AI-Driven Network Slicing Framework for 6G: A Reinforcement Learning Approach Using NS-3 Traffic Simulation for 5G Network

K Venkata Bharadwaj 121CS0011
Rama Krishna Aare 121CS0060

Under the supervision of: **Dr. R. Anil Kumar**

Department of Computer Science & Engineering
Indian Institute of Information Technology Design and Manufacturing Kurnool

Introduction

- This project simulates a 6G network slicing scenario using NS-3, capturing dynamic traffic across multiple service classes.
  It generates realistic synthetic datasets for enhanced Mobile Broadband (eMBB), Ultra-Reliable Low-Latency Communication (URLLC), and massive Machine-Type Communication (mMTC).
- We implement a baseline Software-Defined Networking (SDN) controller that allocates resources based on queue occupancy and latency thresholds.
  This establishes a deterministic performance reference against which AI-driven methods are benchmarked.
- A Deep Q-Network (DQN) reinforcement learning agent is trained on the NS-3 dataset to optimize real-time bandwidth allocation.
  The DQN learns adaptive policies to minimize latency violations and packet drops under variable load conditions.
- The trained DQN model is reintegrated into NS-3 via PyBind11 for live inference in slice orchestration.
  Comparative analysis highlights significant improvements in latency, reliability, and overall Quality of Service (QoS).

Problem Statement & Objectives

- Traditional SDN controllers rely on static threshold-based resource allocation, failing to adapt to heterogeneous and dynamic 6G traffic demands.
  This inflexibility leads to frequent Quality of Service (QoS) violations in latency-sensitive and throughput-critical slices.
- **Objective 1:** Develop an NS-3 simulation that generates slice-specific traffic for eMBB, URLLC, and mMTC, logging detailed QoS metrics.
  Ensure the dataset captures realistic scenarios across varying load patterns for robust algorithm training.
- **Objective 2:** Train a Deep Q-Network (DQN) agent to learn adaptive bandwidth allocation policies that minimize latency spikes and packet drops.
  Leverage a custom Gym environment for sequential state-action-reward processing and network performance optimization.
- **Objective 3:** Reintegration of the trained DQN model into NS-3 via PyBind11 for live slice orchestration.
  Conduct comparative analysis against the heuristic SDN baseline to quantify improvements in latency, reliability, and resource utilization.

Motivation Behind the Problem Statement

- The growing heterogeneity of 6G applications, from ultra-low latency URLLC to high-throughput eMBB and massive IoT mMTC, demands dynamic resource management.
  Static allocation methods cannot satisfy such diverse QoS requirements simultaneously, leading to unpredictable performance.
- Heuristic slicing based on fixed thresholds often fails under bursty or peak traffic patterns, resulting in increased latency and packet loss for critical services.
  This unpredictability undermines the reliability of slices designed for mission-critical applications like autonomous vehicles.
- Enhancing adaptability in bandwidth allocation can significantly improve both user experience and overall network efficiency.
  Adaptive controllers can proactively reassign resources to the most demanding slices, ensuring smoother service.
- Reinforcement learning offers a data-driven approach to autonomously learn optimal slicing policies, reducing reliance on manual threshold tuning.
  An RL agent can continuously refine its decisions based on real-time feedback, outperforming static heuristics in dynamic environments.

Evolution from 4G to 5G to 6G

- **4G (2009)**: Peak speeds up to 100Mbps, latency 50ms.
  Enabled standard mobile broadband services like HD video streaming.
- **5G (2020)**: Peak speeds 1–10Gbps, latency 1ms.
  Supports eMBB, URLLC for autonomous systems, and mMTC for IoT connectivity.
- **6G (anticipated)**: Peak speeds 100Gbps, latency lt; 0.1ms.
  Envisions AI-native network slicing, integrated sensing, and THz-enabled communication.

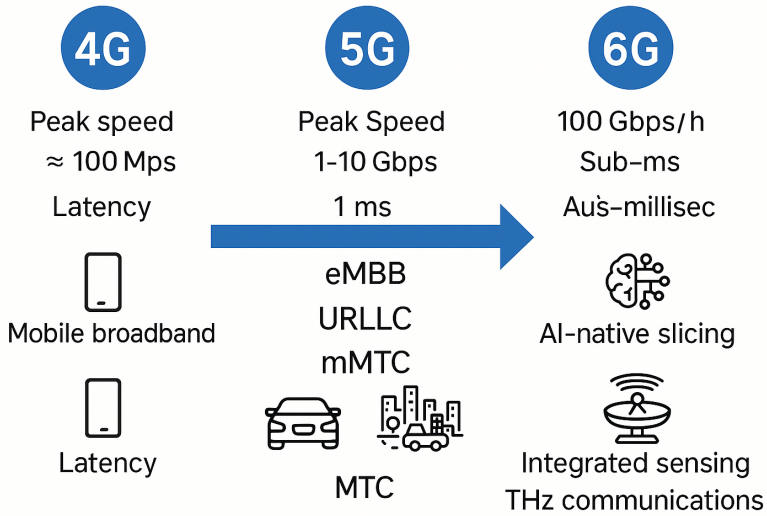


Figure: Evolution of mobile generations with key parameters and use cases

Network Slicing: Definition and Purpose

- **Definition:** Logical partitioning of shared physical network resources into isolated virtual slices.
  Each slice offers custom Quality of Service (QoS) profiles tailored to specific application needs.
- **Purpose:** Enables co-existence of heterogeneous services (eMBB, URLLC, mMTC) on a unified infrastructure.
  Ensures resource isolation, security, and SLA enforcement without hardware over-provisioning.
- **Real-Time Orchestration:** Dynamic slice lifecycle management (creation, scaling, release) via SDN/NFV.
  Controllers adjust slice parameters in response to live traffic metrics and service demands.

Types of Network Slices and Practical Use Cases

- **eMBB Slice:** High bandwidth (up to 100Mbps per UE), suitable for video streaming and AR/VR.
  Example: 4K video conferencing in enterprise and entertainment applications.
- **URLLC Slice:** Ultra-low latency (1ms) and high reliability (99.999%), critical for control loops.
  Example: Autonomous vehicle coordination, industrial automation, remote surgery.
- **mMTC Slice:** Massive connectivity $(10^6 devices/km) with low data rates, energy-efficient$.

Example: Smart metering, environmental sensors, large-scale IoT deployments.

- **Emerging Slices:** Ad-hoc or vertical-specific slices (e.g., private 5G for manufacturing).
  Enables tailored security, performance, and regulatory compliance for industry verticals.

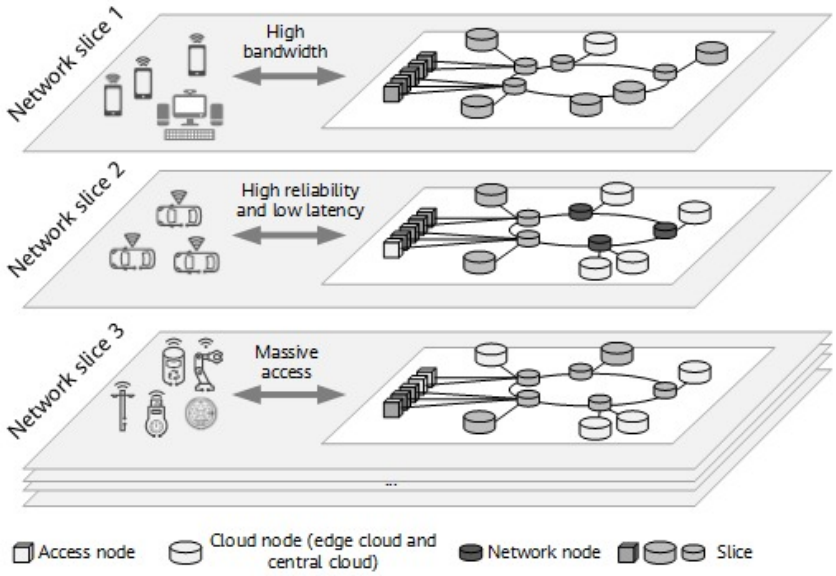


Figure: Examples of eMBB, URLLC, and mMTC deployments

Reinforcement Learning: Focus on Deep Q-Network (DQN)

- **Q-Learning Basis:** DQN extends traditional Q-learning by using a neural network to approximate the Q-value function $Q(s, a)$.
  This enables handling high-dimensional state spaces where tabular methods fail.
- **Experience Replay:** Transitions (s, a, r, s') are stored in a replay buffer and sampled randomly for training.
  Reduces correlation among samples, improving stability and convergence during learning.
- **Target Network:** A separate target network Q_{target} is periodically synchronized with the online network.
  Mitigates oscillations and divergence by providing a stable target for temporal-difference updates.
- **Epsilon-Greedy Policy:** Balances exploration and exploitation via a decaying exploration rate ϵ.
  Ensures the agent explores new actions early and gradually focuses on the best-known actions.
- **Specialty:** Well-suited for discrete action spaces like bandwidth adjustments in slicing (e.g., $\{-10, 0, +10\}$ Mbps).
  Leverages deep learning to scale RL to complex network orchestration tasks in real time.

NS-3 Simulator: Choice and Features

- **Why NS-3?** Open-source, discrete-event simulator with modular C++ core and Python bindings.
  Offers rich support for wireless, wired, and SDN/NFV modules—ideal for realistic 5G/6G research.
- **Why not others?** Alternatives (OMNeT++, QualNet) lack NS-3's integration with latest mmWave and SDN frameworks.
  NS-3's active development community ensures timely support for emerging 6G features and standards.
- **Key Network Parameters:** Supports mmWave and LTE modules, point-to-point links, packet tags, and queue disciplines.
  Enables detailed configuration of bandwidth, latency, queue sizes, and custom traffic-control policies.
- **SDN Integration:** Native support for OpenFlow and custom SDN controllers via NetDevice and Socket APIs.
  Facilitates real-time slice orchestration with controller logic (heuristic or DQN) embedded in simulation.

Literature Survey: AI 6G for Emergencies and Smart Cities

A. Alnoman et al. (2024), IEEE Access

Topic: AI-enhanced user localization in emergency scenarios using NILM and federated learning.
Solution: Combines NILM, deep learning, and federated approaches to infer indoor user positions accurately.
Gaps: Limited testing under dynamic urban disaster conditions and real-time system constraints.
Insights: Federated model preserves privacy while tapping heterogeneous sensor data, guiding our privacy-aware log design.

A. M. Alwakeel A. K. Alnaim (2024), Sensors

Topic: Strategic 6G slicing framework for IoT in smart cities with security focus.
Solution: Uses mathematical resource allocation models with encryption and authentication enhancements.
Gaps: Allocation logic is opaque and may not scale to dynamic real-time traffic.
Insights: Highlights need for transparent policies, informing our interpretable RL approach.

Literature Survey: AI/ML Integration and Security

R. Chataut et al. (2024), Sensors

Topic: AI/ML, mmWave, THz, and 3GPP standardization for 6G resource management.
Solution: Theoretical analyses and simulation studies of AI-native air interfaces and mobility controls.
Gaps: Energy overhead and lack of end-to-end trials under realistic traffic scenarios.
Insights: Informs our choice of lightweight DQN to balance performance and computational cost.

E. N. A. M. S. Chuan et al. (2024), IEEE Comm. Mag.

Topic: Blockchain and AI/ML for securing Open RAN in 6G networks.
Solution: Implements a tamper-proof ledger for RAN data and ML-based anomaly detection.
Gaps: No slice-specific localization security and unclear real-time performance impact.
Insights: Guides our secure logging and anomaly resilience in slice orchestration.

Literature Survey: Hybrid AI Models and Industry Alliances

R. Dangi P. Lalwani (2024), Journal of Supercomputing

Topic: CNN-BiLSTM for adaptive slicing allocation on IP flow datasets.
Solution: Uses convolutional layers for feature extraction and BiLSTM for temporal demand prediction.
Gaps: High compute complexity and limited explainability in real-time scenarios.
Insights: Validates deep approaches, motivating our interpretable and efficient DQN strategy.

E. Obiodu et al. (2024), AI-RAN Alliance Blog

Topic: NVIDIA's AI-RAN Alliance and 3GPP AI/ML initiatives for future RAN.
Solution: Presents AI-native RAN architecture on the Aerial platform, focusing on channel estimation.
Gaps: Lacks real-world mobility and traffic validation.
Insights: Shapes our integration of 3GPP standards and realistic mobility models in NS-3.

Literature Survey: AI-native Architectures and Interpretability

W. Wu et al. (2022), IEEE Wireless Communications

Topic: AI-native slicing architecture for 6G slice and AI service orchestration.
Solution: Simulation-based case studies demonstrating slice lifecycle management with AI loops.
Gaps: Energy inefficiencies and absence of transparent decision logs.
Insights: Encourages transparent logging of policy decisions in our RL integration.

M. Zhao et al. (2025), arXiv Preprint

Topic: InSlicing: interpretable slice configuration via knowledge-assisted networks.
Solution: Combines global search with local refinement for cost-effective slice setups.
Gaps: Does not integrate mobility or localization factors.
Insights: Informs our MDP design to include mobility-aware state features.

Research Gaps Identified

- Lack of realistic validation of AI-driven slicing in dynamic 6G scenarios such as urban mobility and emergency events.
  This gap motivates our NS-3 simulations incorporating RandomWalk2d mobility and URLLC spike modeling.
- Opaque decision logic in existing slicing frameworks limits operator trust and interpretability.
  We address this by logging per-step slice metrics and resource changes for clear explanation of RL decisions.
- High computational complexity of hybrid deep-learning models hinders real-time orchestration.
  We propose a streamlined DQN with small hidden layers and optimized replay buffer for efficient inference.
- Absence of integrated localization and security mechanisms within slice orchestration.
  We integrate secure logging and NILM-based localization triggers to enable context-aware resource prioritization.

Novelty of This Research

- End-to-end NS-3 slicing framework capturing eMBB, URLLC, and mMTC traffic with detailed QoS logging.
  Provides a robust dataset for training adaptive RL policies.
- Custom Gym environment and DQN agent specifically tailored for discrete bandwidth control.
  Hyperparameters (learning rate, ϵ-decay, buffer size) fine-tuned for rapid convergence.
- Seamless PyBind11 integration enabling live DQN inference within NS-3 every 0.5s simulation time.
  Facilitates real-time slice adjustments responsive to current network conditions.
- Weighted packet classification combining size, latency requirement, rate, and throughput metrics.
  Ensures transparent, interpretable logging of resource allocation decisions.

NS-3 Topology Visualization

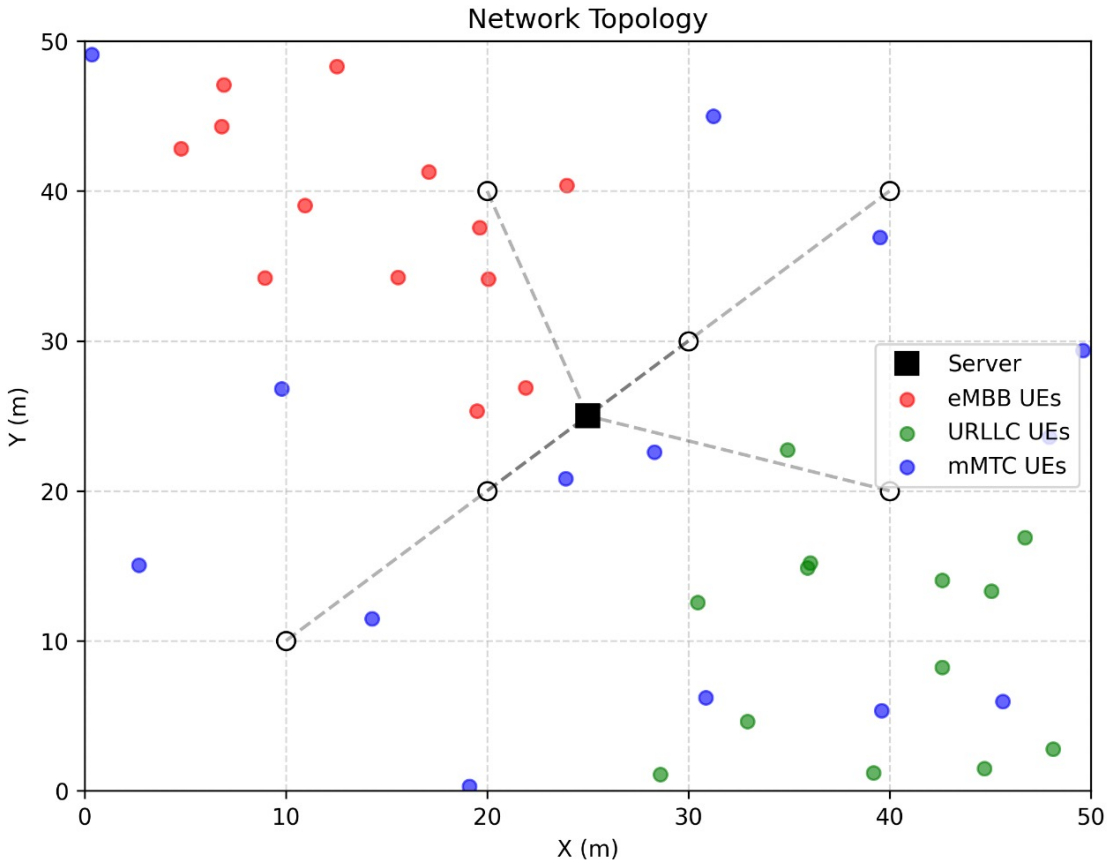


Figure: Graphical layout of UE nodes, mmWave routers, gNodeB, and server placement

Topology Explanation

- UE nodes are distributed randomly within a 50×50m grid, mimicking realistic user movement and load distribution. This ensures that the DQN model sees varied state conditions during training.
- Six mmWave routers are placed in a 2×3 grid for optimal coverage and to introduce line-of-sight variability. Accurate modeling of urban obstacles and link quality fluctuations is achieved.
- A central gNodeB connects all routers to a server via 10Gbps point-to-point links with 1ms delay. Simulates high-speed backhaul for realistic end-to-end latency measurements.
- The combination of mobile UEs and fixed mmWave infrastructure provides a balanced testbed for evaluating both eMBB and URLLC scenarios. This hybrid topology underpins comprehensive performance analysis.

Network Configuration in NS-3

- **Node Creation and Mobility Setup:** Define 20 UE nodes and assign a RandomWalk2d model within a 50×50m area:

```
// Create UE nodes
NodeContainer ueNodes;
ueNodes.Create(20);

// Configure random walk mobility
MobilityHelper mobility;
mobility.SetMobilityModel("ns3::RandomWalk2dMobilityModel",
    "Bounds", RectangleValue(Rectangle(0,50,0,50)),
    "Speed", StringValue("ns3::UniformRandomVariable[Min=1.0|Max=5.0]"));
mobility.Install(ueNodes);
```

- **Router and gNodeB Positioning:** Place 6 router nodes in a grid and keep them stationary, then connect a gNodeB similarly:

```
// Create router nodes
NodeContainer routerNodes;
routerNodes.Create(6);

// Position routers in 2x3 layout
auto posAlloc = CreateObject<ListPositionAllocator>();
for (uint32_t i = 0; i < 6; ++i) {
  posAlloc->Add(Vector(20 + 20 * (i % 3),
                       20 + 20 * (i / 3),
                       0));
}

mobility.SetPositionAllocator(posAlloc);
mobility.SetMobilityModel("ns3::ConstantPositionMobilityModel");
mobility.Install(routerNodes);
```

LatencyTag Class in NS-3

- **Purpose:** Attaches QoS metadata (latency requirement L_r, packet rate R_p, throughput Θ_r) to each packet. Ensures packet-level QoS parameters are accurately considered in slice decisions.
- **Key Methods:**

```
class LatencyTag : public Tag {
public:
  static TypeId GetTypeId() {
    static TypeId tid = TypeId("LatencyTag")
      .SetParent<Tag>()
      .AddConstructor<LatencyTag>();
    return tid;
  }
  void Serialize(TagBuffer buffer) const {
    buffer.WriteDouble(m_latency);
    buffer.WriteDouble(m_rate);
    buffer.WriteDouble(m_throughput);
  }
  void Deserialize(TagBuffer buffer) {
    m_latency = buffer.ReadDouble();
    m_rate = buffer.ReadDouble();
    m_throughput = buffer.ReadDouble();
  }
};
```

- **Realism:** Embeds real packet requirements into NS-3 packets, avoiding synthetic uniform assumptions.

VirtualSlice Class in NS-3

- **Purpose:** Manages slice resources (bandwidth B_s) and queue capacity Q_{max}, tracks performance metrics. Mirrors real router behavior under load with queue limits and drops.
- **Key Methods:**

```
class VirtualSlice {
public:
  bool Enqueue(Ptr<Packet> p) {
    if (bufferedBytes + p->GetSize() > maxQueueBytes) {
      ++droppedCount; return false;
    }
    buffer.push(p);
    bufferedBytes += p->GetSize();
    ++enqueuedCount;
    return true;
  }
  void Resize(uint32_t newBW) {
    bandwidth = newBW;
    // Normalize if total > budget
  }
};
```

- **Realism:** Dynamic resizing and drop tracking emulate real queue management in SDN switches.

PacketGeneratorApp in NS-3

- **Purpose:** Generates slice-specific workload with realistic packet sizes and inter-arrival times. Simulates user/application behavior under varying traffic.
- **Key Code:**

```
void PacketGeneratorApp::GeneratePacket() {
  double size = random->GetValue(minSize, maxSize);
  Ptr<Packet> packet = Create<Packet>(size);
  LatencyTag tag;
  tag.SetLatency(latencyRequirement);
  packet->AddPacketTag(tag);
  socket->Send(packet);
  ScheduleNext();
}
```

- **Realism:** Uses Exponential random for inter-arrival to reflect burstiness and idle periods.

SDNController Class in NS-3

- **Purpose:** Central decision logic that classifies and allocates slice bandwidth per packet. Core of dynamic slice orchestration.
- **Classification Logic:**

```
double scores[3];
for (int k = 0; k < 3; ++k) {
  scores[k] = 0.3 * f_sp(pSize,k)
            + 0.4 * f_lr(latencyReq,k)
            + 0.2 * f_rp(pktRate,k)
            + 0.1 * f_theta(thetaReq,k);
}
int selected = max_element(scores);
slices[selected].Enqueue(packet);
```

- **Realism:** Weighted multi-metric routing mimics advanced QoS schedulers in SDN.

PacketReceiverApp in NS-3

- **Purpose:** Receives packets at server, computes actual end-to-end latency including processing delay. Critical for precise reward calculation.
- **Key Snippet:**

```
socket->SetRecvCallback(
  MakeCallback(&PacketReceiverApp::HandleRead,this));
...
void HandleRead(Ptr<Socket> s) {
  Ptr<Packet> p = s->Recv();
  TimestampTag tag;
  p->PeekPacketTag(tag);
  double sendTime = tag.GetTimestamp();
  double recvTime = Now().GetSeconds();
  double latency = recvTime - sendTime;
  LogMetrics(latency);
}
```

- **Realism:** Incorporates timestamp tags and processing overhead to reflect real network conditions.

Reinforcement Learning Terminology

- **State (s):** Description of the network at time t, including queue occupancy, allocated bandwidths, and packet attributes. Serves as input for the RL agent to decide actions.
- **Action (a):** Discrete bandwidth adjustments $\Delta B \in \{-10, 0, 10\}$ Mbps per slice. Determines how resources are reallocated in response to state.
- **Reward (r):** Scalar feedback, $+1$ for meeting latency requirements, negative penalty otherwise, with an additional -10 for dropped packets. Drives the agent toward optimal QoS compliance.
- **Episode:** Sequence of 2000 packet events over a 10 s simulation, representing one training cycle. Allows the agent to learn across diverse network conditions.
- **Exploration Strategy:** Epsilon-greedy with ϵ decaying from 1.0 to 0.01. Balances exploration of new actions with exploitation of learned policy.

Data Transformation and Preprocessing

- **Feature Normalization:** Scale continuous features (packet size, latency requirement, packet rate, throughput) to [0,1]. Ensures homogeneous input scales for neural network.
- **Categorical Encoding:** Convert slice type into one-hot vector (eMBB, URLLC, mMTC). Enables network to distinguish slice context.
- **Sequential Data Handling:** Maintain temporal order of events, no random shuffling. Preserves network dynamics during training.
- **Replay Buffer:** Store up to 10,000 transitions, sample mini-batches of 64 for updates. Breaks correlation between sequential samples.

DQN Neural Network Architecture

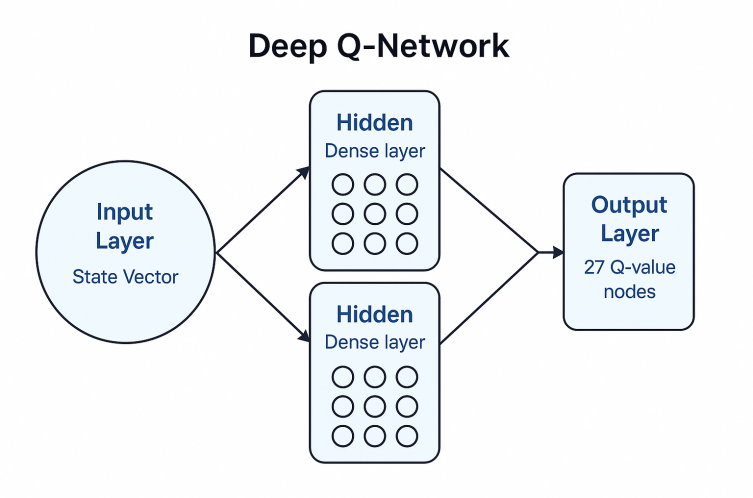


Figure: Input layer (state vector) $\rightarrow$ two hidden ReLU layers (64 units) $\rightarrow$ output layer (27 Q-values)

- Two hidden layers of 64 ReLU neurons each for non-linear feature learning. Balances model complexity and inference speed.
- Output layer with 27 units corresponding to discrete actions. Produces Q-values for each possible action.
- Target network update every 10 training steps to stabilize learning. Prevents divergence and oscillations.

DQN Training: Hyperparameters and Epochs

- Optimizer: Adam with learning rate $= 0.001$.
- Discount factor $(\gamma) = 0.99$ for long-term reward emphasis.
- Epsilon-greedy schedule: $1.0 \rightarrow 0.01$ with decay rate 0.995 per episode.
- Replay buffer size: 10,000; mini-batch size: 64.
- Total episodes: 80 (20 minutes training time); convergence around episode 50.
- Model saved as **dqn_network_slicing.h5** for reintegration.

Reward Convergence Over Episodes

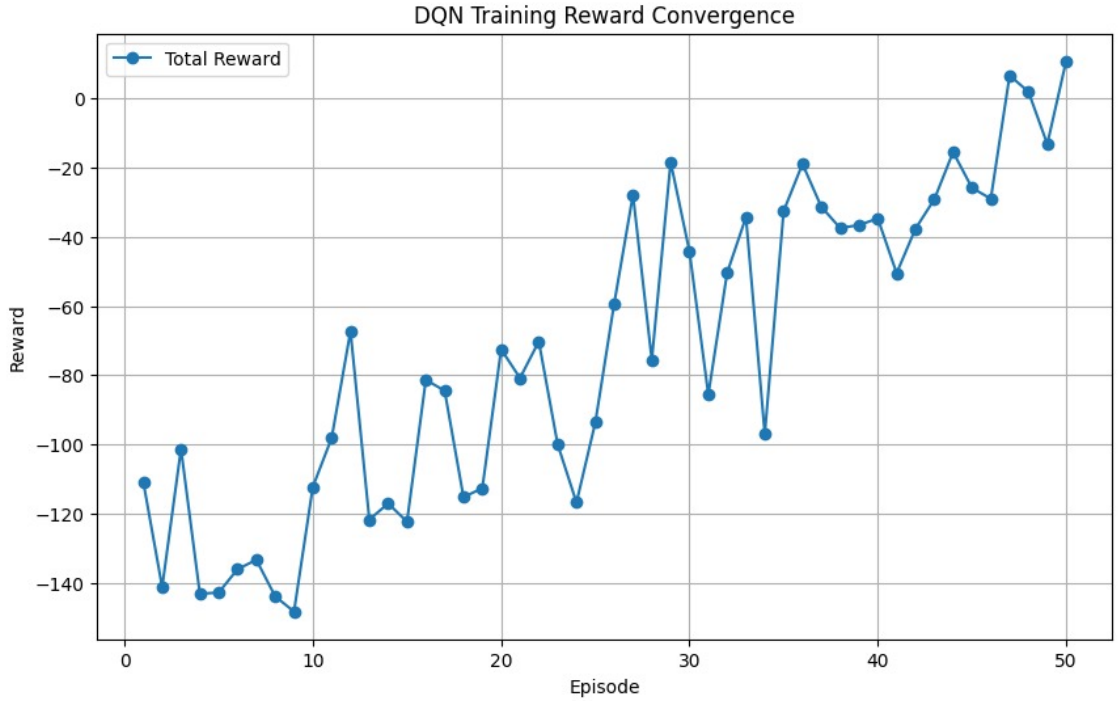


Figure: Average episodic reward increasing steadily, indicating effective learning

Integration of DQN Model via PyBind11

- **Python Binding Setup:** NS-3 embeds a Python interpreter to load the trained model:

```
#include <pybind11/embed.h>
using namespace pybind11::literals;
py::scoped_interpreter guard{};
py::object dqn = py::module::import("tensorflow.keras.models");
.attr("load_model")("dqn_network_slicing.h5");
```

- **Live Inference Loop:** Every 0.5s, extract current state vector and call:

```
std::vector<double> state = controller.GetState();
py::array_t<double> input(state.size(), state.data());
int action = dqn.attr("predict")(input).argmax();
controller.ApplyAction(action);
```

- **SDN Logic Change:** Replaces static heuristic in `MonitorAndAdjust` with DQN-driven `ApplyAction`, enabling adaptive slice resizing.

MDP Components for DQN-based Network Slicing

States:

- s_1 **Balanced Load:** All slices (eMBB, URLLC, mMTC) roughly at target utilization.
- s_2 **URLLC Congested:** URLLC queue buildup causes exceedance of latency thresholds.
- s_3 **eMBB Dominant:** High-volume eMBB traffic; URLLC/mMTC underutilized.
- s_4 **mMTC Overload:** Burst IoT traffic leads to packet drops in mMTC slice.

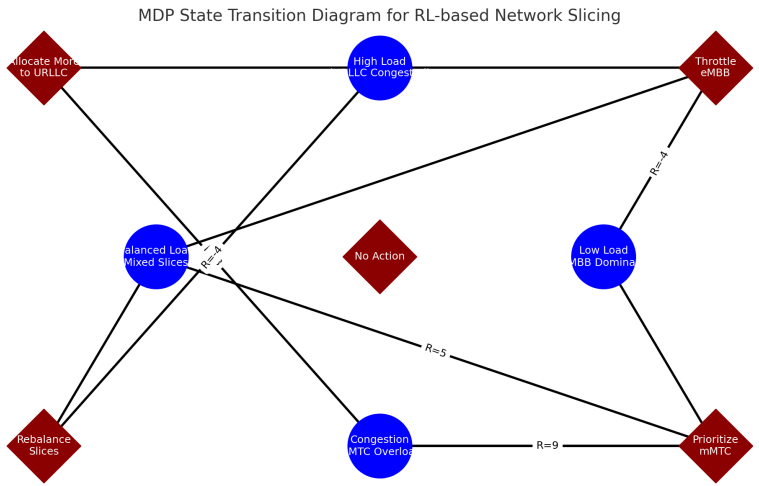


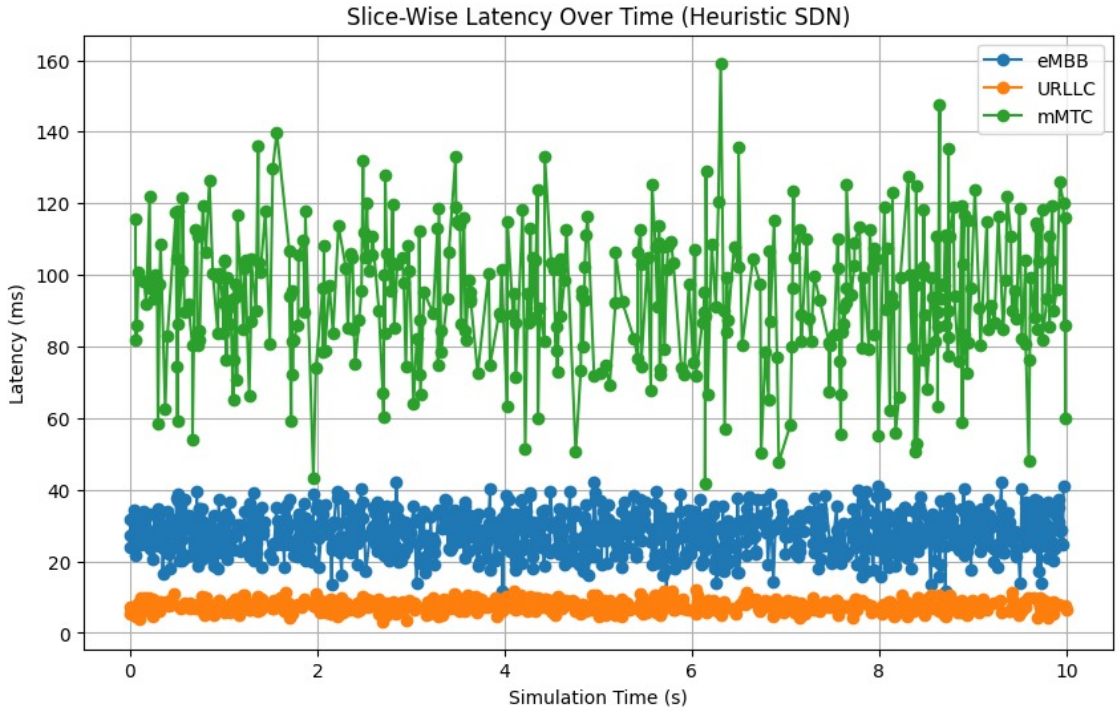
Figure: MDP: states (blue), actions (red diamonds with rewards), arrows show transitions.

Sample of Dataset from ns-3

Time	Packet_ID	PacketSize	LatencyRe	ActualLate	Slice	Enqueued	Dropped	Slice_BW	Queue_Occupancy
0.002596	0	19636	0.349933	0.002596	eMBB	1	0	50	0.01
0.07165	1	24051	0.496923	0.002955	eMBB	1	0	50	0.02
0.101023	2	253	0.000882	0.001023	URLLC	1	0	30	0.01
0.103214	3	72	0.000432	0.001008	URLLC	1	0	30	0.02
0.105818	4	388	0.000736	0.001033	URLLC	1	0	30	0.03
0.119753	5	232	0.000595	0.001021	URLLC	1	0	30	0.04
0.122586	6	305	0.000974	0.001027	URLLC	1	0	30	0.05
0.128512	7	345	0.000191	0.00103	URLLC	1	0	30	0.06
0.132437	8	130	0.000315	0.001013	URLLC	1	0	30	0.07
0.1428	9	34303	0.702752	0.003787	eMBB	1	0	50	0.03
0.167152	10	458	0.000615	0.001039	URLLC	1	0	30	0.08
0.168942	11	113	0.000854	0.001011	URLLC	1	0	30	0.09
0.173329	12	10304	0.862914	0.001837	eMBB	1	0	50	0.04
0.176782	13	112	0.000837	0.001011	URLLC	1	0	30	0.1
0.185027	14	65	0.000212	0.001008	URLLC	1	0	30	0.11
0.186402	15	222	0.000235	0.00102	URLLC	1	0	30	0.12
0.188805	16	448	0.000559	0.001038	URLLC	1	0	30	0.13
0.194566	17	156	0.00048	0.001015	URLLC	1	0	30	0.14
0.199018	18	301	0.000993	0.001026	URLLC	1	0	30	0.15
0.199789	19	239	0.000789	0.001022	URLLC	1	0	30	0.16
0.201044	20	525	6.08167	0.001044	mMTC	1	0	20	0.01
0.201901	21	432	0.000752	0.001037	URLLC	1	0	30	0.17
0.204938	22	78	0.000367	0.001009	URLLC	1	0	30	0.18

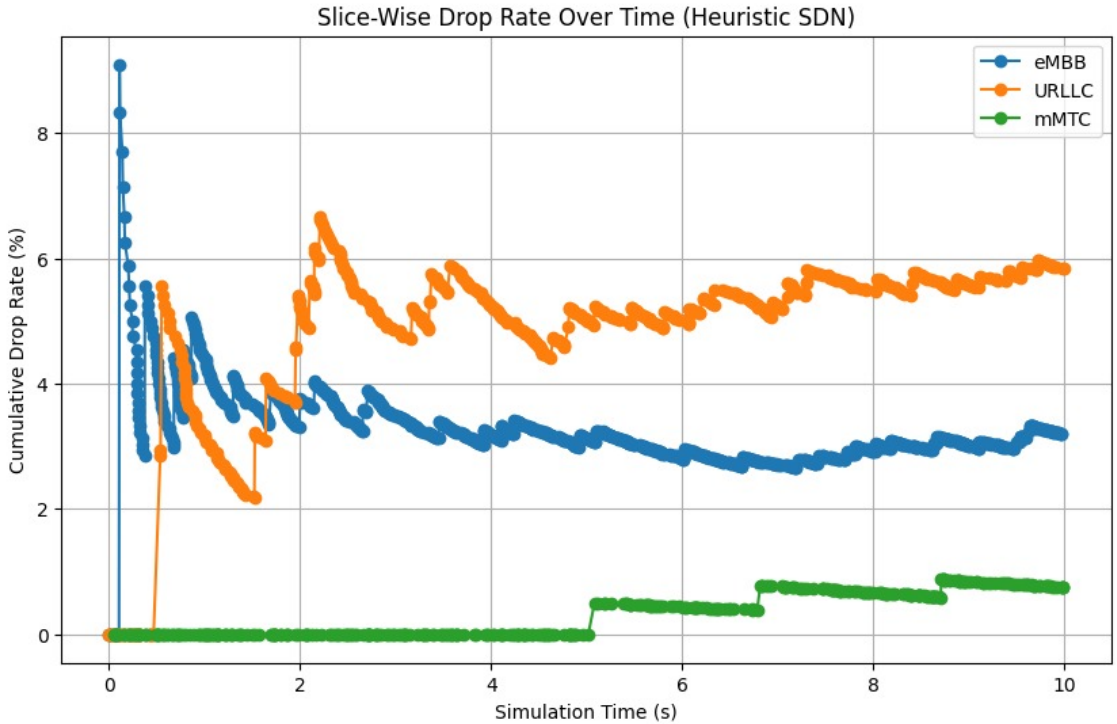
- This data set is generated as heuristic one from the ns-3 data base for training of RL model and analysis

Baseline Heuristic SDN: Slice-wise Latency



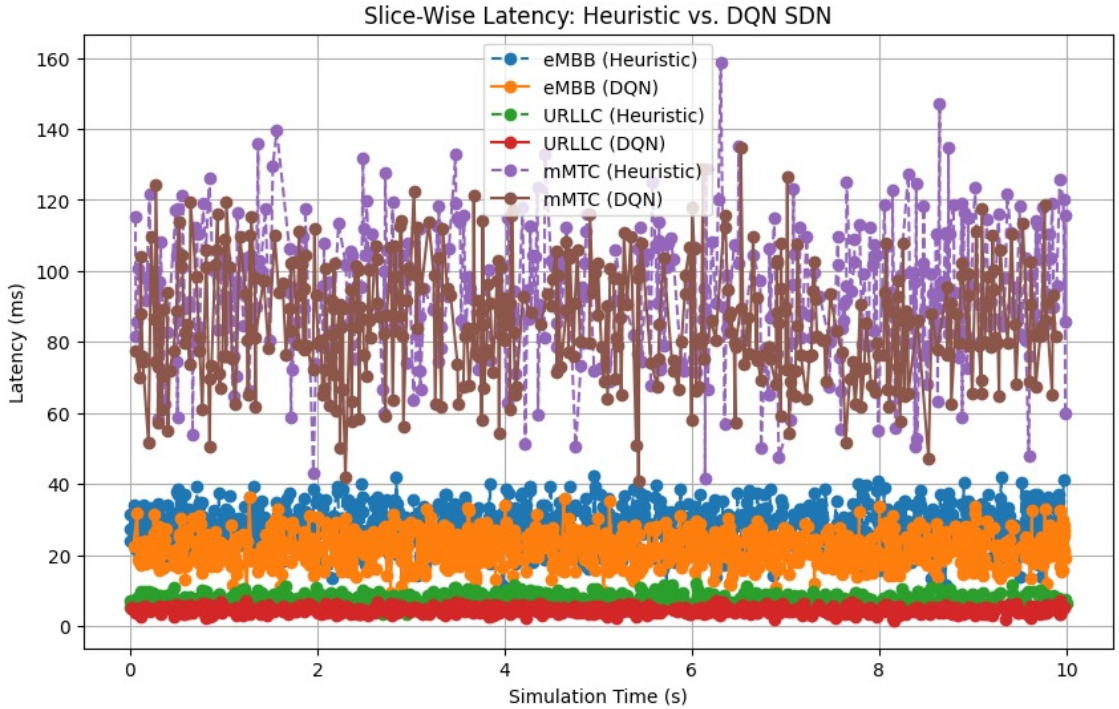
- URLLC latency peaks at 10ms under congestion, violating target 1ms.
- eMBB and mMTC show stable but suboptimal delays (28ms and 95ms respectively).

Baseline Heuristic SDN: Slice-wise Drop Rate



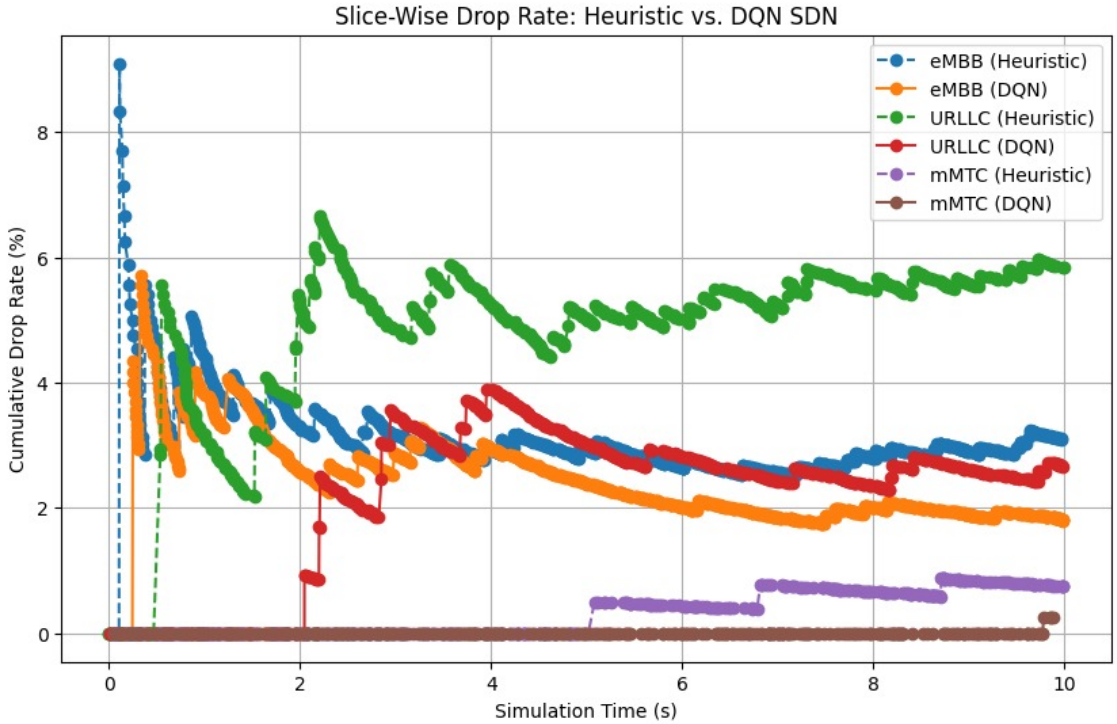
- URLLC drop rate reaches 6%, risking service reliability for critical applications.
- eMBB and mMTC maintain lower drop rates (3% and 0.8% respectively).

DQN-Based SDN: Latency Improvement



- URLLC latency reduced by 40% (from 7.5ms to 4.5ms).
- eMBB and mMTC latency improved by 21% and 11%, respectively.

DQN-Based SDN: Drop Rate Reduction



- URLLC drop rate reduced by 60% (from 5.8% to 2.3%).
- eMBB and mMTC drop rates improved by 38% and 31%, respectively.

Insights and Takeaways

- DQN-driven slicing adapts bandwidth allocations in real time, significantly enhancing QoS for latency-sensitive slices.
- The integration approach via PyBind11 enables seamless model inference with minimal simulation overhead.
- Interpretability through per-step logging provides transparency and aids operator trust in AI-driven decisions.
- The framework is extensible to other RL algorithms (e.g., DDPG, SAC) and real-world 6G testbeds for further validation.

Bibliography (1/4)

- R. Chataut, M. Nankya, and R. Akl, "6G Networks and the AI Revolution—Exploring Technologies, Applications, and Emerging Challenges," *Sensors*, vol. 24, no. 6, p. 1888, 2024.
- W. Wu *et al.*, "AI-native Network Slicing for 6G Networks," *IEEE Wireless Communications*, vol. 29, no. 1, pp. 96–103, 2022.
- S. Kukliński, L. Tomaszewski, R. Kołakowski, and P. Chemouil, "6G-LEGO: A Framework for 6G Network Slices," *Journal of Communications and Networks*, vol. 23, no. 6, pp. 442–453, 2021.
- R. Dangi and P. Lalwani, "Optimizing Network Slicing in 6G Networks Through a Hybrid Deep Learning Strategy," *Journal of Supercomputing*, vol. 80, pp. 20400–20420, 2024.

Bibliography (2/4)

- M. Zhao, Y. Zhang, Q. Liu, A. Kak, and N. Choi, "InSlicing: Interpretable Learning-Assisted Network Slice Configuration in Open RAN," arXiv:2502.15918, 2025.
- Y. Siriwardhana, P. Porambage, M. Liyanage, and M. Ylianttila, "AI and 6G Security: Opportunities and Challenges," in *2021 EuCNC/6G Summit*, 2021, pp. 616–621.
- R. M. Dhanasekaran, J. Ping, and G. P. Gomez, "End-to-End Network Slicing Security Across Standards Organizations," *IEEE Communications Standards Magazine*, vol. 7, no. 1, pp. 40–47, 2023.
- B. Chafika, T. Taleb, C.-T. Phan, C. Tselios, and G. Tsolis, "Distributed AI-Based Security for Massive Numbers of Network Slices in 5G Beyond Mobile Systems," in *2021 EuCNC/6G Summit*, 2021, pp. 401–406.

Bibliography (3/4)

- N. Toumi and T. Dimitrovski, "AI-native Architecture for 6G Networks and Services with Model Dependencies," in *2024 EuCNC/6G Summit*, 2024, pp. 901–906.
- N. F. Mir, "AI-Assisted Edge Computing for Multi-Tenant Management of Edge Devices in 6G Networks," in *2023 6GNet*, 2023, pp. 1–3.
- A. Alnoman, A. S. Khwaja, A. Anpalagan, and I. Woungang, "Emerging AI and 6G-Based User Localization Technologies for Emergencies and Disasters," *IEEE Access*, 2024.
- E. N. A. M. S. Chuan, H. Foh, and R. Tafazolli, "Strengthening Open RAN Security: Exploring Blockchain and AI/ML as Futuristic Safeguards," *IEEE Communications Magazine*, 2024 (preprint).

Bibliography (4/4)

- E. Obiodu, K. Chowdhury, L. Kundu, and X. Lin, "Boosting AI-Driven Innovation in 6G with the AI-RAN Alliance, 3GPP, and O-RAN," online blog, Jul. 2024.
- A. M. Alwakeel and A. K. Alnaim, "Network Slicing in 6G: A Strategic Framework for IoT in Smart Cities," *Sensors*, vol. 24, no. 13, p. 4254, 2024.
- H. S. Hamza and M. E. Manaa, "The Importance of Network Slicing and Optimization in Virtualized Cloud RAN," in *BICITS 2021*, 2021, pp. 245–250.
- R. K. Vaka and A. R. Shankar, "Implementation and Analysis of Wi-Fi Network Slices Based on 5G Network Slicing," in *ICWITE 2022*, 2022, pp. 1–6.

THANK YOU!