the features f and calculates the average μ_f and standard deviation σ_f of this feature for window S_i using the temporal features Δt_{prev} and Δt_{next}. This allows for capturing traffic patterns, such as

the abundance of eMBB traffic and the burstiness of URLLC traffic. Finally, all the temporal, categorical, and rolling features are turned into a single vector x_i for the i-th packet and added to the final matrix X.

- Phase 3 - Clean Up: This final phase is added for the clean-up of the data. Any Not a Number (NaN) values are replaced with 0. The timestamp and user ID columns are then removed to ensure that the model learns general behaviour from the data and not specific times and user ID patterns. Finally, the final feature matrix X is returned to be used in the other modules.

The second module, presented in Algorithm 3 in Appendix B, manages the learning process and returns the final validated model. The training is done over four phases using the training log files F, User Equipment ID sets $\mathcal{U}_{eMBB}$ and $\mathcal{U}_{URLLC}$, and window size W to produce the final bundle $\mathcal{B}$. The algorithm validates that the two UE sets do not overlap, and then calls Algorithm 2 to convert the text into a feature matrix. Algorithm 2 is also used to generate the engineered features for each packet in order to capture the trends throughout the dataset.The model splits the data into a training set and a testing set. The training set is used to fit a LGBM classifier to the features, while the testing set is used to validate the results. The final model bundle $\mathcal{B}$ is returned, as illustrated in Figure 3.11.

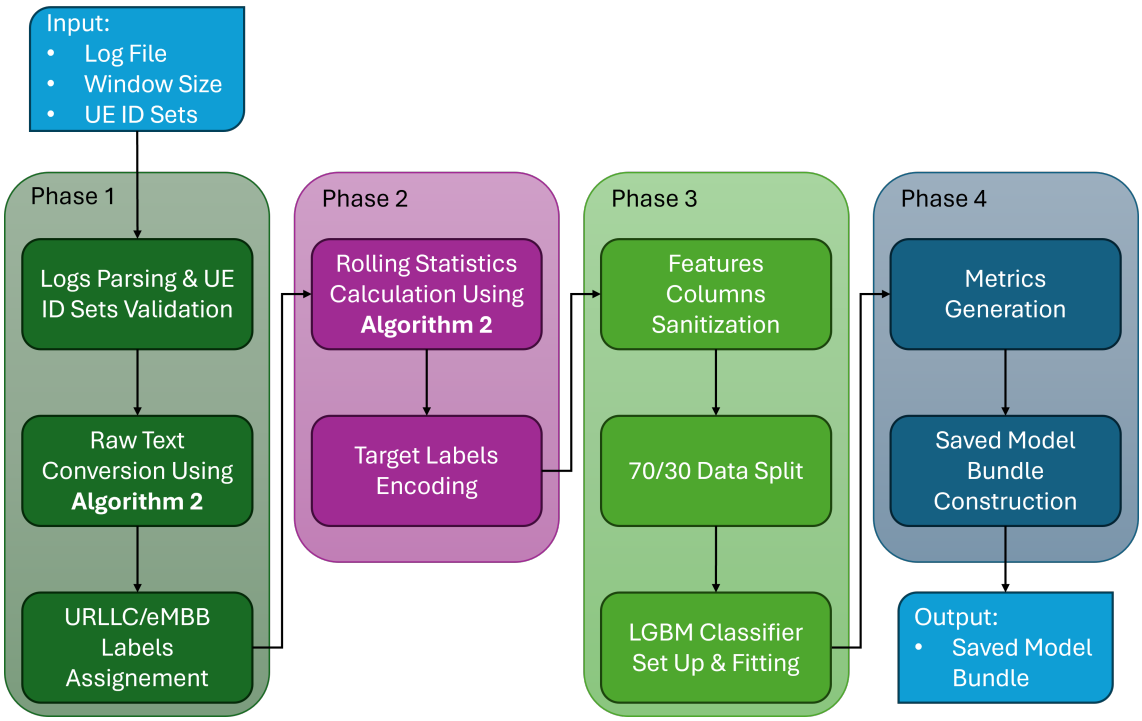


Figure 3.11: Algorithm 3's Pipeline

Algorithm 3 is divided into four phases:

- Phase 1 - Ingestion and Labelling: First, the algorithm goes over the User Equipment ID sets $\mathcal{U}_{eMBB}$ and $\mathcal{U}_{URLLC}$ and ensures that no ID is assigned to both slices. Then an empty master dataset D_{master} is created in order to store the processed packets. The algorithm then iterates through every file f in the log files F and calls Algorithm 2 to convert the raw text from the file f into a feature matrix. It then loops through the packets and labels the packets as URLLC or eMBB following their UE ID. Meanwhile, every packet with $harq = -1$ is filtered out since it is system information and not user data.

- Phase 2 - Feature Engineering and Encoding: In this phase, the algorithm isolates the labels $D_{master}.label$ into a list Y_{raw}, from the feature list D_{master}. The rolling window function is then called from Algorithm 2 for the feature list D_{master} and window size W. The algorithm then converts the text labels "URLLC" and "eMBB" into numerical values $(0, 1)$ in order for the ML algorithm to be able to process them.

- Phase 3 - Alignment and Training: In the third phase, the algorithm sanitises the feature names to ensure that they are compatible with the LGBM library. The feature list X and the label list Y are then each split into two separate lists so that the model trains on 70% of the dataset (X_{train} and Y_{train}), leaving the remaining 30% for the testing (X_{test} and Y_{test}). The LGBM classifier $\mathcal{H}$ is then set up with the parameters defined in Table 3.11. Finally, the model builds decision trees iteratively $\mathcal{H}.fit(X_{train}, Y_{train})$ in order to minimise the objective function based on the training data.

- Phase 4 - Validation and Artefact Generation: In this fourth phase, the algorithm tries to predict the labels of the unlabeled 30% test data. The predicted labels $\hat{Y}$ are then compared to the actual labels Y_{test} in order to calculate the accuracy, precision, and recall. A bundle $\mathcal{B}$ is then created, containing the training model $\mathcal{H}$, the label encoder, the columns order used $X.columns$, and the window size W. The bundle $\mathcal{B}$ is finally saved to be used in Algorithm 4.

Algorithm 4 in Appendix B is composed of five phases in which it classifies new unknown traffic L_{new} using bundle $\mathcal{B}$ from algorithm 3. The algorithm uses Algorithm 2 to parse the data and loads the saved model bundle $\mathcal{B}$ from Algorithm 3 to determine the most likely slice for each packet. The model outputs a list of probabilities P_{probs} for every possible case, and it then chooses the confidence score $\max(P_{probs})$ as the highest value in that list in order to quantify the model's certainty. The model finally provides a log with the timestamps and predictions, as can be seen in Figure 3.12.

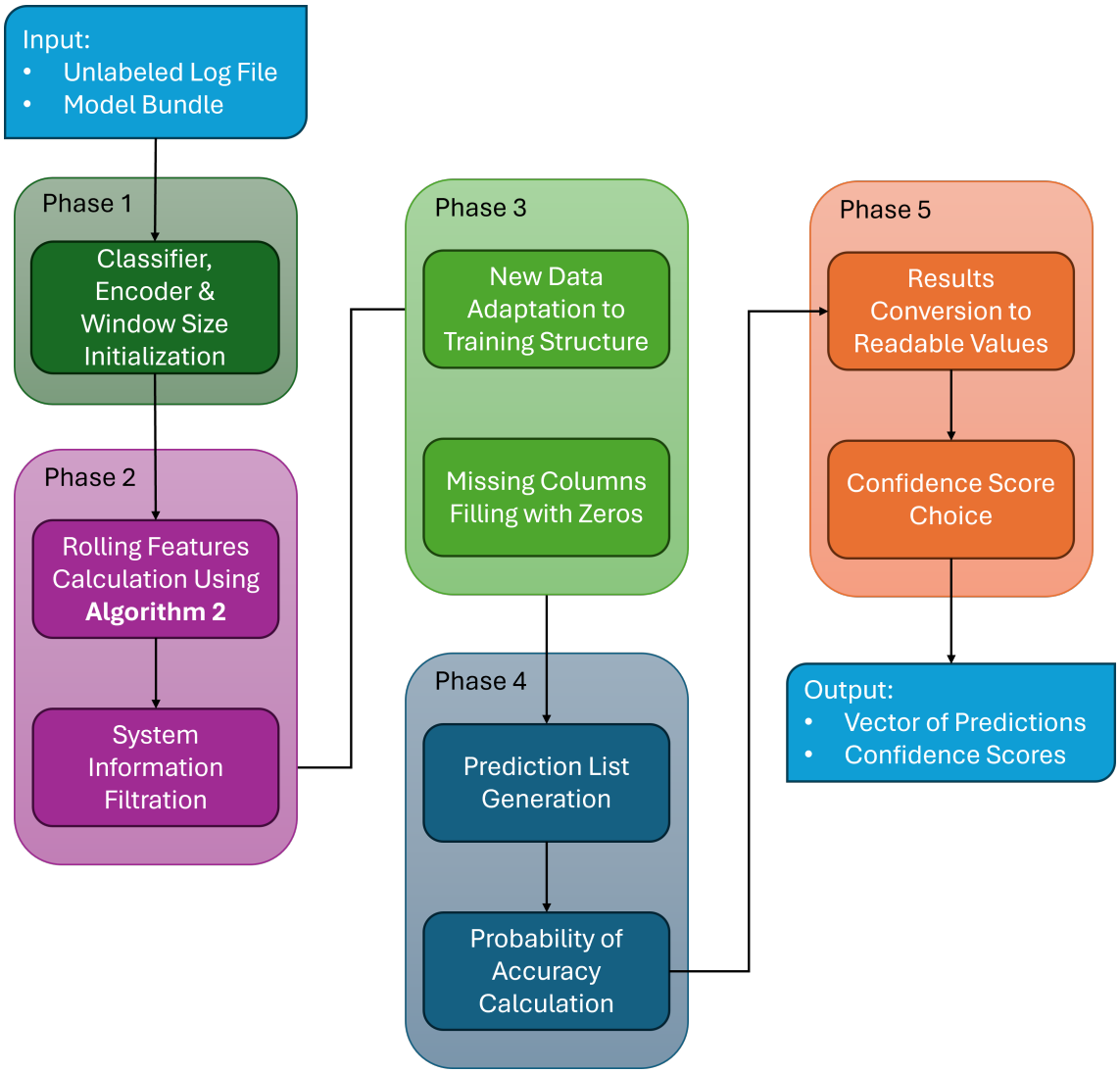


Figure 3.12: Algorithm 4's Pipeline

Algorithm 4 is divided into five phases:

- PhasInitializationsation: In this phase, the algorithm calls the classifier $\mathcal{H}$, the label encoder, the training columns, and the window size W, from the bundle $\mathcal{B}$.
- Phase 2 - Data Preparation: In this second phase, the algorithm runs Algorithm 2 in order to run the new logs L_{new} into a structured list of packets. It then removes the system packets and generates the temporal and rolling features for these new packets so that the model has the needed context to make a decision.
- Phase 3 - Schema Alignment: Phase three is a necessary safety check. The algorithm forces the new data to match the training data structure exactly, since in the real logs, some columns might be missing if an event did not occur. To avoid the error, the algorithm forces the new matrix $X_{aligned}$ to have the same columns, in the same order, as the matrix used in the training. If a column is missing, the algorithm fills it with zeros.

- Phase 4 - Prediction: In the fourth phase, the model processes the matrix $X_{aligned}$ and outputs a list $\hat{y}_{encoded}$ of predicted values 0 or 1. The model then calculates the probability of accuracy P_{probs}.
- Phase 5 - Decoding and Output: In this last phase, the results are converted back into readable information. The algorithm uses the label encoder to turn the numeric predictions back into the labels "URLLC" and "eMBB". The confidence score is then chosen as the highest probability value $\max(P_{probs})$. The algorithm finally prints, for every packet, its timestamp, the predicted slice, and the confidence of the model in its prediction.

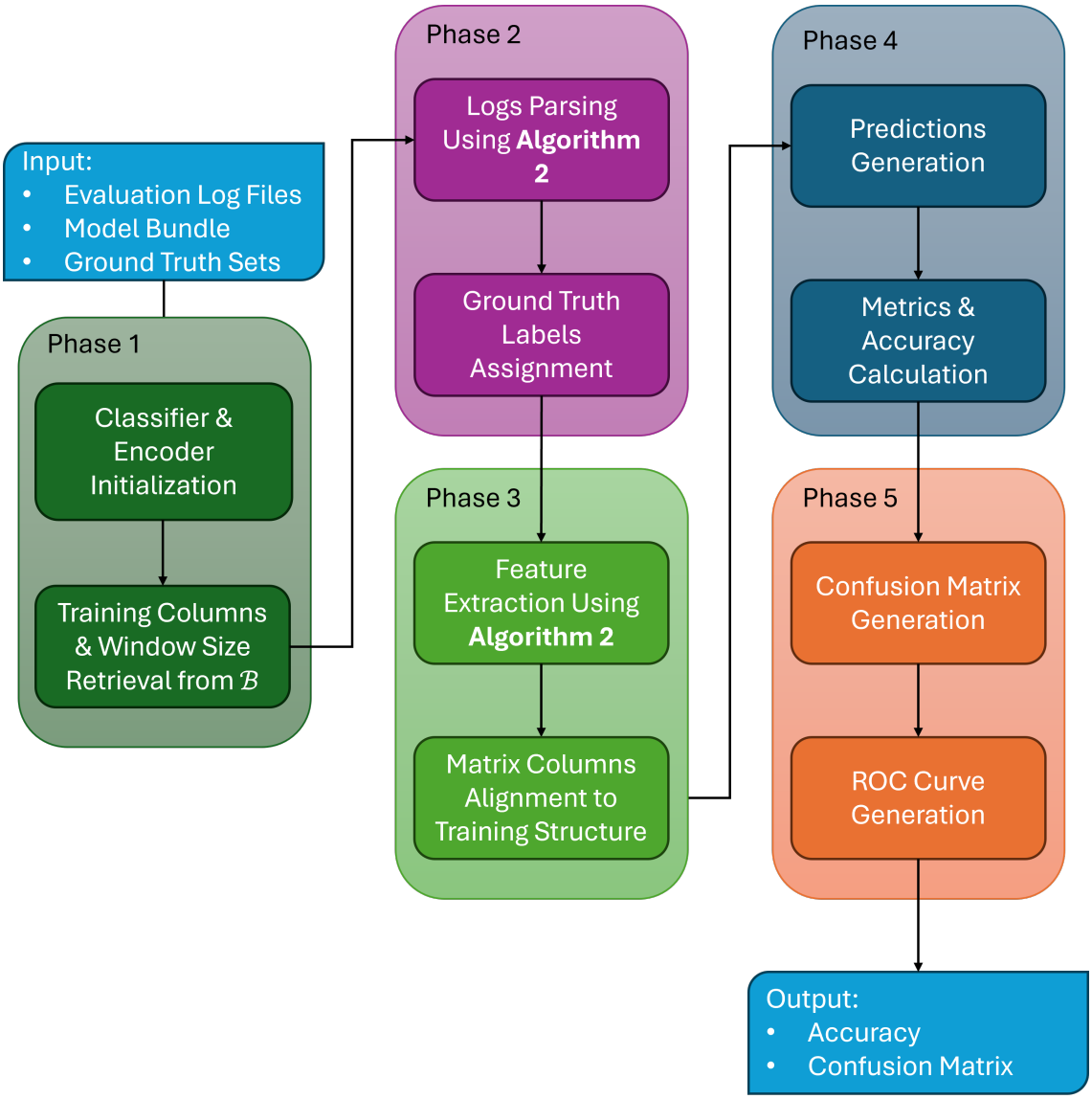


Figure 3.13: Algorithm 5's Pipeline

The final step is the evaluation of the model as can be seen in Algorithm 5 in Appendix B, where the algorithm uses the evaluation log files $\mathcal{L}_{eval}$, saved model bundle $\mathcal{B}$, and User Equipment ID sets $\mathcal{U}_{eMBB}$ and $\mathcal{U}_{URLLC}$ to generate the accuracy score A, and the confusion matrix CM. This module contains five main phases, as can be seen in Figure 3.13. The algorithm starts by retrieving the

information from the saved bundle $\mathcal{B}$, then it calls Algorithm 2 to parse the data and engineer the features. The model then assigns the true labels Y_{true} using the UE ID sets and aligns the result matrix with the training structure. Finally, the algorithm makes its predictions $\hat{Y}_{pred}$ and compares them to the true labels Y_{true} in order to compute the metrics and the accuracy A, and plot the confusion matrix and the ROC curve.

Algorithm 5 is divided into five phases:

- Phase 1 - Initialisation: In this first phase, the algorithm retrieves the trained classifier $\mathcal{H}$, the label encoder, and the column structure $Cols_{train}$ from the saved bundle $\mathcal{B}$, in addition to the rolling window size W.
- Phase 2 - Data Ingestion & Ground Truth Labelling: In phase two, the algorithm creates an empty set D_{eval} to hold the packets for the evaluation phase. Then, for every file f in $\mathcal{L}_{eval}$, the model loops through the log files set aside for testing in order to convert raw text logs into a structured packet list by extracting PHY and MAC layer features using Algorithm 2. For every packet parsed, the algorithm checks the UEID. If it belongs to the eMBB set, it is labelled ”eMBB”; if it belongs to the ”URLLC” set, it is labelled ”URLLC”.
- Phase 3 - Feature Engineering & Alignment: In this section, the algorithm extracts the true labels Y_{true} and converts them into numeric values using the label encode. It then calculates the rolling means and standard deviations for each feature of the evaluation packets X_{eval}. The algorithm then forces the evaluation feature matrix to match the training matrix exactly, filling the missing features with zeros.
- Phase 4 - Prediction & Scoring: The model then makes its predictions $\hat{Y}_{pred}$ on the evaluation features $X_{aligned}$, and then calculates the accuracy score, precision, recall, and F1-score by comparing the model's predictions $\hat{Y}_{pred}$ to the true labels Y_{true}. The model finally generates the confusion matrix CM showing the counts of TP, FP, TN, and FN.
- PhasVisualizationsation: In this final phase, the algorithm finally plots the confusion matrix CM seen in Figure 3.21, and the Receiver Operating Characteristic (ROC) curve seen in Figure 3.22. The algorithm finally uses the ROC curve to calculate the Area Under Curve (AUC) to show its ability to differenciate between URLLC and eMBB packets.

To reduce confusion between small eMBB and URLLC packets, the engineered features were computed on a rolling window of $5ms$. That approach allowed the model to study not only the instantaneous traffic properties but also the temporal patterns over a window of $5ms$. In the following section, the impact of these features on the LGBM model is shown, and the results reached after the testing of the model are explained and analysed.

3.3.3 Results and Interpretation

This section presents the results and performance of the LGBM model on the two generated datasets.

A. Results Using the Rohde & Schwarz Mobile and Network Dataset

The dataset generated using the Rohde & Schwarz Mobile and Network was split into two subsets. The designated subset was used to train the model, and the other to test it.

The results of the test are displayed in the confusion matrix in Figure 3.14.

To use the formulas mentioned in Section 3.1.5, URLLC is considered as "Positive" and eMBB is considered as "Negative". The model gave true URLLC $T_U = 2 \times 10^6$, false URLLC $F_U = 190713$, true eMBB $T_e = 2.8 \times 10^6$, and false eMBB $F_e = 69271$, which leads to an accuracy of $A = 0.9486$.

The metrics of this test were then calculated and can be found in 3.14.

Table 3.14: Classification Report of the LightGBM Traffic Classifier for the R&S Dataset

Class	Precision	Recall	F1-Score
URLLC	0.91	0.97	0.94
eMBB	0.98	0.94	0.96

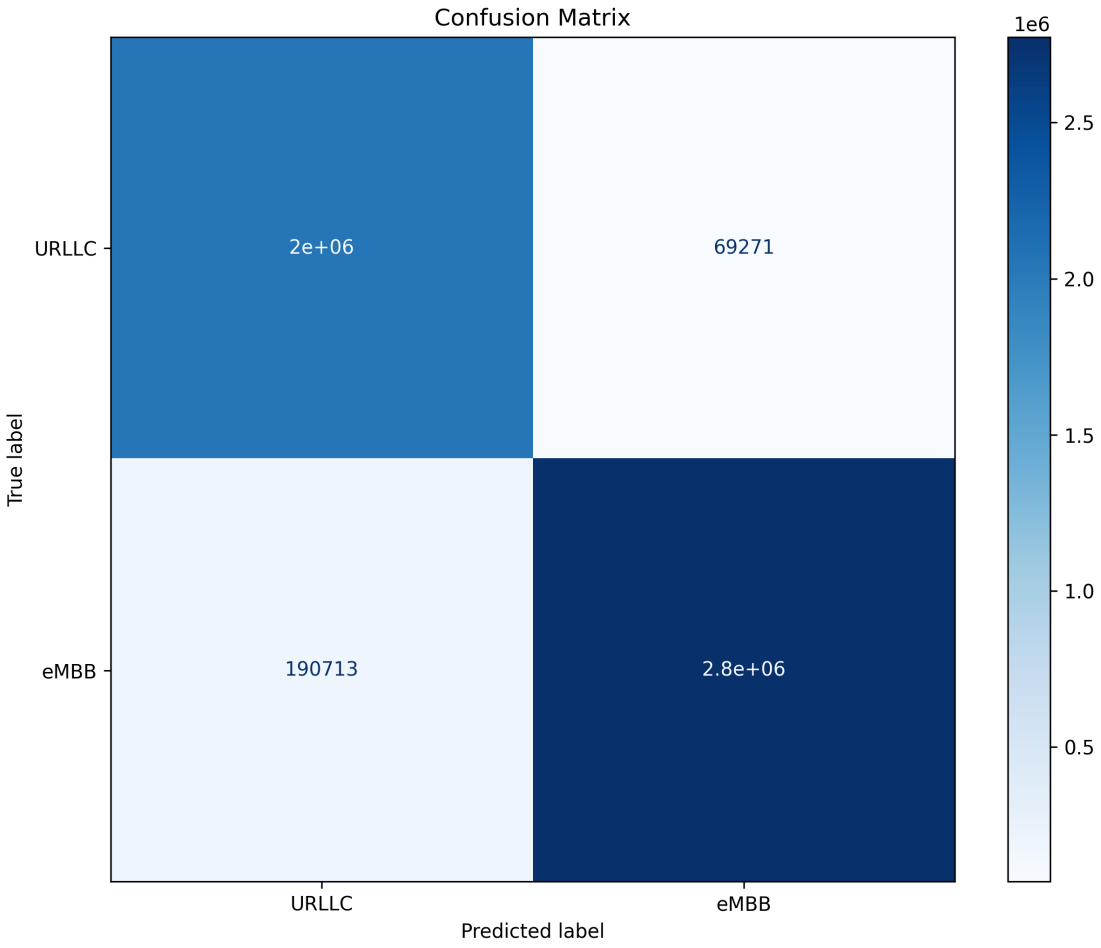


Figure 3.14: Confusion Matrix of the Model for the R&S Dataset

These metrics show that the model is excellent at identifying URLLC packets,

as evident by the high URLLC recall of 97%, but tends to classify around 9% of the eMBB packets as URLLC, as evident by the lower precision of 91%.

As for the eMBB traffic, the precision is very high at 98%, meaning that when the model predicts that a packet belongs to the eMBB traffic, it is right 98% of the time. However, the precision is lower at 94%, which shows that around 6% of eMBB packets are mislabeled by the model as URLLC.

To finally analyse the results of the model, the Receiver Operating Characteristic was plotted for the standard threshold of 0.5 and the Area Under Curve was calculated, as can be seen in Figure 3.15.

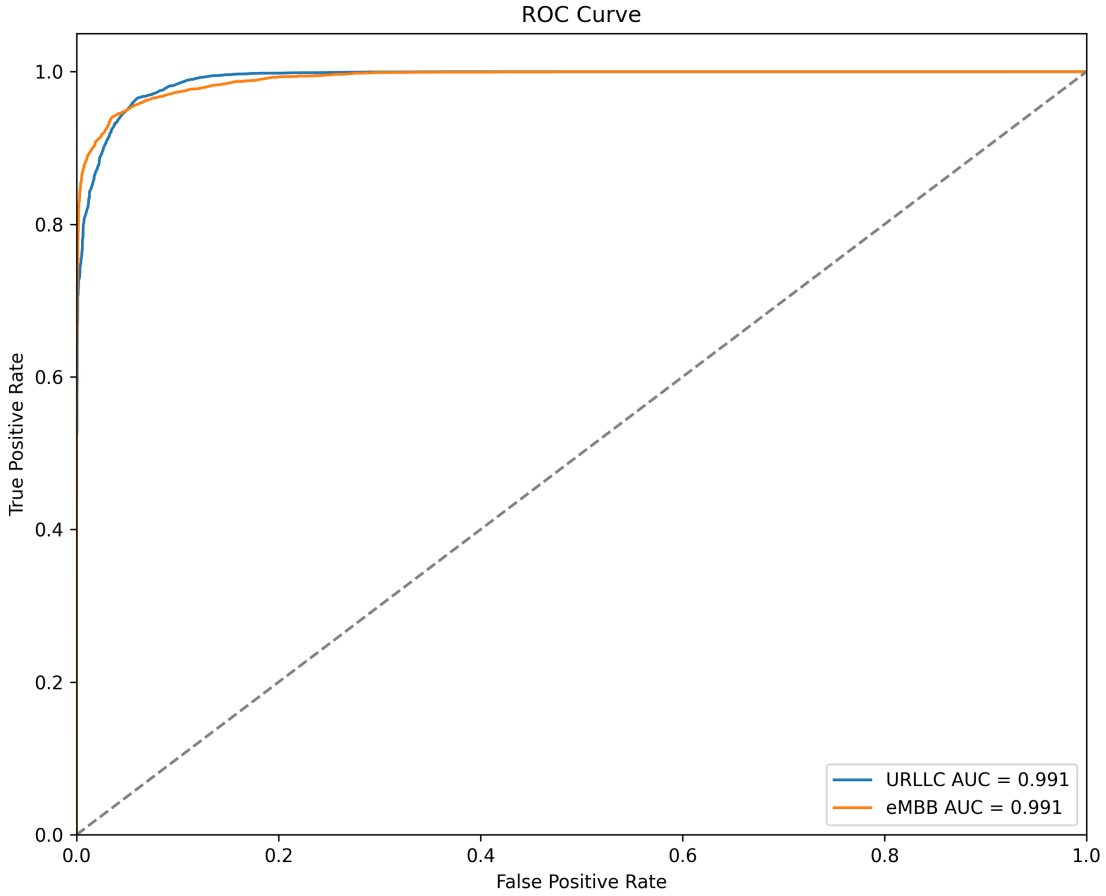


Figure 3.15: ROC Curves of the Model for the R&S Dataset

The URLLC curve is in orange, and the eMBB curve is in blue. The curves both hug the top left corner with an AUC of 0.991, which shows the almost perfect capability of the model to distinguish between the two slices without any bias.

The features that were used the most to classify the traffic can be found in Figure 3.16 by order of importance. It can be seen that the model depended almost exclusively on the Hybrid Automatic Repeat Request (HARQ) to classify the traffic.

This can be justified by the constraints of the Rohde & Schwarz equipment. As the hardware supports only a single terminal, it lacks a multi-user scheduler. Since the network could only handle one user at a time, the HARQ of the single active user became like an identifier of its slice because of the lack of competition. When only a single user is active, there is no noise from other users, meaning

that each traffic type uses a different HARQ unaffected by anything other than the slice type. For the URLLC slice, the importance lies in the low latency, the hardware uses a fast HARQ, expecting an acknowledgement answer as fast as possible, giving it a smaller HARQ. As for the eMBB traffic, the focus is on the data volume and not the speed, which is why the hardware used a standard timing, giving it a higher HARQ. This is explained by the fact that the two features used by the model to determine the slice almost all the time are harq_rolling_std and harq_rolling_mean, the model needed to compare the HARQ of the packet to the ones of the packets in its rolling window in order to place it as small (URLLC) or big (eMBB). This led to the HARQ acting as a slice identifier only applicable in this specific experiment, meaning that the results found in this experiment are not reliable.

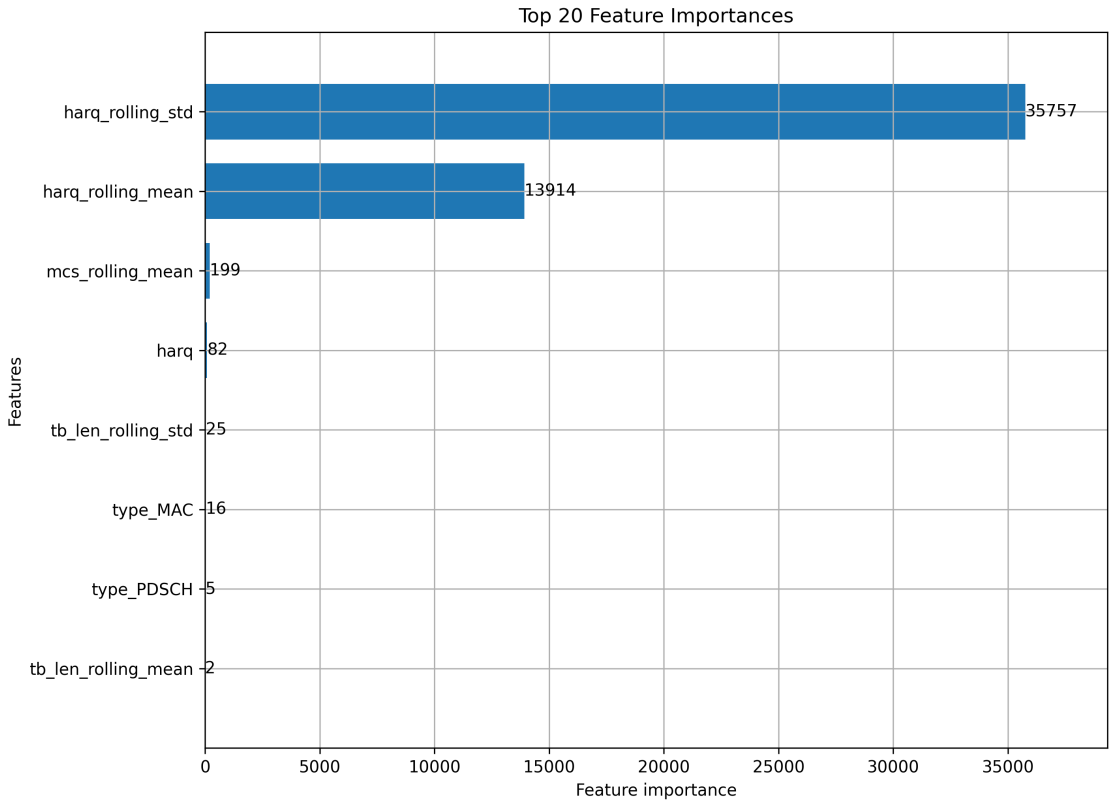


Figure 3.16: Bar Graph of the LightGBM Model Key Features by Importance for the R&S Dataset

The next step was to remove the HARQ related features from the traces and see how the model performs.

B. Results Using the Rohde & Schwarz Mobile and Network Dataset without the HARQ Identifier

This section shows the results of the training and testing of the model on a cut dataset, with all HARQ related features removed.

Figure 3.17 shows the most important features used to classify the traffic now that the model can not depend on the HARQ. Now some of the most prominent

features are tb_len_rolling_std, tb_len_rolling_mean and tb_len, which can be justified by the difference in size between URLLC packets that tend to be small and eMBB packets that have varying sizes, but lean towards the bigger size. Another important feature is the mcs_rolling_mean, which can be justified by the fact that URLLC traffic prioritises reliability, needing a lower MCS to ensure the 10^{-5} loss probability. As for the eMBB traffic, it prioritises throughput, which justifies the need for a high MCS to guarantee high download speeds. Finally, the inter_packet_time_ns and time_to_next_packet_ns features show the tendencies of the traffic. eMBB traffic tends to be continuous with a high frequency of arrival (low and variable inter_packet_time_ns and time_to_next_packet_ns values), while URLLC traffic tends to be periodical with a lower frequency of arrival (high and repetitive inter_packet_time_ns and time_to_next_packet_ns values).

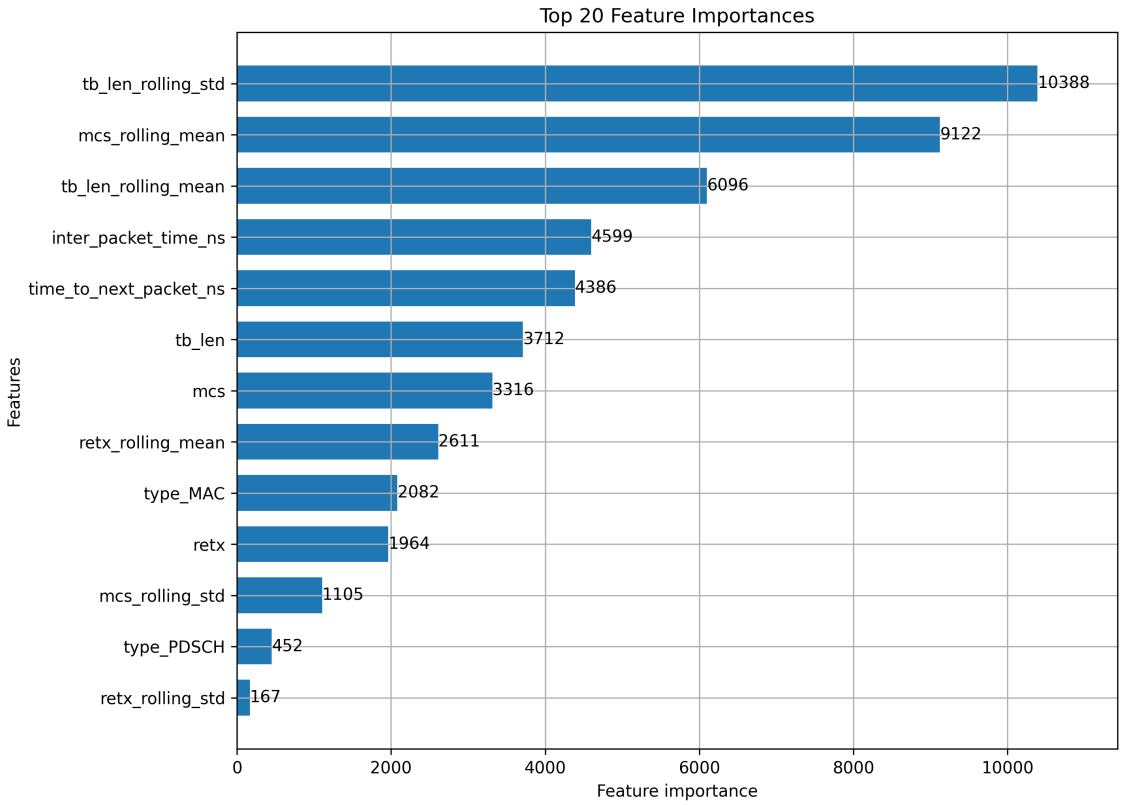


Figure 3.17: Bar Graph of the LightGBM Model Key Features by Importance for the R&S Dataset without the HARQ Identifier

The model used these features in order to classify the traffic between URLLC and eMBB with a less than satisfactory accuracy, as can be seen in the confusion matrix in Figure 3.18.

To use the formulas mentioned in Section 3.1.5, URLLC is considered as ”Positive” and eMBB is considered as ”Negative” as was discussed in Section A.. The model gave true URLLC $T_U = 953405$, false URLLC $F_U = 1.2 \times 10^6$, true eMBB $T_e = 1.8 \times 10^6$, and false eMBB $F_e = 1.2 \times 10^6$, which leads to an accuracy of $A = 0.534$.

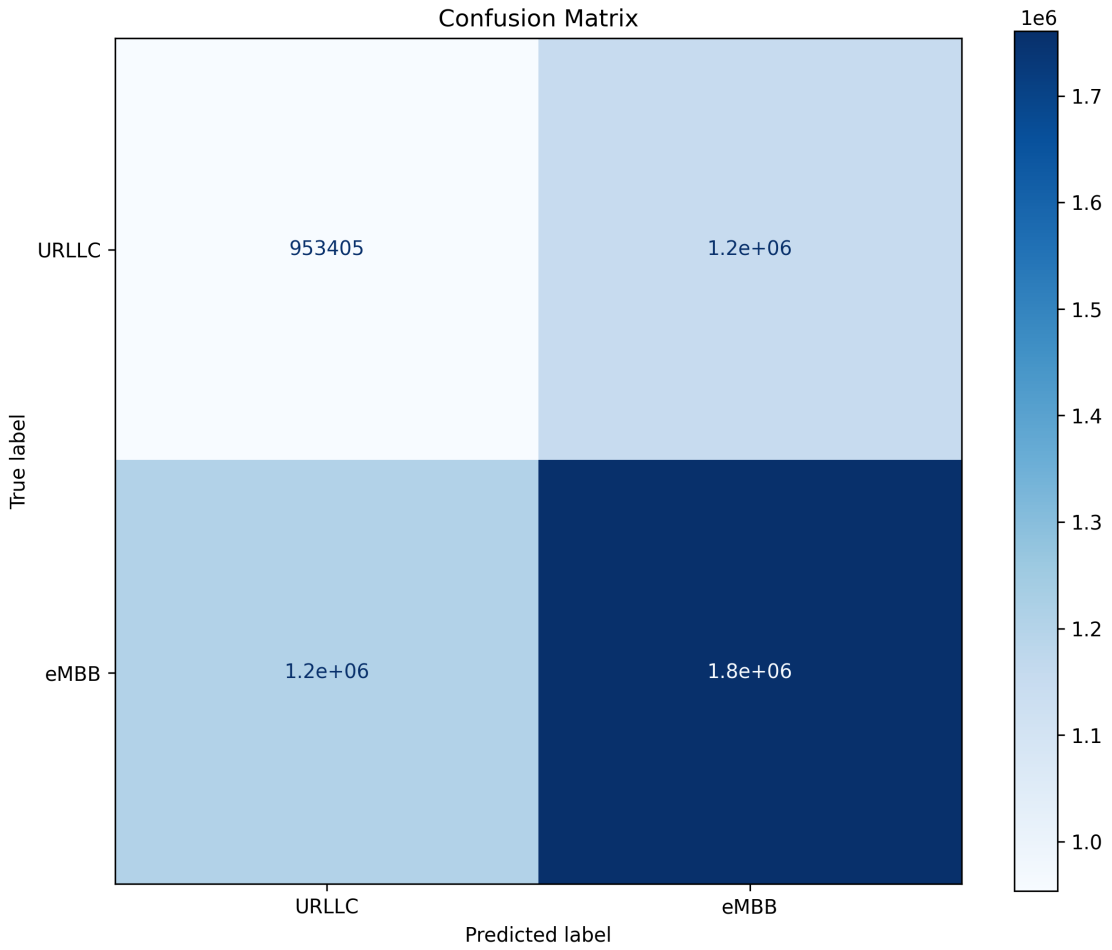


Figure 3.18: Confusion Matrix of the Model for the R&S Dataset without the HARQ Identifier

The precision, recall, and F1-score of both URLLC and eMBB predictions can be found in Table 3.15.

Table 3.15: Classification Report of the LightGBM Traffic Classifier for the R&S Dataset without the HARQ Identifier

Class	Precision	Recall	F1-Score
URLLC	0.44	0.44	0.44
eMBB	0.6	0.6	0.6

These metrics show that the model is less than average at identifying URLLC packets. The low recall of 44% and precision of also 44% show that the model is worse than random guessing for classifying URLLC packets.

As for the eMBB traffic, the precision and recall are slightly higher at 60%, meaning that the model is slightly better than a random guessing classifier.

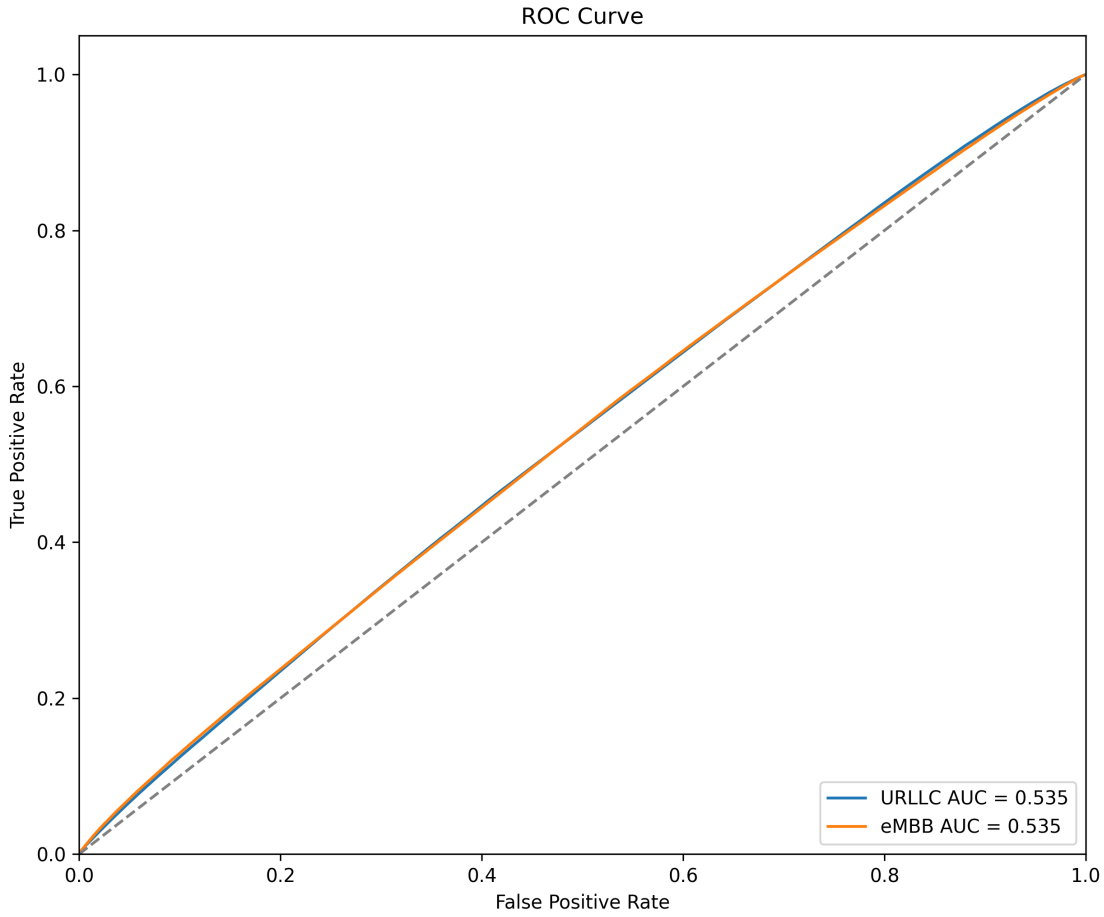


Figure 3.19: ROC Curves of the Model for the R&S Dataset without the HARQ Identifier

The ROC curves were generated in order to evaluate the model better. Both the URLLC curve (blue) and the eMBB (orange) lie between the top right corner and the diagonal line, and both their AUC are 0.535, which means that the prediction is barely better than random guessing, as can be seen in Figure 3.19.

This makes it so that the network can not differentiate between different slices at the PHY and MAC layers, making the distinction between URLLC and eMBB less obvious compared to the dataset used in SectionA..

These results and interpretations led to the need for retraining on a new dataset, in this case, the Amarisoft/Ericsson dataset, in order to produce better accuracy and a model that can better classify the traffic.

C. Results Using then Amarisoft UE Simulator and Ericsson Network Dataset

The model was then trained on the designated dataset from the traffic generated using an Amarisoft UE simulator and an Ericsson network. The model was then tested using the designed dataset, and the features were ranked by how important they were to the classification process, as can be seen in Figure 3.20.

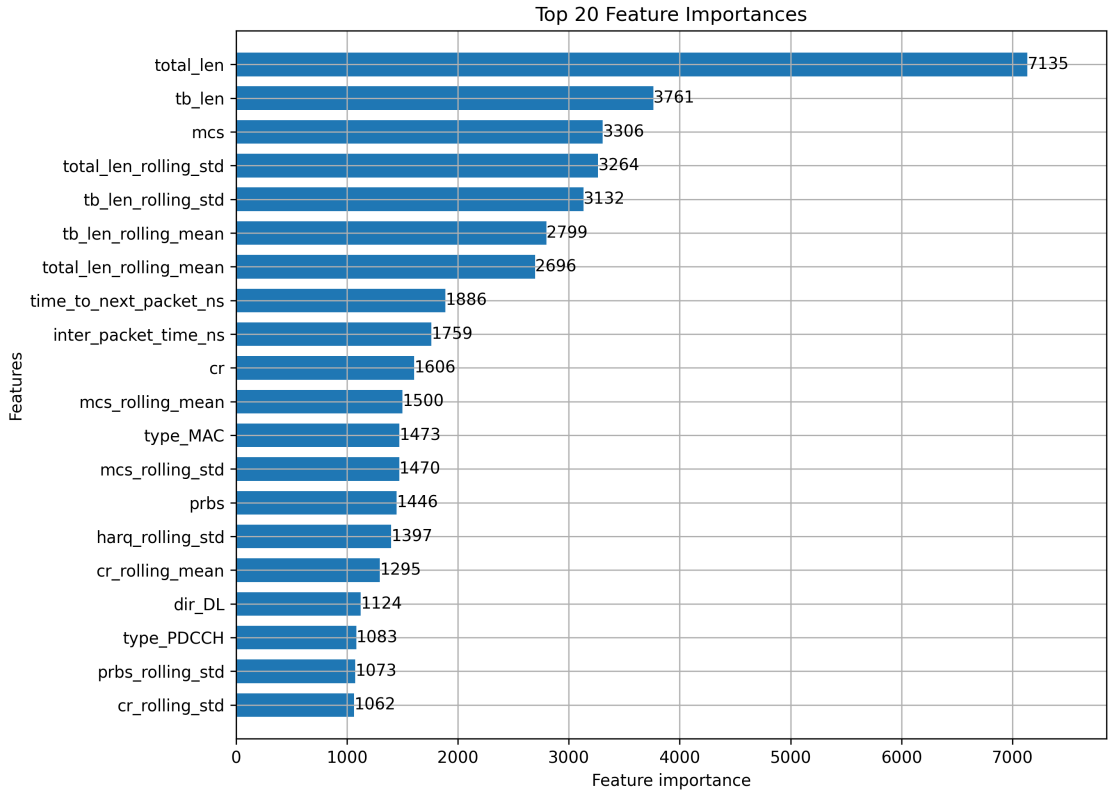


Figure 3.20: Bar Graph of the LightGBM Model Key Features by Importance for the Amarisoft/Ericsson Dataset

The most prominent features are tb_len and total_len. They represent the length of transmitted blocks and MAC blocks, respectively. Since URLLC packets are nearly always small and eMBB packets are mostly large, the size is the first way to distinguish the slice.

The second group of important features is the rolling statistics (tb_len_rolling_std and total_len_rolling_std) since they reveal traffic patterns over time. These indicators show the variability of the feature in the time window, helping detect bursts or stability in the traffic by showing how packet size changes in a sequence of packets. Since a packet's size can overlap between eMBB and URLLC (small eMBB packets, such as a ping), the rolling statistics help differentiate between the two. URLLC traffic being mostly stable would have a rolling standard deviation around zero, as for eMBB traffic and its large packets and variability in size, the standard deviation tends to be high.

Another important feature is harq_rolling_std. It is computed using the IDs of active hybrid HARQ processes in the rolling window. A high variability of this indicator can identify high throughput as multiple HARQ processes are running concurrently. In contrast, a small variability can show very low data rates.

By referring to these features, on top of the other less important features, the LGBM model was able to classify the traffic between URLLC and eMBB with a very high accuracy, as can be seen in the confusion matrix in Fig. 3.21.

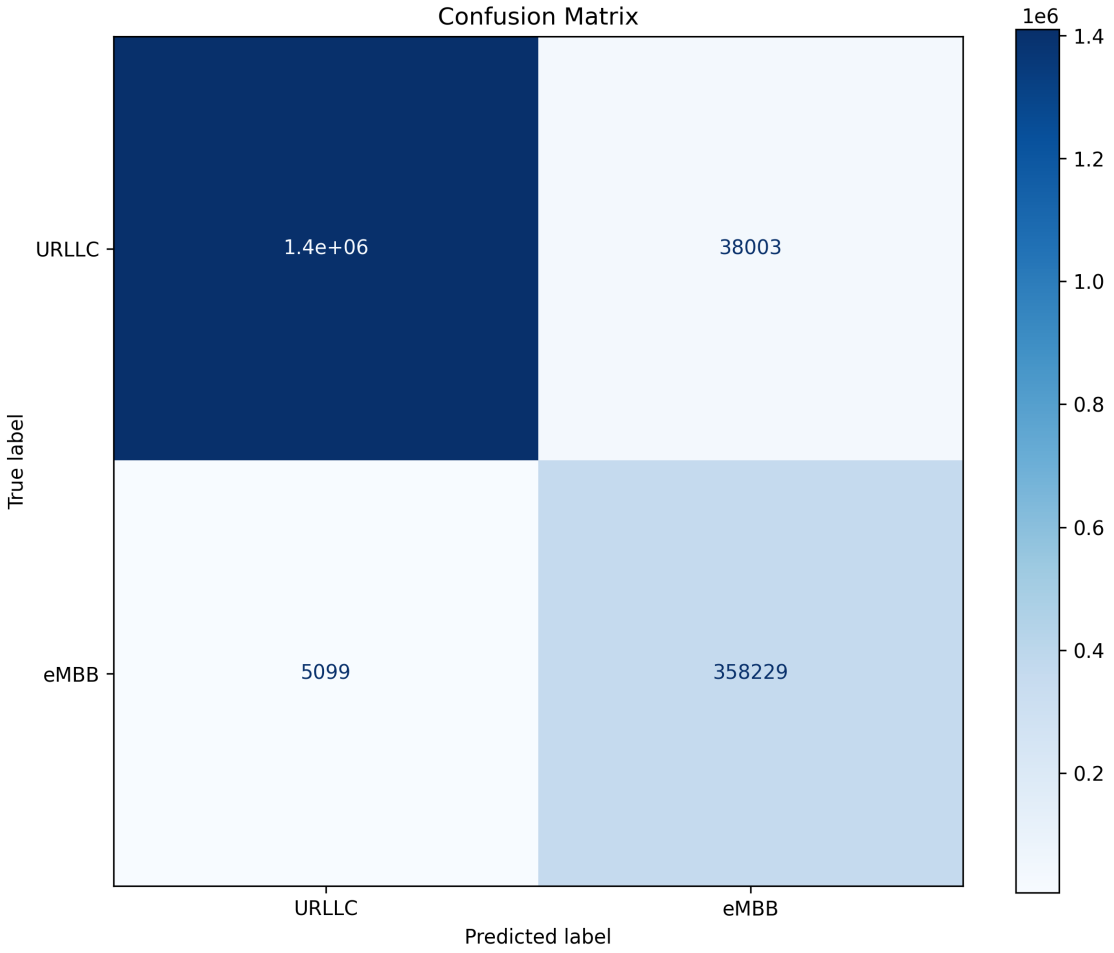


Figure 3.21: Confusion Matrix of the Model for the Amarisoft/Ericsson Dataset

To use the formulas mentioned in Section 3.1.5, URLLC is considered as ”Positive” and eMBB is considered as ”Negative” as was discussed in Section A.. The model gave true URLLC $T_U = 1.4 \times 10^6$, false URLLC $F_U = 5099$, true eMBB $T_e = 358229$, and false eMBB $F_e = 38003$, which leads to an accuracy of $A = 0.976$. An accuracy of 97.6% is higher than the accuracies of the previous works discussed in the state of the art, such as 95% in [**?**] and 89% in [**?**], where an eMBB traffic was classified with access to only the MAC and PHY layers. The same goes for [**?**], where the highest accuracy reached was 87.73% characterises the services in an NSA traffic. As for [**?**], where an accuracy of 100% is reached, it is important to note that the work was done with access to all the layers, especially the KPI features such as the slice type.

Table 3.16: Classification Report of the LightGBM Traffic Classifier

Class	Precision	Recall	F1-Score
URLLC	0.99	0.97	0.98
eMBB	0.90	0.98	0.94

The classification report was grouped in Table 3.16, in which the precision, recall, and F1-score of both slices' predictions are included.

For the URLLC slice's results, the precision is at 0.99, which shows that when the model predicts that a packet is URLLC, it is correct 99% of the time, meaning only 1% of eMBB packets are falsely classified as URLLC. The recall at 0.97 shows that from all the URLLC packets it received, the model correctly labelled 97% of them, only 3% were mislabeled as eMBB. These values led to an F1-score of 0.98, showing that the URLLC traffic was easy to identify due to its stability and small-sized packets.

As for the eMBB slice's results, the precision is at 0.90, meaning that when the model predicts that a packet is eMBB, it is right 90% of the time, 10% of URLLC packets are falsely labelled as eMBB. As for the recall, it is at 0.98, meaning that about 2% of eMBB packets are falsely flagged as URLLC. These values led to an F1-score of 0.94, which is still a high score but lower than the one reached for URLLC traffic. The F1-score reflects the variability of eMBB traffic, given that it can, in some cases, mimic URLLC traffic.

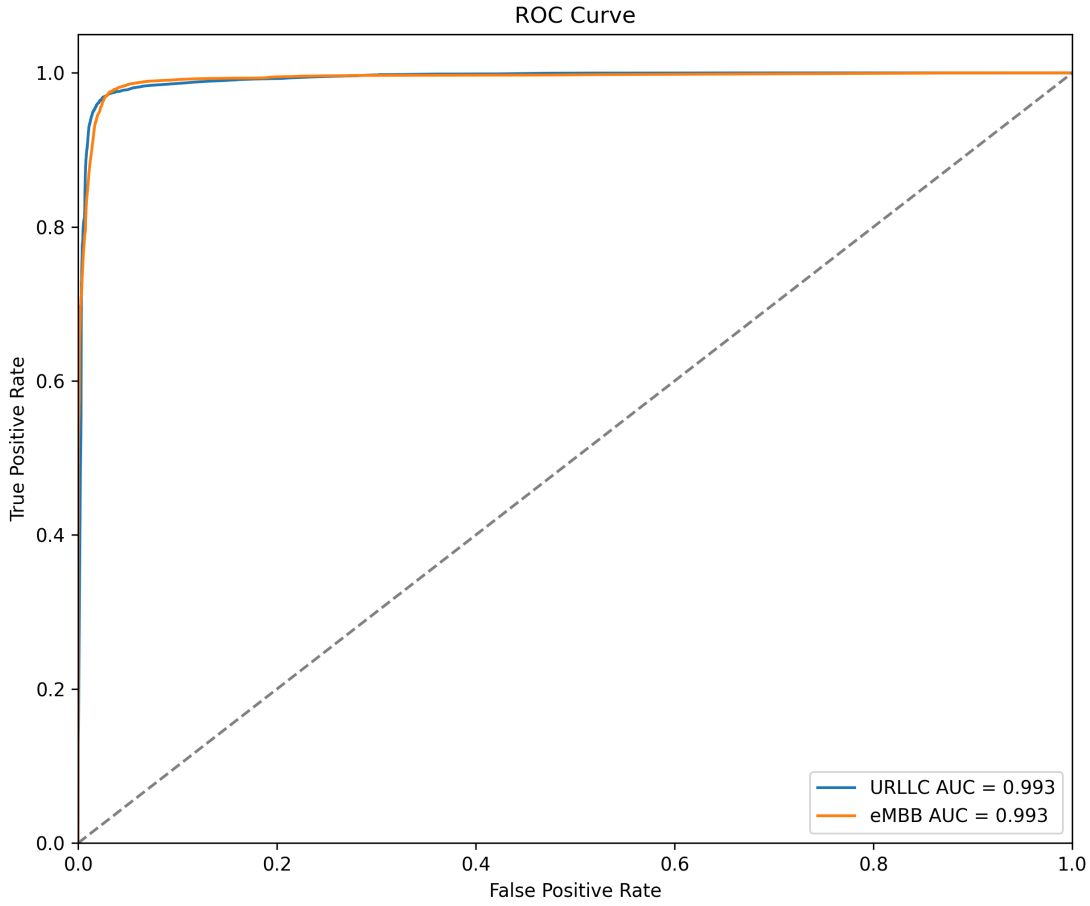


Figure 3.22: ROC Curves of the Model for the Amarisoft/Ericsson Dataset

The threshold of the model was kept as the default 0.5. After the training and the testing, the ROC curves were plotted as can be seen in Figure 3.22. It can be seen that both the URLLC curve (blue) and the eMBB (orange) hug the top left corner, showing that the model has excellent separability between the two slices. The AUC values for both slices, being 0.997, show that the model performs equally well on both slices, showing no bias toward a type of traffic.

These results, combined with the accuracy reached of 97.6%, prove that this

model has a near-perfect classification without overfitting.

3.4 Conclusion

This chapter proposed a LightGBM machine learning model that can distinguish between URLLC and eMBB packets, while having access only to the MAC and PHY layers. Since real-life data was not obtainable, the dataset was generated using realistic radio parameters. The first approach was to generate the data using a Rohde & Schwarz Mobile and Network emulator, which gave a 95% accuracy. By analysing the features used to test the model, it was found that the model used the HARQ as a slice identifier, which rendered the results unreliable. This led to the retaining and testing on a dataset without the HARQ features. Given that the emulator's supposed use is traffic testing and not traffic generation, and it can only handle one user at a time, the dataset produced was not ideal. The model was only able to reach an accuracy of 67.91%. These results revealed the need for another approach to data production. The second attempt was to generate the traffic using an Amarisoft UE Simulator and Ericsson Network, which produced a more realistic dataset. Using this dataset, the LGBM model achieved a 97.6% accuracy, with a higher accuracy for URLLC classification than eMBB, with features such as the size and the rolling statistics being dominant indicators. This model outperforms similar state-of-the-art low-level layers point of view studies and shows that a very high accuracy can be reached without access to high layers.

It is important to mention that this work was done on an experimental dataset. The same code can be used in the future on an operational infrastructure. Moreover, this model only focuses on the two slices URLLC and eMBB. In the future, it could be developed to include more slices, such as mMTC.

The next chapter extends the work of this chapter by integrating a reinforcement learning model that takes the classified traces and returns the optimal number of resource blocks to be assigned to the URLLC traffic to guarantee the chosen probability of loss.

Chapter 4

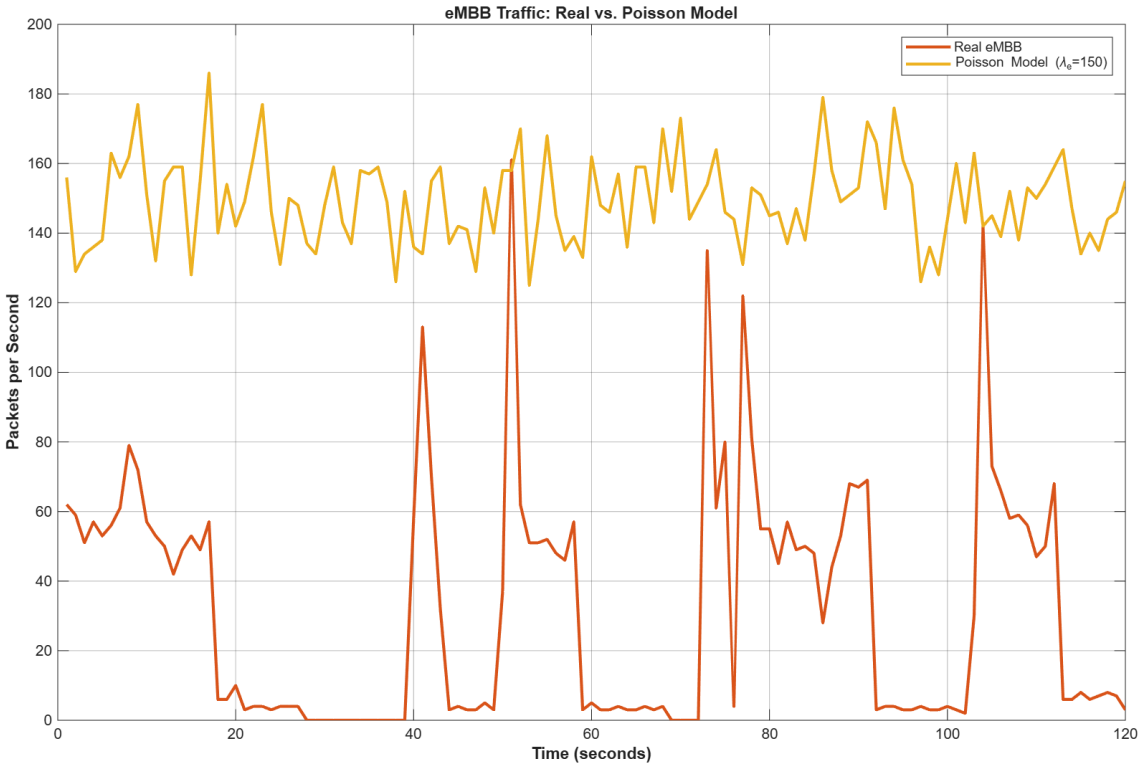
Deep Reinforcement Learning for Resource Allocation Model

4.1 Introduction

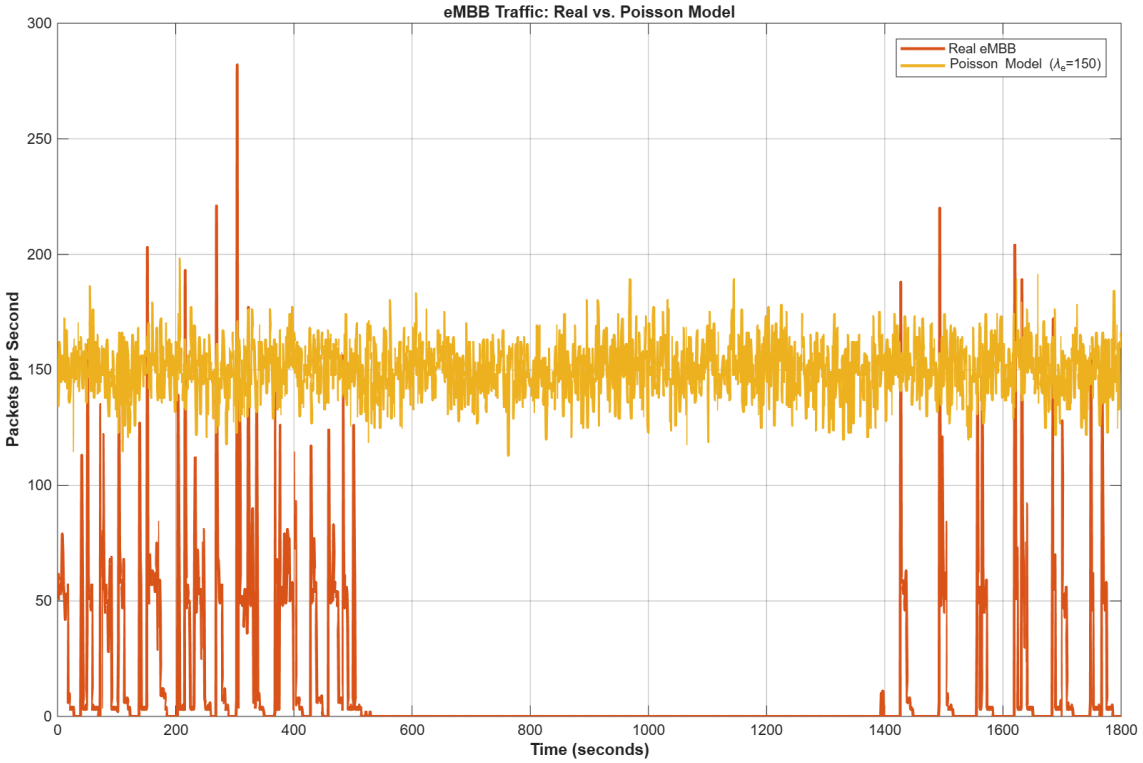
The approach in Chapter 2 is based on queuing theory to determine the optimal number of resource blocks needed to guarantee the chosen URLLC loss probability. That simulation was developed while relying on eMBB and URLLC traffic arrival that follow a Poisson process and an exponential service time. While that assumption allowed the development of a simulation showing the tendencies of the system in theory, it does not reflect real traffic tendencies and needs. Real traffic does not follow Poisson's law, it is more random with occasional bursts of information, as can be seen in Figures 4.1 and 4.2.

This shows a fundamental limitation in analytical models, leading to a need to develop a model that can dynamically allocate resources following real-life traffic.

Moreover, as discussed in Chapter 3, since 5G slicing is not completely deployed, access to real-life traces was unavailable. For this reason, an experimental dataset that mimics real-life traffic was generated and classified between URLLC and eMBB. Entering such a dataset into an analytical model would be very restrictive, given the random nature of the traffic, which does not follow any specific law. This leads to the need for a machine learning resource allocation model that can dynamically adapt the optimal number of resource blocks to reserve for URLLC traffic. Reinforcement Learning (RL), and more specifically Deep Q-Learning, provides a solution for this problem by allowing an agent to observe the system, evaluate the reward and punishment for each action, and step by step develop an optimal policy that minimises the URLLC packet loss while not being too selfish towards eMBB.

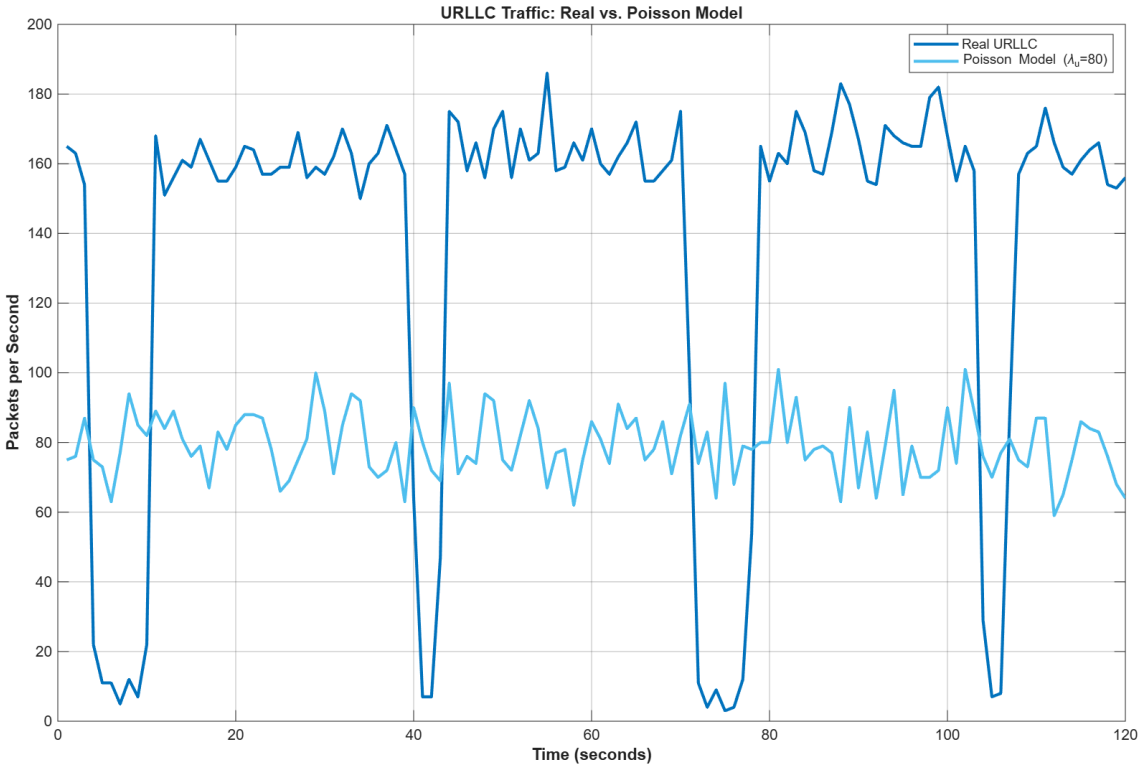


(a) Time window of 2 minutes

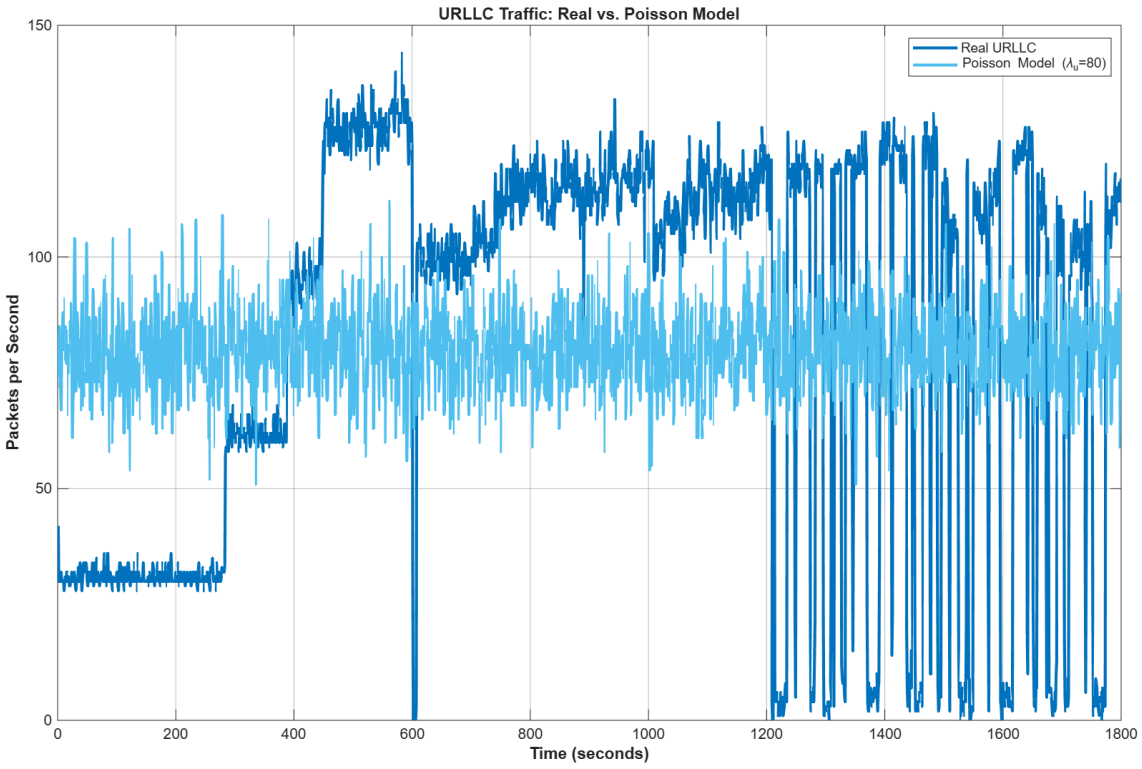


(b) Time window of 30 minutes

Figure 4.1: Comparaison between Poisson traffic and experimentally generated eMBB traffic ($\lambda_e = 150$) for various time windows



(a) Time window of 2 minutes



(b) Time window of 30 minutes

Figure 4.2: Comparaison between Poisson traffic and experimentally generated URLLC traffic ($\lambda_e = 80$) for various time windows

4.1.1 System Notations

Table 4.1 groups the mathematical notations and system parameters used in this chapter.

Table 4.1: Summary of Notations and Parameters used in the Proposed Model

Notation	Description	Unit / Value
S	Total Number of RBs in the System	-
G	Number of Reserved Guard Channels for URLLC	-
λ_e	Arrival Rate of eMBB Requests	packets/s
λ_u	Arrival Rate of URLLC Requests	packets/s
μ_e	Service Rate of eMBB Requests	packets/s
μ_u	Service Rate of URLLC Requests	packets/s
K_e	Number of eMBB Users	-
K_u	Number of URLLC Users	-
n_e	Number of Active eMBB Sessions	-
n_u	Number of Active URLLC Sessions	-
P_{loss}	URLLC Packet Loss Probability	10^{-5}
$\mathcal{S}$	State Space of the DQN Agent	-
s_t	State Vector (n_e, n_u, G) at Time t	-
$\mathcal{A}$	Action Space for Adjusting G	-
a_t	Action at Time t: Increase, Decrease, or Maintain G	-
R	Reward Function Maximizing Utility and Minimizing Loss	-
$r(t)$	Immediate Reward at Time t	-
π	Learning Policy of the RL Agent	-
θ	Online Network Weights	-
θ^-	Target Network Weights	-
η	Learning Rate of the DQN	0.001
E	Total Number of Training Episodes	1000
γ	Discount Factor for the DQN Agent	0.95
$\mathcal{B}$	Experience Replay Buffer Size	10^4
ϕ	Deep Neural Network Parameters (Weights)	-

The rest of this chapter is organised as follows. Section 4.1 gives an overview of the key concepts used in this model, such as Deep Reinforcement Learning (DRL) in Section 4.1.2, the Deep Q-Network (DQN) model in Section 4.1.3 and the metrics used to evaluate it in Section 4.1.4. Section 4.2 discusses the existing state-of-the-art articles on the subject and compares them to the work done in this chapter. Section 4.3 explains the methodology used, such as the training and testing of the DQN model. The simulation and results are presented in Section 4.4. And finally, the work is summarised in Section 4.5 and the conclusions and possible future works are presented.

4.1.2 Deep Reinforcement Learning

As presented in Section 3.1.3 in Chapter 3, Reinforcement Learning (RL) is a type of ML model that uses an agent that progressively learns from the environment by exploring the possible states and actions and maximising the reward given at different steps.

At each step, the agent interacts with the environment based on the current state, action, and reward [**?**]. Since the MDP is the mathematical model for the environment in which the agent operates, the definitions of state, action, and reward correspond to those provided in Section B. of Chapter 2.

- "State (s_t): The state s_t is the representation of the environment at the current moment t.
- Action (a_t): The action a_t is the action chosen by the agent at moment t based on the state s_t of the environment.
- Reward (r_{t+1}): The reward r_{t+1} is the consequence of the action a_t taken by the agent at the moment t. The reward is applied as the agent moves to a state s_{t+1}."

In RL, the environment is typically assumed to be Markovian, indicating that the future state s_{t+1} depends solely on the current state s_t and the action a_t selected by the agent, rather than on any previous states or actions. With each action, the agent receives a reward or penalty. To maximise the cumulative reward, the agent must address the exploration-exploitation dilemma, which requires balancing these two strategies:

- Exploration: The agent chooses new actions in order to develop its knowledge of the system's tendencies.
- Exploitation: The agent chooses to apply the knowledge it already has to the environment in order to choose the optimal next action.

In an MDP environment, the agent's focus is to find an optimal policy π^* that satisfies the Bellman equation (equation 2.5) mentioned in Section C. of Chapter 2. In the static simulation presented in Chapter 2, the Bellman equation was solved using a deterministic search, given the controlled Poisson environment in which the experiment took place. However, in this chapter, given the need for a dynamic allocation and the complexity of the unpredictable experimental data, the environment becomes too complex for a simple deterministic search.

Traditional RL would have been the solution for a smaller dataset, with fewer actions and states, but a complex, heavy dataset calls for the use of Deep Reinforcement Learning (DRL) in order to find the optimal policy π^*.

In contrast to a traditional RL model, which constructs a Q-table of possible state-action pairs, a DRL model employs deep neural networks to approximate the policy and value functions. This method enables the model to identify data patterns, thereby allowing the agent to predict optimal actions for previously unseen states.

4.1.3 Deep Q-Network Algorithm

The Deep Q-Network (DQN) algorithm was first introduced by DeepMind in [**?**] in 2013. It was the first successful algorithm to combine RL with deep neural networks on a large scale.

A. Optimization Using the Bellman Equation

Unlike the gradient-boosting used in LGBM presented in Chapter 3, the DQN model optimises its weights θ by minimising the loss function $\mathcal{L}(\theta)$ derived from the Bellman equation [**?**].

$$\mathcal{L}(\theta) = \mathbb{E}\left[\left(\underbrace{r + \gamma \max_{a'} Q_{\theta^-}(s', a')}_{\text{Target}} - \underbrace{Q_\theta(s, a)}_{\text{Prediction}}\right)^2\right] \tag{4.1}$$

Where s and s' are the current and next state, a and a' are the current and next action, r is the reward, γ is the discount factor, θ and θ^- are the weights of the studied network and a separate target network.

B. Experience Replay

Experience replay is a mechanism of DQN where the experiences of the algorithm from each step are stored in a replay memory [**?**]. This allows the algorithm to learn effectively from a high-dimensional dataset.

According to Mnih et al. [**?**], "the agent's experiences at every step are stored as tuples $e_t = (s_t, a_t, r_t, s_{t+1})$ in a dataset $D = e_1, ..., e_N$ with s_t, a_t and r_t being the state, the action and the reward at time t, and s_{t+1} being the state at time $t + 1$".

Implementing an experience replay has three main advantages over standard Q-learning:

- Greater data efficiency: Each step taken by the agent can be used in multiple weight updates. In standard RL without an experience replay, the agent only learns in the moment, if it finds a highly successful action, it applies a gradient update only once. With experience replay, the successful action is stored in the memory D, where the agent can re-read and apply it in the future. Moreover, standard Q-learning requires many small adjustments to the weights in order to converge to a solution. The introduction of experience replay allows the agent to pair past saved interactions from the buffer with a lower number of new interactions, making it significantly faster in terms of the actions needed for a high-level performance.

- Breaking correlations: In a standard environment interaction, consecutive states tend to be very similar, which can make the agent over-optimise its weights for a specific local area of the environment, leading it to forget how to handle the rest of the environment when it reaches it. The experience replay allows for randomising the samples stored in the replay buffer in order to break these temporal correlations, allowing the agent to better visualise an overall tendency instead of a local one.

- Smoothing the training distribution: In on-policy learning, the agent's current parameters determine its next actions, which can create a biased feedback loop where the training data is limited only to the states the agent currently prefers. This often causes the network to get stuck in local bad policy or suffer from a bad divergence as it oscillates between extremes. Experience

replay solves this by averaging the behaviour distribution over a large history of past states, ensuring the network learns from a stable and diverse set of experiences rather than just its most recent behaviour.

C. Target Network Stability

Target network stability was introduced in another Mnih et al. paper [**?**] published in 2015. In [**?**], the article that introduced DQN in 2013, the same network is used to both predict the current Q-values and the target it was trying to reach. This meant that for every update to the network, the target changed, making the training very unstable by moving the target every time the agent got close. In order to fix this, [**?**] introduced the idea of having two identical neural networks. The first one, the Q-network, can be updated at every training step. The second one, the target network, which has fixed weights, is used to calculate the target values of the loss function. Every few thousand steps, the fixed weights from the target network are replaced with the values of the weights of the Q-network in order to synchronise them. This turned the non-stationary problem into a series of stable learning tasks.

4.1.4 Performance and Stability Metrics

To assess learning progress, performance and stability metrics are employed, including the following:

- Performance Metrics:
  - Cumulative Reward: A metric introduced in RL [**?**]. It is the sum of the rewards received by the agent up until this point in time, showing how well the chosen actions fit the objectives.
  - Success Rate: A metric also introduced with RL [**?**] used to measure the percentage of episodes or trials in which the agent successfully completes a task without violating system constraints.
  - Convergence Rate: A metric that calculates the speed and stability with which an agent converges, indicating that the neural network has found an optimal or near-optimal policy [**?**].
- Stability Metrics:
  - Smooth L1 Loss (Huber Loss): This loss function is used during the optimisation of the neural network to calculate the error between predicted and target values. Unlike the standard mean squared error, it is less sensitive to extreme outliers, which prevents unstable gradient updates when the prediction error is high [**?**]. This robustness helps the model maintain a steadier learning process even in complex environments.
  - Average Predicted Q-Value: A metric used to monitor the convergence of the model, a steady increase in this value suggests that the network is successfully learning to map complex states to future rewards [**?**].

– Exploration-Exploitation Balance: Managed typically through an ϵ-greedy strategy, this ensures the agent sufficiently investigates the state space (exploration) before committing to a consistent, high-performing policy based on its current knowledge (exploitation) [**?**]. It was already explained in Section 4.1.2.

4.2 State of the Art

Traditional resource allocation approaches rely on static analytical models, such as the MDP model presented in Chapter 2. However, the low delay requirements, particularly for URLLC traffic, necessitate more adaptable and dynamic models. Another static approach [**?**] employs a two-dimensional Markov chain to analyse UPF instance requirements based on the number of active PDU sessions. While this method provides a stable solution, it assumes that traffic follows a Poisson distribution and that full core network visibility is available.

The unpredictability of network traffic necessitated the development of more dynamic approaches. Consequently, research has shifted towards Deep Reinforcement Learning, a subset of ML in which an agent interacts with the environment and learns optimal resource allocation policies through experience. In 2018, [**?**] was among the first to argue that 5G slicing is too complex for traditional mathematical approaches, and therefore advocated for the use of an agent-based DRL model. Their approach utilised Deep Q-Learning (DQL) to customise slices for different users by incorporating experience replay and a target network, similar to the algorithm developed in this chapter. In 2019, [**?**] proposed a dynamic scheduling scheme for network slicing. The substantial data generated by 5G mobile devices enabled automated resource allocation to achieve E2E reliability. The DRL agent leverages KPIs found in the layers of the network in order to dynamically adjust the scheduling policy.

The foundational studies by [**?**] and [**?**] established the use of DRL, particularly DQN, for dynamic resource allocation. The primary challenge subsequently became identifying optimal slicing policies for URLLC and eMBB traffic, as these slices often have conflicting requirements and optimizing one can degrade the performance of the other. A notable solution is presented in [**?**], where a Duelling DQN reallocates resources from eMBB to URLLC traffic to satisfy latency requirements. The model in [**?**] addresses the balance among QoE, spectral efficiency, and price satisfaction. This DRL approach achieves high accuracy and converges one hundred times faster than traditional methods.

Beyond service-specific conflicts, the dynamic nature of cellular networks introduces scalability challenges due to fluctuating numbers of active users. To address this, the xSlice model in [**?**] utilises a Graph Convolutional Network (GCN) to model relationships among different traffic sessions. As a result, the agent can make effective slicing decisions at the MAC layer, even as user mobility and network congestion increase. Similarly, [**?**] proposes a Q-learning solution that accounts for both communication and computational resources, emphasising the long-term impact of decisions over a defined time horizon to optimise network utility for diverse requirements.

However, although the solution in [**?**] enables decision-making at the MAC

layer, it still relies on access to all network layers and their respective KPIs, similar to the other approaches discussed. This dependency necessitates a level of network visibility that is typically unavailable to industrial clients, as noted in Chapter 1.

This chapter addresses the visibility gap by employing the ML classifier developed in Chapter 3. Unlike existing literature, which assumes that the network identifies and labels traffic for the agent, the proposed approach infers the necessary environment state directly from low-layer traces (PHY and MAC). By bridging this gap, the DQN agent developed in this chapter can maintain a reliable and autonomous resource allocation policy for industrial clients, such as EDF, even when access to high-level layers is restricted.

The articles mentioned were classified in the table 4.2.

Table 4.2: Summary Comparison of Representative Works on 5G Resource Allocation

Ref.	**Focus**	**Approach**	**Model**	**Layer**	**Goal**
[**?**]	UPF Scaling	Threshold-based scaling and admission control	Markov Chain	All layers	Utilization
[**?**]	Slice Mgmt	Resource orchestration using experience replay	DQL	All layers	Customization
[**?**]	E2E Slicing	Data-driven dynamic scheduling	DRL	All layers	Reliability
[**?**]	URLLC eMBB	Resource perforation and overlay	Dueling DQN	Radio Layers	Latency
[**?**]	Dynamic Traffic	Online slicing, multi-objective metrics	Sigmoid DQN	All layers	QoE/Price
[**?**]	QoS Opt.	Graph embedding for session management	GCN	MAC layer	Perf. Regret
[**?**]	Joint Alloc.	Non-myopic joint comm. and computing	Q-learning	All Layers	Total Utility
This Work	**Slicing**	**Dynamic Guard Channel (G) reservation**	**DQN**	**PHY/MAC**	10^{-5} **Loss**

4.3 System Model

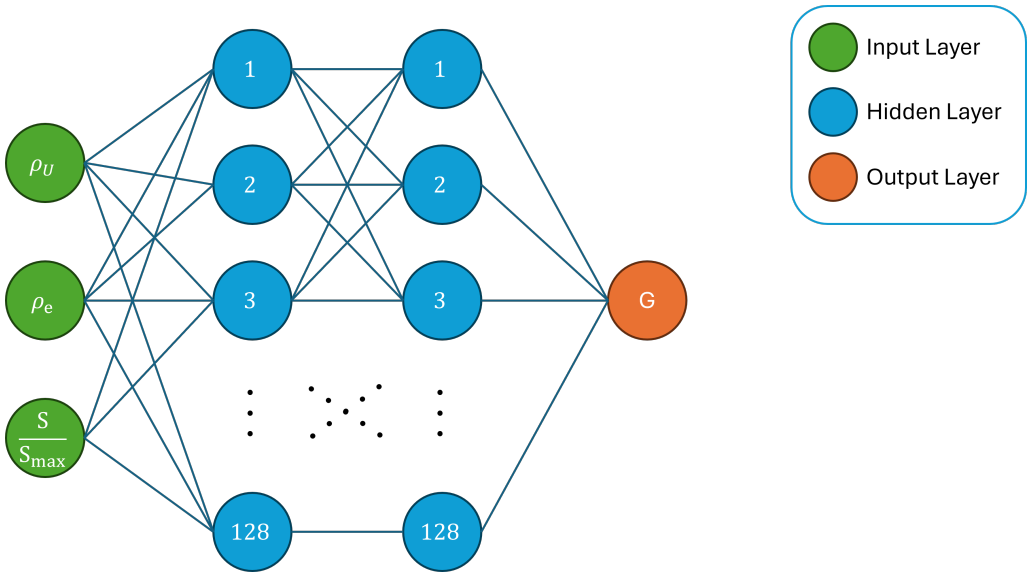


Figure 4.3: Architecture of the DQN used for dynamic guard channel allocation

The environment of this model is similar to the system model presented in Chapter 2, Section 2.3. It is made up of a 5G network cell processing a heterogeneous traffic (URLLC and eMBB) with a fixed number of total available Physical Resource Blocks S. The agent's state space, action space, and reward function can be seen in Figure 4.3 and are as follows:

- State Space: During the training phase, at every episode, the model chooses a new value for the total number of RBs S. This dynamic change allows the algorithm not to overfit the model to a specific network size. At every decision step, the agent observes three dimensions that form its state space:
  - URLLC Load ($\rho_U = \frac{\lambda_U}{\mu_U}$): The current traffic demand for the URLLC traffic
  - eMBB Load ($\rho_e = \frac{\lambda_e}{\mu_e}$): The current traffic demand for the eMBB traffic
  - Normalized Capacity ($\frac{S}{S_{max}}$): The current total capacity normalized against the maximum
- Action Space: Given that the total number of RBs varies with every step, the action space is a relative percentage of the total available resources instead of a discrete number of RBs. The agent has the choice between 21 actions, starting from 0% to 100% with increments of 5%, allowing for the model to converge without having to explore a massively large space. The guard channel value is then calculated as $G = round(action_{fraction} \times S)$.
- Reward Function: The agent has two goals, first and most important to guarantee a URLLC loss probability below the Service Level Agreement (SLA) threshold of 10^{-5}, second, to prevent the eMBB buffer from exploding. The environment uses a penalty and a reward to evaluate the agent's actions:

– SLA Violation Penalty: If the action chosen by the agent results in a URLLC loss exceeding the threshold of 10^{-5}, the agent receives a logarithmically negative reward:

$$r_t^- = -1 - \log_{10}\left(1 + \frac{p_{loss}}{\epsilon}\right) \tag{4.2}$$

With r_t being the reward received at moment t, p_{loss} the current calculated packet loss probability for URLLC traffic, and ϵ the strict maximum URLLC packet loss threshold as can be seen in Figure 4.4.

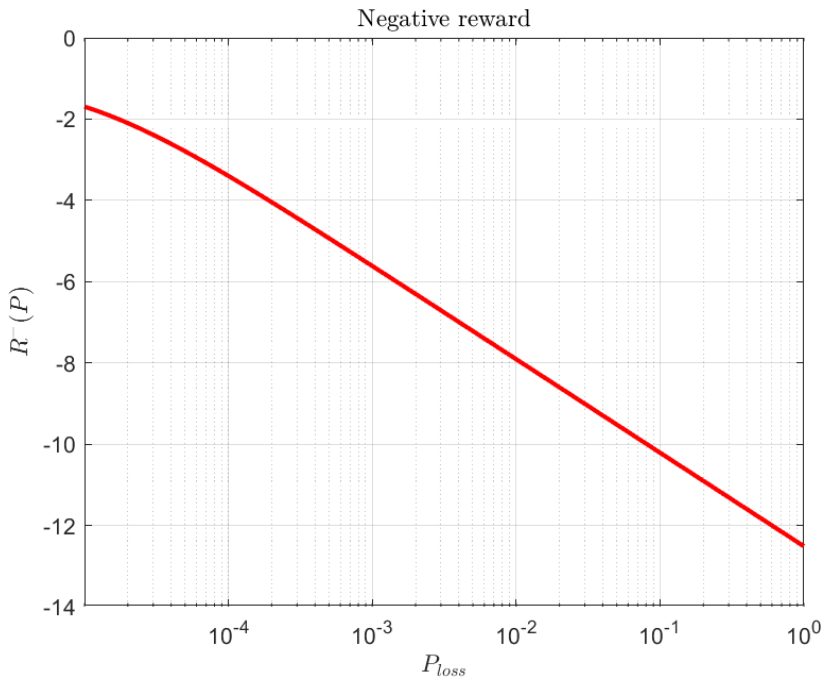


Figure 4.4: Negative Section of the Reward Function

– Efficiency Reward: If the URLLC packet loss probability threshold is respected, the environment looks for the lowest number of guard channels needed G_{best} as can be seen in Figure 4.5. It then rewards the agent based on how close its number of guard channels G_{chosen} is to G_{best}:

$$r_t^+ = \frac{1}{1 + 10 \times \left(\frac{|G_{chosen} - G_{best}|}{S}\right)} \tag{4.3}$$

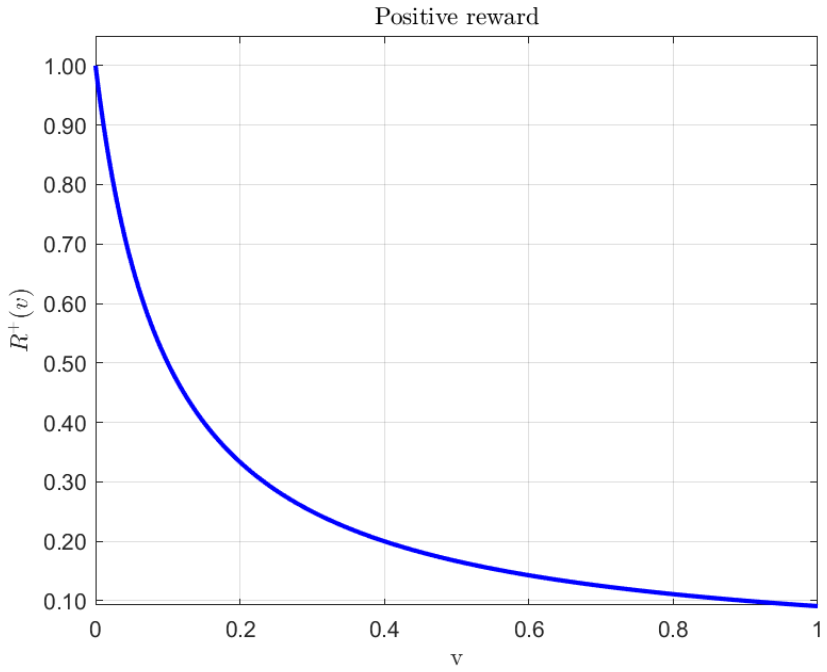


Figure 4.5: Positive Section of the Reward Function

The reward function of the model is then:

$$r_t = \begin{cases} \frac{1}{1+10\times\left(\frac{|G_{chosen}-G_{best}|}{S}\right)} & \text{if } p_{loss} \leq \epsilon \\ -1 - \log_{10}\left(1+\frac{p_{loss}}{\epsilon}\right) & \text{if } p_{loss} > \epsilon \end{cases} \tag{4.4}$$

4.3.1 DQN Model Algorithm

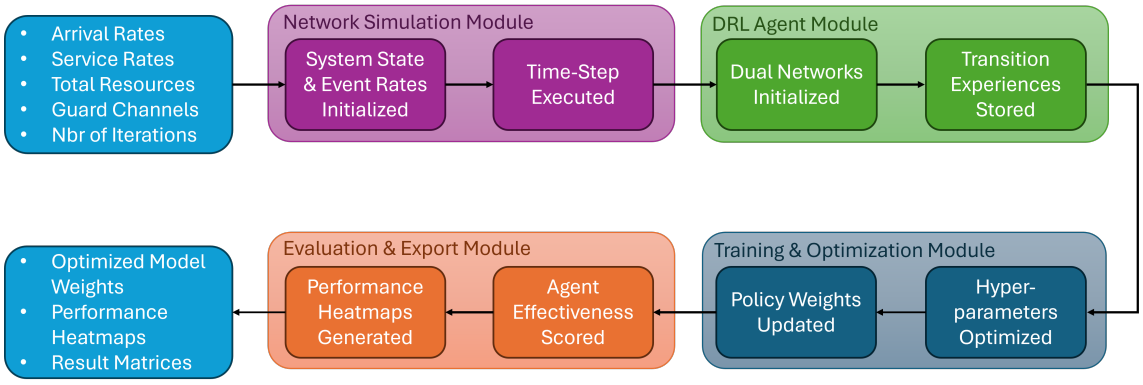


Figure 4.6: DQN Model's pipeline

The DQN model used to calculate the optimal number of resources needed for the URLLC traffic is composed of four different modules, as can be seen in Figure 4.6. The full code for the model can be found on the public GitHub [**?**].

The first module is shown in Algorithm 6 in Appendix C, which uses an adapted version of Algorithm 1 presented in Section 2.4.2 from Chapter 2, as can be seen in Figure 4.7.

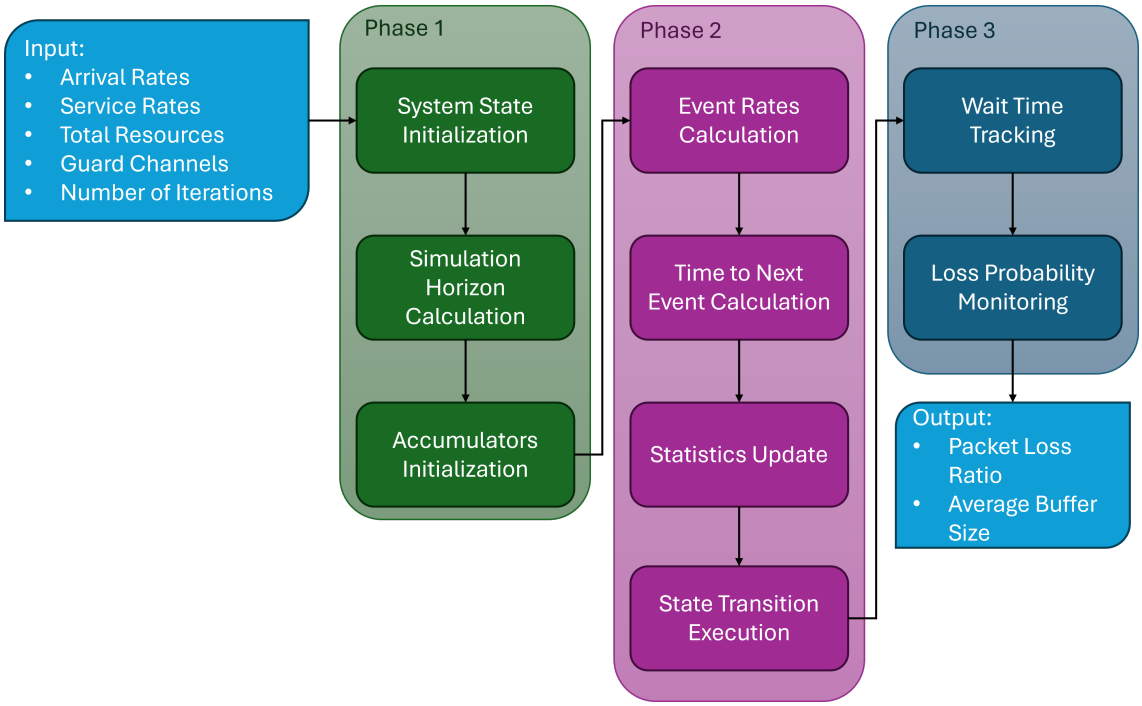


Figure 4.7: Algorithm 6's pipeline

This module uses a Next-Event simulation to generate the static resource allocation needs of the system. The simulation models the system as a Markov Process where the next step depends only on the current number of URLLC packets in service (x_1), eMBB packets in service (x_2), and eMBB packets in the buffer (x_3). Algorithm 6 requires the URLLC and eMBB arrival rates (λ_U,λ_e), URLLC and eMBB service rates (μ_U,μ_e), total number of RBs in the system (S), URLLC guard channels (G), and the number of iterations N_{iter}. It then returns the URLLC packet loss ratio (L), and the average eMBB queue size (W_{avg}) as can be seen in Figure 4.7.

Algorithm 6 is divided into three phases:

- Phase 1 - Initialisation: In the first phase, the algorithm initialises an empty system state $e = x_1 = 0, x_2 = 0, x_3 = 0$, where x_1 is the number of active URLLC packets, x_2 the number of active eMBB packets, and x_3 the number of eMBB packets in the buffer. The model then sets the simulation's horizon H based on the chosen number of iterations N_{iter}, and the sum of the arrival rates $\lambda_U + \lambda_e$. The statistical accumulators, such as the total time T_{total}, the buffer occupancy $Wait_{sum}$, and the time spent in a state where URLLC packets are lost $Cumul_{loss}$, are initialised to zero.

- Phase 2 - Simulation Loop: In this second phase, the model runs the simulation until the total time of the experiment T_{total} reaches the horizon H chosen in the first phase. First, at each step the algorithm calculates the total rate $R_{total} = R_{u_serv} + R_{e_serv} + R_{u_arr} + R_{e_arr}$ where $R_{u_serv} = \mu_U x_1$ is the URLLC service rate, $R_{e_serv} = \mu_e x_2$ is the eMBB service rate, $R_{u_arr} = \lambda_U$ is the URLLC arrival rate, and $R_{e_arr} = \lambda_e$ is the eMBB arrival rate. Next, the model samples a random time Δt to the next event following an exponential distribution, which is added to the total time spent in the simulation

T_{total}. During this interval Δt, the algorithm updates the performance metrics $Wait_{sum}$ and $Cumul_{loss}$. Finally, an event i is chosen with a probability $P_i = \frac{R_i}{R_{total}}$ in order to implement the state transitions presented in the infinitesimal generator matrix presented in Chapter 2, Section 2.3, Equation 2.8.

- Phase 3 - Performance Metrics Extraction: Once the chosen horizon H is reached, the algorithm calculates and returns the packet loss ratio $L = \frac{Cumul_{loss}}{H}$ and the average eMBB queue size $W_{avg} = \frac{Wait_{sum}}{H}$ in order to quantify the performance of the chosen number of guard channels G.

The second module of this model is shown in Algorithm 7 in Appendix C, which details the environment that serves as the interface between the reinforcement learning agent and the network simulation, as can be seen in Figure 4.8.

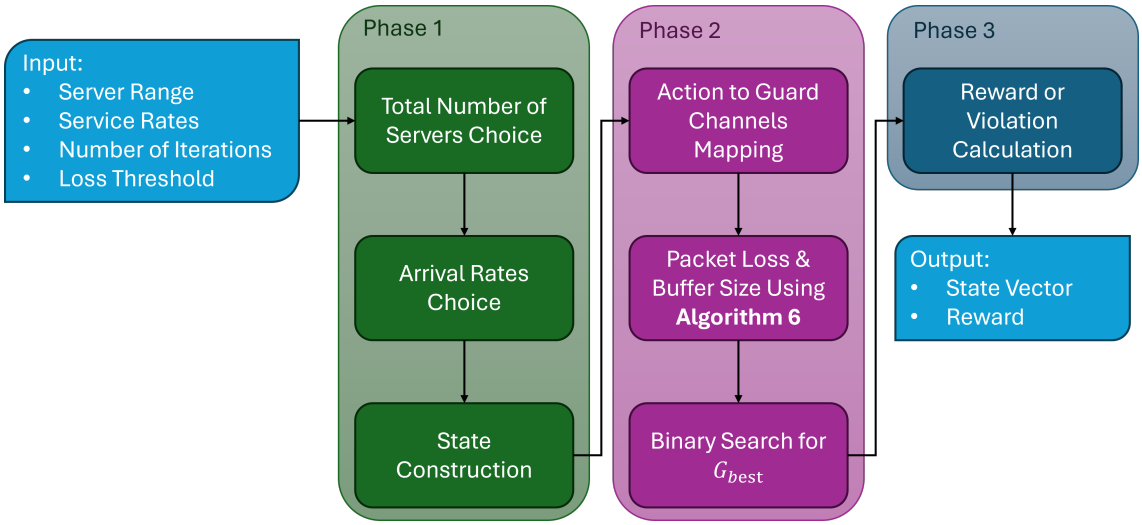


Figure 4.8: Algorithm 7's pipeline

This module manages an environment that maps the agent's actions to a specific number of guard channels (G) and calculates its URLLC packet loss ratio (L_G) and eMBB buffer size (Q_G). It then compares the URLLC packet loss ratio (L_G) to the chosen loss threshold (ϵ) in order to calculate the reward (r) to be granted. Algorithm 7 uses the server range $[S_{min}, S_{max}]$, the URLLC service rate μ_U, the eMBB service rate μ_e, the number of iterations N_{iter}, and the loss threshold ϵ in order to generate the state vectore s and the reward r as can be seen in Figure 4.8.

Algorithm 7 is divided into three phases:

- Phase 1 - Environment Reset: In the first phase, the algorithm resets the environment by choosing a random number of RBs S, and random arrival rates λ_u and λ_e. It then uses the total number of RBs S, the arrival rates λ_u and λ_e and the maximum system capacity S_{max}, in order to generate a state vector s.

- Phase 2 - Step Execution and Simulation: In this second phase, the model receives the action a from the agent and uses it to map the corresponding

number of reserved resources G. The algorithm then calls Algorithm 6 to retrieve the URLLC packet loss ratio L_G and eMBB buffer size Q_G for the given scenario. The model then performs a binary search for the minimum number of RBs to reserve G_{best} while respecting the probability of loss chosen for the current scenario.

- Phase 3 - Reward Calculation: Finally, the model compares the URLLC packet loss ratio L_G to the chosen loss threshold ϵ. If the scenario fails to meet the required loss threshold ϵ, a penality proportional to the URLLC packet loss ratio L_G of $r = -1 - log_{10}\left(1 + \frac{L_G}{\epsilon}\right)$ is applied. On the other hand, if the required loss threshold ϵ is met, a reward based on the proximity of G to G_{best} is granted $r = \frac{1}{1+10\Delta_{norm}}$ with $\Delta_{norm} = \frac{|G-G_{best}|}{S}$ being the normalized distance between G and G_{best}. The model finally returns the state of the system s, and the reward r.

The third module is presented in Algorithm 8 in Appendix C, which details the implementation and training of a Deep Q-Network in order to solve the dynamic resource allocation problem, as can be seen in Figure 4.9.

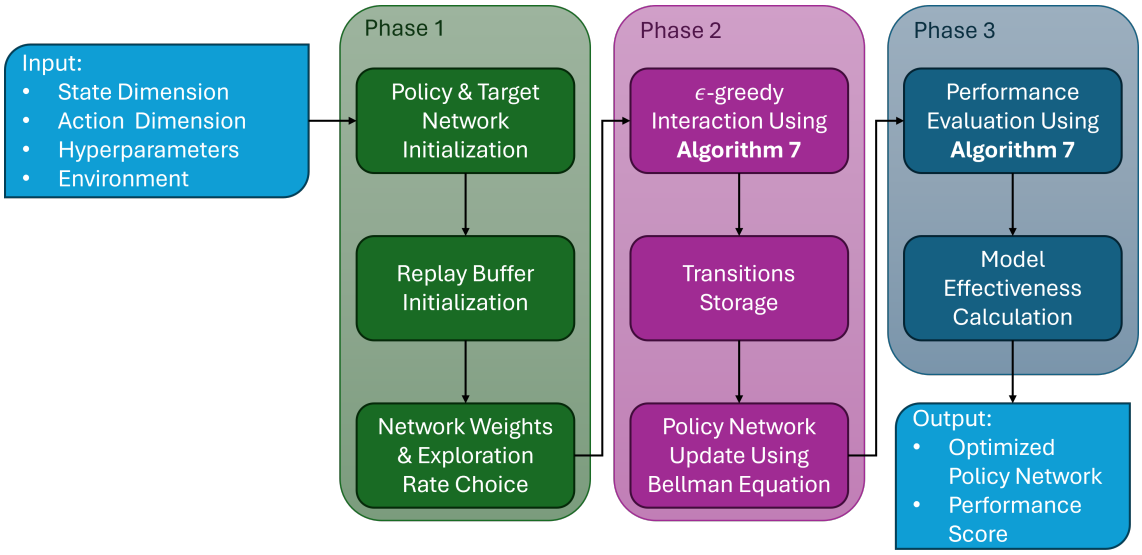


Figure 4.9: Algorithm 8's pipeline

This module implements a DQN that follows an ϵ-greedy strategy in order to ensure a balance between exploitation and exploration. The agent then learns to find the optimal number of guard channels that minimises URLLC packet loss and eMBB buffer size, by storing past experiences in a replay buffer and iteratively updating the policy network. Algorithm 8 uses the state dimension d_s, action dimension d_a, model's hyperparameters $\mathcal{H}$, and the environment $\mathcal{E}$ in order to return the optimized policy network Q_θ and the performance score S_{perf} as can be seen in Figure 4.9.

Algorithm 8 is divided into three phases:

- Phase 1 - Agent and Buffer Initialisation: In the first phase, the algorithm initialises a policy network Q_θ and target network Q_{θ^-} with random weights, and a replay buffer $\mathcal{D}$ to store past experiences. This allows the agent to

learn from random past batches and breaks the time correlation between the consecutive steps. The model also sets the target network weights θ^- and the initial exploration rate ϵ.

- Phase 2 - Training Loop: In the second phase, for every episode e, the agent interacts with the environment using an ϵ-greedy strategy, ensuring a balance between choosing the best known action at the moment, and occasionally exploring a random action in order to discover better options. During each episode, the transitions are stored in the buffer $\mathcal{B}$, and the policy network Q_θ is updated using the Bellman equation.

- Phase 3 - Agent Evaluation: Finally, the model assesses the performance of the agent by first setting the exploration rate $\epsilon = 0$ in order to avoid all the noise of exploration before being tested across multiple scenarios. Then the model's effectiveness is calculated as a performance score S_{perf} that penalises loss violations while rewarding the agent for its proximity to G_{best}. The model finally returns the optimised policy network Q_θ and the performance score S_{perf}.

The fourth and final module is presented in Algorithm 9 in Appendix C, which has the role of orchestrating the cycle of the resource allocation DQN model by linking Algorithms 6, 7, and 8, and producing a final data visualisation as can be seen in Figure 4.10.

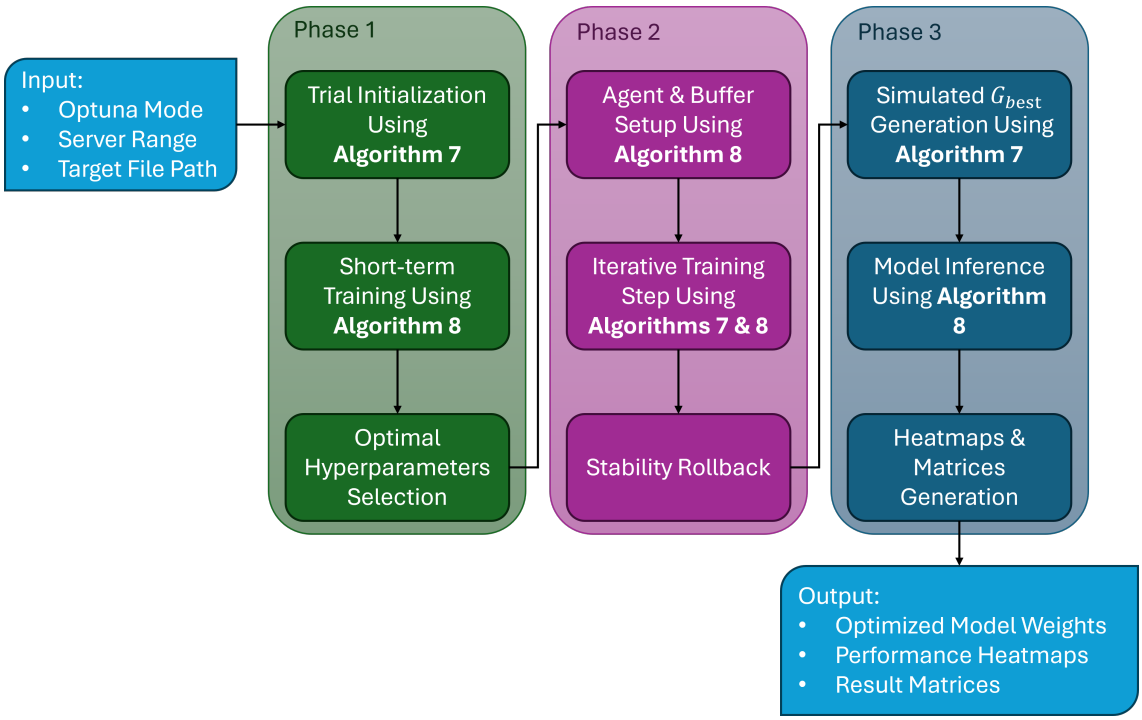


Figure 4.10: Algorithm 9's pipeline

This module automates the selection of the hyperparameters and manages the stability of the ML model over multiple training steps. Specifically, "Optuna is an open-source hyperparameter optimisation framework that automates the search for optimal model parameters using an efficient define-by-run API" [**?**].

Algorithm 9 uses the Optuna mode framework M, the server range $[S_{min}, S_{max}]$, and the target file path P_{model} in order to generate the optimized model weights Θ^*, performance heatmaps H_{map}, result matrices $\mathcal{M}_G$, and the difference matrices $\mathcal{M}_{diff}$ as can be seen in Figure 4.10.

Algorithm 9 is divided into three phases:

- Phase 1 - Hyperparameter Optimisation: In the first phase, the algorithm uses the Optuna framework M to search through the parameters $\mathcal{H}$ in order to maximise the performance score. During each trial, the model initialises the environment $\mathcal{E}$ and an agent to perform a short training cycle of 5000 episodes by calling Algorithm 7. The algorithm then calls Algorithm 8 in order to calculate the performance score S_{perf}, which allows Optuna to identify the optimal set of hyperparameters $\mathcal{H}^*$.

- Phase 2 - Training and Checkpointing: In the second phase, the model uses the optimal hyperparameters $\mathcal{H}^*$ and the replay buffer $\mathcal{D}$ to start a training session. The algorithm assesses the agent's progress by calling *EvaluateAgent* from Algorithm 8 from time to time. If the current score S_{curr} is higher than the previous best score S_{best}, the system saves the new optimised weights to the target file path P_{path}; if it is lower, it rolls back to the best known weights.

- Phase 3 - Results Export and Visualisation: In the final phase, the algorithm uses the optimised weights Θ^* found in the earlier phase to calculate the optimal guard channels G_{best} using the binary search from Algorithm 7 and comparing it to each scenario's agent's results. The results are then exported in the form of decision matrices $\mathcal{M}_G$, difference matrices $\mathcal{M}_{diff}$ and performance heatmaps H_{map}.

4.4 Simulation and Results

The model presented above was trained on two hidden layers, each containing 128 neurons, as can be seen in Figure 4.3. The input layers consist of three nodes, representing the continuous states (URLLC load, eMBB load, normalised capacity) as discussed before. Finally, the output layer is represented by the optimal number of guard channels chosen, G. The parameters used to configure both the 5G simulation environment and the neural network can be found in Table 4.3.

The model was trained over 100 times in order to determine the level where it converges. For every training step, the model calculates the difference between the theoretical number of guard channels G_{th} found in the Poissonian simulation and the experimental number of guard channels G_{exp} found by the agent. This difference is plotted as the percentage of PRBs needed to be added to G_{exp} in order to reach G_{th} for every training step, as can be seen in Figure 4.11.

Table 4.3: DQN Agent and Environment Hyperparameters

Parameter	**Value**	**Description**
Network Architecture & Action Space		
State Space Dimension	3	Input nodes (URLLC load, eMBB load, normalized S)
Neural Network Output	21	Q-values representing percentage fractions (0% to 100% in 5% increments)
Physical Action Decision	G	Final guard channels allocated ($G = \text{round}(\text{action_fraction} \times S)$
Hidden Layers	2	Number of intermediate neural network layers
Neurons per Layer	128	Width of each hidden layer
Environment Configuration		
SLA Loss Threshold (ϵ)	10^{-5}	Strict maximum acceptable URLLC packet loss
Minimum Capacity (S_{min})	10 PRBs	Lower bound of total system capacity
Maximum Capacity (S_{max})	300 PRBs	Upper bound of total system capacity
Simulation Iterations	100000	Number of events simulated per step

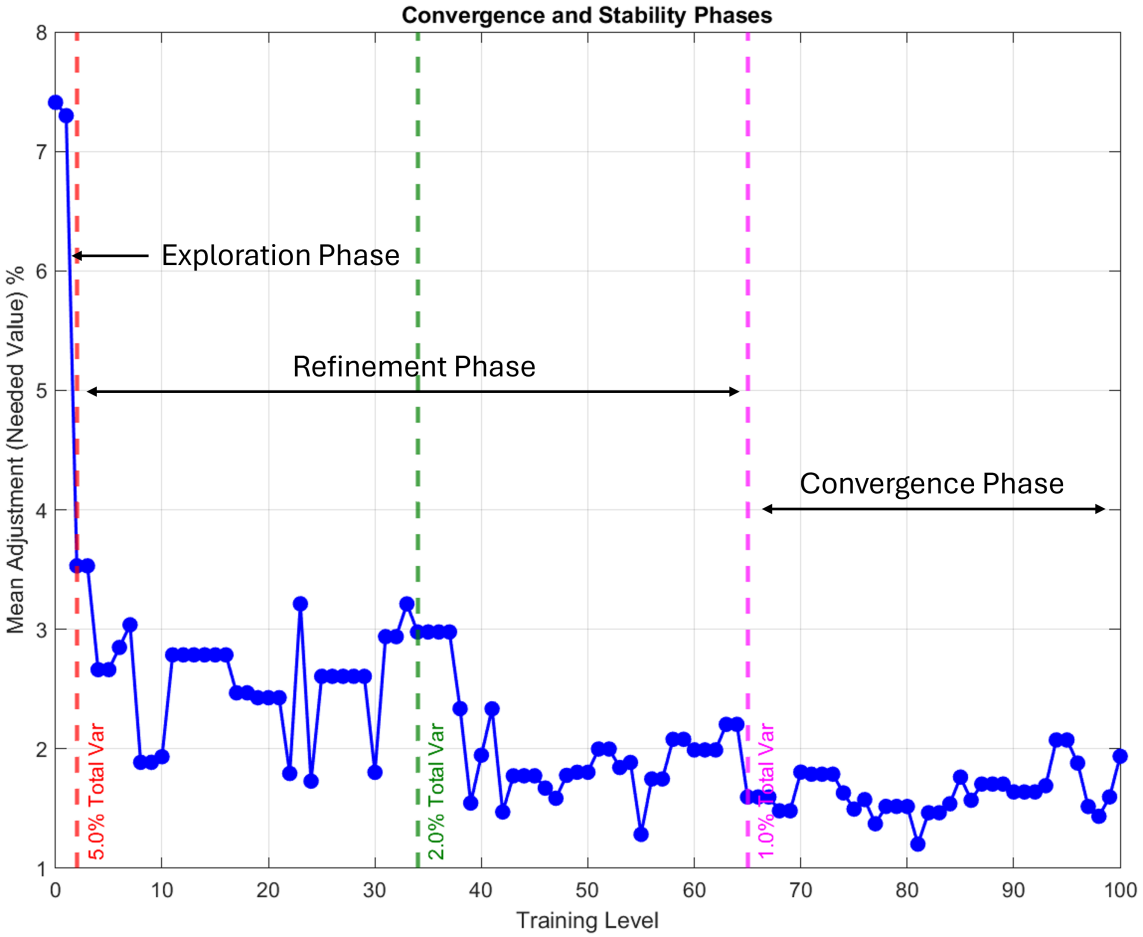


Figure 4.11: Convergence and Stability Analysis of DQN Resource Allocation Model

The convergence of the model was evaluated as the total future fluctuation of this difference over a chosen window. Three stability thresholds were considered: 1%, 2% and 5% as can be seen in Table 4.4.

Table 4.4: Training Convergence Levels by Stability Threshold

Stability Threshold	**Training Step**	**Description**
5.0% ; 13.65 PRBs	2	Initial Policy Discovery
2.0% ; 5.46 PRBs	34	Policy Refinement
1.0% ; 2.73 PRBs	65	Stable Convergence

As can be seen in Figure 4.11, the agent starts with a high rate of learning, satisfying the 5% limit by training step 2. This shows the rapid transition from random exploration to actually finding a functioning policy.

Between step 2 and step 65, the model goes through a refinement phase, and it becomes more and more efficient, but still has some oscillations caused by the burstiness of real-life traffic. At training step 65, the model achieves a real convergence, as the total variation does not exceed 1% (2.73 PRBs) for the remainder of the training.

The chosen trained model was model 65, as it is the first model in the convergence phase. To further extend the research, the model was tested for multiple values of the channel bandwidth, which allowed for results for different values of S. Table 4.5 shows the different channel bandwidth values used and the corresponding number of available PRBs.

Table 4.5: Different Channel Bandwidth Values and Physical Resource Blocks (S) for a Sub-Carrier Spacing of 30kHz

Channel Bandwidth (MHz)	**Total Capacity (PRBs)**
20 MHz	51 PRBs
30 MHz	78 PRBs
40 MHz	106 PRBs
50 MHz	133 PRBs
60 MHz	162 PRBs
70 MHz	189 PRBs
80 MHz	217 PRBs
90 MHz	245 PRBs
100 MHz	273 PRBs

The results for each test were separated into three distinct groups: waste, deficit, and exact. The waste group is the percentage of events where the agent demands more guard channels than the theory, showing that the Poissonian approach is wasteful. The deficit group is the percentage of events where the agents demand more guard channels than the theory, showing that the Poissonian approach is not enough. And the exact group is the percentage of events where

the agent and the theory agree on the amount of guard channels needed, showing that the Poissonian approach is ideal. The results for the different channel bandwidths can be found in Figure 4.12.

The results show that, on average, the theoretical approach is satisfactory around 67% of the time. This means that one third of the time, the theoretical approach is either too pessimistic or too optimistic, which further justifies the need for the DQN model developed in this chapter. Moreover, the results follow the same tendencies, in all cases, the theoretical approach is greatly optimistic. It allocates, on average, 30% of the time, fewer guard channels than needed, leading to a deficit of resources and a URLLC loss probability greater than 10^{-5}. The theoretical approach is pessimistic only around 3% of the time, leading to minimal waste of resources.

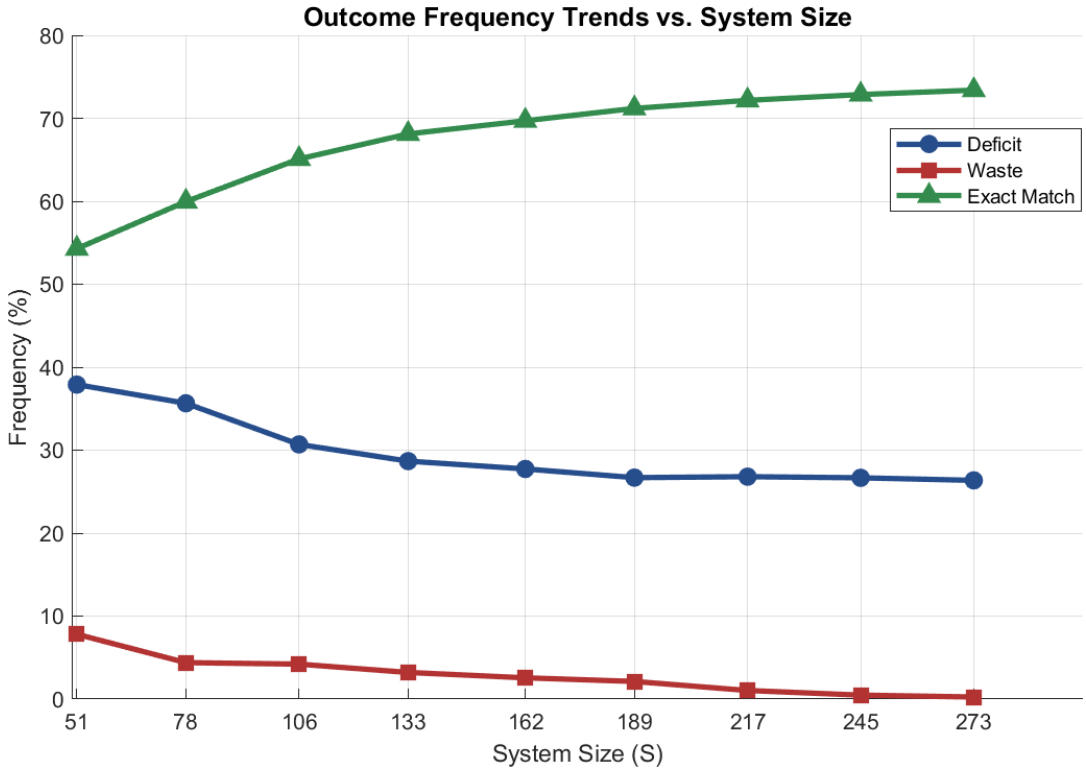


Figure 4.12: Results of the DQN Resource Allocation Model for Different Values of S

Given that over two-thirds of the time the Poissonian approach is satisfactory, 30% of the time it under-allocates, and it almost never over-allocates, it is possible to use the theoretical approach as a baseline and add the lacking resources. Figure 4.13 shows the needed resources to be added to the theoretical value in order to reach the real number of guard channels. The plot also shows the power regressions for each curve.

The formula of the power regression for the target level of 90% is $G^*_{90} = 217.0314 \times S^{-0.7340}$ with an $R^2_{90} = 0.8998$ and $RMSE_{90} = 1.0241$. As for the 95%, $G^*_{95} = 1317.6216 \times S^{-1.0211}$ with an $R^2_{95} = 0.7988$ and $RMSE_{95} = 2.8579$. And finally, for the 99%, $G^*_{99} = 2359.0009 \times S^{-1.0196}$ with an $R^2_{99} = 0.7476$ and $RMSE_{90} = 5.4633$. These results are grouped in Table 4.6.

Table 4.6: The Results of the Regressions for the Different Target Levels

Target Level	**Formula**	R^2	**RMSE**
90	$G^*_{90} = 217.0314 \times S^{-0.7340}$	0.8998	1.0241
95	$G^*_{95} = 1317.6216 \times S^{-1.0211}$	0.7988	2.8579
99	$G^*_{99} = 2359.0009 \times S^{-1.0196}$	0.7476	5.4633

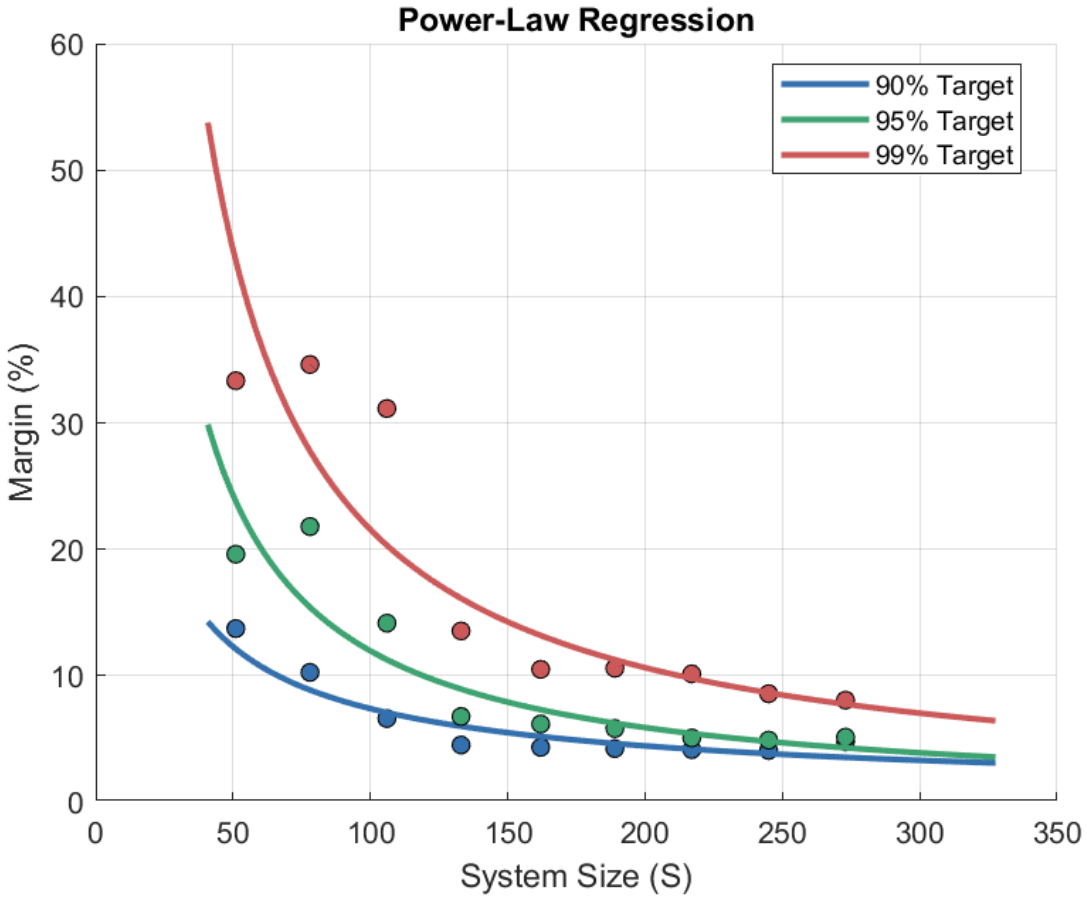


Figure 4.13: Power Regressions for 90%, 95% and 99% Success Targets

The Coefficient of Determination (R^2) and the Root Mean Square Error (RMSE) are used to evaluate the reliability and precision of the power-law regression models across the different guarantee levels. The R^2 metric evaluates how predictably the system state S explains the required resource margin. For a 90% guarantee level, a high R^2 of 0.8998 is reached, showing that moderate traffic behaviour is highly deterministic and well-captured by the proposed formula. In addition, the RMSE measures the absolute average error of these predictions. As for the more strict target reliability of 99%, it can be seen that the R^2 degrades to 0.7476 and the RMSE escalates to 5.4633. This statistical shift reflects the fundamental nature of ultra-reliable traffic. Guaranteeing a 99% reliability requires accommodating extreme traffic bursts that are unpredictable.

4.5 Conclusion

In this chapter, the fundamental limitations of the static, Poisson-based analytical model developed in Chapter 2 were addressed. By analysing the bursty, non-deterministic nature of real-world network traffic, it can be seen that theoretical assumptions fail to capture the extreme fluctuations of modern industrial communications. Relying only on a theoretical queuing model exposes URLLC traffic to an unacceptable deficit in resource allocation around one-third of the time. This shows that the theoretical model can not guarantee the strict 10^{-5} packet loss probability threshold required by industrial clients such as EDF.

To address this issue, a dynamic, realistic approach was introduced through the implementation of a Deep Q-Network (DQN) algorithm trained and tested on generated data instead. As discussed in Section 4.2, existing state-of-the-art solutions require high-level core network visibility. However, the proposed DQN agent operates using only the lower-layer traffic classification model's results from Chapter 3. By observing the system's current URLLC load, eMBB load, and normalised capacity, the agent is able to balance exploration and exploitation in order to step by step develop an optimal resource allocation policy. The training results show a fast convergence and high stability, validating the agent's ability to dynamically allocate the exact number of guard channels needed to minimise the URLLC packet loss without being too selfish towards the eMBB traffic.

Moreover, the difference between the theoretical and the experimental resource needs was quantified and plotted. Using power-law regressions, mathematical formulas were provided to adjust the static theoretical guard channels in order to meet the target success levels of 90%, 95%, and 99%. However, the evaluation metrics of these regressions, specifically the R^2 and the Root Mean Square Error (RMSE) at the 99% level, prove that static adjustments can not fully capture the heavy tails of extreme traffic bursts. This statistical shift reflects the fundamental nature of URLLC traffic, which justifies the need for a dynamic, machine learning-driven agent to guarantee true ultra-reliability.

Finally, with the integration of theoretical queuing limits, ML traffic classification, and DRL dynamic resource allocation, the model forms a robust, end-to-end framework. This framework allows industrial clients to dimension, predict, and dynamically adjust their 5G network slices autonomously using only Poisson's law and the regression formulas provided. Future works could focus on extending this model to a multi-cell environment, or extending the resource allocation to be applicable to any type of slice, not only URLLC and eMBB.

Chapter 5

Conclusion

This thesis focused on the main challenge of 5G network resource allocation, especially between URLLC and eMBB traffic. For industrial clients, such as EDF, the modernisation of smart grids requires highly reliable Protection, Automation, and Control (PAC) systems. However, the limited access to the network's core layers prevents the client from guaranteeing the needed URLLC loss probability. Moreover, the traditional Poissonian approach to resource allocation tends to be greatly optimistic by underallocating the resources. To bridge this gap, this thesis first developed a model for traffic classification with access only to lower network layers, using a Light Gradient-Boosting Machine (LGBM) machine learning algorithm trained on features extracted from packet-level data to distinguish between URLLC and eMBB traffic patterns. The second model focused on agent-based guard channel allocation, where a reinforcement learning agent (specifically, Q-learning) was trained in a simulated real-life environment to dynamically determine the optimal number of guard channels for URLLC. Key findings highlighted that the LGBM classifier achieved high accuracy in separating traffic types without core-layer data, while the reinforcement learning agent identified allocation strategies that better met URLLC loss requirements compared to the static Poisson-based approach.

This led to the need for the generation of a dataset that can be used for training and testing the model presented in Chapter 3. An LGBM model was developed in order to classify network packets between the URLLC and eMBB slice using data extracted from only the PHY and MAC layers from a SA 5G network. The model achieved a 97.6% classification accuracy, showing that industrial clients can successfully monitor their traffic without access to all the KPIs.

Chapter 4 introduces a DQN agent that uses the classified dataset from Chapter 3 in order to train and generate the optimal number of guard channels needed. By navigating a state space defined by the URLLC load, the eMBB load, and the normalised capacity, the agent learned to balance the exploration-exploitation dilemma in order to dynamically adjust the guard channels. The results from this model showed that the theoretical approach is optimistic around 30% of the time, and thus can not be reliable. Furthermore, power-law regressions were generated for a success rate of 90%, 95% and 99%, which allows for the client to simply use the Poisson formula and the regression in order to instantly calculate the realistically needed guard channels.

This thesis opens the horizon for multiple future perspectives. First, the mod-

els presented can be generalised to accept all available types of slices and not just URLLC and eMBB, such as mMTC. Second, the dynamic resource allocation model could be expanded from a single-cell environment to a multi-cell architecture. Finally, a master model can be developed that integrates the three presented models in this thesis, in order to be deployed onto a physical, large-scale smart grid testbed.

Appendix A

Algorithm Presented in Chapter 2

Algorithm 1 Transition algorithm

Require: $\lambda_e, \lambda_U, \mu, S, G$, current state (x_1, x_2, x_3)
Ensure: Holding time Δt and next state (x_1', x_2', x_3')
Initialize next state $\mathcal{S} \leftarrow \emptyset$
Initialize rate $\mathcal{R} \leftarrow \emptyset$
if $x_1 > 0$ **then**
  Add $(x_1 - 1, x_2, x_3)$ to $\mathcal{S}$
  Add rate $2\mu_U x_1$ to $\mathcal{R}$
end if
if $x_2 > 0$ **then**
  if $x_1 + x_2 \leq S - G$ **and** $x_3 > 0$ **then**
    Add $(x_1, x_2, x_3 - 1)$ to $\mathcal{S}$
    Add rate $\mu_e x_2$ to $\mathcal{R}$
  else
    Add $(x_1, x_2 - 1, x_3)$ to $\mathcal{S}$
    Add rate $\mu_e x_2$ to $\mathcal{R}$
  end if
end if
if $x_1 + x_2 < S$ **then**
  Add $(x_1 + 1, x_2, x_3)$ to $\mathcal{S}$
  Add rate λ_U to $\mathcal{R}$
end if
if $x_1 + x_2 < S - G$ **then**
  Add $(x_1, x_2 + 1, x_3)$ to $\mathcal{S}$
  Add rate λ_e to $\mathcal{R}$
else
  Add $(x_1, x_2, x_3 + 1)$ to $\mathcal{S}$
  Add rate λ_e to $\mathcal{R}$
end if
$R \leftarrow \sum \mathcal{R}$
Sample $\Delta t = -\frac{1}{R}\ln(U_1)$, $U_1 \sim \mathcal{U}(0, 1)$
Select index k such that $U_2 \leq \frac{\sum_{i=1}^{k} \mathcal{R}[i]}{R}$, $U_2 \sim \mathcal{U}(0, 1)$
return Δt, $\mathcal{S}[k]$

Appendix B

Algorithms Presented in Chapter 3

Algorithm 2 Data Parsing and Feature Extraction Module

Require: Log File Path F, Window Size W
Ensure: Engineered Feature Matrix X
Phase 1: Log Parsing
Initialize raw packet list $P_{raw} \leftarrow \emptyset$
for all line l in File F **do**
    Match timestamp t and UE ID u in l using Regex
    if match found **and** (l contains "[PHY]" **or** "[MAC]") **then**
        Extract metrics m (HARQ, TB_Len, SNR, PRB, etc.) using Regex
        Create packet object $p \leftarrow \{t, u, m\}$
        Add p to P_{raw}
    end if
end for
Sort P_{raw} by timestamp t
Let $N \leftarrow$ Length(P_{raw})
Initialize feature matrix $X \leftarrow \emptyset$

Algorithm 2 Data Parsing and Feature Extraction Module (Continued)

Phase 2: Feature Engineering
for $i = 1$ **to** N **do**
  Let current packet be $p_i \in P_{raw}$
  1. Temporal Features
  if $i > 1$ **then**
    $\Delta t_{prev} \leftarrow p_i.t - p_{i-1}.t$
  else
    $\Delta t_{prev} \leftarrow 0$
  end if
  if $i < N$ **then**
    $\Delta t_{next} \leftarrow (p_{i+1}.t - p_i.t)$
  else
    $\Delta t_{next} \leftarrow 0$
  end if
  2. Categorical Encoding
  $v_{cat} \leftarrow$ OneHotEncode($\{p_i$.type, p_i.direction$\}$)
  3. Rolling Statistics
  Define window $S_i \leftarrow \{p_k \mid \max(1, i - W + 1) \leq k \leq i\}$
  Initialize vector $v_{stats} \leftarrow \emptyset$
  for all numeric feature $f \in \{$TB_Len, SNR, PRB, ... $\}$ **do**
    $\mu_f \leftarrow$ Mean(f over S_i)
    $\sigma_f \leftarrow$ StdDev(f over S_i)
    Add μ_f, σ_f to v_{stats}
  end for
  Construct vector $x_i \leftarrow [\Delta t_{prev}, \Delta t_{next}, v_{cat}, v_{stats}]$
  Append x_i to X
end for
Phase 3: Clean Up
$X \leftarrow$ FillMissingValues(X) {Handle boundary NaNs}
Remove columns $\{timestamp, ue_id\}$ from X
return X

Algorithm 3 Traffic Classifier Training Module

Require: Training Log Files $\mathcal{F}$, UE ID Sets $\mathcal{U}_{eMBB}, \mathcal{U}_{URLLC}$, Window Size W, Output Path O
Ensure: Saved Model Bundle $\mathcal{B}$
Phase 1: Ingestion and Labelling
Ensure $\mathcal{U}_{eMBB} \cap \mathcal{U}_{URLLC} = \emptyset$ {Validate no overlap}
Initialize $D_{master} \leftarrow \emptyset$
for all file $f \in \mathcal{F}$ **do**
  $D_{file} \leftarrow$ ParseLog(f) {Calls Algorithm 2}
  $D_{master} \leftarrow D_{master} \cup D_{file}$
end for
Assign Labels
for all packet $p \in D_{master}$ **do**
  if $p.ue_id \in \mathcal{U}_{eMBB}$ **then**
    $p.label \leftarrow$ "eMBB"
  else if $p.ue_id \in \mathcal{U}_{URLLC}$ **then**
    $p.label \leftarrow$ "URLLC"
  else
    Remove p from D_{master}
  end if
end for
Remove p where $p.harq = -1$ {Filter System Information}
Phase 2: Feature Engineering and Encoding
Separate targets $Y_{raw} \leftarrow D_{master}.label$
$X \leftarrow$ EngineerFeatures(D_{master}, W) {Calls Algorithm 2}
if Retraining from existing model M_{old} **then**
  $Enc \leftarrow M_{old}.encoder$
  $Cols \leftarrow M_{old}.columns$
  $Y \leftarrow Enc.$transform(Y_{raw})
else
  Initialise new LabelEncoder Enc
  $Y \leftarrow Enc.$fit_transform(Y_{raw})
  $Cols \leftarrow$ GetColumnNames(X)
end if
Phase 3: Alignment and Training
Sanitize column names of X (alphanumeric only)
if Retraining **then**
  $X \leftarrow X.$reindex$(columns = Cols,$ fill_value $= 0)$
end if
Split X, Y into $X_{train}, X_{test}, Y_{train}, Y_{test}$ (70/30 split)
Initialize Classifier $\mathcal{H}$ (LightGBM)
$\mathcal{H}.$fit(X_{train}, Y_{train})
Phase 4: Validation and Artefact Generation
Generate predictions $\hat{Y} \leftarrow \mathcal{H}.$predict$(X_{test})$
Compute Metrics (Accuracy, Precision, Recall) comparing $\hat{Y}$ vs Y_{test}
Construct Bundle $\mathcal{B} \leftarrow$ {model : $\mathcal{H}$, encoder : Enc, columns : $X.columns$, window : W}
Save $\mathcal{B}$
return $\mathcal{B}$

Algorithm 4 Inference and Classification Module

Require: Unlabeled Log File L_{new}, Saved Model Bundle $\mathcal{B}$
Ensure: Vector of Predictions $\hat{Y}$, Confidence Scores C
Phase 1: Initialization
Load components from $\mathcal{B}$:
  Classifier $\mathcal{H} \leftarrow \mathcal{B}.model$
  LabelEncoder $Enc \leftarrow \mathcal{B}.encoder$
  Training Columns $Cols_{train} \leftarrow \mathcal{B}.columns$
  Window Size $W \leftarrow \mathcal{B}.window_size$
Phase 2: Data Preparation
$D_{raw} \leftarrow$ ParseLog(L_{new}) {Calls Algorithm 2}
if "harq" $\in D_{raw}.columns$ **then**
  $D_{clean} \leftarrow \{p \in D_{raw} \mid p.harq \neq -1\}$
else
  $D_{clean} \leftarrow D_{raw}$
end if
if D_{clean} is empty **then**
  return Error: "No classifiable packets found"
end if
$X_{new} \leftarrow$ EngineerFeatures(D_{clean}, W) {Calls Algorithm 2}
Sanitize column names of X_{new} (alphanumeric only)
Phase 3: Schema Alignment
Critical: Force new data to match training structure exactly
$X_{aligned} \leftarrow X_{new}$.reindex$(columns = Cols_{train}, \text{fill_value} = 0)$
Phase 4: Prediction
$\hat{y}_{encoded} \leftarrow \mathcal{H}$.predict$(X_{aligned})$
$P_{probs} \leftarrow \mathcal{H}$.predict_proba$(X_{aligned})$
Phase 5: Decoding and Output
$\hat{Y} \leftarrow Enc$.inverse_transform$(\hat{y}_{encoded})$
$C \leftarrow \max(P_{probs}, \text{axis} = 1)$
for $i = 1$ **to** Length$(\hat{Y})$ **do**
  Output: Packet at $D_{clean}[i].time$: Class=$\hat{Y}[i]$, Conf=$C[i]$
end for
return $\hat{Y}, C$

Algorithm 5 Model Evaluation Procedure

Require: Evaluation Log Files $\mathcal{L}_{eval}$, Saved Model Bundle $\mathcal{B}$, Ground Truth Sets $\mathcal{U}_{eMBB}$,$\mathcal{U}_{URLLC}$
Ensure: Accuracy Score A, Confusion Matrix CM
Phase 1: Initialization
Load components from $\mathcal{B}$:
  Classifier $\mathcal{H} \leftarrow \mathcal{B}.model$
  LabelEncoder $Enc \leftarrow \mathcal{B}.encoder$
  Training Columns $Cols_{train} \leftarrow \mathcal{B}.columns$
  Window Size $W \leftarrow \mathcal{B}.window_size$
Phase 2: Data Ingestion & Ground Truth Labeling
Initialize $D_{eval} \leftarrow \emptyset$
for all file $f \in \mathcal{L}_{eval}$ **do**
  $D_{file} \leftarrow$ ParseLog(f) {Calls Algorithm 2}
  $D_{eval} \leftarrow D_{eval} \cup D_{file}$
end for
Assign Ground Truth Labels
for all packet $p \in D_{eval}$ **do**
  if $p.ue_id \in \mathcal{U}_{eMBB}$ **then**
    $p.true_label \leftarrow$ "eMBB"
  else if $p.ue_id \in \mathcal{U}_{URLLC}$ **then**
    $p.true_label \leftarrow$ "URLLC"
  else
    Remove p from D_{eval}
  end if
end for
Remove p where $p.harq = -1$ {Filter System Information}
if D_{eval} is empty **then**
  return Error: "No valid evaluation data found"
end if
Phase 3: Feature Engineering & Alignment
Extract targets $Y_{true} \leftarrow Enc.$transform$(D_{eval}.true_label)$
$X_{eval} \leftarrow$ EngineerFeatures(D_{eval}, W) {Calls Algorithm 2}
Sanitize column names of X_{eval}
// Align with training schema
$X_{aligned} \leftarrow X_{eval}$.reindex$(columns = Cols_{train}, \text{fill_value} = 0)$
Phase 4: Prediction & Scoring
$\hat{Y}_{pred} \leftarrow \mathcal{H}$.predict$(X_{aligned})$
$A \leftarrow$ AccuracyScore$(Y_{true}, \hat{Y}_{pred})$
$CM \leftarrow$ ConfusionMatrix$(Y_{true}, \hat{Y}_{pred})$
Print Classification Report $(Precision, Recall, F1)$
Phase 5: Visualization
Plot Confusion Matrix(CM)
Plot ROC Curve$(Y_{true}, \mathcal{H}.$predict_proba$(X_{aligned}))$
return A, CM

Appendix C

Algorithms Presented in Chapter 4

Algorithm 6 Continuous-Time Markov Chain (CTMC) Simulation for 5G Slicing

Require: Arrival rates λ_e, λ_u, Service rates μ_e, μ_u, Total resources S, Guard channels G, Iterations N_{iter}
Ensure: Packet loss ratio L, Average eMBB queue size W_{avg}
1: **Phase 1: Initialization**
2: Initialize system state $e \leftarrow \{x_1 : 0, x_2 : 0, x_3 : 0\}$
3: $\{x_2$: active eMBB, x_1: active URLLC, x_3: queued eMBB$\}$
4: Set simulation horizon $H \leftarrow N_{iter}/(\lambda_e + \lambda_u)$
5: Initialize accumulators $T_{total} \leftarrow 0$, $Wait_{sum} \leftarrow 0$, $Cumul_{loss} \leftarrow 0$
6: Seed random number generator using system time and process ID

Algorithm 6 Continuous-Time Markov Chain (CTMC) Simulation (Continued)

Phase 2: Simulation Loop
while $T_{total} < H$ **do**
  1. Calculate Event Propensities (R)
  $R_{u_serv} \leftarrow \mu_u \cdot x_1$, $R_{e_serv} \leftarrow \mu_e \cdot x_2$
  $R_{u_arr} \leftarrow \lambda_u$, $R_{e_arr} \leftarrow \lambda_e$
  $R_{total} \leftarrow R_{u_serv} + R_{e_serv} + R_{u_arr} + R_{e_arr}$
  2. Determine Time to Next Event
  Sample $\Delta t \leftarrow -\frac{1}{R_{total}} \ln(\text{Uniform}(0, 1))$
  $T_{total} \leftarrow T_{total} + \Delta t$
  3. Statistics Update
  $Wait_{sum} \leftarrow Wait_{sum} + (x_3 \cdot \Delta t)$
  if $x_1 + x_2 = S$ **then**
    $Cumul_{loss} \leftarrow Cumul_{loss} + \Delta t$
  end if
  4. State Transition Execution
  Select event i with probability $P_i = R_i / R_{total}$
  if Event i is URLLC Arrival **then**
    if $x_1 + x_2 < S$ **then**
      $x_1 \leftarrow x_1 + 1$
    end if
  else if Event i is eMBB Arrival **then**
    if $x_1 + x_2 < S - G$ **then**
      $x_2 \leftarrow x_2 + 1$
    else
      $x_3 \leftarrow x_3 + 1$
    end if
  else if Event i is eMBB Departure **then**
    if $x_1 + x_2 \leq S - G$ **and** $x_3 > 0$ **then**
      $x_3 \leftarrow x_3 - 1$
    else
      $x_2 \leftarrow x_2 - 1$
    end if
  else if Event i is URLLC Departure **then**
    $x_1 \leftarrow x_1 - 1$
  end if
end while
Phase 3: Performance Metrics Extraction
$L \leftarrow Cumul_{loss}/H$, $W_{avg} \leftarrow Wait_{sum}/H$
return L, W_{avg}

Algorithm 7 5G Slicing Environment Module

Require: Server Range $[S_{min}, S_{max}]$, Service Rates μ_e, μ_u, Iterations N_{iter}, Loss Threshold ϵ
Ensure: State Vector s, Reward r

Phase 1: Environment Reset
Sample total servers $S \leftarrow \text{UniformInt}(S_{min}, S_{max})$
Sample random arrival rates $\lambda_u \leftarrow \text{Uniform}(0, S)$ and $\lambda_e \leftarrow \text{Uniform}(0, S)$
Construct state $s \leftarrow [\lambda_u, \lambda_e, S/S_{max}]$
return s
Phase 2: Step Execution and Simulation
Receive action index $a \in [0, N_{actions} - 1]$
Map action to guard channels: $G \leftarrow \text{round}\left(\frac{a}{N_{actions}-1} \cdot S\right)$
if simulate is **true then**
    1. Network Simulation
    Execute Simu($\lambda_e, \lambda_u, \mu_e, \mu_u, S, G, N_{iter}$) {Calls Algorithm 6}
    Retrieve URLLC loss L_G and eMBB queue average Q_G
    2. Binary Search for Minimal Valid Guard Channels (G_{best})
    Initialize $G_{best} \leftarrow S$
    if $L_G \leq \epsilon$ **then**
        Set $low \leftarrow 0, high \leftarrow G, G_{best} \leftarrow G$
        while $low \leq high$ **do**
            $mid \leftarrow \lfloor low + (high - low)/2 \rfloor$
            Run simulation for mid to get L_{mid}
            if $L_{mid} \leq \epsilon$ **then**
                $G_{best} \leftarrow mid, high \leftarrow mid - 1$
            else
                $low \leftarrow mid + 1$
            end if
        end while
    end if
    Phase 3: Reward Calculation
    if $L_G > \epsilon$ **then**
        Violation Penalty: $r \leftarrow -1 - \log_{10}(1 + \frac{L_G}{\epsilon})$
    else
        Efficiency Reward: $\Delta_{norm} \leftarrow \frac{|G - G_{best}|}{S}$
        $r \leftarrow \frac{1.0}{1.0 + \Delta_{norm} \cdot 10}$
    end if
else
    $r \leftarrow 0.0$
end if
return s, r

Algorithm 8 DQN Training and Agent Evaluation Module

Require: State dimension d_s, Action dimension d_a, Hyperparameters $\mathcal{H}$ (Learning rate α, Batch size B, Discount γ, Epsilon decay), Environment $\mathcal{E}$
Ensure: Optimized Policy Network Q_θ, Performance Score S_{perf}

Phase 1: Agent and Buffer Initialisation
Initialize Policy Network Q_θ and Target Network Q_{θ^-} with random weights
Initialize Replay Buffer $\mathcal{D}$ with capacity C
Set target network weights $\theta^- \leftarrow \theta$
Set initial exploration rate $\epsilon \leftarrow \mathcal{H}_{eps_start}$
Phase 2: Training Loop
for all episode $e \in \{1, \dots, E\}$ **do**
  Observe initial state $s \leftarrow \mathcal{E}.reset()$ {Calls Algorithm 7}
  Select action a using ϵ-greedy strategy:
  $a \leftarrow \begin{cases} \text{UniformInt}(0, d_a - 1) & \text{with probability } \epsilon \\ \arg\max_a Q_\theta(s, a) & \text{with probability } 1 - \epsilon \end{cases}$
  Execute action a in $\mathcal{E}$, observe reward r, next state s', and done flag d {Calls Algorithm 7}
  Append transition (s, a, r, d, s') to $\mathcal{D}$
  if Size($\mathcal{D}$) $\geq B$ **then**
    Sample random batch $\mathcal{B} \leftarrow \{(s_i, a_i, r_i, d_i, s'_i)\}_{i=1}^{B}$ from $\mathcal{D}$
    Compute Target Q-value: $y_i \leftarrow r_i + \gamma \cdot (1 - d_i) \cdot \max_{a'} Q_{\theta^-}(s'_i, a')$
    Update θ by minimizing Smooth L1 Loss: $\mathcal{L} = \text{Huber}(y_i, Q_\theta(s_i, a_i))$
    Perform gradient descent step using Adam optimiser
  end if
  Epsilon Decay: $\epsilon \leftarrow \epsilon_{end} + (\epsilon_{start} - \epsilon_{end}) \exp(-e/\text{decay})$
  if $e \pmod{\text{TargetUpdateFreq}} = 0$ **then**
    Synchronise Target Network: $\theta^- \leftarrow \theta$
  end if
end for
Phase 3: Agent Evaluation
Set $\epsilon \leftarrow 0$ (Pure Exploitation)
for all evaluation episode $j \in \{1, \dots, N_{eval}\}$ **do**
  $s \leftarrow \mathcal{E}.reset()$ {Calls Algorithm 7}
  $a \leftarrow \arg\max Q_\theta(s, a)$
  Execute N_{runs} per scenario to average stochasticity
  Retrieve simulation metrics $\{L_{urllc}, G, G_{best}\}$ from $\mathcal{E}.step(a)$ {Calls Algorithm 7}
  Compute average URLLC loss $\bar{L}$
  if $\bar{L} >$ SLA_LOSS **then**
    Increment Violation Count V
  else
    Accumulate distance $\Delta_G = |G - G_{best}|$
  end if
end for
$S_{perf} \leftarrow -(V/N_{eval}) \cdot 10000 - \text{Avg}(\Delta_G)$
return Q_θ, S_{perf}

Algorithm 9 DQN Training Orchestration and Hyperparameter Optimisation

Require: Mode $M \in \{\text{optuna, train, export, full}\}$, Range $[S_{min}, S_{max}]$, Target File Path P_{path}
Ensure: Optimized Model Weights Θ^*, Performance Heatmaps $\mathcal{H}_{map}$, Result Matrices $\mathcal{M}_G$, Difference Matrices $\mathcal{M}_{diff}$

Phase 1: Hyperparameter Optimisation (Optuna)
if $M \in \{\text{optuna, full}\}$ **then**
  Define Objective Function $f(\mathcal{H})$:
    1. Initialize $\mathcal{E}$ and Agent with trial parameters $\mathcal{H}$ {Calls Algorithm 7}
    2. Perform short-term training ($N_{episodes} = 5000$) {Calls Algorithm 8}
    3. **return** Evaluation score S_{perf} {Calls Algorithm 8}
  Run Study: $\mathcal{H}^* \leftarrow$ Maximize $f(\mathcal{H})$ over N_{trials}
  Save optimal parameters $\mathcal{H}^*$ to JSON
end if
Phase 2: Training and Checkpointing
if $M \in \{\text{train, retrain, full}\}$ **then**
  Load $\mathcal{H}^*$ and Initialize Agent and Replay Buffer $\mathcal{D}$
  if $M =$ retrain **then**
    Load existing weights from P_{path}
  end if
  Initialize $S_{best} \leftarrow \text{EvaluateAgent}(\text{InitialModel})$ {Calls Algorithm 8}
  for all episode $e \in \{1, \ldots, E_{total}\}$ **do**
    Run training step: {Calls Algorithms 7 & 8}
    if $e \pmod{N_{eval}} = 0$ **then**
      $S_{curr} \leftarrow \text{EvaluateAgent}(\text{CurrentModel})$ {Calls Algorithm 8}
      if $S_{curr} > S_{best}$ **then**
        $S_{best} \leftarrow S_{curr}$
        Save CurrentWeights $\rightarrow P_{path}$
      else
        Load weights from P_{path} {Rollback to best known model}
      end if
    end if
  end for
  $\Theta^* \leftarrow$ Weights loaded from P_{path}
end if
Phase 3: Results Export and Visualisation
if $M =$ export **then**
  Load Θ^* from P_{path} and initialize $\mathcal{E}$
  for all Target Servers $S \in \{S_{40MHz}, S_{80MHz}, S_{100MHz}\}$ **do**
    1. Simulated Baseline:
    for all Grid $(\lambda_u, \lambda_e) \in [0, S]^2$ **do**
      Compute G_{best} via binary search in $\mathcal{E}$ {Calls Algorithm 7}
    end for
    2. Model Inference:
    for all Grid $(\lambda_u, \lambda_e) \in [0, S]^2$ **do**
      Observe state $s \leftarrow [\rho_u, \rho_e, S/S_{max}]$
      $G_{model} \leftarrow \text{Agent}_{\Theta^*}(s)$ {Calls Algorithm 8}
      Diff $\leftarrow G_{model} - G_{best}$
    end for
    Generate $\mathcal{H}_{map}$ for G_{model} and $\mathcal{M}_{diff}$
    Save $\mathcal{M}_G$ and $\mathcal{M}_{diff}$ to CSV files
  end for
end if
return Θ^*, $\mathcal{H}_{map}$, $\mathcal{M}_G$, $\mathcal{M}_{diff}$

ECOLE DOCTORALE

Titre: Méthodes de dimensionnement et de déploiement de réseaux mobiles virtualisés : Application à la 5G/6G et optimisation pour les réseaux électriques intelligents

Mots clés: Découpage de réseaux 5G, URLLC, eMBB, Réseaux électriques intelligents

Résumé:
La modernisation des réseaux électriques intelligents (smart grids) industriels chez EDF nécessite des communications hautement fiables pour les systèmes de Protection, d'Automatisation et de Contrôle (PAC). Bien que le découpage de réseau (network slicing) 5G prenne en charge les messages critiques via les communications URLLC, le manque de visibilité des clients industriels sur le cœur de réseau de l'opérateur empêche de garantir la probabilité de perte de paquets requise de 10^{-5}. De plus, les modèles traditionnels d'allocation basés sur la loi de Poisson ne reflètent pas la nature par rafales du trafic réel, entraînant un déficit de ressources 30% du temps. Pour combler ce manque de visibilité, un modèle de classification par apprentissage automatique (LGBM) a été développé. Entraîné sur un trafic expérimental et utilisant exclusivement les données des couches inférieures PHY et MAC, il atteint une précision de $97,6\%$, prouvant que les clients peuvent surveiller leur trafic de manière autonome. Ensuite, un modèle d'allocation dynamique des ressources fondé sur l'apprentissage par renforcement profond (DRL) a été introduit. Un agent DQN interagit en continu avec le réseau dans un espace d'états (charges URLLC/eMBB, capacité de cellule) pour équilibrer le compromis exploration-exploitation. Il alloue ainsi dynamiquement les canaux de garde requis pour minimiser la perte de paquets URLLC tout en préservant le tampon eMBB. Ce travail offre une solution complète permettant aux industriels de prédire, dimensionner et ajuster instantanément leurs ressources 5G.

Title: Methods for Sizing and Deploying Virtualised Mobile Networks: Application to 5G/6G, Optimisation for Smart Electrical Grids

Keywords: Slicing, URLLC, eMBB, Smart grids

Abstract:
Modernising industrial smart grids operated by EDF requires highly reliable communications for Protection, Automation, and Control (PAC) systems. Although 5G network slicing supports mission-critical messages via URLLC, industrial clients lack visibility into the operator's core network, making it hard to guarantee the required 10^{-5} packet loss probability. Furthermore, traditional allocation models based on Poisson's law fail to capture real-world traffic burstiness, causing resource deficits 30% of the time. To bridge this visibility gap, a Machine Learning classification model was developed. A LightGBM model was implemented to classify traffic into URLLC and eMBB slices using experimental data restricted to the lower PHY and MAC layers. Achieving a 97.6% accuracy demonstrates that industrial clients can autonomously monitor traffic without operator-level KPIs. Building on this, a dynamic resource allocation model using Deep Reinforcement Learning (DRL) was introduced. A Deep Q-Network (DQN) agent continuously interacts with the network environment across a continuous state space (URLLC/eMBB loads, cell capacity) to manage the exploration-exploitation trade-off. The agent dynamically allocates the optimal guard channels to minimize URLLC packet loss while maintaining fairness toward the eMBB buffer. Combining queuing limits, ML classification, and DQN, this thesis empowers industrial clients to autonomously predict, dimension, and adjust their 5G network slice resources instantaneously.

Institut Polytechnique de Paris
91120 Palaiseau, France