INTERNSHIP REPORT

Dynamic Radar Task Scheduling using Reinforcement Learning

Electronics & Radar Development Establishment (LRDE), DRDO

Submitted by

Adyasha Mahanta
Ojusav Agarwal

BTech
Computer Science and Engineering
Manipal University, Jaipur

Guided By

Dr. Dyana A
Scientist 'E', LRDE, DRDO

JUNE 2025

ACKNOWLEDGMENTS

First and foremost, we wish to express our sincere gratitude to our esteemed guide, **Sc. Dyana A.**, who has contributed significantly to the successful completion of our Industrial Training through her thoughtful reviews and valuable guidance.

Next, I would like to extend my sincere thanks to **Dr. Neha Chaudhary (Head of Computer Science and Engineering Department), Department Industrial Training Coordinator and faculty members** whose assistance was sought during the training work for their co-operation and encouragement.

Adyasha Mahanta
Ojusav Agarwal

ABSTRACT

Modern radar systems are required to handle a variety of dynamic and time-sensitive tasks, often under strict constraints such as limited power, radar availability, and tight deadlines. Efficient scheduling of these tasks is essential to maximize the utility of radar resources and ensure timely execution. This internship project focused on solving the problem of dynamic task scheduling using reinforcement learning techniques, with an emphasis on adaptability and real-time decision-making.

Three distinct approaches were explored and implemented. The first involved the use of Q-learning, a tabular reinforcement learning method that learns optimal task-radar assignments by exploring discrete state-action spaces. While effective for smaller problems, its scalability becomes limited as complexity grows. To address this, a second approach employed a Deep Q-Network (DQN), which uses neural networks to approximate the Q-values and supports more generalized learning across complex environments. Key features like experience replay and target networks were incorporated to stabilize training and improve learning efficiency.

In addition to these, a custom scenario-based scheduler was developed to handle specific edge cases and integrate heuristic rules with learned policies, ensuring robustness in constrained or exceptional conditions. All three models were evaluated in a simulated environment designed to mimic real-world radar task scheduling challenges.

The results showed that the DQN-based scheduler consistently performed well in terms of reward optimization, radar utilization, and task completion rate. This project demonstrates the significant potential of reinforcement learning in managing real-time radar resources and lays the groundwork for further advancements in adaptive scheduling systems.

Contents

1. INTRODUCTION

With the rapid evolution of modern warfare and surveillance technologies, radar systems are expected to perform increasingly complex functions across dynamic and uncertain environments. In particular, **Airborne and Spaceborne radar** platforms are tasked with detecting, tracking, and classifying multiple targets while operating under stringent constraints such as limited energy, spectrum, processing time, and concurrent task demands. As these platforms become more intelligent and multifunctional, the traditional static or rule-based scheduling approaches often fail to offer the responsiveness and adaptability required in real-time operational scenarios.

In this context, the field of **Cognitive Radar** has emerged as a transformative paradigm that envisions radars capable of perception, learning, and decision-making. Cognitive radars can observe their environment, interpret task priorities, and autonomously optimize their behaviour over time—thereby enhancing mission effectiveness and resource efficiency. One of the key pillars of cognitive radar is the ability to **dynamically schedule radar tasks in real time** while accounting for platform limitations and operational goals.

This project aimed to address this scheduling problem using **Reinforcement learning (RL)**. Through learning-based methods such as Q-learning and Deep Q-Networks (DQN), and by simulating custom scenarios, the goal was to build intelligent scheduling agents that can adaptively manage radar tasks under realistic mission constraints.

1.1. Motivation

Traditional radar scheduling systems often rely on predefined heuristics or rule-based logic, which struggle to cope with highly dynamic environments and conflicting resource demands. With the increasing complexity of radar missions—particularly in airborne and spaceborne applications—there is a growing need for systems that can **autonomously learn and adapt** their scheduling strategies in real time.

The motivation behind this project lies in bridging this gap through **reinforcement learning**, where the radar learns to assign tasks based on trial-and-error interactions with its environment. This aligns directly with the goals of **Cognitive Radar**, which seeks to make radar systems more autonomous, intelligent, and capable of decision-making under uncertainty.

By leveraging RL-based techniques, the project seeks to reduce task conflicts, improve radar utilization, and prioritize mission-critical operations. The ability to simulate various constraints—such as radar availability, task deadlines, power limits, and delay tolerance—makes this approach versatile and scalable to future cognitive radar systems.

2. LITERATURE REVIEW

The field of reinforcement learning (RL) has seen substantial theoretical and practical advancements over the last few decades. The foundational work by Sutton and Barto (2018) provides a comprehensive introduction to the principles of RL, including value-based learning, policy optimization, exploration-exploitation trade-offs, and temporal difference methods [1]. This theoretical grounding laid the basis for the early stages of this project, especially in implementing tabular Q-learning.

As RL evolved, Deep Reinforcement Learning (DRL) emerged as a powerful technique to handle high-dimensional state-action spaces. The survey by Arulkumaran et al. (2017) offers an insightful overview of DRL architectures, such as Deep Q-Networks (DQN), and discusses their applications in robotics, games, and control systems [2]. These ideas became particularly relevant as the limitations of Q-learning—especially its lack of generalization became evident in radar scheduling contexts.

A more recent perspective on RL fundamentals by Ghasemi and Ebrahimi (2024) reinforced the understanding of modern advancements in model-free learning and policy iteration frameworks, aligning well with the problem of radar task allocation [3].

The survey by Hashmi et al. (2023) was pivotal in contextualizing RL within the domain of Radar Resource Management (RRM). This comprehensive review outlines how AI and RL are being integrated into radar functions, especially for adaptive scheduling, waveform selection, and electronic counter-countermeasures [4]. It provided the conceptual link between cognitive radar systems and the application of learning-based methods.

Building on this foundation, the work of Xu and Zhang (2020) became the basis for the initial implementation phase of this project. Their paper presents a model-free RL approach using Q-learning for dynamic task scheduling in multifunction radar networks [5]. Inspired by this, the Q-learning agent in this project was designed to assign radar tasks based on learned state-action values while considering task deadlines, durations, and resource constraints.

However, the execution of the Q-learning approach revealed challenges in scalability and learning efficiency. This motivated a transition to DQN, where neural networks were used to approximate value functions more effectively. The need for hard real-time constraint handling and task prioritization was also

explored through works like Jeauneau et al. (2014) [6], which helped in shaping the reward structure and evaluation metrics.

Furthermore, the concept of multi-stage interaction and adaptive behavior, as discussed in Wang et al. (2019) [7], informed the design of the custom scenario-based scheduler. This hybrid model was built to combine rule-based logic with learned policies for better handling of edge cases and mission-critical decisions.

Overall, this layered literature review guided the progressive refinement of the project—from basic Q-learning to DQN and finally to a more adaptive, scenario-driven scheduling framework—anchored firmly within the domain of Cognitive Radar Systems..

2.1. Outcome of Literature Review

The literature reviewed played a key role in shaping the trajectory of the project. The major outcomes and influences are summarized below:

1. Theoretical Foundation:
    a. Sutton and Barto (2018) provided the fundamental understanding of reinforcement learning, Q-learning, which formed the starting point for the project's initial implementation.

2. Transition to Deep Reinforcement Learning:
    a. Arulkumaran et al. (2017) highlighted the limitations of tabular methods and introduced Deep Q-Networks (DQN) as a scalable alternative.
    b. Ghasemi and Ebrahimi (2024) reinforced conceptual clarity on model-free learning and supported the decision to use function approximation for complex radar environments.

3. Application to Radar Systems:
    a. Hashmi et al. (2023) established the relevance of AI and RL in Radar Resource Management (RRM), aligning the project with the goals of cognitive radar systems.
    b. Xu and Zhang (2020) offered a replicable baseline using Q-learning for radar task scheduling, which was successfully implemented in the early phase of the project.

4. Enhancing Realism and Adaptability:

a. Jeauneau et al. (2014) introduced handling of hard real-time constraints and task prioritization, influencing reward design.
b. Wang et al. (2019) motivated the development of a hybrid, custom scenario-based scheduler to handle edge cases with adaptive logic.

The literature provided a clear roadmap from foundational theory to radar-specific implementations, guiding the evolution from Q-learning to DQN, and finally to a robust, scenario-aware scheduling framework.

2.2. Statement

Airborne and spaceborne radar systems must execute multiple, concurrent tasks—each with strict timing, power, and priority constraints—within a shared resource environment. Traditional static or rule-based scheduling approaches lack the adaptability to respond to dynamic task arrivals, conflicting priorities, and varying radar availability in real-time operations.

This project tackles the problem of dynamic task scheduling for multifunction radar systems by leveraging reinforcement learning (RL) to learn adaptive, priority-aware task allocation policies. Tasks differ in type, deadline, duration, delay tolerance, and power requirements, while radar resources are limited and shared. The objective is to maximize task throughput and radar utilization while respecting mission-critical constraints.

To address this, the project explores and evaluates three scheduling strategies: Q-learning, Deep Q-Networks (DQN), and a custom rule-augmented scenario, all developed within a radar-simulated environment modeled after real-world operational parameters encountered in airborne platforms.

2.3. Research Objectives

This project is driven by the following concrete objectives:

- **To design a radar task scheduling environment** that realistically simulates the operational context of airborne and spaceborne radar systems, incorporating task parameters such as request time, deadline, duration, power requirements, and radar channel availability.
- **To implement a Q-learning-based scheduling agent** that learns task allocation policies through trial-and-error interaction with the environment, using a discrete state-action formulation and a scalar reward signal based on task success and resource efficiency.

- **To address the scalability and generalization limitations of Q-learning** by implementing a Deep Q-Network (DQN) scheduler that approximates Q-values using neural networks and improves convergence using experience replay and target networks.
- **To develop a custom scenario-based scheduler** capable of handling edge cases—such as oversaturated task queues or low radar availability—by integrating heuristic rules with learned policies for robust and fail-safe decision-making.
- **To perform a comparative analysis** of the three approaches based on:
    - Task completion rate
    - Total reward
    - Radar utilization efficiency
    - Performance under varying task loads and constraints
- **To demonstrate the applicability of learning-based scheduling in cognitive radar systems**, with a focus on adaptability, autonomy, and real-time feasibility for airborne mission profiles.

3. METHODOLOGY

3.1. Synthetic Dataset Generation for Q-Learning and DQN

To initiate the task scheduling framework, a synthetic dataset was created to simulate dynamic arrival of radar tasks. The dataset captures diverse radar operations (Detection, Tracking, etc.) under constrained conditions like deadlines, power usage, and delays.

Task Format:

Each task T_i is structured as:

T_i = *(Target_ID, Task_ID, Task_Type, Radar_Type, Priority, Request_Time, Deadline, Duration, Init_Power, Max_Power, Max_Delay)*

Key Attributes:

- **Task types** simulate beam stages (e.g., a=search, b=detect, etc.).
- **Request time**, **deadline**, and **duration** reflect scheduling urgency.
- **Initial and max power** impose operational constraints.
- **Max delay** defines slack.

Dataset Creation:

- Random generation of task parameters within specified bounds.
- Multiple tasks allocated across 3 radars (A, B, C).
- Stored as CSV and read during environment initialization.

3.2. Environment Design (OpenAI Gym)

A **custom Gym environment** was implemented to simulate the radar task scheduler.

Environment Features:

- **Observation Space**: Encodes active tasks, current time step, radar statuses (idle/busy), and pending task queue.
- **Action Space**: Discrete actions representing assignment of a task to a radar, or no-op.
- **Step Logic**:
    - Advances time.
    - Updates radar and task statuses.
    - Applies reward and penalty based on task feasibility, constraints, and delay.

Constraints Enforced:

- Radar exclusivity: one task per radar at any time.
- Deadline and delay bounds.

- Power usage must stay within $[P_{\text{init}}, P_{\text{max}}]$

3.3. Q-Learning Implementation

The initial implementation used **tabular Q-learning** to learn radar-task assignment policies.

Key Details:

- **State Space**: Discrete encoding of radar availability and current task attributes.
- **Action Space**: Discrete choices mapping task-radar assignments.
- **Reward Design**:
    - Positive reward for successful scheduling within time and power constraints.
    - Penalty for constraint violations or idle steps.

Limitations Observed:

- Poor scalability with growing state space.
- Q-table sparsity led to slow learning and overfitting to seen tasks.
- Inability to generalize across varying task arrival sequences or unseen combinations.

3.4. Deep Q-Network (DQN) Implementation

To overcome the limitations of tabular learning, a **Deep Q-Network (DQN)** was implemented using **Stable-Baselines3**.

Environment Enhancements:

- Continuous observation space: Vectorized task and radar status.
- Valid action masking: Prevent illegal task-radar assignments.
- Replay buffer and target network used to stabilize learning.

Reward Function:

- Rewarded on-time task assignment, penalized deadline misses or radar overuse.
- Incorporated delay sensitivity and power usage within bounds.

Benefits Over Q-Learning:

- Generalized to unseen task sets and orderings.
- Converged faster and achieved higher task completion rates.

3.5. Dataset generation for Custom Scenario

As the task complexity increased, a new dataset was created to simulate a realistic radar beam sequencing scenario. This supported the modelling of radar revisits, beam update cycles, and sequential operations (Search → Detect → Track → Classify).

Each task T_i is structured as:

T_i = *(Target_ID, Task_ID, Task_Type, Radar_Type, Priority, Request_Time, Deadline, Duration, Init_Power, Max_Power, Max_Delay, Stage_Count)*

3.6. Custom Scenario-Based Scheduling

Moved to a **realistic radar scenario** incorporating domain-specific logic.

A new **custom Gym environment** was developed to model radar operations under the constraints of sequential beam logic and realistic revisit timing.

Environment Features:

- Tracks each target's progress through radar stages.
- Enforces sequential beam execution: Search → Detect → Track → Classify.
- Includes beam-specific execution counts.
    - 4 search hits
    - 4 detect hits
    - 5 tracking hits
    - Classification pass

DQN Agent Customization:

- Input space includes radar states, target progress, and timing constraints.
- Action space combines beam type, radar selection, and task assignment.
- Reward shaped to maximize task completion and penalize invalid sequences or delays.

3.7. Evaluation Metrics

For each approach (Q-learning, DQN, Custom-DQN), performance is evaluated on:

- **Task Completion Rate**: % of tasks completed before deadlines.
- **Average Delay**: Time difference between request and assignment.
- **Constraint Violations**: Count of power, delay, or radar conflicts.
- **Reward Evolution**: Cumulative reward per episode.

4. IMPLEMENTATION

4.1. Q-Learning Approach

4.1.1. Baseline Dataset for Q-Learning and DQN

The baseline synthetic dataset was generated to simulate radar task scheduling in a controlled environment suitable for discrete Q-learning and DQN algorithms. It abstracts the complexities of radar beam sequencing, focusing instead on time-bounded and resource-constrained task assignment to radars with defined capabilities. The dataset is designed to provide sufficient variety and noise in task parameters to enable reinforcement learning agents to learn efficient scheduling policies under typical operational conditions.

Task Data Format

Each task is represented as a structured tuple:

T_i = *(Target_ID, Task_ID, Task_Type, Radar_Type, Priority, Request_Time, Deadline, Duration, Init_Power, Max_Power, Max_Delay)*

Attribute Definitions

Field	Description
Target_ID	Identifier for the associated target. Multiple tasks may share the same target ID.
Task_ID	Unique identifier for each task.
Task_Type	Type of operation required. Values: Detection, Classification, Tracking.
Radar_Type	Radar platform assigned or compatible with the task. Chosen based on radar-task compatibility.
Priority	Integer between 1 (highest) and 5 (lowest), denoting urgency.
Request_Time	Time (in simulation units) when the task enters the system.
Deadline	Latest time by which the task must be completed.
Duration	Time required to execute the task.
Init_Power	Minimum power required to initiate the task.
Max_Power	Upper limit of power allowed for the task execution.
Max_Delay	Acceptable delay from **Request_Time**, calculated as: **Max_Delay = Request_Time - Duration**

Radar-Task Compatibility Constraints

Each radar is configured to support a subset of the possible task types, as follows:

Radar	Supported Task Types

Radar1	Detection, Classification
Radar2	Classification, Tracking
Radar3	Tracking, Detection

During task generation, the radar for each task is randomly chosen from the set of radars capable of performing the given task type.

Temporal Constraints

- **Task Generation**: Tasks are generated sequentially with a small random time increment between each request (simulating real-time streaming input).
- **Deadline Constraint**: Every task has a deadline calculated as:

  $$\textbf{\textit{Deadline = Request\_Time + Duration + Slack}}$$

  where slack is sampled from a uniform range [3,10] time units.
- **Max_Delay**: Represents the scheduling flexibility window and is calculated directly from the slack duration.

Resource Constraints

- **Power Constraint**: For each task, Max_Power is defined as:

  $$\textbf{\textit{Max\_Power = Init\_Power + }}\Delta\textbf{\textit{P, }}\Delta\textbf{\textit{P}}\in\textbf{\textit{[0,20]}}$$

  This ensures variability in allowable power usage for different tasks.
- **Duration Constraint**: Duration is sampled from a fixed range [2,8], representing variability in computational/operational effort required per task.

Priority-Based Scheduling Implication

While priority is not enforced explicitly in the dataset (i.e., it does not affect deadlines or durations directly), it is included to allow the RL agent to learn preferences or bias toward higher-priority tasks. In a more complex reward design, priority could be incorporated into the reward signal or action-value weighting.

Dataset Generation Summary

- **Number of Targets**: 10
- **Number of Tasks**: 100
- **Arrival Pattern**: Randomized but sequential with small time increments.
- **Radar Assignment**: Based on compatibility mapping.
- **Parameter Ranges**:
  - Priority: Integer in [1, 5]
  - Duration: Integer in [2, 8]
  - Init Power: Integer in [10, 50]

- Max Power: Init Power + [0, 20]
- Slack for Deadline: Integer in [3, 10]

Each attribute is generated under randomized but bounded distributions to simulate realistic and learnable scheduling behavior.

4.1.2. Environment Design for Q-Learning

The Q-learning environment simulates a radar task scheduling system operating under discrete constraints. Tasks are generated synthetically and are scheduled on a fixed set of radar platforms, each with task-specific capabilities. The environment enables tabular Q-learning through a reduced and discrete state-action representation, allowing for value iteration over feasible radar-task assignments.

State Space

The environment state is discretized to ensure tractable exploration using Q-tables.

State Representation :

Each state vector is constructed from:

1. **Task Features** (Discrete Buckets):
   - Priority level (1–5)
   - Time until deadline (binned)
   - Remaining slack (delay) (binned)
2. **Radar Availability** (Binary Flags):
   - Radar1: 0 or 1 (available or busy)
   - Radar2: 0 or 1
   - Radar3: 0 or 1
3. **Current Time** (Rounded/bucketed if used)

This results in a manageable, multi-dimensional discrete state space suitable for tabular learning.

Action Space

The action space is discrete and finite:

- Each action represents assigning a task to one of the available radars.
- Optionally includes a “no-op” or “skip” action to simulate idling.

Total number of actions:

- ***A = Num Radars + 1 (for skip/idle)***

Invalid actions (e.g., assigning a task to a radar that cannot handle the task type or is currently busy) are discouraged via negative rewards.

Task Constraints Modeled

For every task T_i, the following constraints must be checked:

1. **Radar Compatibility**: $Task\_Type \in Radar\_Capabilities$
2. **Timing Constraint**: $Start\_Time \leq Deadline \text{ and } ActualDelay \leq MaxDelay$
3. **Power Constraint**: $P_{init} \leq P_{assigned} \leq P_{max}$
4. **Radar Availability**: R_j *must be idle at the task's start time*

If any constraint is violated, the task assignment is deemed invalid.

Reward Strategy

The reward function is central to guiding the Q-learning agent toward optimal scheduling behavior.

Reward Components:

Condition	Reward
Valid assignment within constraints	**+1.0**
Invalid assignment (violates any constraint)	**-1.0**
Idle action (skip/no-op)	**-0.1**
Assignment after deadline	**-1.0**
Assignment near deadline (but valid)	**+0.5** (optional shaping)

This formulation ensures the agent is incentivized to:

- Prioritize valid, timely task assignments.
- Learn to avoid infeasible actions.
- Minimize idleness or unnecessary skips.

4.1.3. Training & Learning Strategy

- **Q-Table Initialization**: All state-action pairs initialized to 0.
- **Learning Update Rule**:

$$Q(s, a) \leftarrow Q(s, a) + \alpha \, [\, r + \gamma \max Q(s', a') - Q(s, a) \,]$$

  where:
  - α: learning rate
  - γ: discount factor
  - r: immediate reward
  - s′: next state
- **Exploration Policy**: ε-greedy
  - Selects random actions with probability **ε**, otherwise chooses the highest-value action.
  - ε, decays over time to shift from exploration to exploitation.

Episode and Environment Dynamics

- **Initialization**:
  - A fixed number of synthetic tasks are loaded.
  - All radars are initialized as idle.
  - Time starts from zero.
- **At Each Step**:
  - A task is fetched from the queue.
  - The agent selects an action (radar assignment or idle).
  - Environment checks constraints, updates radar status and time.
  - Reward is assigned.
  - Q-table is updated.
  - The next state is formed and returned.
- **Termination Conditions**:
  - All tasks are processed or skipped.
  - Simulation time exceeds maximum window (if defined).

Limitations of Q-Learning Environment

- **State Explosion**: As more task features are added or made continuous, Q-tables become unmanageable.
- **Lack of Generalization**: Tabular methods cannot extrapolate to unseen states.
- **Discrete Actions**: Cannot model nuanced radar-task interactions (e.g., partial allocation, load balancing).
- **No Sequential Dependency**: Tasks are atomic; beam sequence logic is not handled.

4.2. Deep Q-Network Approach

4.2.1. Dataset

We'll be using the same dataset used in Q-learning discussed is Section 4.1.1.

4.2.2. Environment Setup

To overcome the limitations of tabular Q-learning—particularly its inability to generalize across continuous or high-dimensional state spaces—a Deep Q-Network (DQN)-based scheduling agent was implemented using a modified OpenAI Gym environment. This environment simulates a multifunction radar task scheduling problem under resource, timing, and power constraints. The use of neural function approximation allows the agent to operate on vectorized state representations and learn scalable policies that can generalize to unseen task combinations.

The environment was integrated with the Stable-Baselines3 DQN implementation, enabling modular experimentation with reward strategies, policy architectures, and evaluation schemes.

State Space Design

Unlike the discrete state structure used in Q-learning, the DQN environment employs a continuous, vectorized state representation suitable for input into a neural network.

State Vector Features

Each observation (state) includes:

1. **Task Features** (for current or candidate task):
    - Priority (scaled between 0 and 1)
    - Time until deadline (normalized)
    - Duration (normalized)
    - Initial Power and Max Power (scaled)
    - Max Delay (scaled)
2. **Radar Features**:
    - Radar availability (binary or scaled time-to-free)
    - Radar-task compatibility (binary flags or learned embeddings)
3. **Global Features**:
    - Current time step (normalized to total horizon)
    - Task queue length (optional)
    - Time since last radar assignment (optional)

All numerical features are normalized to [0, 1] or standardized for effective training.

Action Space Design

The action space is defined as a flat discrete space:

- Each action represents the assignment of a task to a compatible radar.
- Optionally includes an "idle" or "no-op" action when no valid assignments exist or skipping is optimal.

For N tasks and M radars, the action space size is $N \times M+1$ *(with the +1 for idle).*

Invalid actions (e.g., assigning a task to an incompatible or busy radar) are filtered or penalized during training.

Constraints Enforced by Environment

Each action must satisfy several real-time operational constraints:

Constraint Type	**Condition**
Radar Compatibility	Task type must be supported by the selected radar.
Radar Availability	Radar must be idle at the current time.
Timing Feasibility	Current time + duration ≤ deadline.

Delay Constraint	Actual delay ≤ max_delay.
Power Constraint	Init_Power ≤ power ≤ Max_Power.

Invalid actions are handled by:

- Either masking them before sampling.
- Or returning negative rewards when selected (in case of naive exploration).

Reward Strategy

The reward function is designed to be dense and informative, providing gradient signals for both good and bad decisions.

Reward Components

Scenario	Reward Value
Valid assignment within constraints	+1.0
Valid assignment with delay near limit	+0.5
Idle step (no-op)	-0.05
Assignment violates constraints	-1.0
Deadline missed	-1.0
All tasks completed on time (final bonus)	+5.0 (optional)

Reward Shaping (Optional Enhancements)

- Penalty proportional to actual delay:

$$r = -0.1 \times \left(\frac{\text{Actual Delay}}{\text{Max Delay}} \right)$$

- Reward boost for high-priority tasks scheduled early.

This shaping encourages the agent to:

- Learn urgency-aware scheduling.
- Avoid idle or late execution.
- Prefer satisfying high-priority tasks.

4.2.3. Training and Learning Pipeline

RL Algorithm: DQN from Stable-Baselines3

Neural Network Policy Architecture

- **Input Layer**: Size equal to the length of the flattened state vector.
- **Hidden Layers**: 2 or 3 layers with 64–128 ReLU units (tunable).

- **Output Layer**: One output per action (Q-value estimate for each).

Learning Configuration

Parameter	**Value**
Learning Rate	1e-3 (default)
Discount Factor (γ)	0.99
Replay Buffer Size	100,000 transitions
Batch Size	64
Target Network Update	Every 500 steps
Exploration Strategy	ε-greedy with decay (ε from 1.0 to 0.05)

Exploration Strategy

- The agent begins with random exploration (ε = 1.0).
- ε decays linearly or exponentially to 0.05.
- Ensures initial coverage and later exploitation.

Episode and Simulation Logic

- **Initialization**:
  - Environment loads a list of radar tasks.
  - All radars are initialized as idle.
  - Current time is set to 0.
- **Step Function**:
  - Agent selects an action (radar-task assignment).
  - Environment checks feasibility and updates radar state.
  - If valid, task is marked as scheduled; radar becomes busy for duration.
  - Time advances.
  - Reward is computed.
  - New state is returned.
- **Termination Conditions**:
  - All tasks are scheduled or rejected.
  - Simulation reaches a predefined time horizon.

Key Environment Advantages Over Q-Learning

Feature	Q-Learning	DQN
State Representation	Discrete (hand-coded)	Continuous (learned features)
Generalization	Poor	High
Action Masking	Basic	Advanced
Sequential Learning	Limited	Flexible
Scaling to large problems	Infeasible	Achievable

4.3. Custom Scenario DQN Approach

4.3.1. Realistic Dataset for Custom Scenario DQN

This dataset models a realistic and progressive radar task scheduling environment suitable for training Deep Q-Network (DQN) agents in a constrained and dynamic setting. Unlike the baseline dataset where tasks are independent, this dataset emphasizes sequential radar operations per target, time-sensitive execution, radar-task compatibility, and resource-bounded planning. It simulates real-world operational behavior of a multifunction radar system, enabling learning agents to schedule tasks across a time horizon with delayed task appearances, strict ordering, and infeasibility modeling.

Data Generation Configuration

- **Number of Unique Tracked Targets**: 50
- **Standalone Search Tasks**: 180
- **Must-Drop Tasks**: 50
- **Maximum Simulation Time**: ~180 minutes
- **Initial Request Time per Tracked Target**: Distributed over [5.0, 60.0] minutes
- **Average Time Between Tracked Target Tasks**: Randomized between 0.5 to 1.0 minutes
- **Standalone Search Request Times**: Evenly spread over [0.0, 180.0] minutes with ±0.5 jitter
- **Stage Progression**:

  Search → Detection → Tracking → Classification → Locked

  With the required counts to move to the next stage:
  - Search → 1 pass
  - Detection → 4 passes
  - Tracking → 5 passes
  - Classification → 6 passes

Task Tuple Format

Each task is represented as:

T_i = *(Target_ID, Task_ID, Task_Type, Radar_Type, Priority, Request_Time, Deadline, Duration, Init_Power, Max_Power, Max_Delay, Stage_Count)*

Attribute Definitions

Attribute	Description
Target_ID	Unique target ID (0 for standalone tasks).
Task_ID	Encoded as TargetID_TaskType_StagePass (e.g., 12_D3).
Task_Type	One of: Search, Detection, Tracking, Classification.
Radar_Type	Assigned radar from a compatible set.
Priority	Search = 1, Detection = 2, Tracking = 3, Classification = 4.
Request_Time	Time when task is available (floating-point, minutes).
Deadline	Latest completion time. Depends on slack and duration.
Duration	Task duration sampled from task-type-specific range.
Init_Power	Sampled from type-specific range.
Max_Power	Calculated as: Init_Power + random(5, 20).
Max_Delay	Derived from: Deadline - (Request + Duration).
Stage_Count	Number of times this stage has occurred for the target.

Radar-Task Compatibility

Radar	Supported Task Types
Radar1	Search, Detection, Classification
Radar2	Detection, Tracking, Classification
Radar3	Search, Detection, Tracking

Tasks are only assigned to radars that are capable of performing the required operation.

Task Timing Constraints

Deadline Configuration (Task-Type Dependent):

Task Type	Slack Range (minutes)
Search	[5.0, 10.0]
Detection	[4.0, 8.0]
Tracking	[3.0, 7.0]
Classification	[2.0, 5.0]

$$Deadline = Request\_Time + Duration + Offset$$

Duration (per task type):

Task Type	Range (minutes)
Search	**[1.0, 2.0]**
Detection	**[1.5, 3.0]**
Tracking	**[2.0, 4.5]**
Classification	**[3.0, 6.0]**

Power Constraints

Task Type	Init Power Range
Search	**10–25**
Detection	**25–40**
Tracking	**40–60**
Classification	**50–75**

- **Init Power**: Sampled fromabove ranges
- ***Max_Power = Init_Power + random (5,20)***

Delay Constraints by Priority

Priority	Task Type	Max Delay Multiplier Range
1	Search	[2.0, 4.0] × Duration
2	Detection	[1.0, 2.0] × Duration
3	Tracking	[0.5, 1.0] × Duration
4	Classification	[0.1, 0.5] × Duration

$$Max\_Delay = Duration \times Random\ Multiplier_{priority}$$

$$Max\_Delay = Deadline - Request\_Time - Duration$$

Always non-negative and reflects slack. Must-drop tasks have Max_Delay = 0.0.

This constraint ensures that higher-priority tasks (e.g., classification) are more urgent, with less flexibility in scheduling.

Must-Drop Task Configuration

To simulate infeasible or mission-critical tasks, ~ **5%** of the tasks are generated as **must-drop tasks**.

To model infeasible or mission-critical overload:

- **Total Must-Drop Tasks: 50**

- **Only applied to: Tracking and Classification**
- **Assignment Conditions:**
    - Task is randomly marked while generating tracked tasks.
- **Modifications:**
    - Deadline = Request_Time + Duration - Random (0.5, 2.0)
    - Duration: Unmodified
    - Init/Max Power: Standard range
    - Max_Delay: Set to 0.0
    - Forces the agent to learn avoidance behavior.

Standalone Search Tasks

- Target_ID = 0
- Task_Type = Search
- Count: 180
- Request Times: Uniformly spread across [0, 180] minutes
- Jitter: ±0.5 minutes
- Used to simulate background radar load

Dataset Summary

Property	Value
Tracked Targets	50
Must-Drop Tasks	50 (~5%)
Standalone Search Tasks	180
Max Simulation Time	~180 minutes
Task Types	4
Stage Progression	Enforced (S→D→T→C→Locked)
Stage Count Tracking	Yes
Radar Compatibility	Enforced
Deadline, Power, Delay	Type-dependent
Must-Drop Behavior	Yes (explicitly modeled)

4.3.2. Environment Setup

The custom scenario environment models a **realistic multifunction radar task scheduling system** using a custom OpenAI Gym-compatible interface. This environment incorporates several operational complexities such as:

- **Sequential beam operations per target** (Search → Detection → Tracking → Classification)
- **Radar-task compatibility constraints**
- **Beam revisit logic**
- **Stage-dependent task progression**
- **Dynamic task arrival and power/time constraints**

The environment supports reinforcement learning agents (specifically DQN) capable of learning adaptive scheduling policies in this high-fidelity scenario.

State Space Design

The state representation is structured, high-dimensional, and continuous, capturing both target-level task progression and radar status.

State Vector Components:

1. **Target State Features** (per active task):
    - Current stage (encoded as integer or one-hot)
    - Stage count (progress made in current stage)
    - Request time, deadline, duration
    - Remaining slack: ***Deadline = (Current Time + Duration)***
    - Init power and max power
    - Priority (from 1 to 4)
    - Whether target is already "Locked" (stage complete)
2. **Radar State Features**:
    - Radar availability (0 or 1)
    - Radar busy time remaining
    - Current radar orientation (azimuthal slip position)
    - Whether radar can cover the task's sector (if sector-based logic is implemented)
3. **Global Features**:
    - Current simulation time (normalized)
    - Number of active tasks
    - Task queue size

This full state vector is flattened and passed as input to the DQN agent.

Action Space Design

Each action represents a tuple: ***A=(Target ID, Radar ID, Task Type)***

In implementation, this is **flattened into a single discrete action index**, where:

- The environment internally maps each index back to (target, radar, task_type).

- Invalid combinations (e.g., assigning classification before search is complete) are dynamically masked or penalized.

Size of action space: $|A| = N_{targets} \times N_{radars} \times N_{task\_types}$

Beam Stage Progression Logic

Each target must progress through the following sequence:

Search (1 pass) → Detection (4 passes) → Tracking (5 passes) → Classification (6 passes)

- A task of a given type can only be scheduled if all required passes of the **previous type** are completed.
- Stage count is tracked per target and per stage.
- After classification is complete, the target is marked as **Locked** and removed from scheduling.

Constraint Enforcement

For each task assignment, the following constraints must be satisfied:

Constraint Type	**Condition**
Stage Feasibility	Required number of previous-stage passes completed.
Radar Capability	Radar must support the current task type.
Radar Availability	Radar must be idle.
Deadline	*Current Time + Duration ≤ Deadline*
Delay	*Actual Delay ≤ Max Delay*
Power	$P_{init} \leq P_{used} \leq P_{max}$

Assignments violating any of these constraints result in a penalty or ignored scheduling.

Reward Strategy

A multi-level reward structure is employed to guide the agent in this complex environment.

Reward Components:

Scenario	**Reward**
Stage pass (valid assignment)	**+1.0**
Completion of final stage (Locked)	**+5.0**
Valid assignment close to deadline	**+0.5**
Invalid assignment (any constraint violated)	**−1.0**
Assignment to wrong radar/task stage	**−1.0**
Idle step (no-op)	**−0.05**
Attempt to reassign Locked target	**−1.0**

Shaping Enhancements (Optional):

- Reward scaled by remaining delay margin:

$$r = +1.0 \times \left(1 - \frac{\text{Actual Delay}}{\text{Max Delay}}\right)$$

- Bonus for early-stage completions or meeting full-stage thresholds earlier than required.

4.3.3. Training Pipeline

Component	Details
RL Algorithm	DQN (Stable-Baselines3)
Policy Network	MLP with 2–3 hidden layers (64–128 units)
Replay Buffer	100,000 transitions
Target Network	Updated every 500 steps
Batch Size	64
Exploration	ε-greedy, ε decaying from 1.0 to 0.05
Learning Rate	1e-3
Discount Factor γ	0.99

The training proceeds episodically, where each episode simulates a full scheduling session over multiple targets and tasks.

Episode and Step Logic

1. **Initialization**:
    - All radars are idle.
    - Time starts at 0.
    - Task list is loaded and initialized.
2. **Step**:
    - Agent selects an action.
    - Environment checks feasibility.
    - Updates radar state, task queue, stage counts.
    - Advances time.
    - Computes reward and returns new state.
3. **Termination**:
    - All targets reach the Locked state, or
    - Simulation time exceeds maximum limit (e.g., 1500 units).

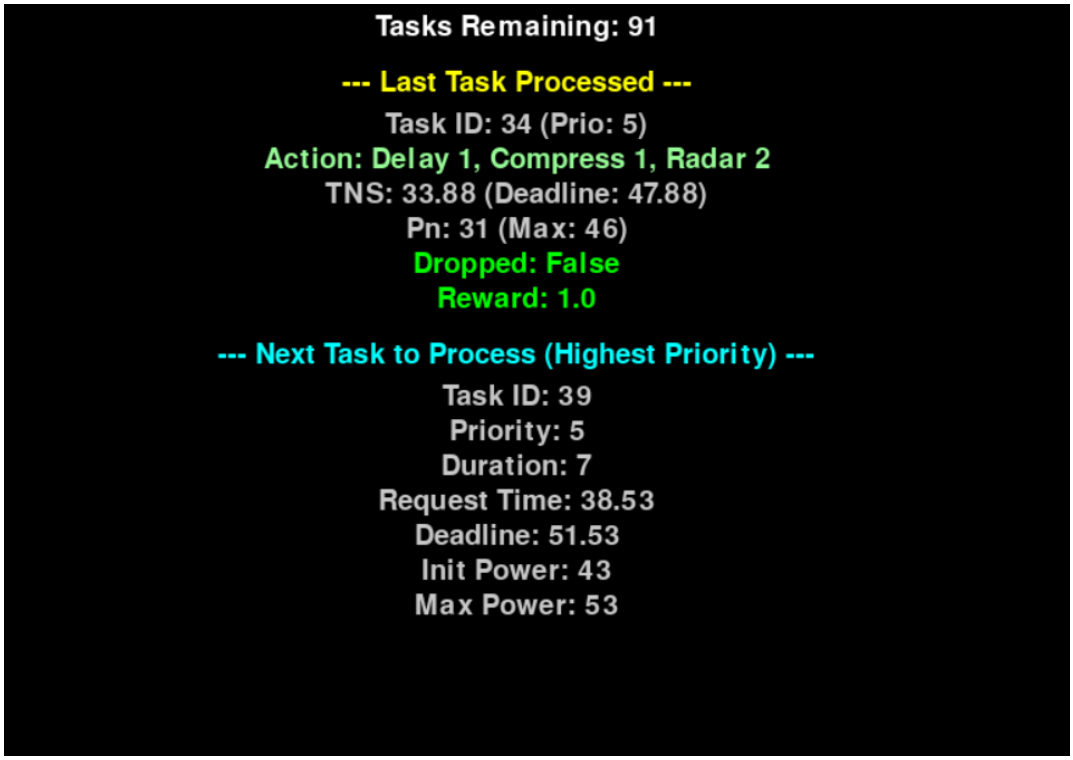
Advantages and Capabilities

Feature	Description
Sequential Task Logic	Supports multi-stage radar beam assignment per target.
Radar Constraints	Models radar-task compatibility and availability.
Beam Slip and Revisit	Simulates radar scanning constraints.
Dynamic Task Arrival	Tasks appear over time, not all at once.
Must-Drop Tasks	Injects infeasible, high-priority tasks to test policy robustness.
Action Masking	Only feasible actions allowed or heavily penalized.

5. RESULTS

5.1. Q-Learning Results

Training with rendering snapshot -



Final task scheduling of all the tasks, snapshot of few of the tasks scheduled:

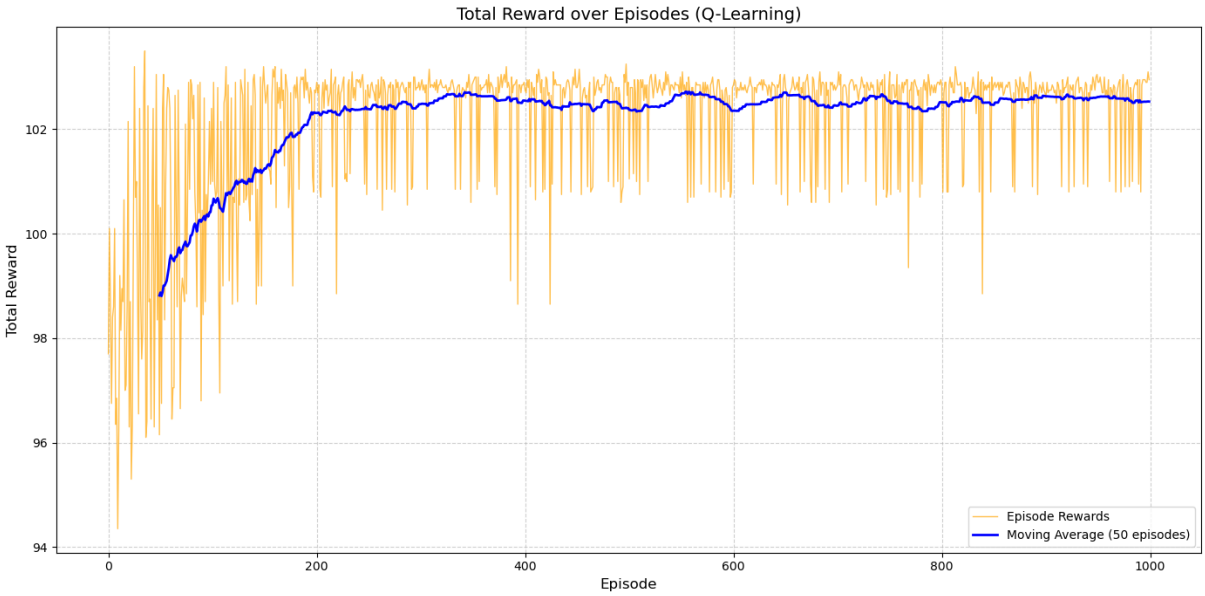
--- Final Task Schedule (from the last completed training episode) ---

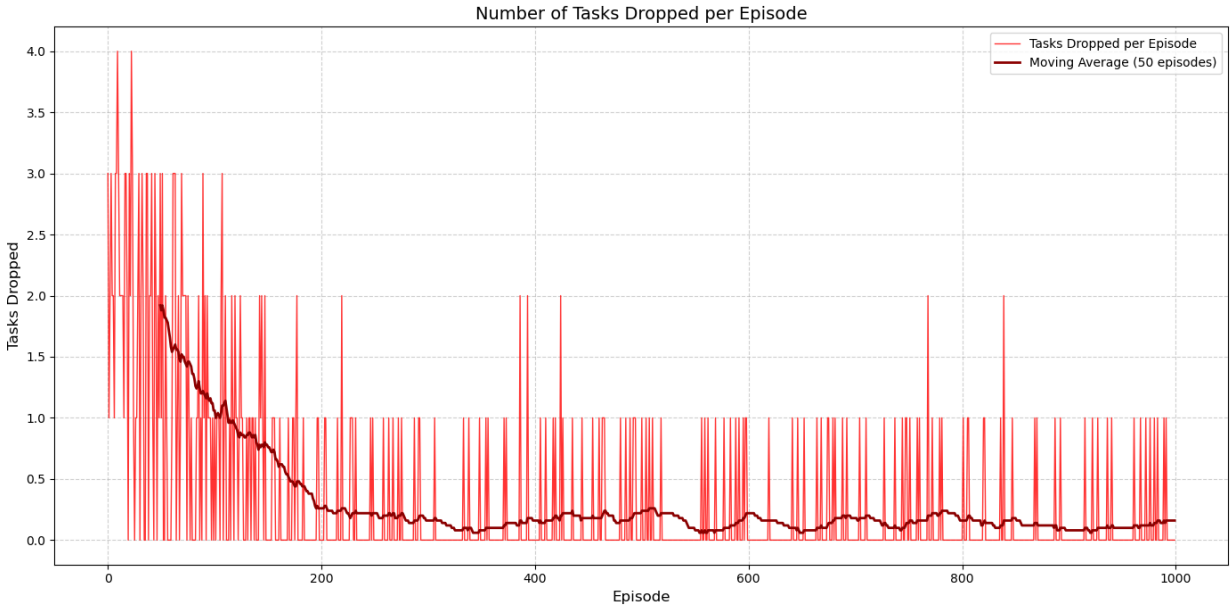
Task_ID	Priority	Action_Delay	Action_Compress	Action_Radar	Dropped	Final_TNS	Final_Pn	Original_Deadline	Original_Max_Power
1	5	1	0	2	False	2.34	28	15.34	33
3	5	0	1	1	False	3.86	32	15.86	51
7	5	1	1	1	False	8.24	14	15.24	17
9	5	0	0	0	False	9.19	13	17.19	32
14	5	0	0	0	False	14.17	46	26.17	54
17	5	2	2	1	False	19.20	34	25.20	38
19	5	0	2	0	False	19.00	18	31.00	22
31	5	2	1	2	False	32.17	51	45.17	63
34	5	1	2	2	False	33.88	32	47.88	46
39	5	0	2	1	False	38.53	45	51.53	53
48	5	0	0	0	False	49.06	40	56.06	40
50	5	1	2	0	False	52.28	23	68.28	24
61	5	1	2	0	False	62.75	19	68.75	35
62	5	0	0	1	False	62.49	11	72.49	28
63	5	0	2	0	False	63.46	12	73.46	24
87	5	2	1	2	False	88.97	21	97.97	28
97	5	1	0	2	False	97.60	10	111.60	23
99	5	0	0	1	False	98.11	28	113.11	46
100	5	1	0	1	False	99.69	33	106.69	40
6	4	0	0	1	False	6.61	18	20.61	33
10	4	1	1	0	False	11.47	28	15.47	41
11	4	2	1	1	False	13.22	12	19.22	20
12	4	0	1	2	False	11.81	24	21.81	30
41	4	0	0	1	False	40.78	42	50.78	48
45	4	1	0	0	False	46.75	45	55.75	48
47	4	0	1	1	False	47.95	43	54.95	52
49	4	0	2	0	False	50.50	29	66.50	30

Performance metric after task scheduled:

```
--- Performance Metrics ---
                            Metric   Value
           Total Tasks in Dataset      100
Tasks Attempted (in last episode)      100
     Tasks Successfully Completed      100
                    Tasks Dropped        0
    Percentage of Tasks Completed  100.00%
      Percentage of Tasks Dropped    0.00%
Environment closed.
```

Plot for reward and tasks dropped:





5.2. Results of Deep Q Network

Final task schedule after DQN Training:

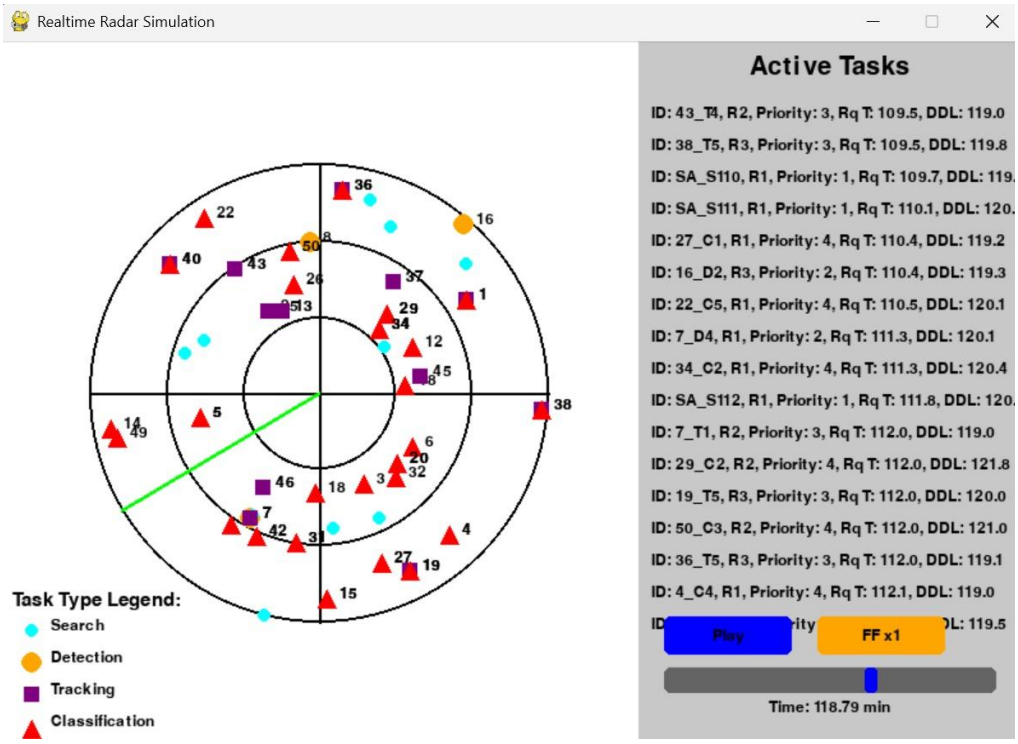
```
--- Final Task Schedule (from last evaluation) ---
 Task_ID  Priority  Action_Delay  Action_Compress  Action_Radar  Dropped  Final_TNS  Final_Pn  Original_Deadline  Original_Max_Power
       1         5             0                0             2    False       1.34        28              15.34                  33
       3         5             0                0             2    False       3.86        31              15.86                  51
       7         5             0                0             2    False       7.24        13              15.24                  17
       9         5             0                0             2    False       9.19        13              17.19                  32
      14         5             0                0             2    False      14.17        46              26.17                  54
      17         5             0                0             2    False      17.20        32              25.20                  38
      19         5             0                0             2    False      19.00        16              31.00                  22
      31         5             0                0             2    False      30.17        50              45.17                  63
      34         5             0                0             2    False      32.88        30              47.88                  46
      39         5             0                0             2    False      38.53        43              51.53                  53
      48         5             0                0             2    False      49.06        40              56.06                  40
      50         5             0                0             2    False      51.28        21              68.28                  24
      61         5             0                0             2    False      61.75        17              68.75                  35
      62         5             0                0             2    False      62.49        11              72.49                  28
      63         5             0                0             2    False      63.46        10              73.46                  24
      87         5             0                0             2    False      86.97        20              97.97                  28
      97         5             0                0             2    False      96.60        10             111.60                  23
      99         5             0                0             2    False      98.11        28             113.11                  46
     100         5             0                0             2    False      98.69        33             106.69                  40
       6         4             0                0             2    False       6.61        18              20.61                  33
      10         4             0                0             2    False      10.47        27              15.47                  41
      11         4             0                0             2    False      11.22        11              19.22                  20
      12         4             0                0             2    False      11.81        23              21.81                  30
      41         4             0                0             2    False      40.78        42              50.78                  48
      45         4             0                0             2    False      45.75        45              55.75                  48
      47         4             0                0             2    False      47.95        42              54.95                  52
      49         4             0                0             2    False      50.50        27              66.50                  30
```

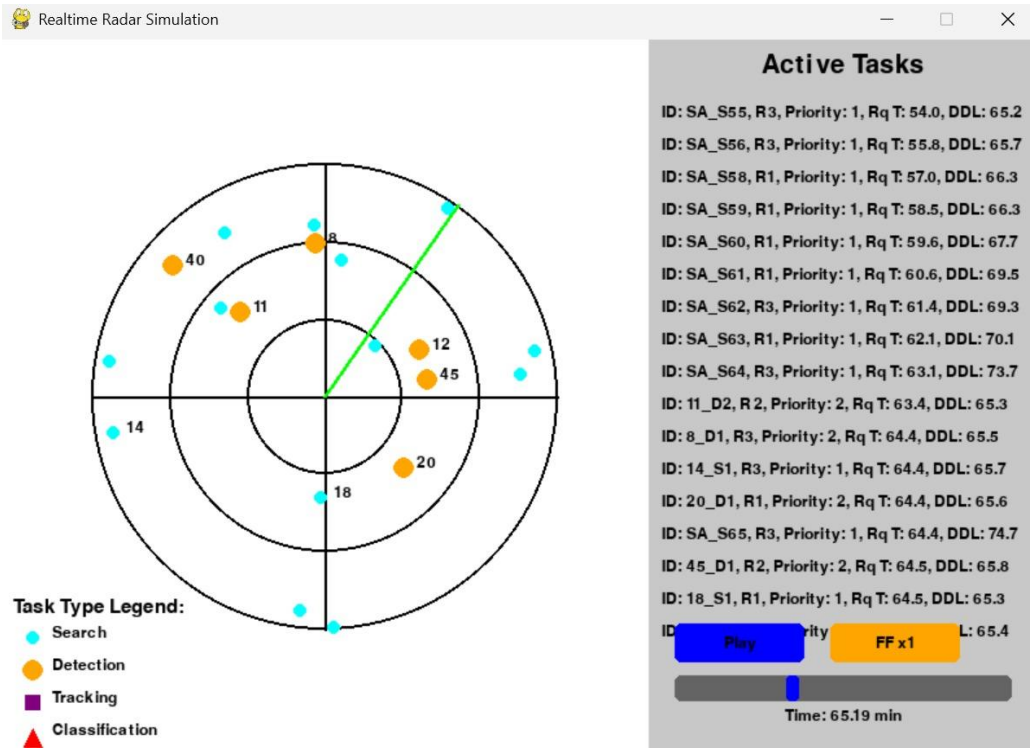
Evaluation metric after DQN Training:

```
--- Performance Metrics (from last evaluation) ---
                                Metric   Value
                  Total Tasks in Dataset     100
Tasks Attempted (in last evaluation)     100
           Tasks Successfully Completed     100
                          Tasks Dropped       0
         Percentage of Tasks Completed 100.00%
            Percentage of Tasks Dropped   0.00%
All environments closed.
```

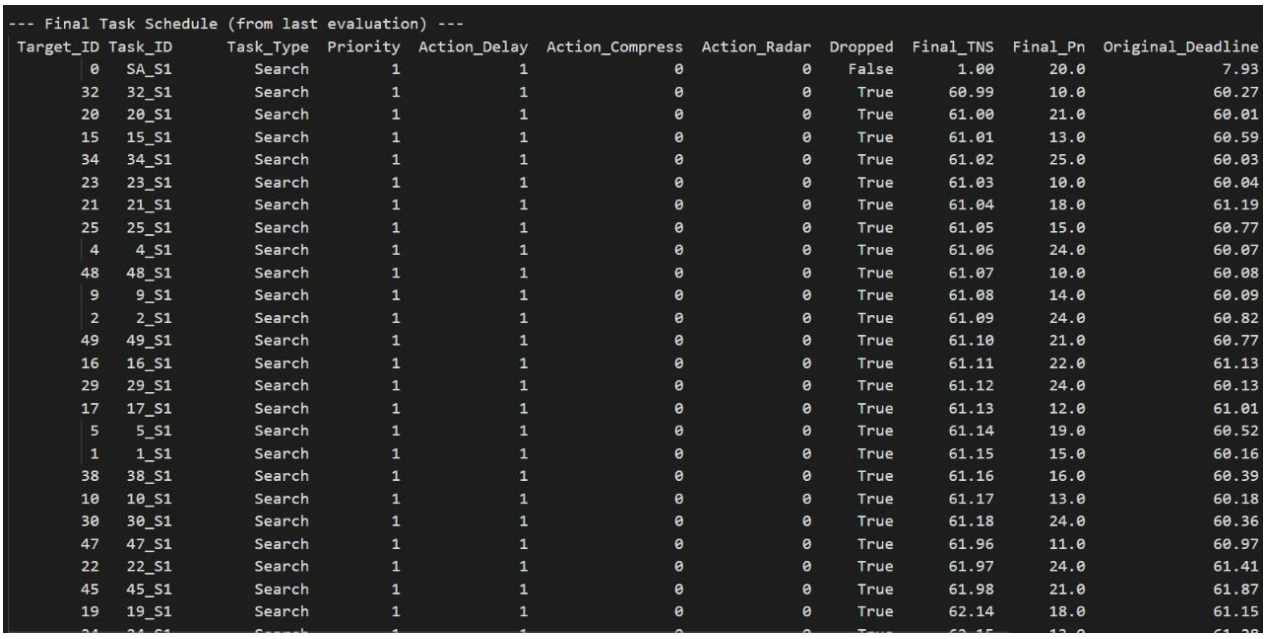
5.3. Results of Custom Scenario-Based DQN

Snapshot of the Custom-Based Scenario Data:





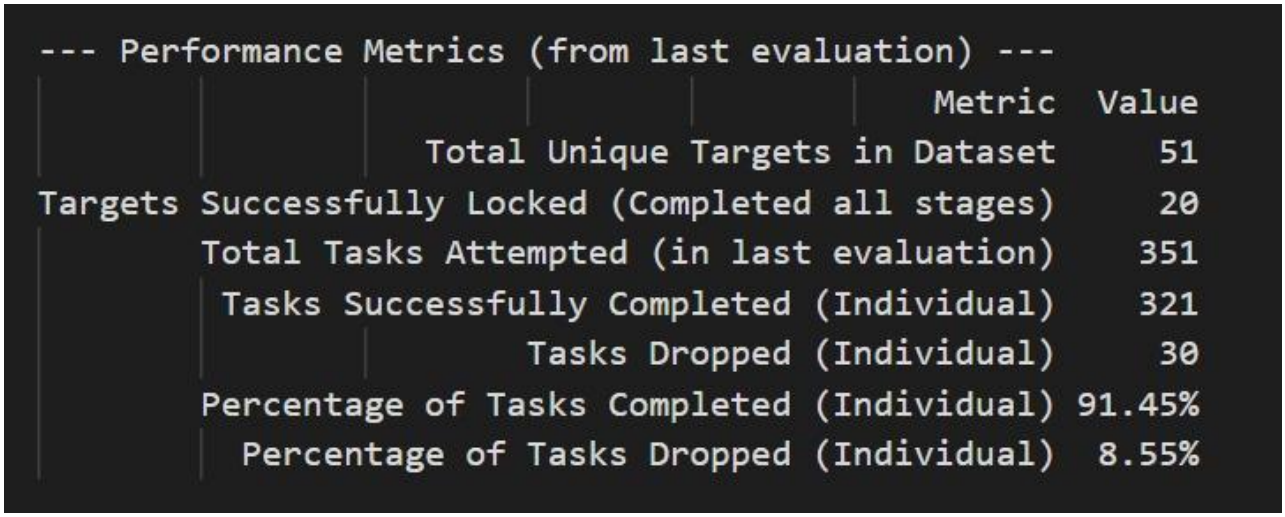
Snapshot of final task schedule:



--- Final Task Schedule (from last evaluation) ---

Target_ID	Task_ID	Task_Type	Priority	Action_Delay	Action_Compress	Action_Radar	Dropped	Final_TNS	Final_Pn	Original_Deadline
0	SA_S1	Search	1	1	0	0	False	1.00	20.0	7.93
32	32_S1	Search	1	1	0	0	True	60.99	10.0	60.27
20	20_S1	Search	1	1	0	0	True	61.00	21.0	60.01
15	15_S1	Search	1	1	0	0	True	61.01	13.0	60.59
34	34_S1	Search	1	1	0	0	True	61.02	25.0	60.03
23	23_S1	Search	1	1	0	0	True	61.03	10.0	60.04
21	21_S1	Search	1	1	0	0	True	61.04	18.0	61.19
25	25_S1	Search	1	1	0	0	True	61.05	15.0	60.77
4	4_S1	Search	1	1	0	0	True	61.06	24.0	60.07
48	48_S1	Search	1	1	0	0	True	61.07	10.0	60.08
9	9_S1	Search	1	1	0	0	True	61.08	14.0	60.09
2	2_S1	Search	1	1	0	0	True	61.09	24.0	60.82
49	49_S1	Search	1	1	0	0	True	61.10	21.0	60.77
16	16_S1	Search	1	1	0	0	True	61.11	22.0	61.13
29	29_S1	Search	1	1	0	0	True	61.12	24.0	60.13
17	17_S1	Search	1	1	0	0	True	61.13	12.0	61.01
5	5_S1	Search	1	1	0	0	True	61.14	19.0	60.52
1	1_S1	Search	1	1	0	0	True	61.15	15.0	60.16
38	38_S1	Search	1	1	0	0	True	61.16	16.0	60.39
10	10_S1	Search	1	1	0	0	True	61.17	13.0	60.18
30	30_S1	Search	1	1	0	0	True	61.18	24.0	60.36
47	47_S1	Search	1	1	0	0	True	61.96	11.0	60.97
22	22_S1	Search	1	1	0	0	True	61.97	24.0	61.41
45	45_S1	Search	1	1	0	0	True	61.98	21.0	61.87
19	19_S1	Search	1	1	0	0	True	62.14	18.0	61.15

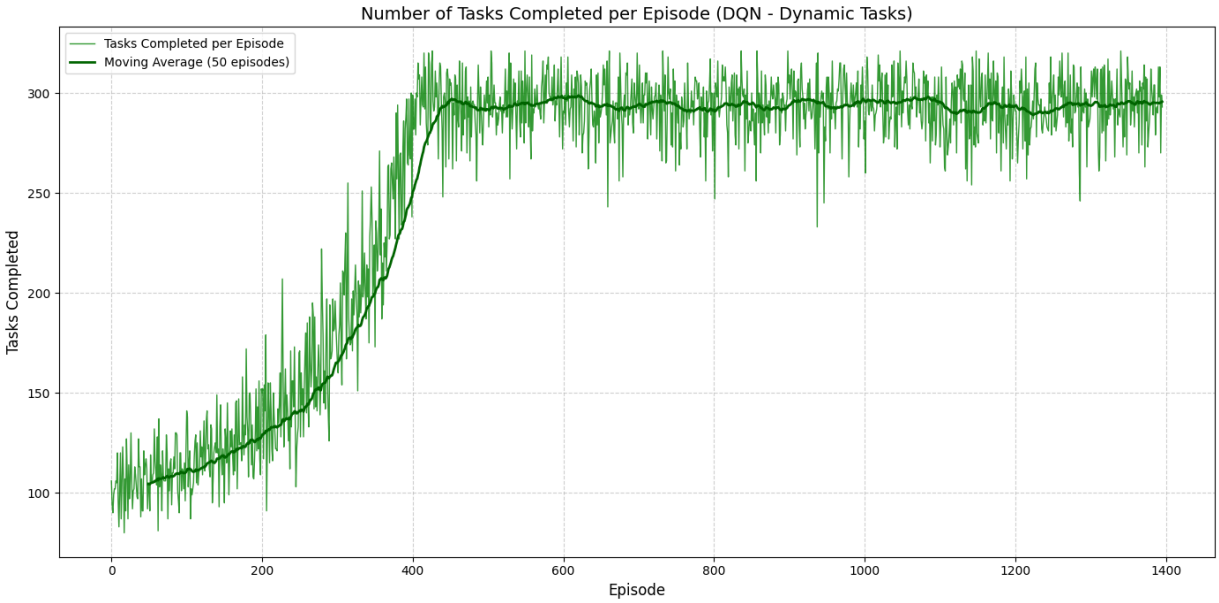
Snapshot of performance metrics after evaluation:

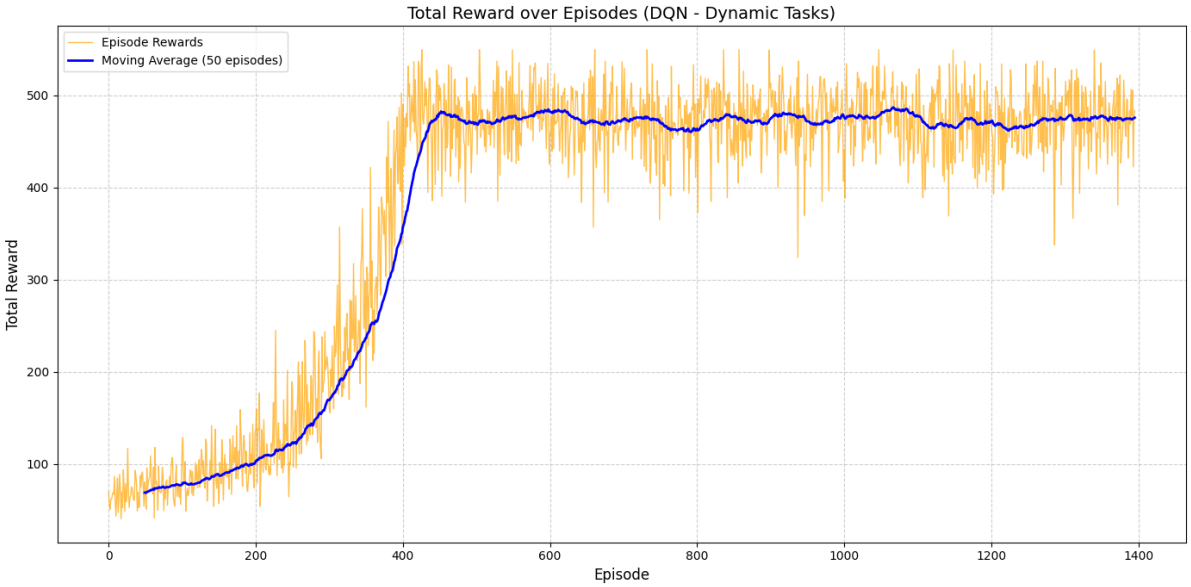


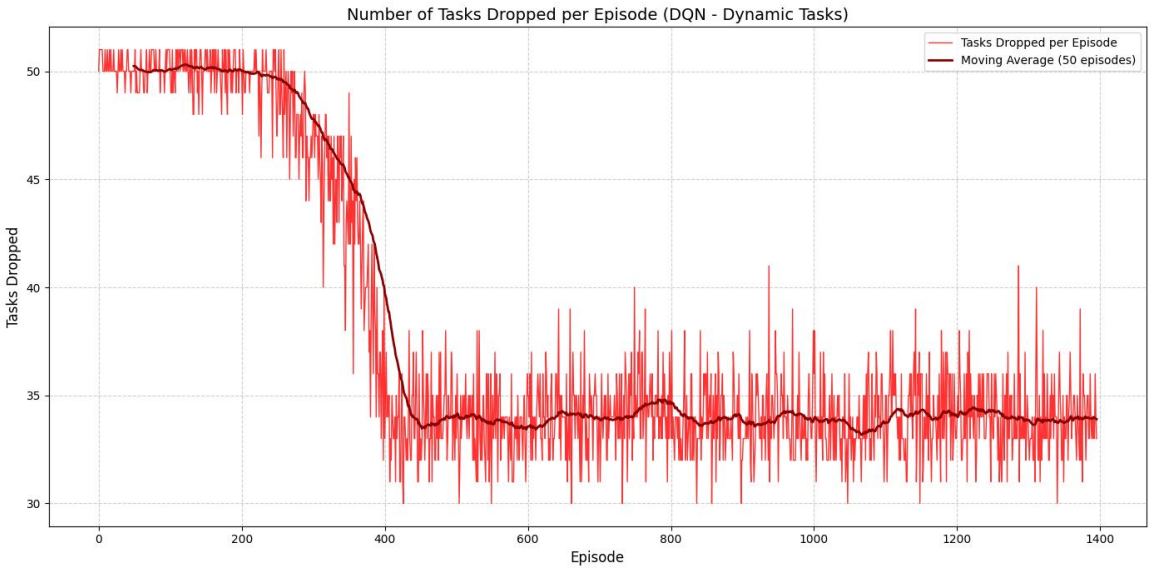
--- Performance Metrics (from last evaluation) ---

Metric	Value
Total Unique Targets in Dataset	51
Targets Successfully Locked (Completed all stages)	20
Total Tasks Attempted (in last evaluation)	351
Tasks Successfully Completed (Individual)	321
Tasks Dropped (Individual)	30
Percentage of Tasks Completed (Individual)	91.45%
Percentage of Tasks Dropped (Individual)	8.55%

Snapshot of reward, task drops and task performed plots:







6. CONCLUSION AND FUTURE WORK

6.1. Conclusion

This project applies reinforcement learning to dynamic task scheduling in multifunction radar systems. Starting with Q-learning in a discrete environment, the approach evolved to a DQN-based model for better generalization and constraint handling. A custom scenario environment was developed, modeling realistic radar beam sequences, power and delay constraints, radar-task compatibility, and must-drop tasks. The trained DQN agent demonstrated effective scheduling policies, stage-aware task progression, and robust handling of infeasible tasks—showcasing the potential of deep reinforcement learning for intelligent radar resource management.

6.2. Future Work

While the current research demonstrates the applicability of reinforcement learning for radar task scheduling under various constraints, several extensions and improvements are envisioned to enhance realism, scalability, and performance:

1. Multi-Agent Reinforcement Learning (MARL)

Future implementations can model each radar as an independent agent within a cooperative or competitive MARL framework. This would allow distributed learning and decentralized decision-making, better reflecting real-world radar networks.

2. Integration of Continuous Action Spaces

Current DQN implementations operate on discrete action spaces. Using algorithms like Deep Deterministic Policy Gradient (DDPG) or Proximal Policy Optimization (PPO) can support continuous task parameters such as power levels or beam steering angles.

3. Adaptive Reward Shaping

Dynamic reward shaping based on mission context or target importance can help the agent prioritize critical tasks more effectively. Incorporating priority-weighted rewards and penalties could improve learning in high-stakes environments.

4. Explainability and Interpretability

As reinforcement learning models are typically opaque, integrating explainable RL techniques could help interpret agent decisions, especially in critical defense applications where traceability is essential.

7. REFERENCES

1. Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction* (2nd ed.). MIT Press.

2. Arulkumaran, K., Deisenroth, M. P., Brundage, M., & Bharath, A. A. (2017). A brief survey of deep reinforcement learning. *arXiv preprint arXiv:1708.05866*. https://doi.org/10.48550/arXiv.1708.05866

3. Ghasemi, M., & Ebrahimi, D. (2024). Introduction to reinforcement learning. *arXiv preprint arXiv:2408.07712*. https://doi.org/10.48550/arXiv.2408.07712

4. Hashmi, U. S., Saeed, H., Ahmed, A., Rana, Z. A., & Riaz, S. (2023). Artificial intelligence meets radar resource management: A comprehensive background and literature review. *IET Radar, Sonar & Navigation, 17*(2), 153–178. https://doi.org/10.1049/rsn2.12273

5. Xu, L., & Zhang, T. (2020). Reinforcement learning based dynamic task scheduling for multifunction radar network. In *2020 IEEE Radar Conference (RadarConf20)* (pp. 1–6). IEEE. https://doi.org/10.1109/RadarConf2043947.2020.9266345

6. Jeauneau, V., Barbaresco, F., & Guenais, T. (2014). Radar tasks scheduling for a multifunction phased array radar with hard time constraint and priority. In *2014 International Radar Conference* (pp. 1–6). IEEE. https://doi.org/10.1109/RADAR.2014.7060309

7. Wang, Y., Zhang, T., Xu, L., Tian, T., Kong, L., & Yang, X. (2019). Model-free reinforcement learning based multi-stage smart noise jamming. In *2019 IEEE Radar Conference (RadarConf19)* (pp. 1–6). IEEE. https://doi.org/10.1109/RADAR.2019.8835772

8. Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W. (2016). OpenAI Gym. *arXiv preprint arXiv:1606.01540*. https://doi.org/10.48550/arXiv.1606.01540

9. Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., & Dormann, N. (2021). Stable-Baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research (JMLR)*. Retrieved from https://stable-baselines3.readthedocs.io