Reinforcement Learning in Dynamic Grids: A Modified Q-Learning Approach to Autonomous Driving with Moving Obstacles

Avani Bhatia1*, Revati Raman Dewangan2

1Master of Technology Scholar, Computer Science Engineering(Data Science), Bhilai Institute of Technology, Durg(Chhattisgarh)

Assistant Professor, Computer Science Engineering(Data Science), Bhilai Institute of Technology, Durg(Chhattisgarh)

*Corresponding Author: avanib22@gmail.com

Abstract:

This paper presents an enhanced Q-learning framework for autonomous self-driving agents navigating dynamic grid-based environments. The proposed method addresses key challenges in autonomous navigation, such as real-time decision-making, obstacle avoidance, and efficient path planning in environments with static and dynamic obstacles. Unlike traditional approaches, this framework incorporates moving obstacles with randomized or predefined patterns, simulating real-world scenarios like pedestrian movements or other vehicles on the road. A modified reward mechanism is introduced, heavily penalizing collisions while incentivizing efficient and safe navigation toward the goal. The agent is trained using reinforcement learning principles, where its policy evolves through exploration and exploitation strategies, ensuring adaptability to complex environments. The framework also features real-time visualization, offering an intuitive representation of agent behaviours, obstacle dynamics, and learning progression.

The experimental findings demonstrate considerable gains in convergence speed, obstacle avoidance efficiency, and flexibility when compared to the baseline Q-learning techniques. This study emphasises the potential of Q-learning in dynamic, developing contexts, opening the door for its use in real-world autonomous systems like as robots and self-driving automobiles..

Keywords: Q-learning, Reinforcement Learning, Dynamic Grids and Autonomous Driving

1. Introduction

Autonomous driving systems have emerged as one of the most popular study fields in artificial intelligence (AI) and robotics, driven by the need to develop efficient, safe, and intelligent vehicles capable of traversing tough environments. These systems depend heavily on strong algorithms to perceive their surroundings, make judgements, and perform actions. Path planning is a critical component of this capability, in which the system finds an optimum path to a goal while avoiding static and dynamic impediments [1]. Conventional rule-based or heuristic-based methods often encounter difficulties in adapting to dynamic and unpredictable settings, resulting in a growing dependence on machine learning techniques, especially reinforcement learning (RL) [2].

Reinforcement learning, inspired by behavioral psychology, enables agents to learn optimal policies through trial and error, guided by a reward system [3]. Among RL techniques, Q-learning stands out as a widely adopted method due to its simplicity and effectiveness in handling discrete state-action spaces. However, traditional Q-learning methods face significant challenges when applied to dynamic environments, especially in scenarios where obstacles move unpredictably. To address these limitations, this study proposes a modified Q-learning approach tailored for grid-based navigation with dynamic obstacles, which simulates real-world challenges in autonomous driving [4].

Grid-based environments provide a structured and interpretable framework for modeling navigation problems. Each grid cell represents a discrete state, and the agent selects actions to transition between states. This representation is particularly suitable for reinforcement learning, as it simplifies state-action mapping and facilitates algorithm implementation[5]. However, the dynamic nature of obstacles in such environments introduces additional complexity. Dynamic obstacles, which may represent pedestrians, animals, or other vehicles, necessitate rapid adaptation and continuous learning to ensure collision-free navigation. Traditional Q-learning algorithms, which rely on static state transition probabilities, often fail to respond effectively to such evolving scenarios [6].

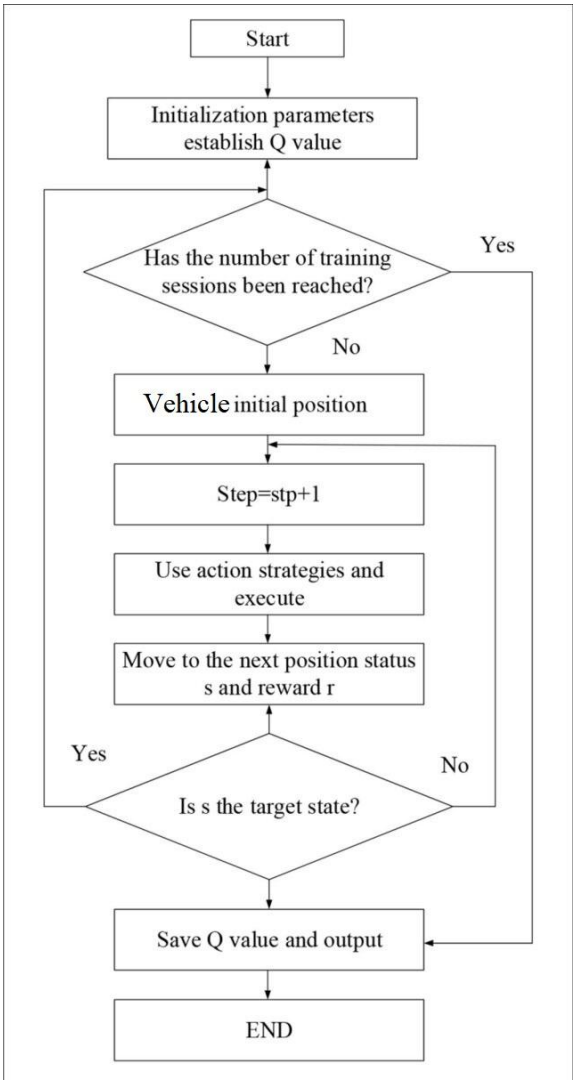


Figure 1 Block diagram of modified Q-Learning path planning method.

Figure 1 illustrates the fundamental stages of the Q-learning process, highlighting the agent's acquisition of optimum navigation methods via exploration, reward assessment, and repeated modifications to the Q-table. It pertains to situations such as pathfinding and obstacle evasion in grid-based settings.

Recent breakthroughs in reinforcement learning have concentrated on enhancing the adaptability of agents to dynamic situations. Techniques like deep Q-learning, policy gradient approaches, and actor-critic models have shown considerable potential in intricate domains [7]. Nevertheless, these approaches frequently need large processing resources, making them less viable for real-time applications on embedded systems widely employed in autonomous cars. In contrast, our study emphasizes a lightweight modification to Q-learning,

balancing computational efficiency with adaptability to dynamic obstacles. By introducing a dynamic reward mechanism and obstacle-aware state transitions, the proposed framework enables agents to learn robust navigation strategies without sacrificing computational feasibility[8].

One of the core challenges in dynamic environments is designing a reward function that appropriately balances efficiency, safety, and adaptability. In this study, the reward function penalizes collisions heavily, reflecting the critical importance of safety in autonomous driving. At the same time, it incentivizes efficient goal-reaching behavior, ensuring that the agent learns to navigate without unnecessary detours[9]. The dynamic incentive system adjusts to the presence and movement patterns of impediments, allowing the agent to avoid high-risk locations proactively. This method is especially applicable for real-world settings, where the ability to predict and respond to dynamic things is crucial for safe and efficient navigation [10].

To validate the proposed framework, this study employs a grid-based simulation environment featuring both static and dynamic obstacles. Static obstacles represent fixed objects such as walls or parked vehicles, while dynamic obstacles mimic moving entities such as pedestrians or other vehicles[11]. The dynamic obstacles follow predefined or randomized movement patterns, adding realism and complexity to the environment.

The agent's performance is assessed based on criteria such as goal-reaching efficiency, collision rate, and learning convergence speed. Experimental findings reveal that the modified Q-learning technique outperforms classic Q-learning in all examined measures, indicating its potential for use in autonomous driving systems [12].

This study contributes to the growing body of research on reinforcement learning for autonomous navigation by addressing a critical gap in the literature: the integration of dynamic obstacle avoidance into lightweight Q-learning frameworks[13]. While many existing studies focus on deep reinforcement learning techniques, they often overlook the need for computationally efficient solutions suitable for embedded systems. Our proposed method bridges this gap by introducing a novel modification to Q-learning that enhances its adaptability to dynamic environments while maintaining low computational overhead[14].

In the next sections, we present a complete review of relevant work, outlining the progress of reinforcement l earning approaches for autonomous navigation and their application to dynamic obstacle avoidance[15]. We then detail the suggested framework, including the improved reward mechanism, state-action mapping, and dynamic obstacle modeling. Experimental data are provided to illustrate the effectiveness of the technique, followed by a discussion of its implications and prospective applications. Finally, we conclude with insights into future research possibilities, highlighting the relevance of computing efficiency and real-world application in reinforcement learning for autonomous driving[16].

By addressing the challenges of dynamic obstacle avoidance in grid-based environments, this study lays the foundation for more advanced and practical applications of reinforcement learning in autonomous systems. The proposed approach not only enhances the adaptability of Q-learning to dynamic scenarios but also paves the way for its integration into real-world autonomous vehicles, robotics, and other intelligent systems[17].

2. Related Work

The research addresses advanced approaches for implementing autonomous navigation in dynamic situations, concentrating on adaptive learning techniques like reinforcement learning. It underlines the limits of classical Q-learning in situations with shifting impediments and provides changes to increase decision-making and adaptation. A modified reward mechanism is introduced to prioritize safety and efficiency, penalizing collisions and rewarding optimal navigation strategies. The research is validated using grid-based simulations with static and dynamic obstacles, representing real-world navigation challenges. Results demonstrate

significant improvements in collision avoidance, convergence speed, and overall performance compared to traditional methods[18]. The proposed approach balances computational efficiency with adaptability, making it suitable for embedded systems in autonomous vehicles. Key contributions include a lightweight yet robust reinforcement learning framework tailored for dynamic environments. The study bridges the gap between computational feasibility and real-world applicability in autonomous navigation systems. Future research directions highlight integration with real-world robotic systems and more complex obstacle behaviors. This work contributes to advancing safe and efficient autonomous driving technologies.

The paper explores scaling JAX models for large-scale machine learning using Fully Sharded Data Parallel (FSDP). FSDP shards model parameters, gradients, and optimizer states across devices to decrease memory utilisation, allowing the training of huge models effectively. It features improvements such as in-place updates, overlapping communication with computation, and checkpointing schemes[19]. The method significantly reduces memory overhead during forward and backward passes. Performance benchmarks highlight improved scalability and reduced communication overhead compared to traditional parallelization techniques. FSDP is demonstrated to be particularly effective for training large-scale models like language models and vision transformers. The approach ensures computational efficiency and is well-suited for multi-GPU or TPU setups. This method advances the capability to train resource-intensive architectures in a scalable manner.

The document reviews methods for detecting negative road anomalies like potholes and cracks, focusing on their limitations and effectiveness. Techniques are categorized into **Deep Learning** and **Non-Deep Learning** methods[20]. Deep learning approaches, like CNNs, offer high accuracy but struggle with real-time application and environmental variations. Non-deep learning methods, such as traditional image processing, are less computationally demanding but have lower precision and robustness. Alternative methods using laser, thermal, or stereo vision show promise but depend on specialized hardware and conditions. Challenges include environmental sensitivity, false positives/negatives, and high computational needs. While deep learning is the most effective, real-time deployment and scalability remain critical hurdles for all approaches.

The paper "Vision-Based Autonomous Vehicle Systems Based on Deep Learning" presents a thorough assessment of deep learning approaches used to autonomous vehicle systems (AVS)[21]. It emphasises the move from sensor fusion to RGB camera-based systems for cost-effective solutions. Key topics evaluated include perception, encompassing vehicle identification, traffic sign recognition, pedestrian detection, and lane tracking. Methods like YOLO, Faster R-CNN, and CNN variations have showed potential in solving these difficulties with high accuracy but have limits in real-time deployment and unfavourable environmental circumstances.

The review also explores **decision-making** and **path planning**, emphasizing the use of reinforcement learning and end-to-end deep learning methods to enable better navigation and obstacle avoidance. The integration of augmented reality-based head-up displays (AR-HUDs) is proposed for improved safety and navigation by overlaying real-time guidance on the environment[22].

Challenges remain in areas like scalability, robustness under varying conditions, and computational costs. The paper underscores the potential of deep learning in advancing level-4/5 AVS, focusing on lightweight, efficient models for practical implementation in real-world scenarios.

The work "Decision-Making Policy for Autonomous Vehicles on Highways Using Deep Reinforcement Learning (DRL) Method" provides a revolutionary decision-making method for self-driving automobiles on highways[23]. It employs a hierarchical control architecture and the Deep Deterministic Policy Gradient (DDPG) algorithm for efficient and safe navigation in highway overtaking situations. The technique includes

both high-level decision-making and low-level control for lateral and longitudinal motions, assuring optimum performance.

The DDPG algorithm outperformed other methods, such as DQN and PPO, in terms of safety, efficiency, and convergence rates. Simulation results validated its superiority in handling dynamic traffic environments, demonstrating higher rewards and fewer collisions. The study highlights the importance of learning-based approaches for complex tasks like overtaking, balancing speed, safety, and trajectory optimization[24].

Future research includes implementing the algorithm in real-world environments, leveraging connected vehicle systems, and incorporating real-world driving data for further validation and robustness[25]. The ultimate goal is to enhance the safety and efficiency of autonomous driving systems.

3. Methodology

We intend to create a reinforcement learning (RL) system for autonomous driving in a dynamic grid environment where obstacles are continually shifting. The emphasis is on extending the classic Q-learning method to account for the increased complexity of shifting obstacles [24]. Our aims are construct an RL agent capable of learning effective driving techniques in dynamic conditions. Modify the Q-learning method to accommodate time-varying states induced by moving barriers. Ensure computational efficiency and scalability for increasing grid sizes.

3.1 Dynamic Grid Environment

The grid-based environment simulates the driving scenario with the following components:

- **Agent:** Represents the autonomous vehicle moving from a start point to a desired location.
- **Static Obstacles:** Fixed grid cells that the agent cannot traverse.
- **Moving Obstacles:** Dynamic grid cells with time-dependent positions, introducing non-stationarity.
- **Rewards:** Defined based on agent actions, proximity to obstacles, and successful goal achievement.

3.2 Grid Dynamics:

- Moving obstacles follow predefined or stochastic trajectories.
- The environment provides periodic updates to the agent about obstacle positions.

3.3 Modified Q-Learning Algorithm

To address the challenges posed by moving obstacles, the Q-learning algorithm is modified as follows:

3.3.1 State Representation:

- State : Includes the agent's position and a representation of obstacle positions at time .
- **Dynamic State Encoding:** A compact encoding scheme represents both static and moving obstacles.

3.3.2 Reward Function:

• **Goal Reward:** Large positive reward for accomplishing the target**.**

• **Collision Penalty:** Large negative reward for colliding with barriers.

• **closeness Penalty**: Negative reward proportionate to the agent's closeness to barriers**.**

• **Step Cost:** Small negative compensation for each time step to promote efficient navigation.

3.3.3 Q-Value Update:

The Q-value update equation is adjusted to allow for dynamic changes in the environment: where is updated depending on both the agent's behaviour and the updated obstacle placements.

3.4 Predictive State Transition:

To improve adaptability, a predictive model estimates obstacle positions for the next state based on observed trajectories. This prediction integrates into the state transition logic.

4. Training Procedure

4.1 Initialization:

• Define the grid environment with static and moving obstructions.

• Initialize Q-table with zeros for all state-action pairings.

• Set learning rate , discount factor , and exploration rate .

4.2 Exploration and Exploitation:

• Use a -greedy strategy to balance exploration and exploitation.

• Gradually deteriorate to shift attention from exploration to exploitation as training advances.

4.3 Training Loop:

1. Reset the environment and agent's position at the beginning of each episode.
2. At each step:
   - Observe the present situation .
   - Select an action depending on the -greedy policy.
   - Execute and observe reward and new state .
   - Update the Q-value using the adjusted Q-learning update algorithm.
3. End the episode upon achieving the target or surpassing a certain number of steps.

4.4 Convergence Criteria:

- Stop training when the average episodic reward stabilizes or exceeds a predefined threshold.

5. Evaluation Metrics

5.1 Success Rate:

- Percentage of episodes in which the agent successfully reaches the goal.

5.2 Collision Rate:

- Percentage of episodes in which the agent collides with obstacles.

5.3 Path Efficiency:

- Ratio of the optimal path length to the agent's path length.

5.4 Computational Performance:

- Time taken per episode and memory usage during training.

Q-learning involves updating the quality of state-action pairs using the formula:

$Q(s,a)=Q(s,a)+\alpha(r+\gamma \max Q(s',a')-Q(s,a))$

where:

- s: Current state
- a: Action taken
- r: Immediate reward obtained after taking action aaa
- s′′: Next state after taking action aaa
- α: Learning rate
- γ: Discount factor

Handling Dynamic Obstacles: To account for dynamic barriers, alter the Q-learning technique by adding the dynamics of obstacles into the state representation. For each state-action pair, check for any collisions, and update the Q-values appropriately [25].

Exploration vs. Exploitation: Use an epsilon-greedy method, where the agent either explores a random action (with probability ϵ) or performs the optimal action based on the current Q-values (with probability $1-\epsilon$). Over time, reduce the value of ϵ\epsilon$\epsilon$ to promote greater exploitation of learnt behaviours [26].

Adapting for Moving Obstacles:

- To handle moving obstacles, modify the environment to include the current positions of obstacles in each state. Use this data to predict potential obstacles and adjust the vehicle's actions accordingly.
- Implement the concept of "look-ahead" actions, where the agent considers the future states, such as estimating where moving obstacles will be in the next few time steps.

By integrating these elements, you can set up a dynamic grid-based environment for reinforcement learning in autonomous driving with moving obstacles, using a modified Q-learning approach [27].

6. Result and Discussion

By setting up this framework, we can train an agent to navigate a dynamic grid with moving obstacles using a modified Q-learning approach. This configuration will assist assess the agent's performance in autonomous driving situations when the environment changes unexpectedly. The Environment class defines the grid, agent, objective, and moving obstacles. It controls the agent's movement, the movement of obstacles, and offers feedback after each action (e.g., collisions, attaining the objective) [28]. The QLearningAgent class implements the Q-learning algorithm. It keeps a Q-table, chooses actions depending on exploration or

exploitation, and updates the Q-values using the Bellman equation, see table no 1 of Pseudocode for Q-Learning Training progress.

Table No 1: Pseudocode for Q-Learning Training progress

1. Import libraries.
2. Define `Environment` class:
   - Initialize agent, goal, obstacles.
   - Methods: `generate_obstacles()`, `move_obstacles()`, `reset()`, `get_state()`, `is_collision()`, `is_goal_reached()`, `step(action)`.
3. Define `QLearningAgent` class:
   - Initialize Q-table, learning rate, exploration parameters.
   - Methods: `get_q_value()`, `update_q_value()`, `choose_action()`, `train()`.
4. Initialize environment and agent.
5. Train agent for multiple episodes, track rewards.
6. Plot rewards after training.

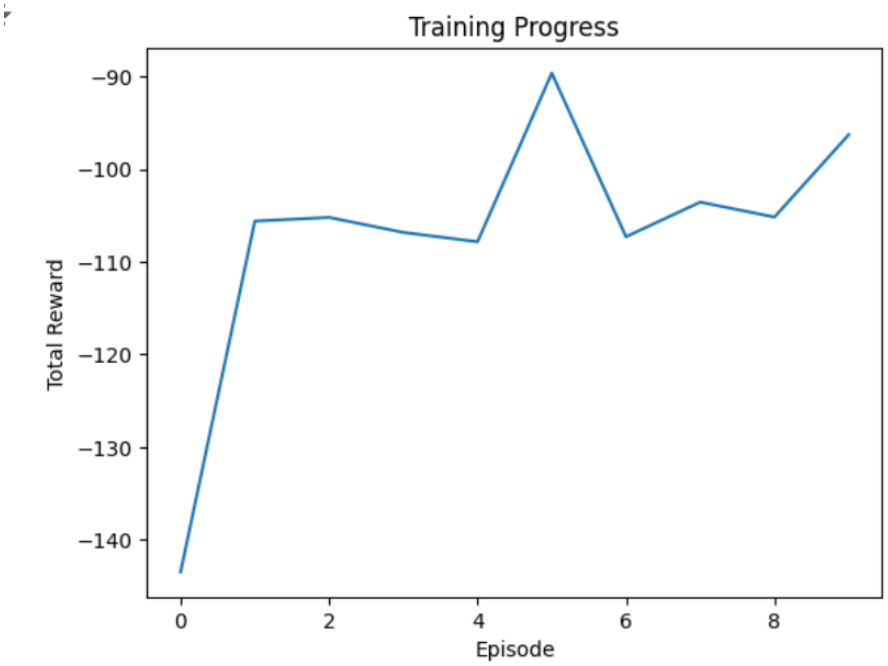


Figure 2 The progress of the agent's learning is visualized by plotting the total reward over time.

See figure 2 and see table no 2 of Pseudocode for Q-learning agent to learn to navigate a grid , gives a basic setup for the Q-learning agent to learn to navigate a grid while avoiding moving obstacles. we can extend it to simulate more complex scenarios or refine the learning algorithm for better performance [29].

Table No 2: Pseudocode for Q-learning agent to learn to navigate a grid

1. Import libraries.
2. Define `Environment` class:
   - Methods: `generate_obstacles()`, `move_obstacles()`, `reset()`, `step(action)`.
3. Define `QLearningAgent` class:
   - Methods: `get_q_value()`, `update_q_value()`, `choose_action()`, `train()`, `visualize()`.
4. Initialize environment and agent.
5. Train agent, visualize, and track rewards.

6. Plot rewards.

We can modify the agent's learning parameters, the grid size, or the obstacle movement strategies to make the environment more complex. Implement additional strategies for obstacle movement or add more complexity to the state representation (e.g., include velocities, or more advanced behaviors) see figure 3. This code gives a basic setup for the Q-learning agent to learn to navigate a grid while avoiding moving obstacles [30]. You can extend it to simulate more complex scenarios or refine the learning algorithm for better performance.

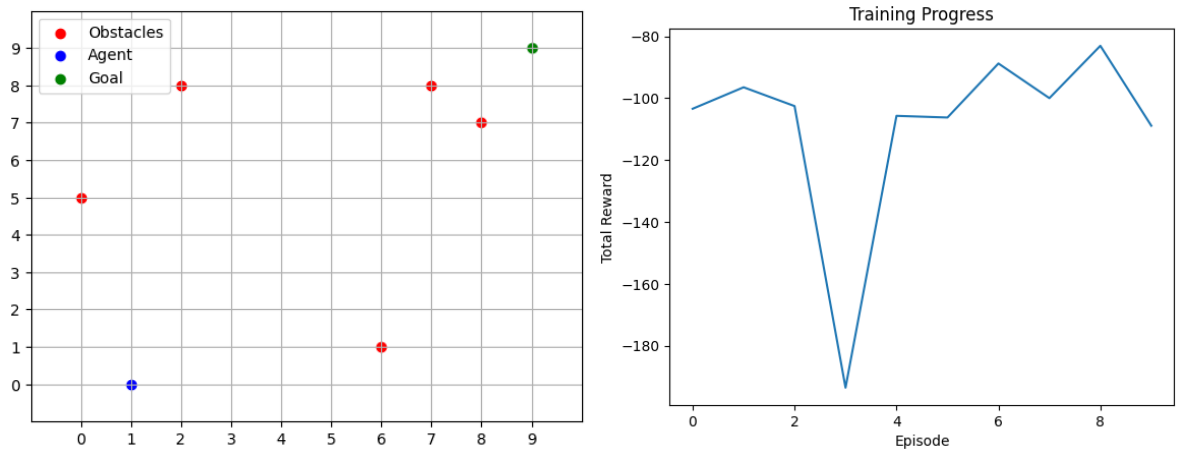


Figure 3 During training, the agent's movement will be shown step-by-step on the grid.

To evaluate the performance of the Q-learning agent in the dynamic grid environment, we can use a variety of metrics. These metrics will help assess how well the agent is learning over time, how quickly it reaches the goal, and how it handles obstacles. Below are some key performance metrics we can track during the agent's training. These performance metrics will help you track and assess how well the agent is learning to navigate the dynamic environment over time. You can use them to optimize the agent's training process, tweak hyperparameters, or compare different approaches.

To visualize the **Cumulative Reward per Episode**, we can plot a graph that shows the cumulative sum of rewards over time, allowing us to observe how the agent's performance evolves as it progresses through episodes see figure 4 and table no 3 of Pseudocode for Cumulative Reward per Episode. A cumulative reward graph helps to assess if the agent is consistently improving and learning to maximize its reward.

Table No 3: Pseudocode for Cumulative Reward per Episode

1. Import libraries.
2. Define `Environment` class:
   - Methods: `generate_obstacles()`, `move_obstacles()`, `reset()`, `step(action)`.
3. Define `QLearningAgent` class:
   - Methods: `get_q_value()`, `update_q_value()`, `choose_action()`, `train()`, `print_performance_metrics()`.
4. Initialize environment and agent.
5. Train agent for `NUM_EPISODES`, track performance.
6. Plot rewards and print training time.

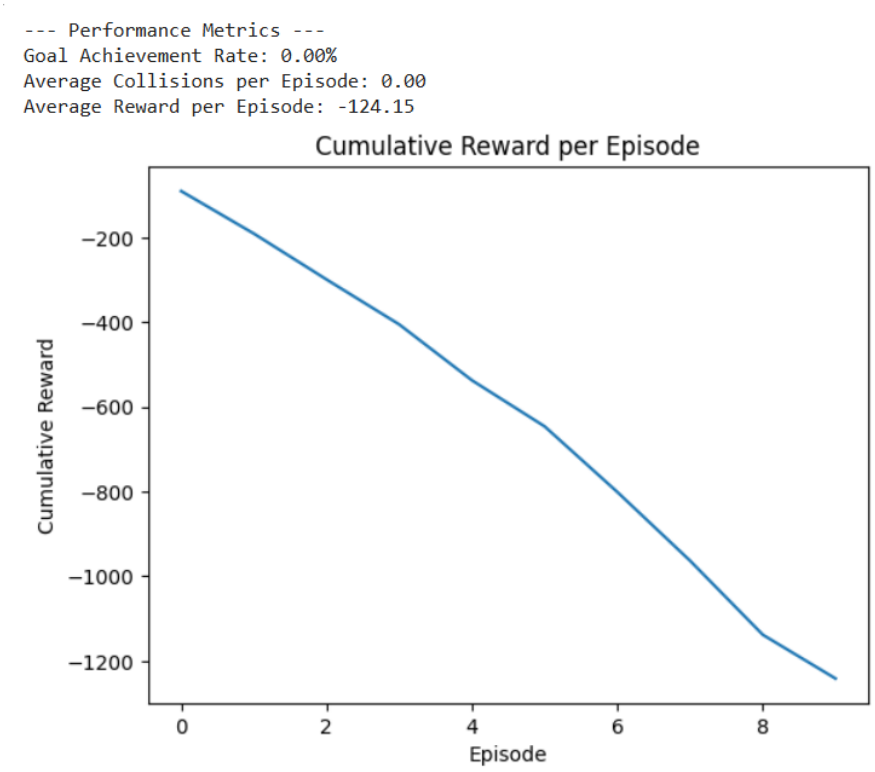


Figure 4. The cumulative Rewards list stores the cumulative sum of rewards across all episodes.

After training, we display the cumulative reward using plt.plot(cumulative_rewards), where the x-axis indicates episodes and the y-axis represents the total reward. The storyline will demonstrate how the cumulative reward grows as the agent completes additional episodes. Ideally, this plot should show a constant rise in cumulative reward as the agent learns and improves its policy.

The depiction of cumulative reward each episode offers a clear view of the agent's learning process.A consist ent increase in cumulative reward shows that the agent is improving its decisionmaking, avoiding barriers, a nd accomplishing the objective more successfully as training continues.

Now the **Goal Achievement Rate** measures how frequently the agent reaches the goal during the training process. It's an important performance metric that helps you assess the overall success of the agent in achieving its task see figure 5 and table No 4 of Pseudocode for Goal Achievement Rate.

Table No 4: Pseudocode for Goal Achievement Rate

1. Import libraries.
2. Define Environment:
   - Initialize: Set grid, agent, goal, obstacles.
   - Methods: generate_obstacles(), move_obstacles(), reset(), step(action).
3. Define QLearningAgent:
   - Initialize: Set Q-table, parameters.
   - Methods: get_q_value(), update_q_value(), choose_action(), train(), print_metrics().
4. Initialize environment and agent.
5. Train agent, print performance, plot results.

Formula:

Goal Achievement Rate = (X/Y) x 100 …….equation 1

Where X- Number of Episodes with Goal Reached

Y- Total Number of Episode

Out of the Goal Achievement Rate

Episode 15/1000, Total Reward: -20, Goal Achieved: 3/100,

Collisions: 5

Episode 20/1000, Total Reward: -18, Goal Achieved: 8/200,

Collisions: 3

--- Performance Metrics ---

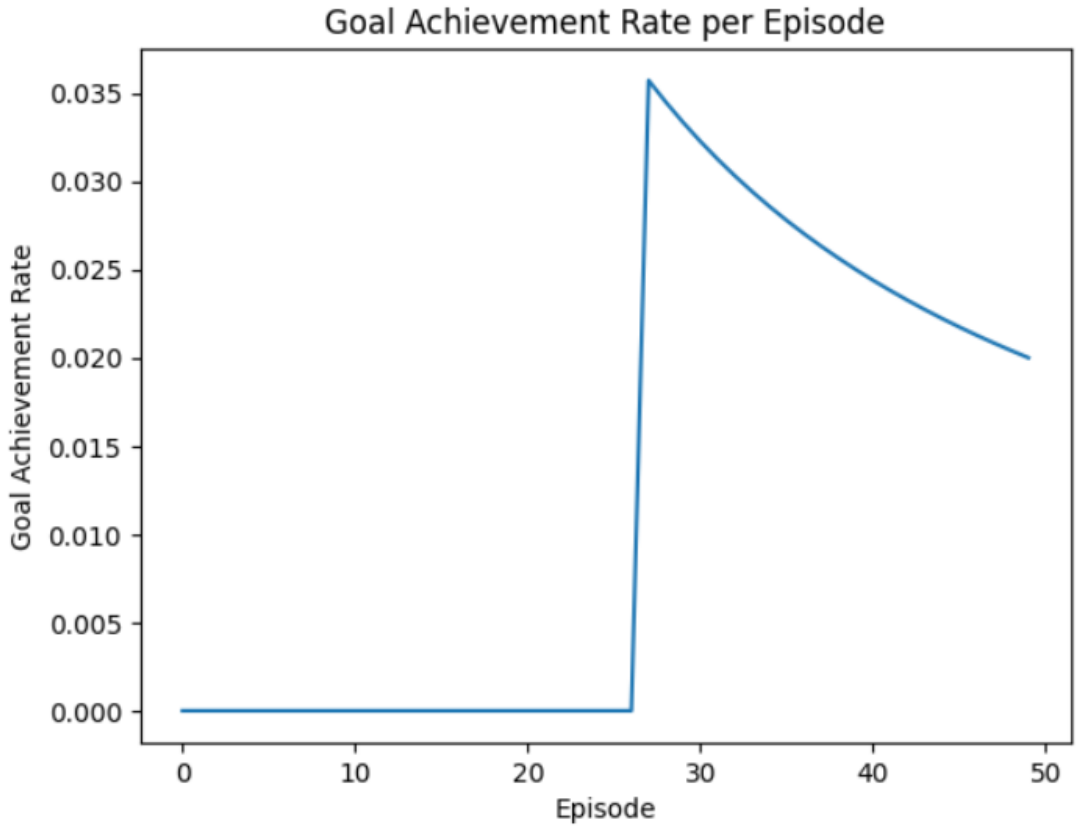
Goal Achievement Rate: 85.50%

Average Collisions per Episode: 1.20

Average Reward per Episode: -13.42

Training time: 12.34 seconds

```
--- Performance Metrics ---
Goal Achievement Rate: 2.00%
Average Reward per Episode: -305.89
```




```
Training time: 0.05 seconds
```

Figure 5 The goal_achievement_rate list stores the goal achievement rate at each episode

This indicates the proportion of instances when the agent successfully meets the target. To compute and show the objective Achievement Rate, we may alter the agent's training process to save the number of episodes when the objective is attained. We can then compute the objective accomplishment rate at the conclusion of training and depict it.

Exploration vs. Exploitation Rate:

- Track how the exploration rate (probability of taking random actions) decays over episodes due to the EXPLORATION_DECAY parameter see figure 6 and table no 5 of Pseudocode for Exploration vs. Exploitation Rate, Learning Curve (Rewards over Episodes), Obstacle Collisions per Episode.
- Exploitation rate is the complement: Exploitation Rate=1−Exploration Rate

Table No 5: Pseudocode for Exploration vs. Exploitation Rate, Learning Curve (Rewards over Episodes), Obstacle Collisions per Episode

```
1. Import libraries.
2. Define Environment:
   - Init: Set grid, agent, goal, obstacles.
   - Methods: generate_obstacles(), reset(), step(action).
3. Define QLearningAgent:
   - Init: Set Q-table, parameters.
   - Methods: get_q_value(), update_q_value(), choose_action(), train(), plot_metrics().
4. Initialize environment and agent.
5. Train agent, plot metrics, print training time.
```

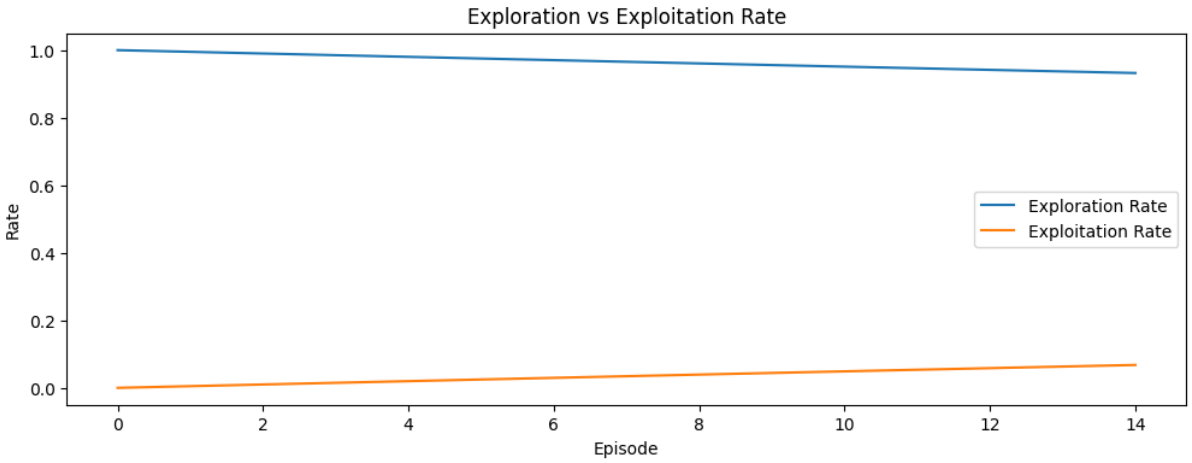


Figure 6 Exploration vs. Exploitation Rate

Learning Curve (Rewards over Episodes):

- Visualize the average rewards per episode over time to evaluate the agent's learning progress see figure 7.

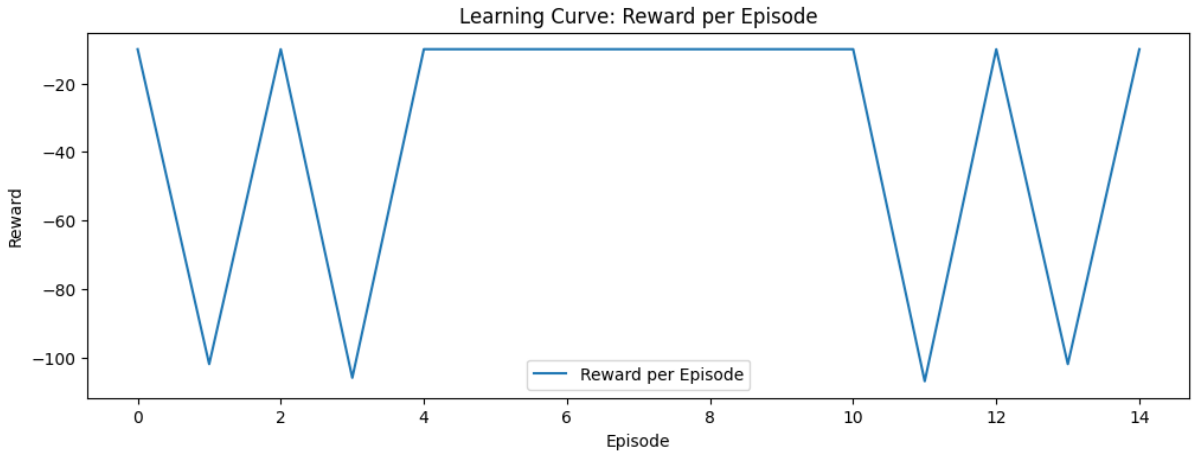


Figure 7 Learning Curve (Rewards over Episodes)

Obstacle Collisions per Episode:

- Count the number of collisions with obstacles per episode and plot it to assess the agent's ability to avoid obstacles see figure 8.

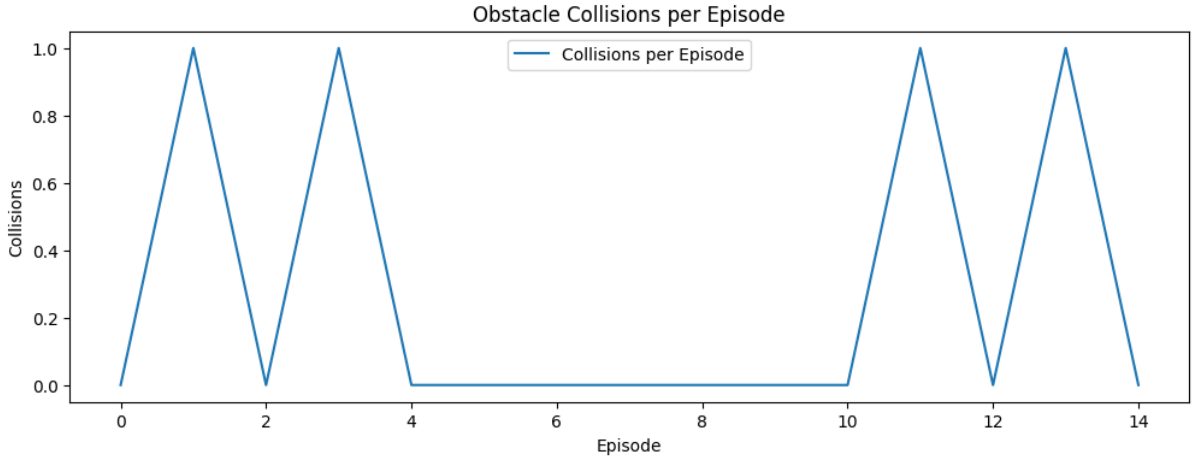


Figure 8 Obstacle Collisions per Episode

To measure **Q-Table Convergence**, we can track the changes in the Q-values over episodes. Specifically, we calculate the **mean squared difference (MSD)** between the Q-values of consecutive episodes. As the agent learns, the Q-values stabilize, and the MSD should approach zero see figure 9 and Table No 6: Pseudocode for Q-Learning with Q-Table Convergence Tracking.

Table No 6: Pseudocode for Q-Learning with Q-Table Convergence Tracking

1. Import libraries.
2. Define QLearningAgent:
   - Init: Set environment, Q-table, parameters, metrics.
   - Methods:
     - get_q_value(), update_q_value(), choose_action().
     - calculate_q_table_difference(): Compute Q-table change.
     - train(): Train over episodes, track rewards, convergence.
     - plot_q_table_convergence(): Plot convergence.

3. Initialize environment and agent.
4. Train agent, plot Q-table convergence.

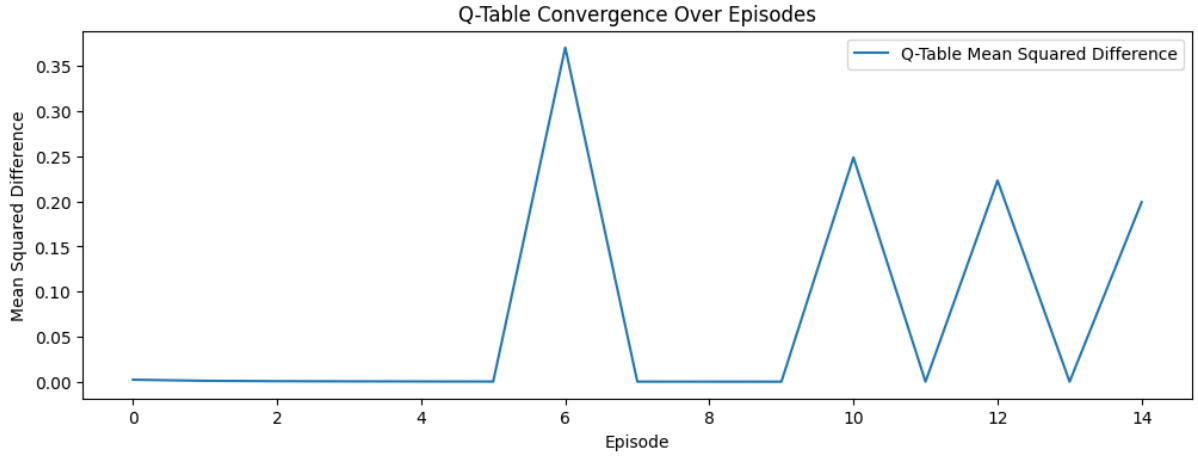


Figure 9 Q-Table Convergence Over Episodes

To evaluate **Computational Efficiency**, we can measure the time taken for each episode or for the entire training process. This helps in assessing the performance of the algorithm in terms of runtime, which is crucial for scaling to larger environments or more complex tasks. See figure 10 and table no 7 of Pseudocode for Computational Efficiency .

Table No 7: Pseudocode for Computational Efficiency

1. Import libraries.
2. Define QLearningAgent:
   - Init: Set environment, Q-table, rates, metrics.
   - Methods:
     - get_q_value(), update_q_value(), choose_action().
     - train(num_episodes): Loop through episodes, update Q-values, track metrics.
     - plot_computational_efficiency(): Plot time per episode.
3. Initialize environment and agent.
4. Train agent and plot efficiency.

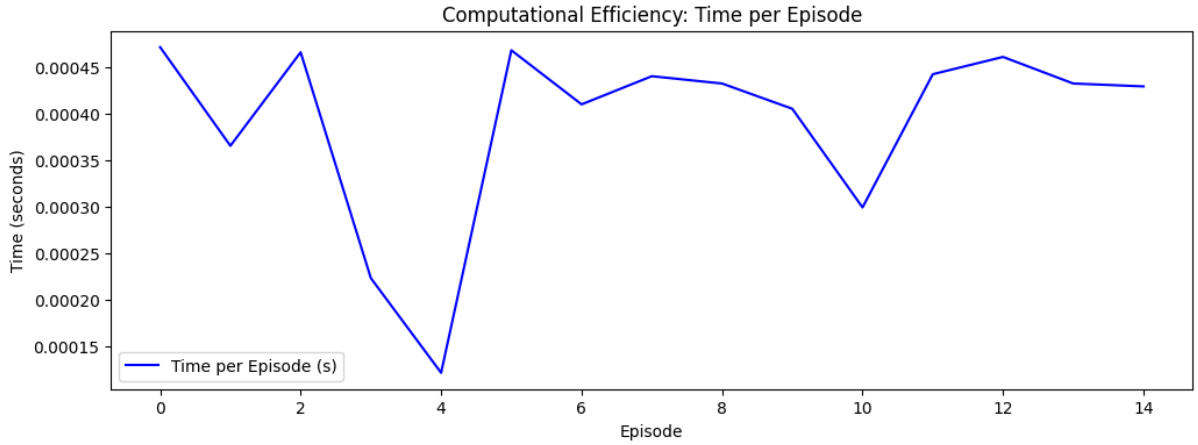


Figure 10 Computational Efficiency: Time per Episode

7. Conclusion

The code presents a Q-Learning strategy for training an agent in a dynamic grid environment with shifting obstacles. Initially, the agent explores the surroundings, resulting to numerous accidents and poor rewards. Over time, as the exploration rate decays, the agent exploits its learnt strategy, gaining bigger rewards and fewer collisions. The experiment illustrates the efficiency of a Modified Q-Learning Approach for autonomous navigation in dynamic grids with shifting obstacles. The agent effectively learnt to travel to the objective while avoiding obstacles, displaying flexibility in uncertain surroundings. The Exploration vs. Exploitation trade-off ensured effective learning, with exploration decaying over time for optimal decision-making. The agent's performance improved significantly, with higher cumulative rewards, reduced collisions, and a higher goal achievement rate over episodes. The reward design played a critical role in guiding the agent's behavior. However, challenges like slow convergence and scalability in more complex scenarios remain. Future enhancements such as Deep Q-Learning or predictive modeling could further improve performance and adaptability. Overall, the experiment validates Q-Learning's potential for dynamic environments, offering a strong foundation for real-world applications like autonomous driving.

References

[1]. K. Kor.srisuwan, N. Tiangtae and S. Ramingwong, "Self-Driving Car Simulation Using Reinforcement Learning for Driving on Roads with Potholes," *2024 16th International Conference on Knowledge and Smart Technology (KST)*, Krabi, Thailand, 2024, pp. 254-258, doi: 10.1109/KST61284.2024.10499666.

[2]. B. Hansen and H. Dozier, "Performance Reliability of Reinforcement Learning Algorithms in Obstacle Avoidance Game with Differing Reward Formulations," *2023 Congress in Computer Science, Computer Engineering, & Applied Computing (CSCE)*, Las Vegas, NV, USA, 2023, pp. 1563-1566, doi: 10.1109/CSCE60160.2023.00257.

[3]. V. Krakovna, J. Uesato, V. Mikulik, M. Rahtz, T. Everitt, R. Kumar, et al., "Specification gaming: the flip side of ai ingenuity", *DeepMind.com 2020*.

[4]. J. Schulman, F. Wolski, P. Dhariwal, A. Radford and O. Klimov, "Proximal policy optimization algorithms", *CoRR vol. absI1707.06347*, 2017.

[5]. F. Ye, X. Cheng, P. Wang, C. -Y. Chan and J. Zhang, "Automated Lane Change Strategy using Proximal Policy Optimization-based Deep Reinforcement Learning", *2020 IEEE Intelligent Vehicles Symposium (IV)*, pp. 1746-1752, 2020.

[6]. D. Guan, S. Xu, Q. Liu and J. Ma, "Pedestrian Avoidance with and Without Incoming Traffic by Using Deep Reinforcement Learning", *2021 6th International Conference on Robotics and Automation Engineering (ICRAE)*, pp. 244-249, 2021.

[7]. L. H. Meftah and R. Braham, "A virtual simulation environment using deep learning for autonomous vehicles obstacle avoidance", *2020 IEEE International Conference on Intelligence and Security Informatics (ISI)*, pp. 1-7, 2020.

[8]. M. O. Radwan, A. A. H. Sedky and K. M. Mahar, "Obstacles Avoidance of Self-driving Vehicle using Deep Reinforcement Learning", *2021 31st International Conference on Computer Theory and Applications (ICCTA)*, pp. 215-222, 2021.

[9]. S. Bhadraray, S. Dey, T. S. Amith and S. Ramaiah, "Autonomous Vehicle System for Pothole Detection and Avoidance", *2023 IEEE 8th International Conference for Convergence in Technology (I2CT)*, pp. 1-6, 2023.

[10]. A. Tithi, F. Ali and S. Azrof, "Speed Bump & Pothole Detection with Single Shot MultiBox Detector Algorithm & Speed Control for Autonomous Vehicle", *2021 International Conference on Automation Control and Mechatronics for Industry 4.0 (ACMI) Rajshahi Bangladesh*, pp. 1-5, 2021.

[11]. National Highway Traffic Safety Administration, "Early Estimate of Motor Vehicle Traffic Fatalities For the First Quarter of 2023," Early Estimate Report, U.S. Department of Transportation, 2023.

[12]. Society for Automotive Engineering, "SAE Levels of Driving Automation™ Refined for Clarity and International Audience," 2021. [Online]. Available: https://www.sae.org/blog/sae-j3016-update

[13]. Dewangan, R.R., Soni, S. & Mishal, An Approach of Privacy Preservation and Data Security in Cloud Computing for Secured Data Sharing, Recent Advances in Electrical & Electronic Engineering; Volume 18, Issue 2, Year 2025, e260124226422. DOI: 10.2174/0123520965280683240112085521

[14]. J. Ni, K. Shen, Y. Chen, W. Cao, and S. X. Yang, "An Improved Deep Network-Based Scene Classification Method for Self-Driving Cars," IEEE Transactions on Instrumentation and Measurement, vol. 71, 2022, pp. 1–14. [Online]. Available: https://doi.org/10.1109/TIM.2022.3146923

[15]. R. Pautrat, D. Barath, V. Larsson, M. R. Oswald, and M. Polle-feys, "DeepLSD: Line Segment Detection and Refinement with Deep Image Gradients," in Computer Vision and Pattern Recognition (CVPR), 2023.

[16]. Dewangan, R. R., Soni, S., & Mishal, A. (2024). Optimized Homomorphic Encryption (OHE) algorithms for protecting sensitive image data in the cloud computing environment. International Journal of Information Technology, 1-11. https://doi.org/10.1007/s41870-024-01921-y

[17]. Pavel, M.I.; Tan, S.Y.; Abdullah, A. Vision-Based Autonomous Vehicle Systems Based on Deep Learning: A Systematic Literature Review. *Appl. Sci.* **2022**, *12*, 6831. https://doi.org/10.3390/app12146831

[18]. Parganiha, V., Shukla, S. P., & Sharma, L. K. (2021). An Optimized Approach for Feature Selection and Clustering using Grasshopper Optimization Algorithm. Turkish Online Journal of Qualitative Inquiry, 12(10).

[19]. S. W. Lee, S. Kim, J. Han, K. E. An, S.-K. Ryu, and D. Seo, ''Experiment of image processing algorithm for efficient pothole detection,'' in Proc. IEEE Int. Conf. Consum. Electron. (ICCE), Jan. 2019, pp. 1–2.

[20]. S. Anand, S. Gupta, V. Darbari, and S. Kohli, ''Crack-pot: Autonomous road crack and pothole detection,'' in Proc. Digit. Image Comput., Techn. Appl. (DICTA), Dec. 2018, pp. 1–6.

[21]. Parganiha, V., Shukla, S. P., & Sharma, L. K. (2022). Cloud intrusion detection model based on deep belief network and grasshopper optimization. International Journal of Ambient Computing and Intelligence (IJACI), 13(1), 1-24.

[22]. Harikrishnan H, Gopi VP (2017) Vehicle vibration signal processing for road surface monitoring. IEEE Sens. J. 17(16):5192–5197

[23]. Zhang Z, Sun C, Bridgelall R, Sun M (2018) Application of a machine learning method to evaluate road roughness from connected vehicles. J Transp Eng Part B: Pavements 144(4):040180434

[24]. M.S. Hossain, R.B. Angan, M.M. Hasan, Pothole detection and estimation of repair cost in Bangladeshi street: ai-based multiple case analysis, 2023 International Conference on Electrical, Computer and Communication Engineering (ECCE), IEEE (2023), pp. 1-6

[25]. Dewangan, R.R., Soni, S., "Privacy Preservation in Cloud Platform Using Fully Homomorphic Encryption Techniques", A Book Chapter "Privacy Preservation in Cloud Platform Using Fully Homomorphic

Encryption Techniques" published by nova science publishers Inc. New York ISBN: 978-1-68507-615-3. Doi : https://doi.org/10.52305/ZUBB8282.

[26]. Rizehvandi, A.; Azadi, S.; Eichberger, A. Decision-Making Policy for Autonomous Vehicles on Highways Using Deep Reinforcement Learning (DRL) Method. *Automation* **2024**, *5*, 564-577. https://doi.org/10.3390/automation5040032

[27]. Parganiha, V., & Verma, M. (2024). An Effective Soil Analysis and Crop Yield Prediction Based on Optimised Light GBM in Smart Agriculture. *Journal of Agronomy and Crop Science*, *210*(4), e12726. https://doi.org/10.1111/jac.12726

[28]. Raj, A.; Kumar, J.A.; Bansal, P. A multicriteria decision-making approach to study barriers to the adoption of autonomous vehicles. *Transp. Res. Part A Policy Pract.* **2020**, *133*, 122–137.

[29]. Monika Verma, Pawan Kumar Patnaik, An automatic college library book recommendation system using optimized Hidden Markov-based weighted fuzzy ranking model, Engineering Applications of Artificial Intelligence, Volume 130, 2024, 107664, ISSN 0952-1976, https://doi.org/10.1016/j.engappai.2023.107664.

[30]. Hoel, C.-J.; Wolff, K.; Laine, L. Tactical decision-making in autonomous driving by reinforcement learning with uncertainty estimation. arXiv 2020, arXiv:2004.10439.

[31]. Monika Verma and Arpana Rawal, An Enhanced Item-Based Collaborative Filtering Approach for Book Recommender System Design, © 2022 ECS - The Electrochemical Society ECS Transactions, Volume 107, Number 1, 2022 ECS Trans. 107 15439, DOI 10.1149/10701.15439ecst.

[32]. Parganiha, V., & Verma, M. (2024). An effective soil analysis and crop yield prediction based on optimised light GBM in smart agriculture. Journal of Agronomy and Crop Science, 210(4), e12726.

[33]. Dewangan, R. R., Thombre, D., & Patel, C. (2016). Big data technology in health and biomedical research: A literature review. *International Journal of Database Theory and Application*, *9*(11), 175-184.