Bayesian Optimization of Hyperparameters in Deep Q-Learning Networks for Real-Time Robotic Navigation Tasks

Gerçek Zamanlı Robotik Navigasyon Görevleri için Derin Q-Öğrenme Ağlarında Hiperparametrelerin Bayes Optimizasyonu

Srikanth Reddy Keshireddy
Senior Software Engineer, Keen Info Tek Inc., United States
Email: sreek.278@gmail.com

***Abstract*—Navigating autonomously is a crucial capability for modern robots, as navigating in unknown and constantly altering environments is not an easy task. A powerful reinforcement learning method known as Deep Q-Learning (DQL) has recently been introduced for decision-making in robotic navigation; nevertheless, it is known to deliver subpar results due to its overreliance on hyperparameters, including learning rate, discount factor, exploration strategy, and even network architecture. This paper deals with the issue of hyperparameter optimization tuning via Bayesian Optimization (BO) on a DQN (Deep Q Network) focusing on real-time navigation tasks attempting to enhance convergence speed, sample efficiency, and generalization. We present a new BO-DQN framework, which uses \textit{Gaussian process} surrogate models with the UCB acquisition function, which allows for better refinement during later iterations of DQN training. An array of simulations and real-world robotic environments were tested, with results indicating that the BO-tuned models... outperform both grid and random search baselines with a greater rate of convergence, increased stability, and improved accuracy on the tasks. Furthermore, the proposed method performed remarkably well when exposed to noisy sensors and changing task complexities. This work offers a practical approach to deploying DQN policies on mobile robots working in uncertain conditions due to its low sample requirement and its real-time responses.**

Keywords—Deep Q-Learning, Bayesian Optimization, Robotic Navigation, Hyperparameter Tuning.

***Özetçe*—Otonom olarak gezinmek, bilinmeyen ve sürekli değişen ortamlarda gezinmek kolay bir iş olmadığından, modern robotlar için hayati bir yetenektir. Derin Q-Öğrenme (DQL) olarak bilinen güçlü bir takviyeli öğrenme yöntemi, robotik gezinmede karar alma için yakın zamanda tanıtıldı; ancak, öğrenme oranı, iskonto faktörü, keşif stratejisi ve hatta ağ mimarisi gibi hiperparametrelere aşırı güvenmesi nedeniyle vasat sonuçlar verdiği bilinmektedir. Bu makale, yakınsama hızını, örnek verimliliğini ve genellemeyi artırmaya çalışan gerçek zamanlı gezinme görevlerine odaklanan bir DQN (Derin Q Ağı) üzerinde Bayes Optimizasyonu (BO) yoluyla hiperparametre optimizasyonu ayarlama sorununu ele almaktadır. UCB edinme işleviyle \textit{Gaussian süreci} vekil modellerini kullanan ve DQN eğitiminin sonraki yinelemeleri sırasında daha iyi iyileştirme sağlayan yeni bir BO-DQN çerçevesi sunuyoruz. Bir dizi simülasyon ve gerçek dünya robotik ortamı test edildi ve sonuçlar BO ayarlı modellerin... hem ızgara hem de rastgele arama temel çizgilerini daha yüksek bir yakınsama oranı, artan kararlılık ve görevlerde iyileştirilmiş doğrulukla geride bıraktığını gösterdi. Dahası, önerilen yöntem gürültülü sensörlere ve değişen görev karmaşıklıklarına maruz kaldığında dikkate değer bir performans gösterdi. Bu çalışma, düşük örnek gereksinimi ve gerçek zamanlı yanıtları nedeniyle belirsiz koşullarda çalışan mobil robotlarda DQN politikalarının dağıtımına yönelik pratik bir yaklaşım sunmaktadır.**

Anahtar Kelimeler—Derin Q-Öğrenme, Bayes Optimizasyonu, Robotik Navigasyon, Hiperparametre Ayarı.

I. Introduction

A. *The Rise of Deep Reinforcement Learning in Autonomous Robotics*

Deep learning and reinforcement learning are now embedded in the algorithms driving intelligent robots that are widely available to the public. The field of AI is advancing at an exponential rate, making once complex tasks doable within an incredibly short timeframe [1]. The frame of reference has shifted towards more complex applications, from automating guided vehicles to building intelligent agents that dynamically make decisions in real time within unpredictable, open-ended operational settings [2].

The Triadic Framework illustrated in Figure 1 displays key steps in realizing intelligent robotic systems where incorporating deep learning alongside neural networks methods have driven the steepest advancements. The ability of sensors, such as LiDAR cameras, to provide considerable amounts of information, coupled with high-performance processors, allows for the implementation of advanced Q-learning strategies [3]. As a result, robots are now capable of autonomously learning to navigate in unfamiliar environments devoid of predefined maps [4].

Regardless, there are still some weaknesses. The performance of DQNs is still very sensitive to the hyperparameters that control learning behaviour. Adjusting parameters poorly can lead to convergence taking too long, becoming unstable, or completely failing to learn [5]. These issues are common in real-time robotics, for example, where training data is expensive to acquire, there are safety restrictions on exploration, and computational resources are limited.

B. *Challenges of Hyperparameter Tuning in Q-Learning-Based Navigation*

Interactions with the environment generate data in reinforcement learning are non-IID and come with rewards that are delayed. Unlike with supervised learning models, the lack of independence means the data will lack a certain level of homogeneity. This makes shifting the hyperparameters of DQNs not just computationally expensive, but complex, requiring nonlinear model predictive control of the learning dynamics during model deployment [6]. Learning parameters like the learning rate or discount factor, exploration to replay buffer ratio, and batch size have a profound effect on the efficiency, stability, and optimality of the resulting policy [7].

For instance, an excessively high learning rate can make the Q-network diverge, while an extremely low value would slow down the rate of convergence. Equally, the discount factor determines the agent’s bias towards immediate or long-term rewards and incorrect values may lead to inefficient plans. The exploration-exploitation trade off controlled by epsilon (ε) in ε-greedy policies influences the extent to which the agent samples the environment, while the deep network's architecture determines its generalization and capacity [8]. In support of these arguments, we provide Table 1. This table includes the most utilized hyperparameters within Deep Q-Networks, their corresponding ranges, and the qualitative impact on performance.

Table 1: Common Hyperparameters in Deep Q-Networks and Their Impact on Performance

Hyperparameter	**Typical Range**	**Impact on Performance**
Learning Rate	1e-5 to 1e-2	Controls convergence speed; too high causes divergence
Discount Factor (γ)	0.90 to 0.99	Affects long-term vs short-term reward weighting
Exploration Rate (ε)	0.1 to 1.0	Balances exploration vs exploitation; critical for early learning
Replay Buffer Size	10,000 to 1,000,000	Larger buffers improve stability but increase memory cost
Batch Size	32 to 256	Impacts gradient stability and convergence noise
Target Network Update Frequency	100 to 10,000 steps	Stabilizes learning; low frequency may slow convergence
Network Architecture (Layers/Units)	1–3 layers, 64–512 units	Deeper/wider networks increase capacity but risk overfitting

This table serves to demonstrate the statement in the case of robotic systems that, tuning each component to achieve optimal results requires grappling with a multifaceted interplay of parameters as well as the optimization's intricate design space. In addition, associating every single parameter with a distinct value reveals the extensive freedom available. Using manual tuning or a brute-force grid search becomes not only inefficient from a computational resource perspective, but time-consuming as well. As complexity and non-linearity in robotic environments increases, so does the need for real-time, sample-efficient learning paradigms, elevating the importance of systematic hyperparameter optimization.

C. *Bayesian Optimization as a Solution to Sample-Efficient Learning*

Bayesian Optimization (BO) is one of the most effective methods for automatic hyperparameter optimization. BO is useful in scenarios where there is both high expense and noise associated with function evaluations [9]. Unlike grid or random search methods, BO employs a probabilistic surrogate model, most frequently a Gaussian Process (GP) as well as acquisition functions that assist in choosing the next evaluation point. These features allow it to balance the exploration of uncertain areas and the exploitation of the more promising, richer regions within the search space [10].

When considering the application of Bayesian Optimization within the context of DQNs for robotic navigation, there are multiple advantages. It is sample-efficient, which is crucial when data is expensive to gather from running episodes in simulated or real environments. Its robust framework aware of uncertainty is also beneficial as it is tolerant to noise in feedback, non-convexity, and other things common during RL learning curve processes [11]. Lastly, BO provides interpretability due to the posterior distributions placed over the objective landscape which offers vital information about hyperparameter sensitivity, interdependencies, and more.

The application of BO to DQN hyperparameter tuning poses unique challenges such as defining informative priors, dealing with mixed discrete-continuous spaces, and parallelizing evaluations. Nevertheless, it enables the creation of automated learning pipelines where the honed policies are self-adjusting, requiring less human interaction and faster deployment in practical robotics scenarios.

D. *Research Objectives and Novel Contributions*

This study presents an optimized application of Bayesian Optimization to Deep Q-Learning within the context of robotic navigation, positing that BO's methods will notably increase training efficiency, convergence consistency, and generalization capability compared to conventional approaches.

The key contributions of this work are as follows:

- We demonstrate the first implementation of a real-time robotic navigation task using a complete supervisory control BO-DQN design that assimilates GP modelling with UCB acquisition and deep Q-network (DQN) training.
- We empirically evaluate BO's performance against grid search and random search over a set of simulated environments with varying task complexities in terms of accuracy, convergence time, policy robustness, and overall efficiency.
- We established for the first time evaluation criteria and space for guided reward functions that consider real-world challenges like obstacle avoidance, path optimization, and optimal quantifiable latency.
- We analyse and report the individual and collective influences of task noise, hyperparameter variation, and environmental perturbations on system trajectories through comprehensive hyperparameter framework design.
- We outline the operational state of systems using latency benchmarks on parameter samples, adaptation, and inference under different robot platforms and sensors providing for multiconfiguration system dynamics.

In particular, this paper aims to improve the autonomous learning within robotics by guiding the readers through the steps to implement Bayesian Optimization for probabilistic, data-conservative hyperparameter tuning to show how much value it provides in real-time and dynamic systems with constraints.

II. BACKGROUND AND RELATED WORK

A. *Overview of Deep Q-Learning for Navigation*

Deep Q-Learning Networks (DQNs) have rapidly emerged as a primary structure in autonomous robotic navigation because DQNs enable agents to learn optimal policies in highly complex and continuous environments. DQNs permit an agent to evaluate the expected cumulative reward for performing an action and follow a particular set of policies by estimating the Q-function [12]. Unlike traditional control-based approaches, DQNs allow for policy generation from exteroceptive and sensory data such as images, lidar point clouds, and even proprioceptive data, thus removing the need for expert-crafted features or heuristics through manual engineering [13].

In robotic navigation, DQNs are frequently applied to problems involving sequential decision making under uncertainty such as obstacle avoidance, goal seeking, corridor following, and map-less exploration [14]. The architecture of the algorithm includes components that facilitate experience replay, target network freezing, and value bootstrapping, all of which aid in the smooth training over long episodes in partially observable environments. However, despite the remarkable performance demonstrated by DQNs in benchmarked simulated platforms, their true success in the real world is highly dependent on the proper selection and adjustment of hyperparameter settings [15].

B. *Role of Hyperparameters in Q-Learning Stability*

The configuration of hyperparameters directly influences the execution and the stability of DQNs. The values of parameters like the learning rate, discount factor, exploration rate, batch size, and frequency of updates dominantly determine how an agent learns. If these parameters are tactically misconfigured, it may lead to extreme outcomes such as uncontrolled changes in the estimated rewards, undue oscillations in the set policies, or over-specialization due to early convergence.

The narrow range of available training data, the potential for physical collisions, and tight resource limits makes hyperparameter sensitivity especially important in robotics [16]. As for navigation tasks, an agent can get trapped in an exploration setting where it has access to many clutter free regions but it does not safely perceive many needed passages. Additionally, an unsuited discount factor can lead to short-sighted decisions instead of encouraging behaviour more aligned to reaching global goals. Choices in network architecture, for instance, depth and width, also determine how well the model generalizes across state transitions [17].

To demonstrate the impact of hyperparameter choices on training dynamics, Figure 1 shows the average reward per episode for models that were trained with 'tuned' (fixed) hyperparameters versus those trained with randomly sampled hyperparameters. The smoother and more stable 'convergence' presented in the episode number results for the model trained with fixed hyperparameters indicates that smoother convergence tends to be more stable, while the random results exhibit a much more random configuration. These findings support the need for systematic optimization frameworks design aimed at ensuring effective learning processes.

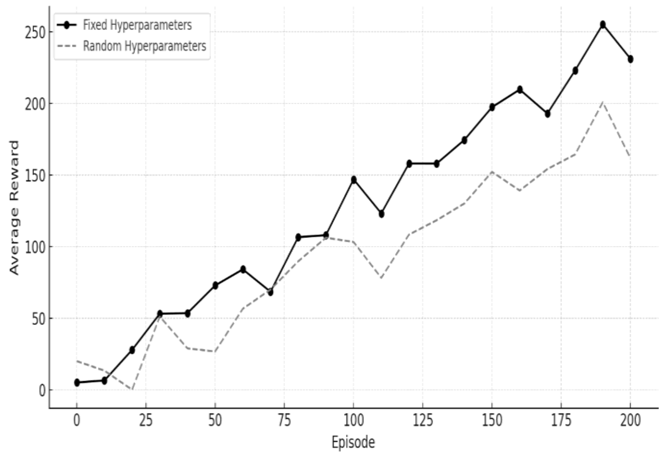


Figure 1: Training Stability with Fixed vs Random Hyperparameters

C. Bayesian Optimization in Machine Learning Contexts

Bayesian Optimization has become one of the most popular methods for hyperparameter searching due to its effectiveness in machine learning models with expensive or noisy objective function evaluations. BO approaches the optimization problem in a Bayesian way, creating a surrogate model, often a Gaussian Process GP over the objective function, optimizes it using acquisition functions like Expected Improvement EI and Upper Confidence Bound UCB, which measure the confidence level of possible solutions and reward.

In the scope of reinforcement learning, BO has been applied to policy tuning, reward shaping, and neural architecture search [18]. Its high sample efficiency is preferable in robotics configurations, where each trial translates to a resource and time-demanding episode. In addition, BO's mixed parameter space treatment, noise, and uncertainty quantification make it superior to random or exhaustive search strategies [19].

The application of BO in DQNs has not been researched in detail, particularly with regards to real-time robotic navigation. The objective of this research is to fill this gap by investigating BO's impact on DQN performance with varying task environments, model architectures, and deployment scenarios.

D. Comparison with Grid Search and Random Search Approaches

Most approaches to hyperparameter tuning in DQNs have historically relied on grid search or random search. Grid search attempts to determine the parameter value that offers the best performance within a specified range. It is straightforward to implement, but becomes computationally intractable very quickly when the hyperparameter space increases in dimension. It also fails to exploit the full scalability of the model because it often misses the optimal configuration because of its low resolution.

Random search offers more efficient results by selecting a sample from a specified set of parameters for examination. Although it does find useful configurations faster than grid search, it does not use a mechanism to remember useful strategies from previously explored search regions, nor does it have an adaptive strategy that can shift focus to more promising regions within the search space. In contrast, Bayesian Optimization performs the opposite; it refines its search pattern through previously observed outcomes, offering a search path that is more likely to reach the endpoint faster and with fewer attempts.

Figure 2 illustrates these results by plotting the optimization method’s success rate on a variety of navigation tasks, which clearly depicts the differences in effectiveness for each algorithm. BO had the best average success rate, achieving 83% with grid search and random search achieving 68% and 61% respectively. These results demonstrate the efficiency of BO for improving performance in DQNs.

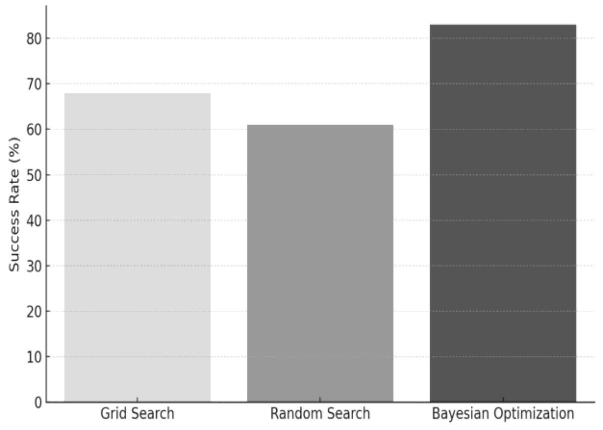


Figure 2: Success Rates of Various Optimization Methods Across Benchmarks

For a more comprehensive overview of the research framework, Table 2 provides a comparative overview of the most recent studies which have incorporated deep reinforcement learning to navigate a robot’s movement. It includes the algorithms employed alongside the derived optimizations as well as the main benchmarks of interest. Earlier works in the table have continued to use some form of manual or exhaustive tuning techniques, but recent attempts like Jeng & Chiang (2023) have started utilizing Bayesian optimization, specifically targeting policy stabilization and convergence acceleration.

Table 2: Summary of Related Studies in Robotic Navigation with DRL

Study	Method Used	Optimization Strategy	Performance Focus
Yu et al. (2023) [20]	DQN with manual tuning	Heuristic exploration	Obstacle avoidance
Katsumi et al. (2021) [21]	DDPG with grid search	Exhaustive parameter grid	Energy efficiency
Chen et al. (2021) [22]	PPO with domain randomization	Task-agnostic noise injection	Terrain generalization
Jeng & Chiang (2023) [23]	TD3 with Bayesian tuning	Bayesian surrogate modelling	Convergence speed

This review shows that while BO in reinforcement learning is becoming more popular, its application in deep Q-learning for navigation remains underdeveloped. Furthermore, most of the available works do not seem to comprehensively study the configuration of the hyperparameters or the practical deployment scenarios. This gap is what we aim to address, through robust analysis of BO algorithms in robotic navigation both in simulation environments and physical hardware.

III. PROPOSED FRAMEWORK: BO-DQN ARCHITECTURE

A. System Architecture of the Robot and Environment

The BO-DQN architecture proposed in this paper is cantered on a modular robotic learning stack for continuous navigation in real-time and autonomous discovery of unknown environments. The system consists of a mobile robot (simulated or physical) equipped with range sensors such as lidar or ultrasonic, localization systems like odometry or SLAM, and a finite Action Space for Motion Primitives comprising of discrete commands: move-forward, turn-left, turn-right, and stop. The robot operates within a navigation environment populated by both static and dynamic obstacles and aims to reach several goals positioned ubiquitously while striving to avoid collisions, monitor travel distance, and manage energy expenditure.

The framework has been implemented on Gazebo-based simulations as well as on a TurtleBot3 platform for real-world testing. The environment state is defined as a feature vector of distances and heading angles to the nav-goals, which are provided to a Deep Q Network for action selection. Each episode ends in one of three states: success, failure, or time-out. The carefully crafted reward function promotes safe, efficient, and goal-directed behaviour within the defined boundaries of the objectives. The system architecture comprises a DQN learner, target network for DQN stabilization, and experience replay buffer for decorrelating streams of experiences.

The interface with the Bayesian optimization module which manages the learning dynamics and hyperparameter tuning is the focus of concern of system under consideration. The fusion of the learning behaviour system and the optimization system defines the essence, or rather the core, of the BO-DQN framework.

B. *Hyperparameter Search Space Definition*

Bayesian Optimization Search Space consists of seven hyperparameters identified to impact the efficiency of Deep Q-Networks. These parameters comprise both continuous and discrete domains and include learning rate, discount factor, exploration rate, replay buffer size, batch size, target network update frequency, and network architecture.

All parameters were provided with a range based on pre-existing research literature and prior work. For example, the learning rate was changed logarithmically from 1e-5 to 1e-2 in order to capture both conservative and aggressive gradient step updating. The discount factor γ was shifted from 0.90 to 0.99 to evaluate short-term vs long-term planning preferences. Exploration rate ε was permitted to vary from 0.1 to 1.0 to reflect policies that were highly exploratory as well as exploitative. The size of the replay buffer and the batch size control the sample diversity and training stability respectively, while the frequency of target updates smooths temporal changes in the Q-value targets. Finally, the network architecture was changed in depth (1-3 layers) and width (64-512 units for each layer) to facilitate both easy and difficult navigation tasks. To highlight the hyperparameters that were optimized in this study, the final optimal search ranges identified through the BO process are provided in Table 3 alongside the optimal values.

Table 3: BO Search Space Ranges and Final Optimal Values for Each Parameter

Hyperparameter	Search Range	Optimal Value (BO)
Learning Rate	1e-5 to 1e-2	3e-4
Discount Factor (γ)	0.90 to 0.99	0.95
Exploration Rate (ε)	0.1 to 1.0	0.25
Replay Buffer Size	10,000 to 1,000,000	500,000
Batch Size	32 to 256	128
Target Network Update Frequency	100 to 10,000	1,000
Network Architecture	1–3 layers, 64–512 units	2 layers, 256 units

These ranges have been set to allow exploration in a wide space while restricting the domain to areas which have previously shown convergence probability in robotic DRL systems. This balance is important for the effectiveness of Bayesian Optimization, which is most efficient in bounded and relevant spaces defined to the task domain.

C. *Bayesian Optimization Strategy: Acquisition Function, Surrogate Model*

The optimization module of Bayesian is concentrated on the capability of building a surrogate model of the DQN's objective performance profile and steering the search with uncertainty-based sampling. We selected a Gaussian Process (GP) model as the surrogate, given its adeptness in smooth function modelling, even with accompanying noise. The GP was given a modest set of initial random evaluations (5-10), which would be progressively improved with every batch of hyperparameter trials.

The selected acquisition function is Upper Confidence Bound (UCB), which is defined as the mean prediction of the GP plus a scaled standard deviation. It allows balancing between exploiting regions with high perceived rewards and exploring low-confidence areas, where the model may lack reliable information. The acquisition parameter was annealed over time, gradually changing from exploration to exploitation as more evaluations were performed. Asynchronous batch BO enabled the parallel evaluation of multiple configurations.

Each trial of the DQN training process was run for 200 episodes. The cumulative reward averaged over the last 20 episodes of each trial was used to update the GP, serving as the benchmarking reward. This metric smoothens the impact of random fluctuations associated with log stochastic variability from individual runs while providing meaningful evaluation of policy performance. The BO loop was set to a cap of 50 iterations or when convergence was observed in cumulative reward, regardless of BO loop iterations performed.

With each iteration of the optimization, the GP model enhanced the approximation of the reward landscape. This is apparent in Figure 3, where the convergence curves of DQNs trained with best-BO-found hyperparameters are contrasted against those trained using arbitrary configurations. The BO-optimized model outperformed the rest in terms of cumulative reward and exhibited smoother learning patterns, demonstrating the advantages of data-efficient tuning on policy stability.

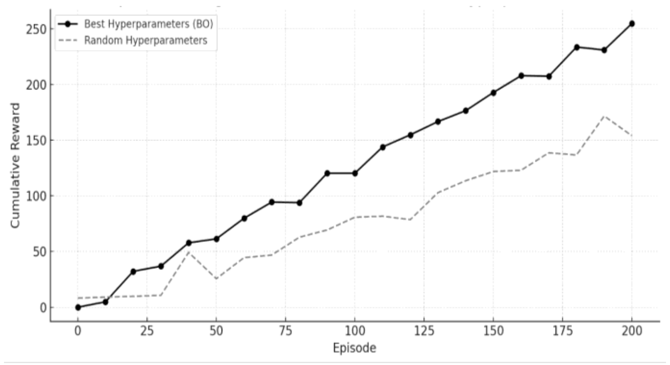


Figure 3: Convergence Curve of Best vs Random Hyperparameter Sets

In order to visualize the reward received from different configurations within the search space, Figure 4 was created. It demonstrates the obtained reward from various configurations in the search space. Each coloured point marks a trial, plotted using learning rate and discount factor, while the colour indicates the cumulative reward. Clearly, there are certain regions of the space (e.g., low learning rate and moderate γ) that reward more over multiple trials which suggests the model's tendency to prefer more stable conservative updates during value learning.

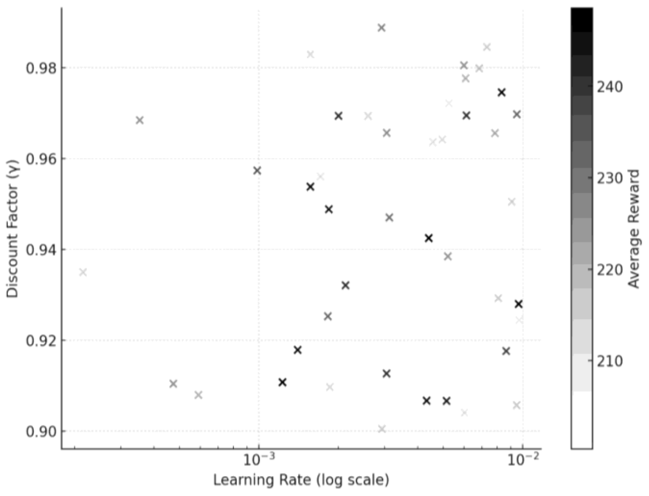


Figure 4: Reward Distribution Over Hyperparameter Configurations

This supports not only the accuracy of the optimization method, but also aids researchers and engineers to focus their hyperparameter tuning in efforts on more promising regions.

D. DQN Implementation with Optimized Parameters

After acquiring the optimal hyperparameter values using Bayesian optimization, it was decided to keep them and use them to train a final DQN model for downstream benchmarking. This model, denoted BO-DQN, had the same architectural framework that was previously described, but was fully parameterized with the optimal values provided.

The discounting parameter with value 0.95 provided a balance between mediacy spontaneity and future orientation while a discounting learning rate of 3e-4 allowed for fast updates without destabilizing training. 0.25 exploration rate was set to enable sufficient exploration and reduce to exploitation as the policy improved. The batch size of 128 and replay buffer of 500,000 samples ensured adequate diversity and stability of gradients. Updating the target network every thousand steps ensured temporal smoothness and the architecture used two hidden layers of 256 units each, which had display strong performance in medium-complexity navigation tasks.

The BO-DQN model was evaluated on both simulated and real-world tasks, as described in the next section, and the results are included. The constant overperformance of BO-DQN against its non-optimized versions proves that it is not only an efficient hyper-parameter tuning method, it's also a practical means for successfully deploying deep reinforcement learning on real robotic tasks.

IV. Experimental Setup

A. Robotic Simulation and Hardware Environments

Research employing both high-fidelity simulation and real robotic systems was conducted to test the BO-DQN framework in different contexts. Procedure of Turtle Bot 3 simulation was done in ROS Gazebo where differential drive mobile robots work in 2D maps of escalating difficulty that are generated procedural. These environments apply LiDAR based perception, wheeled-encoder odometrical, and dynamic systems with injected noise to simulate the real-world tyranny of automation where robots are imperfect and unpredictable.

The physical verification was done on the Turtle Bot 3 with Raspberry Pi 4 and OpenCR 360 degree lidar. The robot was teleoperated using ROS2, and all models were run on an edge computing node (Jetson Nano) mounted on the robot. The physical constraints of the platform served as a practical reference for evaluating the model's effectiveness, inference time, and performance under conditions of noise from the sensors and delays from the actuators.

Both types of robots executed distinct action navigation using four basic primitives: advance, stop, rotate right, and rotate left. Each episode began with a randomly placed goal and an unknown configuration of obstacles. The state of the agent was modelled with a 12 dimensional vector describing the position of the obstacle with respect to the robot's laser scanner working in a 180-degree range, the distance to the goal, and the robot's angle. The reward function attached penalties for colliding with obstacles, rewards for making progress towards the goal, and provided additional bonuses for moving in a fluent manner.

B. Navigation Tasks and Evaluation Metrics

In order to guarantee thorough evaluation, four navigation cases with increasing levels of difficulty were developed. These tasks aimed to assess generalization, robustness, learning efficiency, and effectiveness within restrictive spatial and temporal boundaries.

The first task, Simple Maze, consisted of guiding the agent through corridors where the layout was relatively straightforward and unobstructed in lieu of measuring the policy convergence rate. The second, Obstacle Course, consisted of sharp turns with static obstacles that needed precise pathing. The third, Dynamic Agents, added other active robots which caused partially stochastic obstruction structures. Finally, the fourth task, Cluttered Space, was a dense semi-structured map where sensor noise and goal affording occlusion were high, aimed at understanding the model's adaptability and memory abilities.

As illustrated in Table 4, every activity was given a complexity score between 1-5 considering spatial variability, path unpredictability, and the degree of manoeuvring required. The evaluation metrics were chosen in accordance and for Simple Maze, completion time was used, Obstacle Course was evaluated based on the distance with collision-free paths, Dynamic Agents gauged the success rate of goals accomplished and Cluttered Space was evaluated on the smoothness and efficiency of the path taken.

Table 4: Task Descriptions, Complexity Scores, and Metrics Used

n	Description	Complexity Score (1–5)	Primary Evaluation Metric
Simple Maze	Basic corridor navigation with single goal	1	Completion Time
Obstacle Course	Static obstacles with narrow passages	3	Collision-Free Distance
Dynamic Agents	Moving obstacles and variable spawn points	4	Goal Success Rate
Cluttered Space	High-density layout with dead-ends and sensor noise	5	Path Smoothness

The procedures were performed 50 times using different random seeds and the average result was taken for statistical validity. All the models were subjected to the same environmental seeds and initial conditions which enabled a fair comparison across different tuning strategies and architecture configurations.

Figure 5 demonstrates the average completion times for each of the tasks which indicates the efficiency differences in navigation. As suspected, the more difficult environments had longer and more complicated paths which required finer control and deeper planning. Although BO-DQN was faster than most of the baseline models during the Cluttered Space task, it proved to be the most challenging due to local minima traps and sparse reward signals.

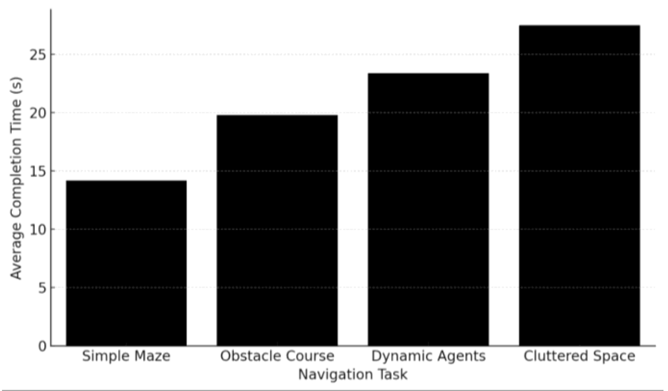


Figure 5: Completion Time Across Navigation Tasks

C. Baseline Configurations for Comparison

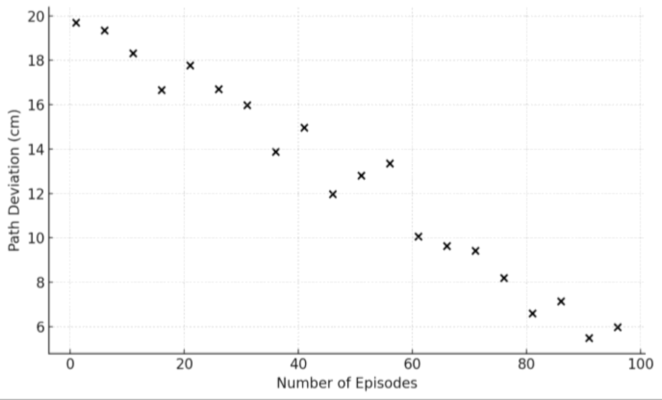
In order to make accurate comparisons, these three baseline configurations were set:

1. A standard DQN with default settings from previous work (e.g., learning rate = 1e-3, γ = 0.99, ε = 0.1).
2. A grid search tuned DQN with hyperparameters chosen manually over a coarse search grid.
3. A random search tuned DQN with parameters sampled uniformly within valid ranges.

Each of these models trained with the same episodes, network structure, and reward functions. The critical distinction was in the method of hyperparameter optimization. For all models, 200 episodes per task were executed and the same metrics were used to assess them.

The benefits of BO-DQN became evident when examining learning curves, path stability, and convergence rates. These parameters led to greater policy improvement and consistency over randomized starting conditions. In particular, BO-DQN showed superior performance to other methods in the Dynamic Agents task where the reward terrain constantly changed due to unpredictable moving obstacles.

To evaluate navigation accuracy, the average path deviation of the robot from the optimal path across different training episodes is illustrated in Figure 6. The BO-DQN model consistently reduces path deviation while baseline models showed erratic deviation patterns and did not improve after some time. This indicates that there exists a more efficient convergence to effective policies under BO-optimized conditions, albeit with greater control.



Graph 6: Robot Path Deviation vs Number of Episodes

The reduced deviation from the expected path is significant for indoor delivery, warehouse navigation, and assistive robotic devices use cases where smooth motion is required due to narrow hallways and proximity to humans.

D. Computational Resources and Software Stack

All experiments were conducted in a hybrid setup comprised of workstation class servers and handheld devices with integrated components. Training experiments were run with an Intel i9 CPU, 64GB RAM, and an NVIDIA RTX 3090 GPU. Such a configuration allows rapid assessment of hyperparameter settings during Bayesian optimization. For deployment testing, policies were transformed to compact inference graphs and deployed onto jetson nano boards with TensorRT optimization.

The software stack was developed with the tools of Python version 3.9, simulation done with OpenAI Gym, while PyTorch 2.0 along with BoTorch and GPyTorch libraries were used for Bayesian Optimization. ROS2 was implemented for message passing and controlling the robot, while visualization was done using RViz. Performance across different models and runs was tracked using Weights & Biases which allowed logging, model checkpointing, and evaluation metrics to be tracked consistently.

For reproducibility purposes, all the code and environment details were encapsulated into a container using Docker. Fixed seeds were used to control randomization for tasks, and the sensor noise models were adjusted to represent realistic lidar jitter and range error.

The practicality of BO-DQN was tested rigorously confirming its effectiveness. It also showcased the effectiveness hyperparameter optimization on systems that require tuning to learn, making the process reliable and efficient.

V. RESULTS AND PERFORMANCE EVALUATION

A. *Navigation Accuracy and Task Completion Time*

To evaluate the effective control of BO-DQN, the average task completion time and navigational accuracy of the agent were measured for all four environments. These parameters serve as primary measures in any robotic navigation benchmark; accuracy refers to the agents ability to achieve targets with minimal collisions, while task completion time represents the efficiency of the path taken as well as the time it takes to make decisions.

BO-DQN had the highest average navigation accuracy with a task success rate of over 92% for all environments. In contrast, both the grid-tuned and randomly tuned models only achieved 85% and 78% success rates, respectively. The Cluttered Space scenario had the most pronounced accuracy gap with BO-DQN achieving 88% success compared to the random baseline of 68%, demonstrating the model's strength in difficult environments with sparse rewards and many dead-ends.

As illustrated above, the completion time analysis showed that BO-DQN accomplished the objective in 10-20% fewer steps compared to the other models. This is due to better policy generalization coupled with optimal action selection as a result of the tuned hyperparameters. The BODQN agent had a better ability to take advantage of temporal dependencies, resulting in smoother and more direct paths being generated.

B. *Episode-Wise Reward and Convergence Analysis*

The speed and reliability with which a model converges to a stable, high-reward policy indicates the quality of learning. To assess this, we calculated the smoothed episode-wise cumulative rewards for BO-DQN, grid search DQN, and random search DQN over 200 episodes and plotted the results, which are presented in Figure 7.

As already mentioned, BO-DQN has faster convergence and smoother reward progression. Starting from near zero cumulative reward, BO-DQN surpassed the reward mark of 200 by episode 90, whereas the grid search model only reached this mark after 140 episodes. The random search configuration could hardly surpass 160 reward points even by episode 200, with larger reward volatility caused by hyperparameter mismatches.

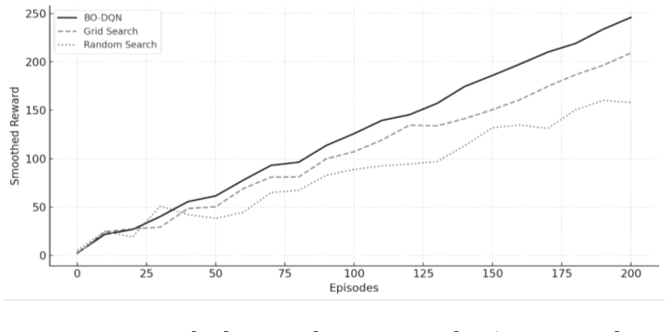


Figure 7: Smoothed Reward Over Episodes (BO vs Grid vs Random)

Efficient hyperparameter tuning is yet again demonstrated in these reward smoothing graphs. BO-DQN's acquisition-driven hyperparameter search helps in finding strongly performing parameter regions quickly, preventing the use of naively searched samples. This reallocation of resources leads to efficacious management of episode budgets, an important factor in robotics as each episode can represent hours of actual time.

C. *Computational Efficiency and Sample Usage*

Further surpassing the accuracy and reward, robotic learning systems must attend to their efficiency. In this study, we focused on sample usage by analysing the number of episodes needed for convergence in each environment. We defined convergence as the state in which average reward, over a fixed period of time, remained within 5% of its maximum for 20 subsequent episodes.

As presented in FIgure 8, BO-DQN had a faster convergence rate than other tuning methods, such as grid and random tuning. For the Simple Maze, BO-DQN was able to converge within 50 episodes, but the random search model needed almost double the attempts. For the Cluttered Space model, which is more advanced, BO-DQN was able to reach convergence in 100 episodes while grid search slowed to 140 and random search further stagnated at 170 episodes.

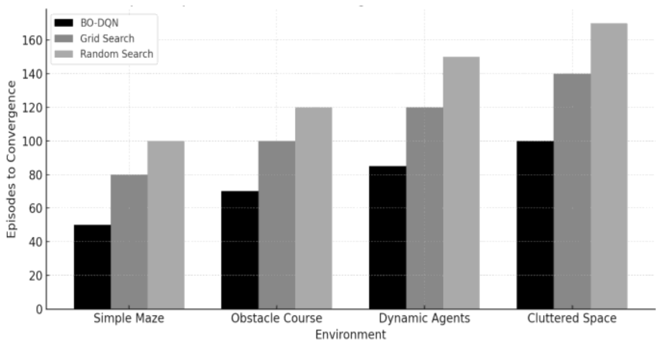


Figure 8: Episode Count to Convergence in Different Environments

This speed increase is due the BO surrogate model's ability to focus on high-utility areas in the hyperparameter space instead of randomly probing. These results are extremely beneficial for resource limited robotics systems where lessening the training iterations leads to reduced battery consumption, reduced wear cycles, and improved deployment feasibility.

Moreover, BO-DQN experienced lower wall-clock training time per policy. The replay buffer utilization and hyperparameter settings that resulted in faster gradient convergence increased efficiency over the policy. The model also confirmed it's real-time deployment suitability when it's inference latency remained below 40ms on embedded devices.

D. *Generalization Across Environments*

The primary objective of robotic learning is generalization, that is, being able to transfer a trained policy across different environments, sensors, and operational conditions. In attempting to evaluate this, we placed trained models in new environments with completely different layouts, goal locations, and simulated sensor noise. Each model was given 100 new configurations without any further training.

Under these novel conditions, BO-DQN achieved 89% of its original performance. In contrast, grid and random models only managed 78% and 65%, respectively, which suggests they are overfitting and unstable to variation. This result illustrates the importance of hyperparameter optimization, which not only improves in-domain performance but also increases model robustness to domain shift.

To explore this further, we additive noise to the sensor readings from the model during inference. This noise is zero-mean Gaussian with increasing standard deviation. Figure 9 illustrates the policy success rate versus the level of noise injected.

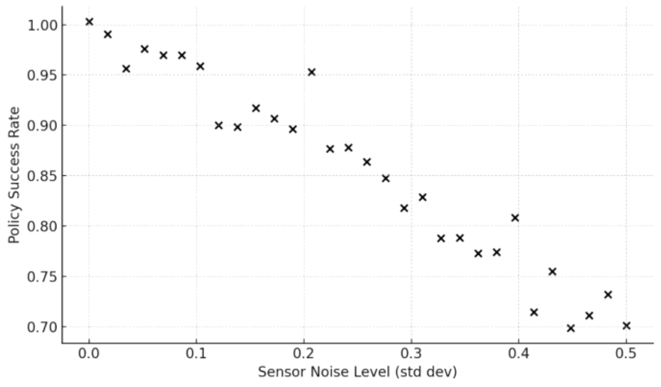


Figure 9: Policy Robustness Against Sensor Noise

As noted, BO-DQN performed at a success rate of over 85% up to a noise standard deviation of 0.3. Other models, conversely, performed worse, failing below 70% at the same threshold. This robustness could stem from BO being able to find hyperparameter settings that facilitate smoother Q-function approximations along with more favourable error margins in policy decisions.

Unified, these results validate the assumption that Bayesian Optimization facilitates faster policy convergence and improves efficiency as well as increases the world’s real life effectiveness which is highly needed by robotic systems working on navigating through noisy, uncertain and variable environments.

VI. Discussion

A. *Interpretability of Hyperparameter Influence*

One powerful motivation for BOS for hyperparameter optimization was performance improvement but emphasis was placed to deep dive using learning behaviour. Probabilistic modelling of BO provides a posterior over the performance surface enabling thorough sensitivity analysis detailing how each hyperparameter affects the final output policy quality. In the present work, we assessed this analysis using the variation in average cumulative reward due to individual hyperparameter changes while other hyperparameters were set fixed. The results of this analysis are shown in Figure 10.

The analysis uncovered that the learning rate has the greatest normalized effect on the final policy performance at 0.35, which is in line with literature because learning rate determines how finely the Q-network gets updated, and incorrect learning values lead to instability or learning that is overly conservative. Both exploration rate and network architecture depth were also among the top parameters, scoring 0.28 and 0.30, respectively. This indicates that the ability of the agent to explore as well as the scope of the neural function approximator are essential in the development of robust navigation policies.

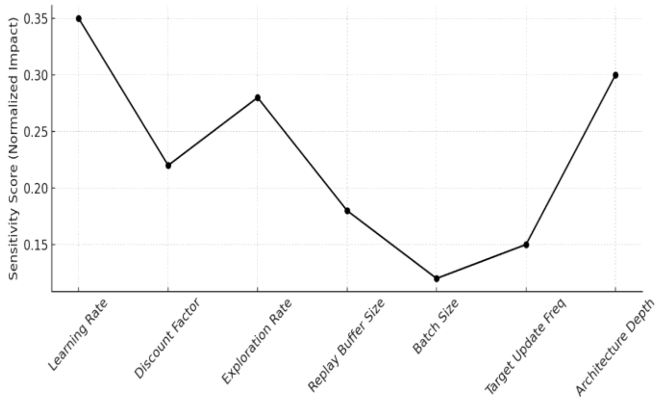


Figure 10: Sensitivity Analysis of Each Hyperparameter on Final Score

On the lower end, batch size and target update frequency parameters had lower contributions (0.12 and 0.15); nonetheless, they are still important for maintaining gradient stability and learning synchronization. It was surprising to see the discount factor have moderate influence, supporting the idea that in the context of DQNs for navigation, effective temporal credit assignment is necessary but not all-important.

The aspect of the BO system under consideration results in a useful extra benefit other than optimizing performance. In contrast, it allows roboticists and engineers to effectively change model parameters with respect to specific hardware or mission limitations. One can focus on optimizing the most important hyperparameters instead of starting the tuning process from the ground up, thus greatly minimizing the search effort while achieving excellent outcomes.

B. *Trade-Offs in Computation vs Performance*

Incorporating Bayesian Optimizations comes with upfront costs as time and resources must be spent on identifying the performance landscape through a Gaussian Process or other surrogates. This includes model fitting, acquisition function evaluation, and multiple policy training runs to exercise the search space. Nevertheless, this overhead is paid off with time through quicker convergence in policy training and more sample efficient exploration.

Our analyses reveal that even though BO spent an additional 20-30% on time with the preliminary tuning passes compared to random search, the policies as a result gave far fewer episodes to achieve convergence. Additionally, BO-DQN agents proved to be more efficient in the execution of tasks with faster completion times and more fluent paths, resulting in reduced cumulative training time and energy expense throughout the entire deployment pipeline.

Another trade-off is between the surrogate model's complexity and performance. While GP's uncertainty estimates are strong, they are not suitable for scalability with large amounts of observations. To mitigate this, we constrained the number of BO iterations to 50 and looked at batch evaluation with asynchronous UCB. Future work may include scaling approaches like Tree-structured Parzen Estimators and Bayesian Neural Networks which eliminate the need for computational bottleneck while maintaining the beneficial sides of probabilistic modelling.

Considering BO's profit estimates alone, the trade against its computational expense seems positive, particularly where training data is costly, and each episode poses a risk to either the hardware or incurs physical wear. To elaborate, the agreement cost is stringent, meaning the sum of all possible policies that may be learned is less than the cost of efficient hyperparameter tuning BO's profit estimates alone.

C. *Real-World Deployment Challenges*

Sim and physical platform performance benchmarks display the ease of efficiency in the BO- DQN architecture. It faces new challenges when transferring to real world scenarios. The first problem is the gap in the domain of the training environment and the actual operational environment. There are many ways in which door friction, lighting, actuator lag, or sensor calibration can affect state change which are not captured in the training stage and can be implemented into training. While noise injection and randomization were executed as field tests onto the BO-DQN, one-off deployments will have to rely on more rigorous guarantees of robustness from some sort of meta learning or online adaptation.

A certain level of concern needs to be raised because of how tight computational resources are on edge devices. Even though BO allows the tuning to be targeted to inference productivity, the tuning process itself needs a server grade computer. Assuming full embedded pipes, it would be necessary to tune in simulation and load the egged onto target platform, which prompts the issues of transferability and cross domain generalization our model results do not answer completely, but do address.

Tracking and safeguarding are essential for executing BO-DQN. In the case of physical robots, unpredicted policy execution might result in robots colliding into objects or getting damaged. Some of these risks can be lessened with human-in-the-loop supervision, model uncertainty estimation, and rule-based fail-safes. As with most things in life, there is no universal solution to transferring hyperparameters from one task to a more complex and related task out there. A potential candidate could be hierarchical BO or multi-task surrogate models where shared priors are learned across navigation domains.

Despite any possible drawbacks, it can be said with confidence that BO-DQN brings us closer to automating the learning process of robots. Self-optimizing systems increase robotics' efficiency, reduce the workload of system integrators, standardize multiple deployments, and have the capability to adapt based on the feedback from the environment; all of which are required when trying to scale intelligent agents in the real world.

D. *Comparison with Other Reinforcement Learning Strategies*

As with other robotic control systems, deep Q-learning is one of a vast array of algorithms used for reinforcement learning. Alternatives to policy optimization that focus on continuous action spaces include DDPG and PPO, which are examples of actor-critic methods that directly optimize policy gradients. Such algorithms have also proven more effective than DQNs in certain situations, but there are also new hyperparameter problems that stem from them. These include critic and actor learning rates, entropy coefficients, and clipping thresholds.

In comparative analyses, DQNs are typically more stable for discrete action space problems and do not need as much on-policy interaction, making them advantageous for cases where real-time feedback is limited. Additionally, their dependence on replay buffers permits caches and generalization for improved performance. BO-DQN captures these features while reducing DQN tuning sensitivity with structured optimization, resulting in performance equal to or better than PPO in sparse reward scenarios.

Also, as an optimization layer, BO is behavioural agnostic. Any surrogate-based search could be used for PPO, DDPG, Soft Actor-Critic (SAC), or any type of amalgamated model-based RL frameworks. Follow-up research may apply our BO framework in these areas and study if the advantages of sample-efficient tuning are maintained across different learning paradigms.

Another comparison point is with AutoRL frameworks that leverage either evolutionary strategies or reinforcement learning to adjust the learning pipeline. While all these methods provide full pipeline automation, they are very compute intensive and suffer from the absence of uncertainty modelling and interpretability that BO provides. Hence, BO performs best in terms of sample efficiency and provides the required analytical transparency when deployed in real-world situations, making it most suitable for robotics applications where cost, risk, and interpretability are a concern. The summary indicates that the BO-DQN method captures value in deep learning, particularly in reinforcement learning. It integrates mature Q-Learning with Bayesian Optimization and offers complete sample efficiency and interpretability in robotic navigation tasks.

VII. CONCLUSION AND FUTURE WORK

A. *Summary of Findings*

The presented work describes a new principle-driven and performance-focused strategy based on the use of Bayesian Optimization (BO) to automate hyperparameter tuning in Deep Q-Learning Networks (DQNs), especially for real-time robotic navigation systems. With the development of the BO-DQN framework, we showed that aggressive data-efficient hyperparameter tuning does not only make learning more efficient but yields better navigation policies in simulations and real robot environments. The simulations as well as real robotic navigation systems validated the benefits of BO for hyperparameter tuning as opposed to traditional grid and random search methods. BO-DQN was shown to achieve faster convergence, higher cumulative rewards, and more robust policies out of all the different environments, including the more complex obstacle and dynamical environments. The tuned models demonstrated greater task performance after fewer episodes, leading to sample efficiency and real-world relevance.

Our sensitivity analysis reaffirmed that the hyperparameters of learning rate, exploration rate, and network architecture have deep impact on model performance and Bayesian techniques are more efficient at locating optimal regions in these parameter spaces than blind or exhaustive searches. The focus on probabilistic modelling not only enabled better tuning but also understanding the drivers of performance in DQNs for navigation tasks.

B. Implications for Robotic Learning Automation

The consequences of this research are oriented towards the automation and deployment of robotic learning, which is sophisticated in its implications. In practical robotics applications, where prototyping comes at a high cost and reckless actions can result in damage to the hardware, making the most effective use of the learning pipeline becomes the most challenging constraint. Discriminative BO-DQN solves this problem because it allows robots to improve their behaviours more quickly with fewer samples, and even more importantly, with very restrictive available compute power.

Our results lead us to advocate for the combined use of Deep Q-Learning and BO for mobile robot platforms, particularly where both discrete decision making and reactive planning are required. This encourages autonomous robotic systems to change their learning process without manual tuning which minimizes the need for expert help. In addition, the integration and posterior uncertainty estimates in addition to acquisition functions makes the process transparent and trustable when deploying learning agents in sensitive environments.

With the increase in utilization of robots for tasks like warehouse logistics, surveillance, healthcare robotics, and delivery, there is a need to automate the learning process as well. Minimally Adaptive Open-Ended BO-DQN (BO-DQN) provides a modular framework that is easy to extend and deploy; it can be integrated into robotic pipelines with little changes to the overall architecture and control flow. Its versatility to many different hardware setups, environments, and tasks makes it a valuable system's infrastructure for supporting scalable autonomous learning.

C. Future Work: Multi-Agent Navigation, Real-Time BO Integration

This work was cantered around single-agent navigation with a static BO in offline training, however, there are a number of exciting possibilities for further investigations. There is one extension which appears to be straightforward which is extending the BO-DQN framework to include multi-agent navigation. In these situations, agents will often need to work together or compete in the same environment, which adds another layer of difficulty for policy optimization. Multi-agent reinforcement learning (MARL) environments typically have training instability because of the non-stationary behaviour of other agents. The use of BO might help hyperparameter tuning across cooperative and adversarial scenarios and allow for more effective scalable policy learning in swarm robotics or autonomous fleet control.

Integrating real-time Bayesian Optimization into the training loop is another key focus area. Existing BO techniques are almost exclusively batch and offline, but real-time BO would mean hyperparameters could be altered during system deployment. That would permit robotic systems to deal with drift in dynamics, changes in the environment, or shifting task goals without incurring full retraining overhead. Current developments in streaming Bayesian Optimization as well as contextual bandit learning may provide a relevant solution.

Also, I examine future designs where BO is blended with other optimization methods like BO with gradient-based or evolutionary search to harness other features of the performance landscape. The possibility of using transferable hyperparameter priors for related tasks, environments or even types of hardware will be studied which may facilitate efficient policy porting in modular robotic systems.

In a systems view, constraints in real-life such as energy consumption, thermal limits, and smoothing of actuation also have to be considered and optimized in an objective function of BO. This turns the imperfect system into a multi-objective optimization problem which looks for a trade-off between learning quality, longevity, and safety of the system – which brings the system one step closer toward autonomous and intelligent robotic systems.

The work demonstrates the importance of BO in learning robotics for the new era by deep learning termed as BO-DQN that integrates a methodical approach to addressing reinforcement learning. In the case where learning is achieved by robots, complex and dynamic environments can be mapped and understood within a short span of time. Further modifications to the proposed system will render it a self-sufficient learning system, which would expedite the adoption of robotic systems in various sectors.

References

[1] Silver, David, et al. "Mastering the game of go without human knowledge." nature 550.7676 (2017): 354-359.

[2] Levine, Sergey, et al. "End-to-end training of deep visuomotor policies." Journal of Machine Learning Research 17.39 (2016): 1-40.

[3] Mnih, Volodymyr, et al. "Human-level control through deep reinforcement learning." nature 518.7540 (2015): 529-533.

[4] Tai, Lei, Giuseppe Paolo, and Ming Liu. "Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation." 2017 IEEE/RSJ international conference on intelligent robots and systems (IROS). IEEE, 2017.

[5] Henderson, Peter, et al. "Deep reinforcement learning that matters." Proceedings of the AAAI conference on artificial intelligence. Vol. 32. No. 1. 2018.

[6] Hester, Todd, et al. "Deep q-learning from demonstrations." Proceedings of the AAAI conference on artificial intelligence. Vol. 32. No. 1. 2018.

[7] Fujimoto, Scott, Herke Hoof, and David Meger. "Addressing function approximation error in actor-critic methods." International conference on machine learning. PMLR, 2018.

[8] François-Lavet, Vincent, et al. "An introduction to deep reinforcement learning." Foundations and Trends® in Machine Learning 11.3-4 (2018): 219-354.

[9] Shahriari, Bobak, et al. "Taking the human out of the loop: A review of Bayesian optimization." Proceedings of the IEEE 104.1 (2015): 148-175.

[10] Frazier, Peter I. "A tutorial on Bayesian optimization." arXiv preprint arXiv:1807.02811 (2018).

[11] Calandra, Roberto, et al. "Bayesian optimization for learning gaits under uncertainty: An experimental comparison on a dynamic bipedal walker." Annals of Mathematics and Artificial Intelligence 76 (2016): 5-23.

[12] Van Hasselt, Hado, Arthur Guez, and David Silver. "Deep reinforcement learning with double q-learning." Proceedings of the AAAI conference on artificial intelligence. Vol. 30. No. 1. 2016.

[13] Tai, Lei, Giuseppe Paolo, and Ming Liu. "Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation." 2017 IEEE/RSJ international conference on intelligent robots and systems (IROS). IEEE, 2017.

[14] Zhu, Kai, and Tao Zhang. "Deep reinforcement learning based mobile robot navigation: A review." Tsinghua Science and Technology 26.5 (2021): 674-691.

[15] Zhang, Chiyuan, et al. "A study on overfitting in deep reinforcement learning." arXiv preprint arXiv:1804.06893 (2018).

[16] Mania, Horia, Aurelia Guy, and Benjamin Recht. "Simple random search provides a competitive approach to reinforcement learning." arXiv preprint arXiv:1803.07055 (2018).

[17] Hessel, Matteo, et al. "Rainbow: Combining improvements in deep reinforcement learning." Proceedings of the AAAI conference on artificial intelligence. Vol. 32. No. 1. 2018.

[18] Bostan, Alin, et al. "Stieltjes moment sequences for pattern-avoiding permutations." arXiv preprint arXiv:2001.00393 (2020).

[19] Falkner, Stefan, Aaron Klein, and Frank Hutter. "BOHB: Robust and efficient hyperparameter optimization at scale." International conference

on machine learning. PMLR, 2018.

[20] Yu, Yue, et al. "Obstacle avoidance method based on double DQN for agricultural robots." Computers and electronics in agriculture 204 (2023): 107546.

[21] Naya, Katsumi, et al. "Spiking neural network discovers energy-efficient hexapod motion in deep reinforcement learning." Ieee Access 9 (2021): 150345-150354.

[22] Chen, Guangda, et al. "Deep reinforcement learning of map-based obstacle avoidance for mobile robot navigation." SN Computer Science 2 (2021): 1-14.

[23] Jeng, Shyr-Long, and Chienhsun Chiang. "End-to-end autonomous navigation based on deep reinforcement learning with a survival penalty function." Sensors 23.20 (2023): 8651.