# Reinforcement Learning approach for decision-making in driver control shifting for semi-autonomous driving

Evaldas Latoškinas
Delft University of Technology
Rijswijk, the Netherlands

**Abstract**

Semi-autonomous driving innovations aim to bridge the gap to fully autonomous driving by co-operating with human drivers to lead to optimal choices on who should drive in different scenarios by offering different automation levels. However, in the present day, known semi-autonomous driving solutions do not generalise to every complex case of driver and AI interaction. This limitation prompted research in attempting to solve the problem using artificial intelligence and machine learning techniques. This paper focuses on providing a reinforcement learning approach to solve one specific decision-making scenario of the driver initiating a shift of control to a different automation level. The decision problem was formulated as a Markov Decision Process, and the problem was solved both by a baseline handcrafted decision tree and a learned reinforcement learning policy using the DQN algorithm. The two policies were compared based on safety, comfort and efficiency metrics in a simulated driving environment. The results were indicative that a reinforcement learning policy generally ensured safety & comfort and has shown increased efficiency over the baseline policy, however, it faced efficiency & comfort issues in outlier cases.

**Keywords:** Reinforcement learning, deep learning, machine learning, AI-based decision making, Markov Decision Process, semi-autonomous driving

## 1 Introduction

With the emergence of self-driving cars, the safety and trust of the novel invention have been of great debate. At present, no consensus has been reached on whether autonomous driving is considered safer than manual driving and how such a comparison could be made reliably [1]. Due to the novelty, recency and lack of certainty in autonomous driving, mutual trust remains a central issue in the field [2], with driver trust varying in different driving scenarios, human and environmental factors [3], [4]. The issues have prompted the emergence of the MEDIATOR[1] project, which

[1]MEDIATOR project - https://mediatorproject.eu/

aims to bridge the gap between manual and autonomous driving. Rather than having cars operate without a driver entirely, semi-automated driving uses multiple automation modes representing different degrees of artificial intelligence autonomy, with lower degrees of automation delegating more tasks to the human driver. Semi-autonomous driving involves a variety of complex decision tasks, such as deciding when to intervene in driving or which automation level to choose to maximise security and comfort for the driver [5]. While general formulations for switching the control between the human and artificial intelligence have been proposed [6], [7], the utilization of artificial intelligence for specific complex scenarios remains an area to be explored.

The research is scoped to reinforcement learning techniques to address novel decision-making problems relevant to the MEDIATOR system. Reinforcement learning (RL) is a widely researched technique, commonly used to find optimal policies for action planning and optimal control [8], with problems generally characterized as Markov Decision Processes (MDP) [9, p. 1-16]. Research has shown that MDPs are applicable in simulating a variety of driving-related decision problems: highway traffic [10], behaviour planning for cars [11] and driver-automation switching in semi-autonomous driving [7]. Recently, RL approaches have been combined with deep learning methods, leading to more effective deep reinforcement learning (DRL) [12] algorithms capable of learning efficient policies in state spaces that would otherwise be computationally infeasible to enumerate. DRL has since been utilised to provide proof of concept solutions to various complex decision-making problems, such as enabling robots and humans to achieve a common goal together efficiently [13], finding optimal policies for motion planning and complex navigation tasks in autonomous driving [14] and autonomous driving within traffic [15]. These previous works suggest that reinforcement learning is suitable for solving semi-autonomous driving decision problems modelled as MDPs. Since the area of semi-autonomous driving is novel and the algorithms are generally applicable to any MDP, the main challenge of the research lies in the formulation & modelling of decision problems.

The paper raises the following hypothesis as the research question: *A reinforcement learning policy learned from a Markov Decision Process is an efficient and feasible decision-making solution for the specific scenario of the driver initiating a shift to the desired automation level.* Validation of this hypothesis guides future work in decision-making problems in semi-autonomous driving by giving further insight into the applicability of machine learning methods for specific driver-AI interaction decision tasks. The results of the research provide a baseline that can be used for comparison, allowing novel solutions to build on top of the research covered in this paper by employing simpler or more accurate solutions and extensions. The output of the research includes a Markov Decision Process formulation of the decision problem and an RL algorithm solving the MDP. Validation of the hypothesis is performed by comparing driving safety & comfort, as well as operational efficiency metrics collected on both the learned policy and a handcrafted decision tree policy.

# 2 Methodology

The research followed a three-step process. First, the problem was formulated as a Markov Decision Process. Then, policies were defined and implemented to solve the MDP. Finally, the experiment was designed and executed to yield results for comparison. The section will focus on the first two steps, while the details of the last step will be described in the follow-up section.

## 2.1 Problem formulation

The study focused on the use case of the driver shifting control to a new automation level in a vehicle. Hence, the decision-making problem can be formulated as: *Which action is the optimal choice when the vehicle is presented with a driver request?* To formulate this decision making problem precisely, the MDP model was chosen, which allows describing stochastic environments, how they change based on actions taken and what rewards are received when the actions are taken. The goal of solving an MDP is to maximise the reward in such a stochastic environment [9, p. 53-59]. MDPs are formally represented as a 4-tuple of elements:

$$\langle S, A, P, R \rangle$$

- $S$ - state space, an instance of $s \in S$ is characterised as a sequence of variables $x_1, ..., x_n$, where each $x_i$ takes a value from a fixed domain $X_i$.
- $A$ - action space, defined as a discrete set of variables $A = \{a_1, a_2, ..., a_m\}$.
- $P(s'|s, a)$ - transition function, representing the conditional probability of reaching state $s'$ in time $t + 1$ when an action $a$ is performed in state $s$ at time $t$.
- $R(s, a, s')$ - reward function returning a numeric value for state $s$, action $a$ and newly reached state $s'$.

#### State space

The state space was largely defined by the MEDIATOR system's components, which capture sensor observations of the human driver, the vehicle and the environment [5]. The state was designed to be the product of observations coming from different contexts. This modular composition allows to more easily reason about different sources of data, as well as to make the extension of the model easier. The state space is presented in Figure 1. Policies interacting with the MDP can gain sufficient visibility of the environment to make decisions, as they are able to observe the safe & comfortable levels of automation based on the driver & automation's capabilities, as well as the driver's current interactions with the system.

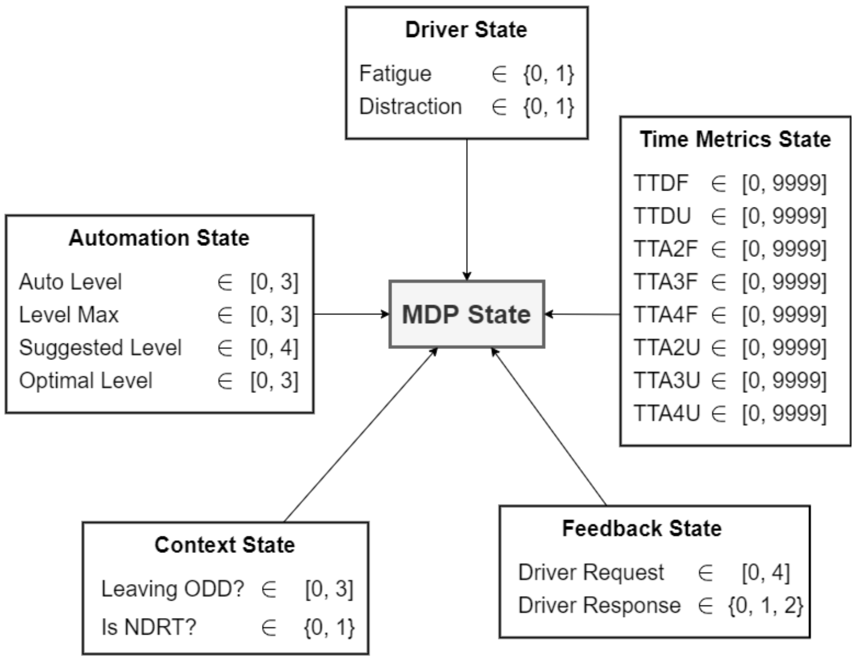


Figure 1: MDP state space, composed of 5 state components

Five main components were included in the state:

- **Driver**: binary variables are captured to track the human driver's condition to drive. A value of 1 indicates that the condition is active (e.g. fatigued)
- **Automation**: captures the vehicle's automation state. Active, maximum and optimal levels are between 0 (L0 - manual driving) and 3 (L4 - maximum automation). The value of 1 represents L2 (partial automation), with L1 skipped due to similarity to L0, as L1 additionally only provides driving assistance such as cruise control [16]. Suggested level values use the same level range, except they are offset by 1 to have 0 represent "no suggestion". The optimal level variable will be explained in the transition function section.
- **Context**: tracks road and environment observations. Leaving ODD indicates that the vehicle will be leaving its operational design domain within 5 minutes for a specific automation level. Leaving ODD for a lower level implies leaving ODD for a higher level (e.g. leaving ODD for L2 means that ODD is also left for L3 and L4). Values are between 0 and 3: 0 indicating the vehicle is not leaving ODD, 1 meaning the vehicle is leaving ODD for the highest automation level (L4) and 3 for the lowest (L2). The NDRT binary variable indicates that a non-driving related task occurred at the specific time step.

- **Feedback**: holds data of driver interactions with the system. Driver request represents a request to shift to a specific level. 0 meaning "no request", 1 indicating L0, and 4 meaning L4 requests. Driver response represents a driver's response to suggestions: 0 as no response, 1 as accepted and 2 as rejected.
- **Time Metrics**: stores time metric estimates of state changes to enable the automation system to make decisions based on near-future forecasts. Each time metric has two components: fitness ("F") and unfitness ("U"). TTDF and TTDU represent time to driver fitness and unfitness respectively, with TTDF indicating the number of seconds remaining until the driver becomes fit to drive (caused by fatigue/distraction changes), whereas unfitness indicates the estimate of seconds until the driver becomes no longer fit to drive in manual mode. TTA metrics represent fitness for each automation level respectively. The values are set in a large enough range for simplicity, capping at 9999 seconds to represent arbitrarily large values.

**Action space**

First, a requirements specification was crafted prior to defining the actions. The requirements focused on providing the capability for the automation system to operate at its full capacity, including both controlling the vehicle and interacting with the user. The defined criteria were as follows:

1. The system should be able to shift the automation level
2. The system should be able to send signals to the driver when shifting is not possible (i.e. suggestions, rejections and alarms)
3. The system should be able to have an action that would perform no change to the state
4. The system should take driver preference into account when shifting and suggesting

The requirements were formalised to 5 MDP actions:

- **Do Nothing (DN)**: the system makes no change to the vehicle's state
- **Reject (RA)**: the system rejects a level shift initiated by the driver
- **Shift Level (SL)**: the system performs a shift to the optimal level
- **Suggest Shift (SSL)**: the system suggests a shift to the optimal level
- **Prepare Driver (PD)**: the system alarms the driver to get ready for the takeover of the vehicle

**Transition function**

Since the use case focuses on human-AI interaction, the MDP was defined to operate on timesteps of 1 second each to account for human & automation reaction times. Moreover, a simplification was made regarding determining the optimal level to shift to: instead of providing an action per automation level, the shift (SL) and suggest shift (SSL) actions choose optimal level $L_{opt}$, defined as follows:

- If no driver request: $L_{opt} = \textit{Auto Level}$
- Define $L_{min}$ as 2 if $TTDF > 0 \;||\; TTDU < 60$, and 0 otherwise. The minimum level is L3 if the driver is unfit, since L0 and L2 require the driver to monitor the driving environment [16].
- Define $L_{comfort\_max} = min(3 - LeaveODD, L_{max})$ to enforce comfortable shifts to a level that will not leave ODD soon
- Let $L_{req} = DriverRequest - 1$, which is the level requested by the driver, rebased to be consistent with the level variables in the state.
- Let $L_{opt} = max(L_{min}, min(L_{comfort\_max}, L_{req}))$
- (if $L_{min} > L_{comfort\_max}$, then set $L_{opt} = \textit{Auto Level}$)

The formulation simplified the decision logic by only shifting to levels that are considered safe and comfortable (in range of minimum and maximum) while optimizing for the level to be as close to the request as possible. The optimal level is additionally stored to the "Optimal Level" variable in the state to inform policies interacting with the MDP. The transitions were formulated per action as follows:

- Define shorthand function *RESET(s)*, which resets the following variables of state $s$: *Driver Request = 0*, *Driver Response = 0* and *Suggested Level = 0.* Intuitively, this resets the driver request state.
- **Do Nothing**: makes no change to the state: $P(s|s, DN) = 1$
- **Reject**: $P(s_{RA}|s, RA) = 1$
  - $s_{RA}$ is constructed by $s_{RA} = RESET(s)$, resetting the driver request.
- **Shift Level**: $P(s_{SL}|s, SL) = 1$
  - $s_{SL}$ is defined with the same variables of $s$, except it is reset using the $RESET$ function and $\textit{Auto Level} = L_{opt}$. Intuitively, this resets the driver request state and shifts the automation level to the optimal choice.
- **Suggest Shift**: makes no change to the state on no driver request, if the optimal level is equal the suggested level, or if the optimal level is equal to the current level. Otherwise:

- Define three states: $s'_0, s'_1, s'_2$. Each represents a different driver response, e.g. 0 for *DriverResponse = 0*
- Set suggested level changes in next timestep: $\forall\, i\ \mathit{SuggestedLevel}_{s'_i} = L_{opt}$
- Driver request changes in the accepted state, imitating the driver accepting the suggestion: $\mathit{DriverRequest}(s'_1) = L_{opt}$
- Define $c_0$ and $c_1$ constants. $c_0$ is the no response probability, and $c_1$ is the base acceptance probability. The two state probabilities should not exceed 1: $\sum_i c_i \leq 1$. In the paper, $c_0 = 0.1$ and $c_1 = 0.8$.
- $P(s'_0|s, SSL) = c_0$. The transition represents the uncommon outcome when the driver does not respond within 1 second, or the case when the automation system delays the signal. The two outcomes are treated equally to learn policies that would factor in both delay scenarios.
- Define $d = |L_{opt} - (\mathit{DriverRequest} - 1)|$. Let $t_1 = min(0, c_1 - 0.25 \times d)$ and $t_2 = 1 - t_1 - c_0$. The values represent the decrease in acceptance probability for larger level differences, while constraining $c_0 + t_1 + t_2 = 1$.
- Define transition probabilities: $P(s'_1|s, SSL) = t_1$, $P(s'_2|s, SSL) = t_2$

- **Prepare Driver**: $P(s_{PD}|s, PD) = 1$
  - $s$ is transformed to $s_{PD}$, with $TTDF(t+1) = max(0, TTDF(t) - 1)$. Intuitively, the driver is prepared by increasing fitness.

The transition function is additionally extended to define terminal states: if $\mathit{DriverRequest}(t) \neq 0$ && $\mathit{DriverRequest}(t+1) = 0$, then the episode ends, as the driver request is satisfied.

**Reward function**

The reward function was represented as a decision tree, with leaf nodes denoting return values, and internal nodes representing conditional branches. This allowed to flexibly compose the effects of actions on the reward. The designed reward tree is summarised in Figure 2. The reward was designed to not be overly complex nor too simple. Simpler reward design in the use case can lead the agent to perform unwanted actions (e.g. if "Prepare Driver" was not penalized in incorrect scenarios, the agent might start using it as a "Waiting" action). On the other hand, reward over-engineering might lead to less efficient learning [17] as the agent overfits to specific correlations, motivating to design the reward based on outcomes rather than specific actions (in this case, tracking driver request satisfaction instead of individual actions). More insights into different reward designs can be found in Appendix A.

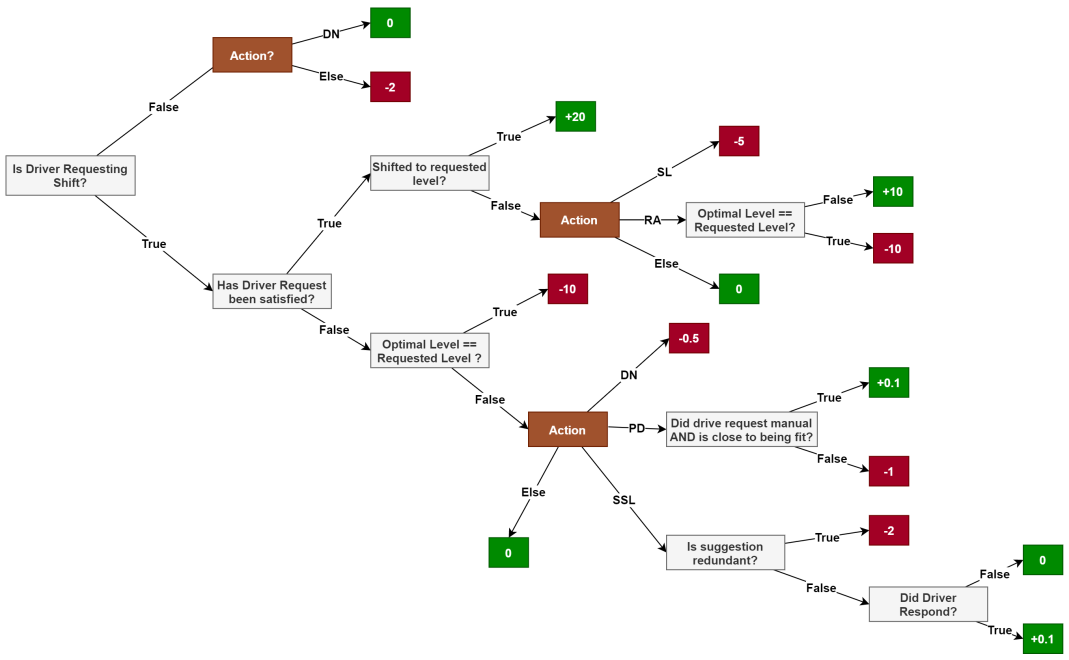

Is Driver Requesting Shift?
False
True
Action?
DN
0
Else
-2
Has Driver Request been satisfied?
True
Shifted to requested level?
True
+20
False
Action
SL
-5
RA
Optimal Level == Requested Level?
False
+10
True
-10
Else
0
False
Optimal Level == Requested Level ?
True
-10
False
Action
DN
-0.5
PD
Did drive request manual AND is close to being fit?
True
+0.1
False
-1
Else
0
SSL
Is suggestion redundant?
True
-2
False
Did Driver Respond?
False
0
True
+0.1

Figure 2: MDP reward decision tree

### Route simulation

In addition to state changes caused by actions, the MDP was extended to have a fixed generated route per episode. The route was set to be of a fixed length of 3 KM, assuming a constant speed of 100km/h. The simplification was made since the sensor data variables are assumed to capture all the necessary data to make an optimal decision on the automation-human interaction level. The 3KM route is furthermore divided into 108 timesteps, with 1 timestep equaling 27.78 meters or 1 second travelled, and a driver request is set to arrive in the first 5 seconds of the route.

For each timestep $t$, a state $s_{route}$ was generated, which was then used in the MDP transition function to determine state values of $s$ at timestep $t$. The $s_{route}$ sequence was generated by fixing various events to occur throughout the route. Simulated events included driver fatigue & distraction events and maximum automation level changes. Time metrics were computed based on the entire simulated route. The route simulation model is visualised in Figure 3. Details of the simulation can be found in Appendix B. The purpose of the fixed-route simulation was to simulate as many different driving conditions and different states as possible, in order to have diverse simulation data to train and test the policies on.

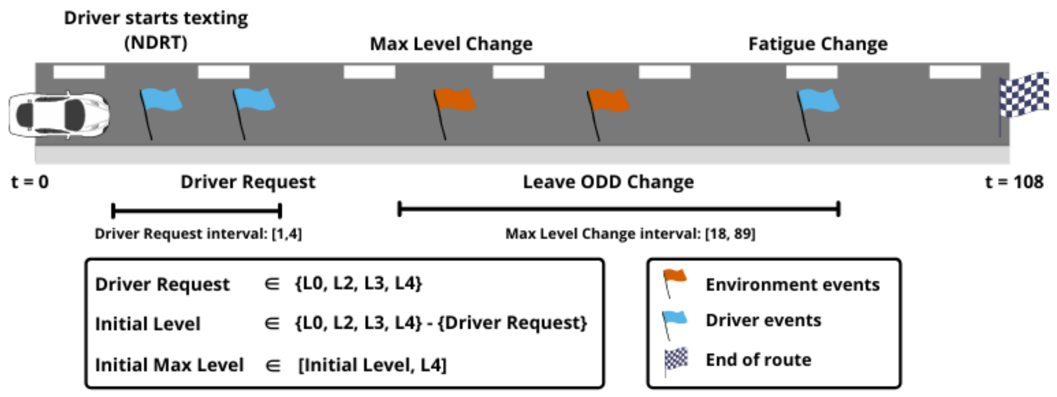

Driver starts texting (NDRT)
Max Level Change
Fatigue Change
t = 0
Driver Request
Leave ODD Change
t = 108
Driver Request interval: [1,4]
Max Level Change interval: [18, 89]
Driver Request ∈ {L0, L2, L3, L4}
Initial Level ∈ {L0, L2, L3, L4} - {Driver Request}
Initial Max Level ∈ [Initial Level, L4]
Environment events
Driver events
End of route

Figure 3: Route simulation model: events are placed around the driver request, and variables are preset for the simulated fixed length route.

## 2.2 Policies

After formulating the problem as an MDP, the next step was to create policies that would solve the MDP. A policy can be thought of as a function that maps a state to an action, allowing to calculate the next action to take in a given state. First, a decision tree policy was made, which defined a fixed set of rules to decide which action to take next. The decision tree provided a baseline comparison for reinforcement learning algorithms and allowed assessing how effective a learned policy is compared to a carefully designed policy. The crafted decision tree is shown in Figure 4. The policy was designed based on intuition: if no request is pending – nothing should be done. Otherwise, the automation should attempt to either prepare the driver if they are close to being fit, suggest an alternative level or reject if it's impossible to satisfy the request or shift directly if the preferred level is available. It should be noted that "soon" conditions were left intentionally vague. This generally depends on the chosen length of the route. In the case of the study, "Automation fit soon" means that the automation is fit within 2 seconds, and "driver close to being fit" means that the driver will be fit within the next 30 seconds.

Multiple reinforcement learning algorithms were considered to solve the task. Since the formulated MDP had a continuous state space (due to time metrics being continuous) and a discrete action space, multiple algorithms fit well for the task, such as Deep Q Network (DQN [12]), Proximal Policy Optimization (PPO [18]) and Actor-Critic with Experience Replay (ACER [19]). The main characteristic of the mentioned algorithms is that they all use function approximators to determine how good an action is based on the current state, therefore enabling to solve problems with very large state spaces in feasible amounts of time and memory. The discretized state space size of the defined MDP is very large (roughly $1.536 \times 10^{37}$), therefore motivating to use an algorithm using approximators.

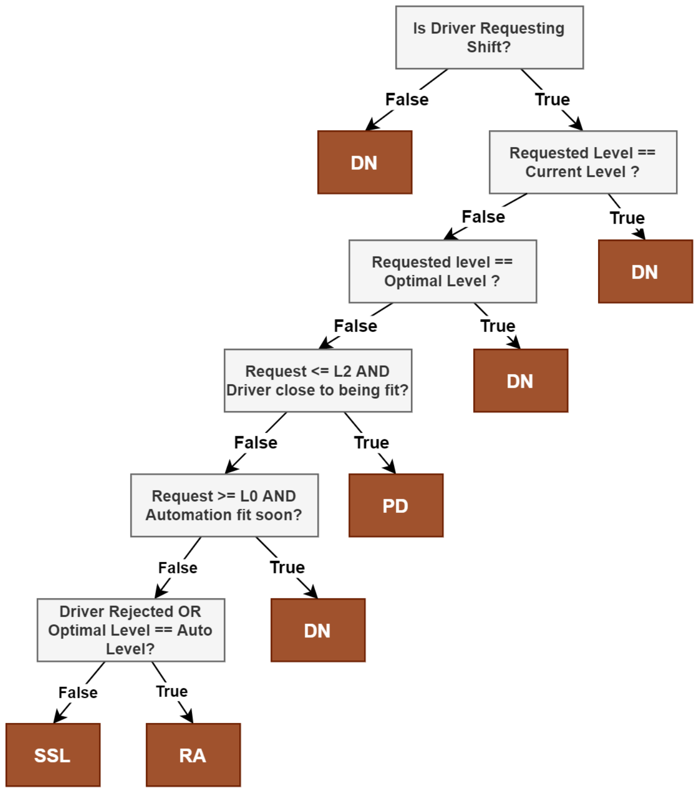

Is Driver Requesting Shift?
False
True
DN
Requested Level == Current Level ?
False
True
DN
Requested level == Optimal Level ?
False
True
DN
Request <= L2 AND Driver close to being fit?
False
True
PD
Request >= L0 AND Automation fit soon?
False
True
DN
Driver Rejected OR Optimal Level == Auto Level?
False
True
SSL
RA

Figure 4: Decision tree policy: fixed set of rules to pick an action based on state

After running a few initial experiments, as seen in Appendix C, the DQN algorithm was chosen based on the algorithm being more sample efficient and effective in solving the decision problem over the limited amount of trials. Additionally, extensions were added to the DQN, which increased the performance considerably: double network [20], dueling network [21] and prioritized experience replay [22]. The DQN algorithm was then run more times with different hyperparameters to find a more performant configuration (the process is detailed in Appendix C). The hyperparameter details of the most efficient observed policy are shown in Appendix D.2. The *Stable Baselines*[2] implementation of the algorithm was used (trained for 1 million timesteps), together with the *VecNormalize*[3] environment wrapper to normalize state observations. Normalization was needed in order to make the policy less sensitive to outlier values during training (e.g. TTA metric values might reach 9999, while other values were usually small, mostly in the range [0, 3]).

[2]Stable Baselines framework - https://stable-baselines.readthedocs.io/

[3]VecNormalize wrapper - https://stable-baselines.readthedocs.io/en/master/guide/vec_envs.html#vecnormalize

## 3 Results

In order to compare the reinforcement learning policy to the baseline decision tree, metrics were defined based on the problem domain. The metrics covered 3 main aspects of semi-autonomous vehicle capabilities: safety, driver comfort and efficiency. The policies were run on the route simulation environment defined in the previous section over 1 million episodes and then the metrics were calculated for each policy separately. Experiment parameters, configuration details and seeds can be found in Appendix D. The section will proceed to cover each metric results in more detail.

### Safety

The safety of an automated agent was measured in terms of unsafe shifts throughout the simulated episodes. An unsafe shift is defined as a shift in the automation level to a level outside the safe level range. The safe level range at time $t$ was defined as $[L_{min}, L_{max}]$, where $L_{max}$ is the maximum level as defined in the state, and $L_{min}$ is L3 if the driver is unfit, and L0 otherwise. As the range was defined by the time metrics, driver and automation state, it gave a complete indication of safety in the defined decision problem. The metric tracked is the ratio of unsafe shifts over all the simulated episodes. The results are presented in Table 1.

### Comfort

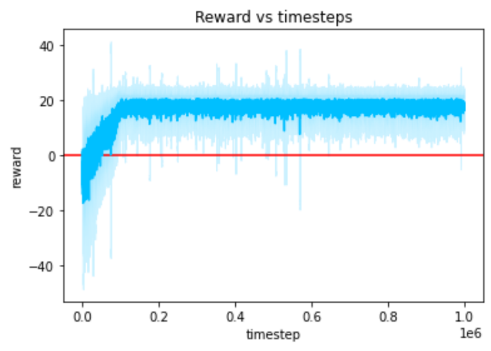
Similarly to safety, comfort was measured by the ratio of uncomfortable shifts throughout all episodes. An uncomfortable shift was defined as a change in the level which is outside the range of comfortable levels. A comfortable level was defined to be a level in which the fitness lasts for at least 1 minute. The small measure of time (1 minute) was chosen since the simulated environment is short, only 108 seconds, thus giving a fair indication of comfort. Additionally, if at any point in the request a *redundant prepare* action was taken (a *Prepare Driver* action when the driver is not shifting to a manual driving level, or TTDF is 0), the request was also considered to be uncomfortable. The comfort metrics are summarised in Table 1.

| **Algorithm** | **Satisfied** | **Unsafe** | **Uncomfortable** |
|---|---|---|---|
| Decision Tree | 100% (1 000 000) | 0% (0) | 0% (0) |
| DQN | 99.9998% (999 998) | 0% (0) | 0.0014% (14) |

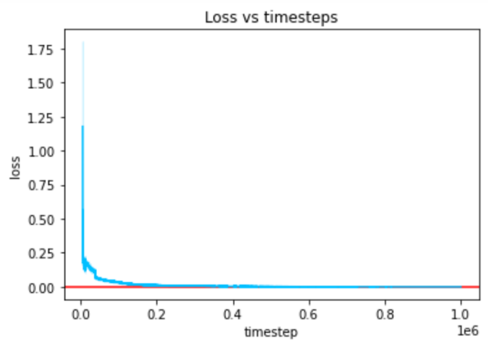
Table 1: Driver request distribution metrics over 1 million episodes

## Efficiency

The efficiency of the algorithms was assessed against two aspects - domain-specific performance and algorithmic performance. The latter was measured both by training performance (mean reward and loss over training timesteps) and test performance (mean and standard deviation of reward after training). Reward metrics were collected both during and after training to assess how well the policy performs outside of a training setting, and the reward metric was chosen as it is the optimization goal of RL algorithms. The training performance is shown in Figure 5, while the test performance is given in Table 2. Note that the decision tree has no training performance, as the policy is pre-defined and requires no training.



(a)



(b)

Figure 5: Training performance graphs: (a) - mean episode reward over timesteps, (b) - mean loss over timesteps. The lighter color draws the 99% confidence interval.

| **Algorithm** | **Reward Mean ± SD** | **Reward Median** | **Reward Range** | **Episode Length Mean ± SD** |
|---|---|---|---|---|
| Decision Tree | 18.46 ± 4.18 | 20 | [8, 23.9] | 7.5 ± 6.3 |
| DQN | 18.67 ± 3.84 | 20 | [-104, 23.9] | 7.02 ± 5.99 |

Table 2: Policy testing performance metrics over 1 million episodes

The domain-specific efficiency of the policy was measured by how long it takes to satisfy the driver's request. The metric indicates how fast the agent can perform its tasks to reach the desired goal. The time metrics are summarized in Table 3. Furthermore, the expectation is that the agents can satisfy all incoming driver requests. Thus, another metric was captured in Table 1 to track how many requests were satisfied. More specific action distribution metrics were captured in Table 4, tracking problematic actions: idle (request pending, but the agent does nothing), shifts missed (optimal level equals requested level, but the agent performs another action), redundant prepares and false rejects (rejections without suggestion, or rejections when a shift is feasible).

| **Algorithm** | **Mean ± SD** | **Median** | **Range** |
|---|---|---|---|
| Decision Tree | 5s ± 6.19s | 1s | [1s, 40s] |
| DQN | 4.52s ± 5.88s | 1s | [1s, 63s] |

Table 3: Driver request satisfaction time metrics (seconds) over 1 million episodes

| **Algorithm** | **Idle** | **Shifts Missed** | **Redundant Prepares** | **False Rejects** |
|---|---|---|---|---|
| Decision Tree | 0.003% (236) | 0% (0) | 0% (0) | 0% (0) |
| DQN | 1.33% (93104) | 0.00036% (25) | 0.00057% (40) | 0.087% (6119) |

Table 4: Policy problematic action distribution metrics

Finally, additional insights into outliers are summarised by the graphs shown in Appendix E. Further, Appendix F provides an additional comparison of the action distributions of the two policies.

## 4 Discussion

The section will evaluate the results seen in the previous section and afterwards reflect on the responsible research aspects of the study. The section concludes by suggesting future work.

### 4.1 Evaluation

The goal of the study was to provide a model and a policy that would ensure safe, comfortable and efficient actions in guiding the driver to their initiated automation level. With regards to safety, it is evident from Table 1 that the property was guaranteed by both policies, since there were no unsafe shifts. This is due to the problem being modelled to enforce the constraint.

With regards to efficiency, generally, the DQN learned policy was more efficient since it managed to satisfy requests quicker, as seen in Table 3. However, Table 4 indicates that the DQN fails in outlier cases since it has more problematic actions than the Decision Tree policy. The *redundant prepares* metric also caused issues in the comfort of the DQN policy, as seen by the uncomfortable shifts metric in Table 1, rendering the DQN generally less comfortable than the Decision Tree policy. Adding the percentages of problematic actions (all in Table 4 except idle time), the DQN falls short in efficiency in roughly 0.09% of cases. The negative outlier effect is also evident in Table 2 and Figure E.2, since the DQN sometimes collects negative rewards even after training. However, it still managed to achieve a higher mean reward than that of the decision tree, indicating that the method is generally more efficient. Lastly, Table 1 exemplifies the negative outlier effect as the DQN policy did not manage to satisfy 2 out of 1 million driver requests, while the decision tree managed to satisfy all of the requests.

Overall, the DQN policy appears to be a more optimal policy than the decision tree. However, the policy fails to outperform the decision tree in outlier cases, indicating a limitation in the reinforcement learning approach for the use case. When placing the two policies in a more uncertain environment (e.g. time metrics with estimation errors), it is expected that the DQN would perform even better than the decision tree policy, since it is much harder for human-crafted logic to deal with unseen cases of uncertainty.

## 4.2 Responsible Research

While the conducted study shows interesting results for the future of automated driving, reproducibility is another important component of the study. Throughout the study, the experiment was designed carefully to allow for reproducibility and repeatability of the experiments. The model of the problem (see Section 2) was provided as a non-ambiguous mathematical and algorithmic formulation, with each assumption documented. The reinforcement learning algorithms were utilized from the publicly available *Stable Baselines*[4] framework, with the hyperparameters, hardware, software and seeds thoroughly documented and explained in Appendix D. Moreover, the results were run over a large sample size to minimize variance and bias in the data. By the Law of Large Numbers [23], the results would be close to the actual mean, meaning that even with randomness, repeat observations would be similar with large sample sizes. Although the listed measures help reproduction immensely, the results should not be looked at with too much optimism and the study should be conducted over larger numbers of test samples by multiple researchers in order to be fully confident about the result. Lastly, it should be noted that the source code is proprietary, which could cause issues in reproducibility, however, it can be provided upon request.

Even though the algorithmic techniques presented in the study show promising results for performing decisions on behalf of human drivers, the novelty of driving automation poses some ethical concerns. A notable example is the social dilemma of an autonomous vehicle having to choose between 2 choices that would both result in human casualties in the case of driving accidents [24]. Ethical issues are also prevalent in the use case discussed in the study. The strong constraints of safety

[4]Stable Baselines framework - https://stable-baselines.readthedocs.io/

and comfort might not be applicable in all scenarios. For instance, the driver might feel much safer in less comfortable scenarios and would be willing to make the trade-off of sacrificing comfort for safety, while the automation system would reject the driver from doing so as it optimizes for both factors. To mitigate this, the system would have to be made more flexible, and cooperate more with the driver, either through learning from experience or through configured preferences. Furthermore, there might be a scenario in the larger MEDIATOR system where the vehicle would have multiple options. For example, it may either choose to force a shift to a safe level, or it may accept the driver's request to a potentially unsafe level. Depending on how the decision is resolved, it may pose ethical issues for human safety. Essentially, there is a trade-off between catering to the user's request and providing safety. The right balance should be struck between the two when integrating the use case into a larger system.

### 4.3 Future Work

Due to the agent failing on outliers, future work could focus on exploring solutions to the problem. A solution could potentially be achieved by crafting a hybrid approach utilizing both the RL agent and the human-designed decision tree, or by utilizing shielding [25]. Furthermore, the presented model assumes that the sensor data under simulation is fully accurate and correct. This assumption would not hold in the real world, thus models capable of dealing with uncertainty could be explored, such as POMDPs [26]. Additionally, stronger RL formulations could be explored to bring the model closer to reality, for example, safety-constrained RL [27] or multi-objective optimization [28]. Additionally, future research could build on top of the problem formulation by attempting to make trade-offs, for example by allowing the driver to take over to increase their personal feeling of safety while decreasing comfort. Finally, it should be noted that the use case presented in the paper is very specific, thus future research is needed on how to effectively integrate the proposed model into the larger MEDIATOR system.

## 5 Conclusion

The paper raised the hypothesis that a reinforcement learning (RL) approach is an efficient and safe solution to the specific scenario of the driver initiating a shift of control in semi-autonomous vehicles. The paper presented a problem formulation as a Markov Decision Process (MDP), an RL application to solve the MDP and a baseline decision tree policy to compare the RL policy against. The hypothesis was confirmed, as the resulting policy was fully safe and comfortable, and was more efficient than a baseline handcrafted decision tree. However, the reinforcement learning approach was limited in a few outlier cases (roughly 0.09%), where it failed to perform efficiently. Future studies should focus on exploring more realistic models and techniques to enforce desired efficiency constraints to prevent outlier cases.

## References

[1] L. P. Robert, "Are automated vehicles safer than manually driven cars?" *AI & SOCIETY*, vol. 34, no. 3, pp. 687–688, 2019.

[2] S. Shahrdar, L. Menezes and M. Nojoumian, "A survey on trust in autonomous systems," in *Science and Information Conference*, Springer, 2018, pp. 368–386.

[3] S. Sheng, E. Pakdamanian, K. Han, B. Kim, P. Tiwari, I. Kim and L. Feng, "A case study of trust on autonomous driving," in *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, IEEE, 2019, pp. 4368–4373.

[4] L. Petersen, H. Zhao, D. Tilbury, X. J. Yang, L. Robert *et al.*, "The influence of risk on driver's trust in semi-autonomous driving," 2018.

[5] M. Christoph, D. Cleij, H. Ahlström, B. Bakker, M. Beggiato, A. Borowsky, R. van Egmond, E. van Grondelle and H. de Ridder, "Mediating between human driver and automation: State-of-the artand knowledge gaps: D1. 1 of the h2020 project mediator," 2019.

[6] D. Vermunt, "A markov decision process approach to human-autonomous driving control logic," 2020.

[7] F. van Wyk, A. Khojandi and N. Masoud, "Optimal switching policy between driving entities in semi-autonomous vehicles," *Transportation Research Part C: Emerging Technologies*, vol. 114, pp. 517–531, 2020.

[8] F. Woergoetter and B. Porr, *Reinforcement learning.* [Online]. Available: `http://www.scholarpedia.org/article/Reinforcement_learning` (visited on 23/04/2021).

[9] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*, 2nd ed. MIT press, 2015.

[10] C. You, J. Lu, D. Filev and P. Tsiotras, "Highway traffic modeling and decision making for autonomous vehicle using reinforcement learning," in *2018 IEEE Intelligent Vehicles Symposium (IV)*, IEEE, 2018, pp. 1227–1232.

[11] S. Brechtel, T. Gindele and R. Dillmann, "Probabilistic mdp-behavior planning for cars," in *2011 14th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, IEEE, 2011, pp. 1537–1542.

[12] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[13] S. Reddy, A. D. Dragan and S. Levine, "Shared autonomy via deep reinforcement learning," *arXiv preprint arXiv:1802.01744*, 2018.

[14] B. R. Kiran, I. Sobh, V. Talpaert, P. Mannion, A. A. Al Sallab, S. Yogamani and P. Pérez, "Deep reinforcement learning for autonomous driving: A survey," *IEEE Transactions on Intelligent Transportation Systems*, 2021.

[15] A. E. Sallab, M. Abdou, E. Perot and S. Yogamani, "Deep reinforcement learning framework for autonomous driving," *Electronic Imaging*, vol. 2017, no. 19, pp. 70–76, 2017.

[16] Synopsys. (n.d.). "The 6 levels of vehicle autonomy explained," [Online]. Available: `https://www.synopsys.com/automotive/autonomous-driving-levels.html` (visited on 27/05/2021).

[17] D. Amodei, C. Olah, J. Steinhardt, P. Christiano, J. Schulman and D. Mané, "Concrete problems in ai safety," *arXiv preprint arXiv:1606.06565*, 2016.

[18] J. Schulman, F. Wolski, P. Dhariwal, A. Radford and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[19] Z. Wang, V. Bapst, N. Heess, V. Mnih, R. Munos, K. Kavukcuoglu and N. de Freitas, "Sample efficient actor-critic with experience replay," *arXiv preprint arXiv:1611.01224*, 2016.

[20] H. Van Hasselt, A. Guez and D. Silver, "Deep reinforcement learning with double q-learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 30, 2016.

[21] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot and N. Freitas, "Dueling network architectures for deep reinforcement learning," in *International conference on machine learning*, PMLR, 2016, pp. 1995–2003.

[22] T. Schaul, J. Quan, I. Antonoglou and D. Silver, "Prioritized experience replay," *arXiv preprint arXiv:1511.05952*, 2015.

[23] R. Routledge, *Law of large numbers*, 2005. [Online]. Available: `https://www.britannica.com/science/law-of-large-numbers`.

[24] J.-F. Bonnefon, A. Shariff and I. Rahwan, "The social dilemma of autonomous vehicles," *Science*, vol. 352, no. 6293, pp. 1573–1576, 2016.

[25] M. Alshiekh, R. Bloem, R. Ehlers, B. Könighofer, S. Niekum and U. Topcu, "Safe reinforcement learning via shielding," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, 2018.

[26] M. T. Spaan, "Partially observable markov decision processes," in *Reinforcement Learning*, Springer, 2012, pp. 387–414.

[27] A. Wachi and Y. Sui, "Safe reinforcement learning in constrained markov decision processes," in *International Conference on Machine Learning*, PMLR, 2020, pp. 9797–9806.

[28] H. Mossalam, Y. M. Assael, D. M. Roijers and S. Whiteson, "Multi-objective deep reinforcement learning," *arXiv preprint arXiv:1610.02707*, 2016.

## A Alternate Reward Designs

The appendix describes additional observed results with alternative reward designs. The observations further motivate the final reward design chosen in Section 2.

### A.1 Engineered reward

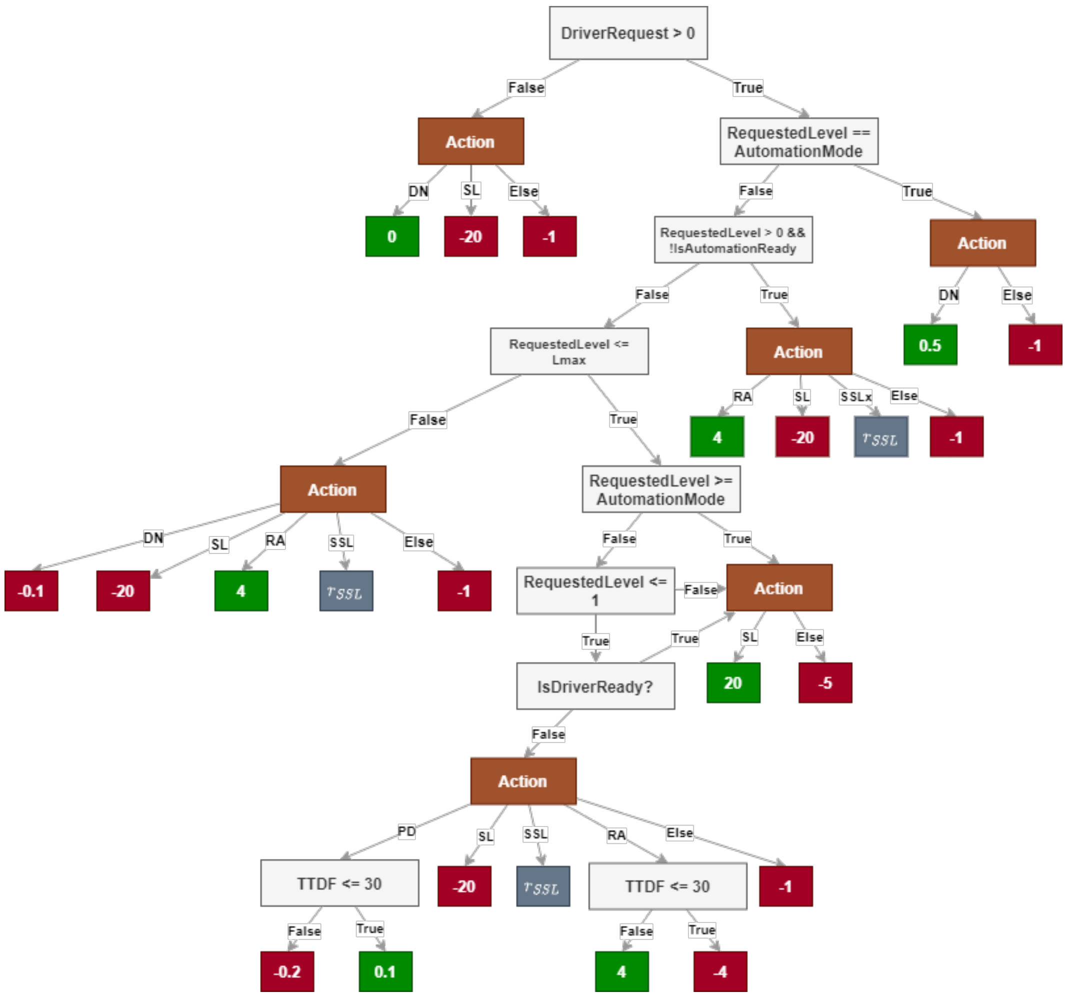


Figure A.1: Alternative reward design: highly branched, engineered reward. $r_{SSL}$ in the figure penalizes redundant suggestions and rewards correct suggestions by -2 and +2 respectively.

The reward design proved to be hard to track and debug due to its complexity. However, apart from that, the reward design is flawed because it has the potential to force the agent to interact in a predetermined way, effectively disabling the agent from learning and exploring to achieve the reward on its own. The reward design was simplified in favour of the original reward design shown in Section 2.

## A.2 Time-based reward

```
if is_initiating:
    request_satisfied = current_state.fb_state.Driver_Request != 0 \
        and next_state.fb_state.Driver_Request == 0
    shifted_to_requested = requested_level == \
        next_state.automation_state.Automation_Mode

    if request_satisfied:
        rewards += 5

        if shifted_to_requested:
            rewards += 15
    else:
        costs -= 0.1
else:
    if action != 0: # Not DN
        costs -= 2
```

Figure A.2: Time-based reward in code

The shown reward is a simplification of the final reward design. The idea is that the main goal of satisfying the driver request is rewarded, and other actions are penalized, encouraging the agent to reach the goal faster. However, Figure A.3 shows that the agent did not successfully learn with the defined reward, as the agent collects a large number of negative rewards after training for 1 million timesteps.

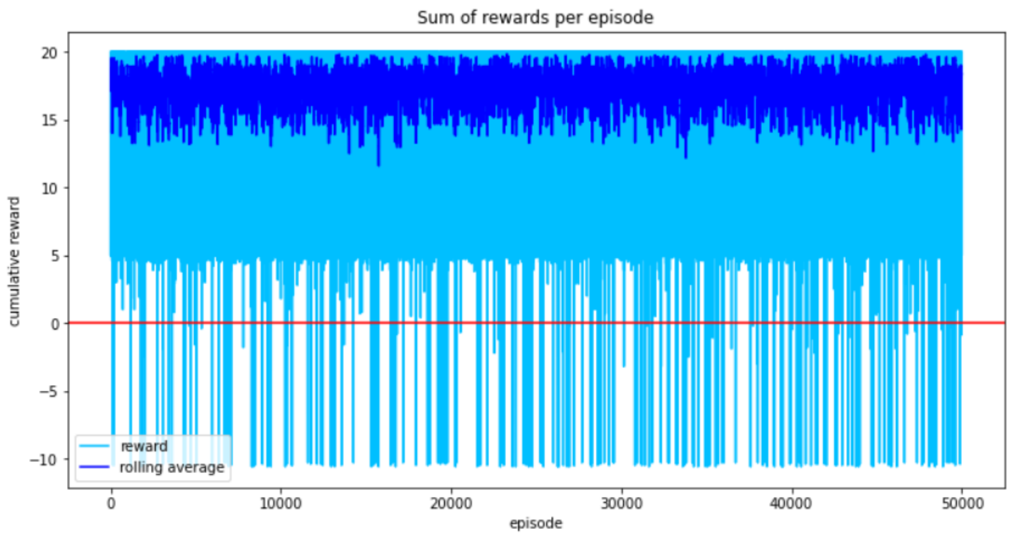


Figure A.3: Time-based reward: collected test rewards after training DQN for 1 million timesteps

## A.3 No penalty for DN

The final reward design in Section 2 was taken, but the DN action was not penalized when a request was pending. Interestingly, the change led to more shifts being missed, as the agent learned to abuse the "Do Nothing" more often. More precisely, after only 50 000 test episodes, 3742 out of 365018 actions (1.03%) were missed shifts, which is far inferior to the data seen in Table 4.

## A.4 Balanced DN and PD penalty

In the reward design presented in Section 2, the penalties for DN and PD are uneven. DN is penalized by a factor of -0.5 when a driver request is pending, while PD is penalized by -1 if used incorrectly. Previously, the reward design was balanced: both penalties were -1. This led to a higher percentage of *redundant prepares* (see Section 3 for details on *redundant prepares*) in outlier scenarios (roughly 0.42%), effectively leading to a less comfortable policy. The reward was changed to give less penalty to DN, indicating to the agent that this is a better action in case waiting is necessary, which decreased "redundant prepares" significantly. Furthermore, the reward did not incorporate the -10 penalty when the request is not satisfied when the optimal level equals the requested level. Adding this penalty improved the learned policy of the agent by reducing missed shifts and false rejects.

## B Route Simulation

Routes used in the MDP were generated procedurally, attempting to simulate various scenarios that could occur in driving. The algorithm is as follows:

1. Generate driver fatigue in the route
   (a) With probability 0.75: set driver's state to *not fatigued* (value of 0), and with probability 0.25 set to *fatigued* (value of 1)
   (b) If the driver is not fatigued, with probability 0.1: make driver *fatigued* from randomly chosen time step $t_{fatigued}$ until the end of the route.
   (c) (3 distinct scenarios are covered: driver is not tired, driver becomes tired and driver is tired from the beginning)
2. Determine driver request scenario. Randomly pick either *Prefer Automation* or *Prefer Manual* scenarios.
   (a) In the *Prefer Manual* scenario, the driver request is set to 1 (L0)
   (b) In the *Prefer Automation* scenario, the driver request is set to a value in $[2, 4]$, representing L2, L3 and L4 requests.
   (c) A random timestep $t_{req}$ is picked in the interval $[1, 4]$. At this time step, the driver request value is set to the generated value.
3. Generate automation levels for the route
   (a) Pick an automation level *auto_level* to be one level from the set $\{L0, L2, L3, L4\} - L_{requested}$.
   (b) Determine the maximum level of the route as a value from the interval $[auto\_level, L4]$
   (c) The level determination is done for the entire route. If at time $t$ the fatigue is 1, then the only valid levels are $L3$ and $L4$. A random level in $\{L3, L4\} - L_{requested}$ is picked for the segment where the driver is fatigued.
   (d) (Driver requested level is excluded from the active level set because then the scenario would end because the driver request would become satisfied automatically, regardless of what decision is made)
4. Generate automation level change events
   (a) With probability 0.4: no event generated
   (b) With probability 0.4: generate a persistent level change event. A random time step $t_0$ is picked from the interval $[18, 89]$ of the fixed-length route of 108 timesteps. Then, an *auto_level* and max level are picked according to the same rules as determining the automation level of the route (except

now the automation level prior to the event is also excluded from the valid choices to enforce a level change). The automation level and the new selected max level are changed for the entire route from point $t_0$ until the last timestep.

(c) With probability 0.2: generate a "tunnel" level change event. A random time step $t_0$ is picked from the interval $[18, 64]$. Then, a maximum level $y$ lower than the current maximum level is picked (according to the same rules as the initial level generation). Then, a random time step $t_1$ is chosen within $[t_0 + 18, t_0 + 27]$. The strip from $t_0$ to $t_1$ is changed to have the maximum level $y$, and a newly generated automation level.

(d) (Three main scenarios are covered: no maximum level changes, long maximum level changes to represent significant driving condition changes, and temporary maximum level changes to represent temporary hindrances such as unclear lane markings)

5. Simulate distraction events, sweeping over the entire route from timestep 0 until the last timestep of the route:

   (a) If the level at time $t$ is L3 or L4, proceed to the next timestep

   (b) Else: trigger a distraction with probability 0.2. If no distraction is triggered, proceed to the next timestep. Otherwise:

      i. Pick random $l \in [1, 5]$

      ii. Set Distraction to 1 in $t$ to $t + l$, unless the automation level changes to L3 or L4, then pick the nearest time step where the automation level is still L0/L2.

      iii. Advance the timestep by 18 from $t + l$ to simulate delays in distraction.

   (c) (Distractions are simulated to occur for random intervals of time, at any time during driving. The problem is simplified by not tracking distractions when the automation is in control of the vehicle)

6. Simulate non-driving related tasks (NDRTs)

   (a) Pick random number of NDRTs that will occur in the route: 0 (probability 0.3), 1 (probability 0.6), 2 (probability 0.1)

   (b) With probability 0.01 (if 1 or 2 NDRTs are triggered): trigger single NDRT after the driver request

   (c) Depending on the previous branch outcome, trigger the remaining NDRTs to occur in a random timestep up until the driver request. If 2 NDRTs are picked, they are constrained to occur at least 5 time steps between each other

   (d) (NDRT occurring after driver request is considered rare, because the driver is initiating a takeover, so it is triggered with a very small probability)

7. Calculate time metrics: TTAU, TTAF, TTDU, TTDF (see Figure B.1 for TTA metrics, and Figure B.2 for TTD metrics)

8. Simulate Leave ODD

   (a) If $TTAxU \leq 300$ at time $t$, then set the leave ODD variable to true for level $x$. Do this for all levels.

   (b) (Interval of 300 is chosen to simulate that the automation is able to forecast ODD changes in the upcoming next 5 minutes)

```
def find_timestep_where_lmax_decreases(
    from_timestep,
    target_level
):
    """
    Returns the first timestep greater than the
    passed in 'from_timestep' in which the max level
    is lower than the 'target_level'.

    Returns -1 if no such timestep exists.
    """
    # ...

# Set TTAF to 0 for all levels up until the active level,
# except level 0 since the metric is not tracked for
# manual mode. 0 indicates that the automation is
# currently fit to drive in the active level.
for l in range(1, auto_level + 1):
    TTAF[l] = 0

Lmax = 4
for l in range(1, Lmax):
    decrease_timestep = find_timestep_where_lmax_decreases(
        current_timestep, l
    )

    if decrease_timestep != -1:
        # Automation will become unfit after reaching
        # the timestep in which the level decreases.
        TTAU[l] = (decrease_timestep - current_timestep)
    else:
        # Indicate that the level is not close to being unfit
        TTAU[l] = 9999

return TTAF, TTAU
```

Figure B.1: TTA metric calculations for a single timestep

```
if auto_level == "L0" or auto_level == "L2":
    if driver_distracted:
        TTDU = 360
    else:
        TTDU = 0

    if driver_fatigued:
        TTDU = 0
    else:
        TTDU = min(TTDU, 1800)

    # TTDF 0: driver is already fit to drive,
    # since they are in control of the vehicle.
    return 0, TTDU
else: # L3/L4
    if is_ndrt:
        # Different NDRT types:
        # Messaging: 5 seconds
        # Obstruction: 10 seconds
        # Immersion: 10 seconds
        # Obstruction + Immersion: 20 seconds
        TTDF = random.choice([5, 10, 10, 20])
    else:
        TTDF = 0

    # TTDU 9999: driver unfitness is irrelevant,
    # since the automation is in control of the vehicle
    return TTDF, 9999
```

Figure B.2: TTD metric calculations for a single timestep

## C Choosing Algorithms & Hyperparameters

The section describes how and why specific algorithms and hyperparameters were chosen.

### C.1 Algorithms

In the initial stages of the experiments, multiple reinforcement learning algorithms were considered and evaluated when training over 5 million timesteps. The only requirement was that the algorithms supported continuous state spaces and discrete action spaces. The algorithms were evaluated based on their overall trends on the rewards acquired over small sample sizes of 50 thousand episodes after training. It should be noted that the experiment is not fully sound because it was run on an MDP with a more complex reward design (see Figure A.1), however, it was close enough to make a decision as the final state space was the same, excluding new state variables introduced for NDRTs and the optimal level. The simplified experiment provided guidance due to distinct trends observed in the figures below.

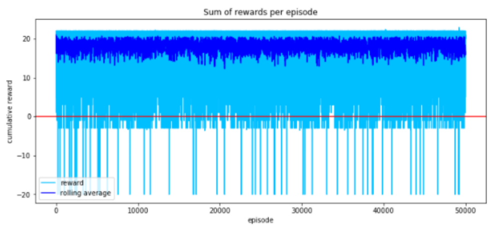


Figure C.1: ACER

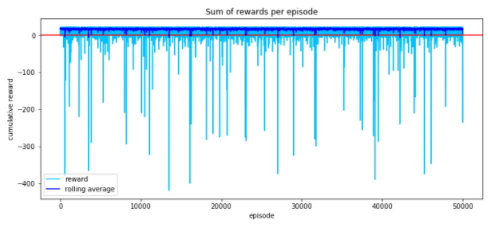


Figure C.2: TRPO

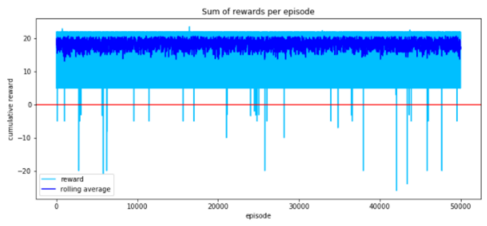


Figure C.3: PPO

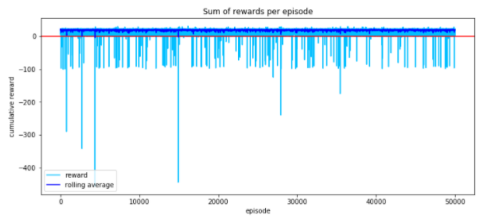


Figure C.4: DQN

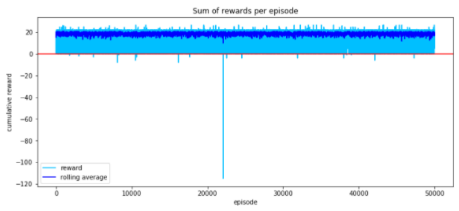


Figure C.5: Double DQN

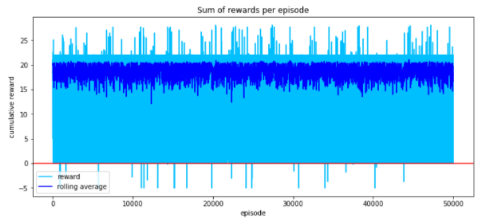


Figure C.6: Double Q-Learning + Duelling Network

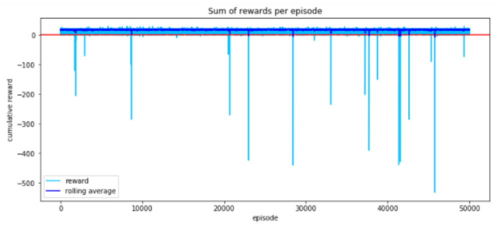


Figure C.7: Double DQN + PER

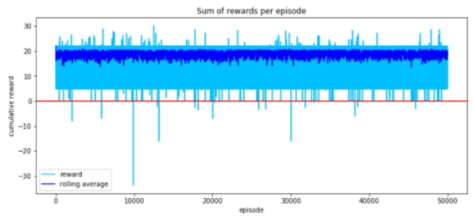


Figure C.8: Double DQN + Duelling Network + PER

The two best-performing algorithms were **PPO** and **DQN** with 3 extensions: Double Q-Learning, Prioritized Experience Replay (PER) and duelling network. Due to the DQN's ability to reach even higher rewards than PPO, it was ultimately chosen as the algorithm to solve the decision making problem.

After the reward was finalized, two additional alternative algorithms were run to gather further evidence whether the DQN was the most suitable found choice during the study. The algorithms under test were ACER and PPO. Both algorithms were trained with 25 million timesteps, more than the DQN algorithm was trained on in the final study. The algorithms posed some interesting issues. The ACER algorithm would not utilize the PD or SSL actions at all, as seen by the action distribution in Figure C.9, whereas the PPO algorithm's observed reward after training was noisier, as shown in Figure C.10. The extra experiment yielded additional evidence that the DQN was more sample efficient and suitable for the use case.

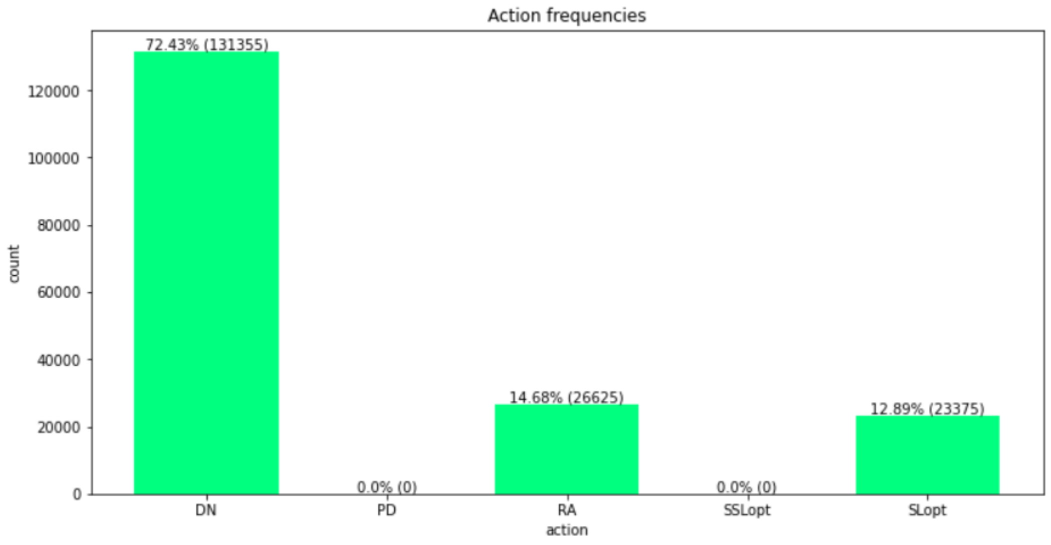


Figure C.9: ACER action distribution after training for 25 million timesteps and testing over 50 000 episodes

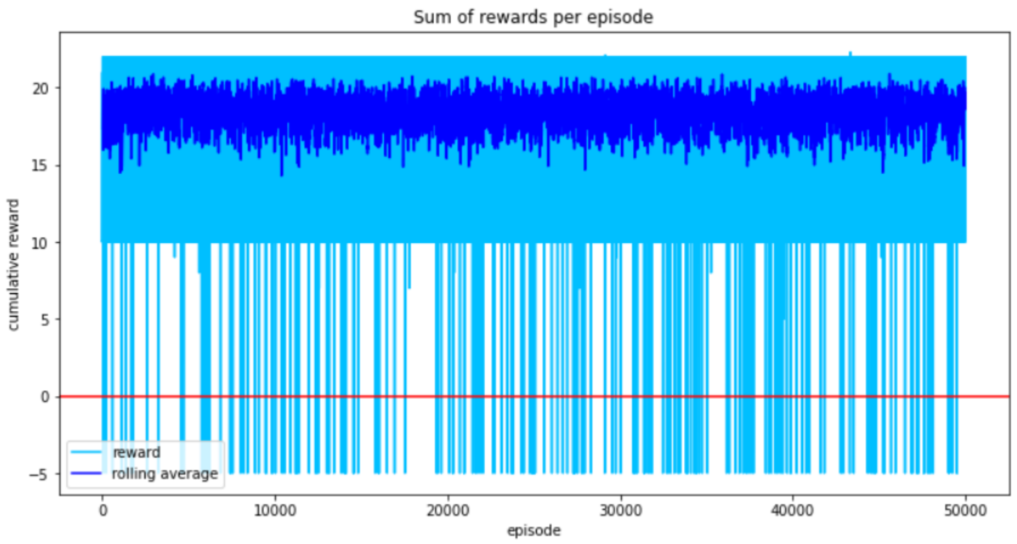


Figure C.10: PPO reward distribution after training for 25 million timesteps and testing over 50 000 episodes

## C.2 Hyperparameters

The hyperparameters were mostly determined by trial and error. Firstly, ranges were defined for the hyperparameters, as seen in Appendix D.2. The ranges were defined to limit the scope of the experiment to intuitive values. For instance, a batch size range of 8 to 128 was picked since values below 8 were deemed too small for mini-batch learning, whereas values above 128 could turn out to be too large. After defining the ranges, multiple experiments were run with different sets of hyperparameters, utilizing the WANDB[5] framework's Bayesian hyperparameter optimization. Experiments involved training the DQN models for 250 thousand timesteps each and measuring reward metrics after training over 5000 episodes. A total of 50 hyperparameter runs were performed, and the correlation coefficients between metrics and hyperparameters were reported by the framework. The correlations are summarized in Table C.1.

| | Mean Reward | Minimum Reward | STDev Reward |
|---|---|---|---|
| $\alpha$ | -0.750 | -0.178 | 0.651 |
| Learning starts | -0.185 | -0.109 | 0.215 |
| Target network update | 0.013 | -0.011 | -0.07 |
| Buffer size | -0.047 | 0.159 | -0.016 |
| Batch size | 0.022 | 0.201 | -0.035 |

Table C.1: Hyperparameter vs metric correlation coefficients over 50 runs

The results provided a good guideline for picking the hyperparameters. The main conclusions were:

- The learning rate should be small, around 2e$-$4 or smaller, as lower learning rates yield higher rewards and smaller variance
- The batch size should be higher due to a high positive correlation with minimum rewards

The remaining metrics did not yield much insight, as the correlation coefficients were too small to lead to any sound conclusions. Over the runs, two configurations with the highest minimum reward had the following parameters:

1. Buffer size: 100 thousand, 70 thousand
2. Learning rate: 0.00022, 0.003
3. Learning starts: both 5000
4. Target network update: 6780, 3040
5. Batch size: 128, 120

The observation led to the final rounded values shown in Table D.2, which were acquired through additional trial and error to lead to the results seen in the final experiment in Section 3.

[5]https://wandb.ai/site

## D Experiment Parameters

### D.1 Hardware & Software

| Hardware/Software | Model/Version |
|---|---|
| CPU | Intel i7-7700HQ @ 2.80GHz |
| GPU | Nvidia GTX 1060 Max Q 6GB VRAM |
| Memory | 16GB 2400 MHz |
| Operating System | Windows 10 |
| Python | v3.7.6 |
| Stable-Baselines | v2.10.2 |
| PyTorch | v1.8.1 |
| Tensorflow | v1.15.5 |

Table D.1: Experiment hardware information and software versions

### D.2 Model Hyperparameters

The DQN network architecture was chosen to have 2 hidden layers, both with 64 hidden neurons. The input dimension equals the state dimension, and the output dimension equals the number of actions. The remaining hyperparameters are shown in the table below.

| Hyperparameter | Range considered | Chosen value |
|---|---|---|
| $\gamma$ (exploration rate) | [0.99, 0.99] | 0.99 |
| $\alpha$ (learning rate) | [0.00001, 0.1] | 0.00005 |
| $\varepsilon_{init}$ (initial exploration rate) | [1, 1] | 1 |
| $\varepsilon_{min}$ (minimum exploration rate) | [0, 0.05] | 0.02 |
| $\varepsilon_{frac}$ (fraction of training with exploration) | [0.1, 0.2] | 0.1 |
| Batch size | [8, 128] | 120 |
| Learning starts | [5000, 10000] | 5000 |
| Buffer size | [25000, 200000] | 100000 |
| Target network update frequency | [10, 10000] | 4000 |

Table D.2: DQN Hyperparameters

### D.3 Seeds

Seeds were used in two parts of the experiment. One part was for training the stable-baselines model to ensure deterministic training, accomplished by setting the model's seed via the *seed*[6] parameter in the framework's model. The other part was for testing a policy by setting a seed before generating test environments, done by setting the seed both to Python's *random* module, as well as the *numpy.random* module. Additionally, the *deterministic* variable was set to *True* in the stable baselines model's *predict*[7] function. The seeds chosen are summarized in Table D.3 below.

| Phase | Seed |
|---|---|
| Training | 492883819 |
| Testing | 1361753209 |

Table D.3: Seed values for training and testing phase

[6]stable-baselines model seeding function - https://stable-baselines.readthedocs.io/en/master/modules/dqn.html#stable_baselines.deepq.DQN.set_random_seed

[7]stable-baselines predict function - https://stable-baselines.readthedocs.io/en/master/modules/dqn.html#stable_baselines.deepq.DQN.predict

## E Outlier Summary

The appendix section further illustrates the outlier cases where the DQN algorithm shows anomalous behavior compared to the decision tree and the mean. The box-plots below are indicative that the DQN agnet achieves better metrics overall: request satisfaction time is generally much lower, and the reward mean is slightly higher. However, a small number of outliers can be observerd where the performance is decreased compared to the worst-case scenarios in the decision tree.

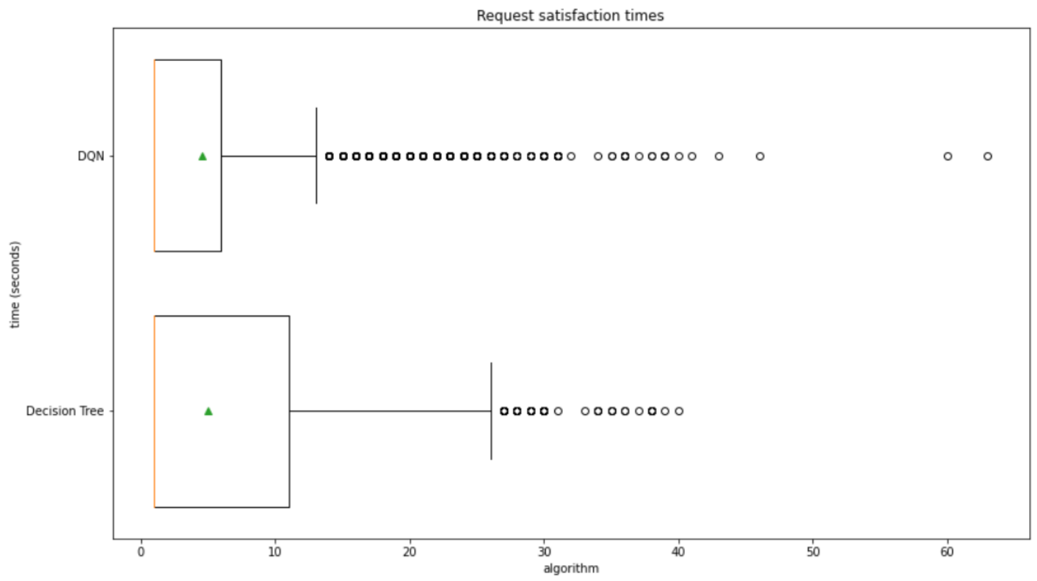


Figure E.1: DQN Agent vs Decision Tree request satisfaction time boxplot

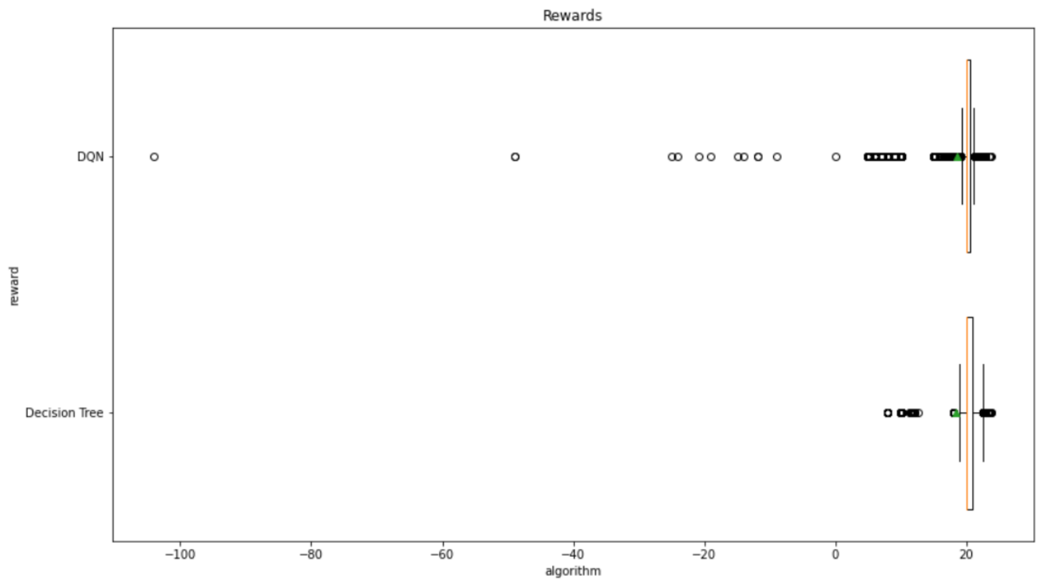


Figure E.2: DQN Agent vs Decision Tree reward boxplot

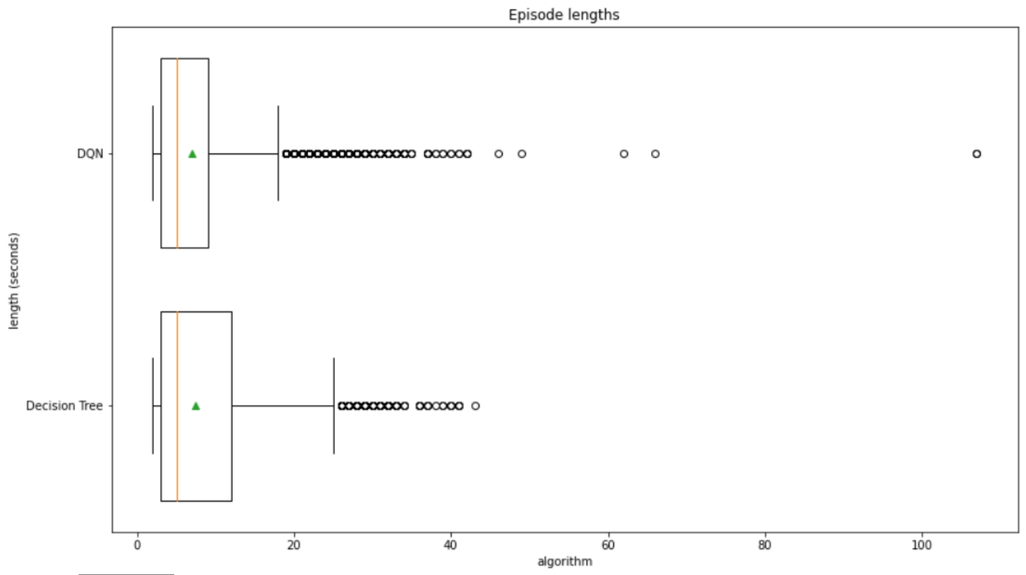


Figure E.3: DQN Agent vs Decision Tree episode lengths boxplot

Figure F.1 provides additional insights into each outlier action. Two apparent trends are that of shifts missed and false rejects. The automation overestimated the value of inappropriate actions too highly. In shift missed, the SLOpt action has no probability of occurring, indicating that the agent did not generalize perfectly.

Another apparent problematic trend is that of redundant prepares – the automation is very close to choosing the SSLopt action, which is much more suitable in cases where the automation cannot shift directly. Changes in reward incorporating additional guidance for the agent in this scenario would likely alleviate the issue.

It should be noted that idle time is generally not an outlier case on its own, as there are scenarios where the action is appropriate (e.g. when no driver request is pending, or when waiting for a few steps would result in a better outcome). The action distribution of idle time, however, indicates that there are cases where the agent is also likely to perform the Prepare Driver action but chooses not to in favour of Do Nothing.

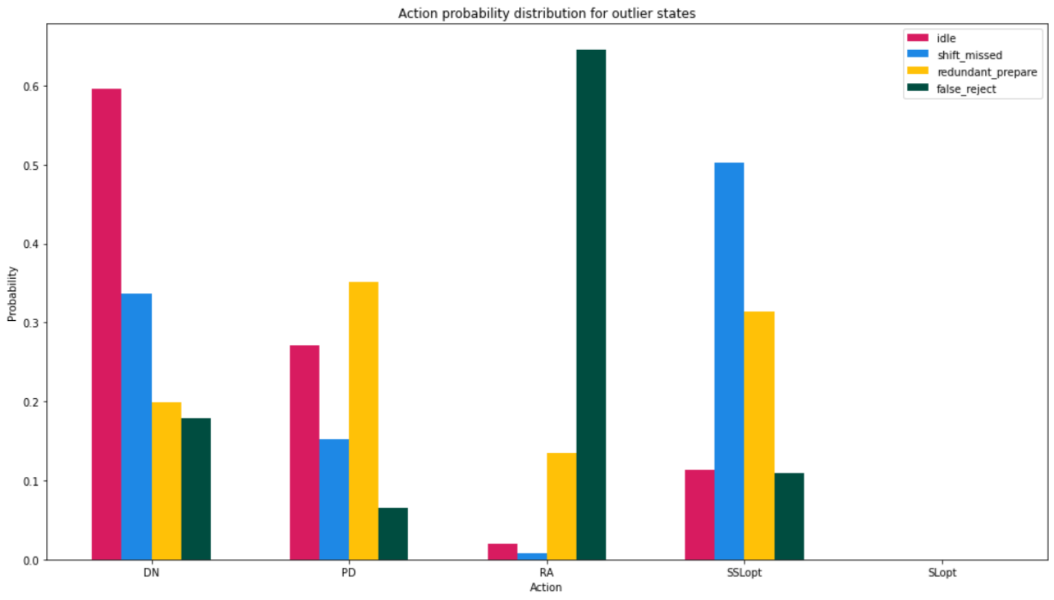


Figure E.4: DQN Agent action distributions per outlier state

## F Policy Action Distribution

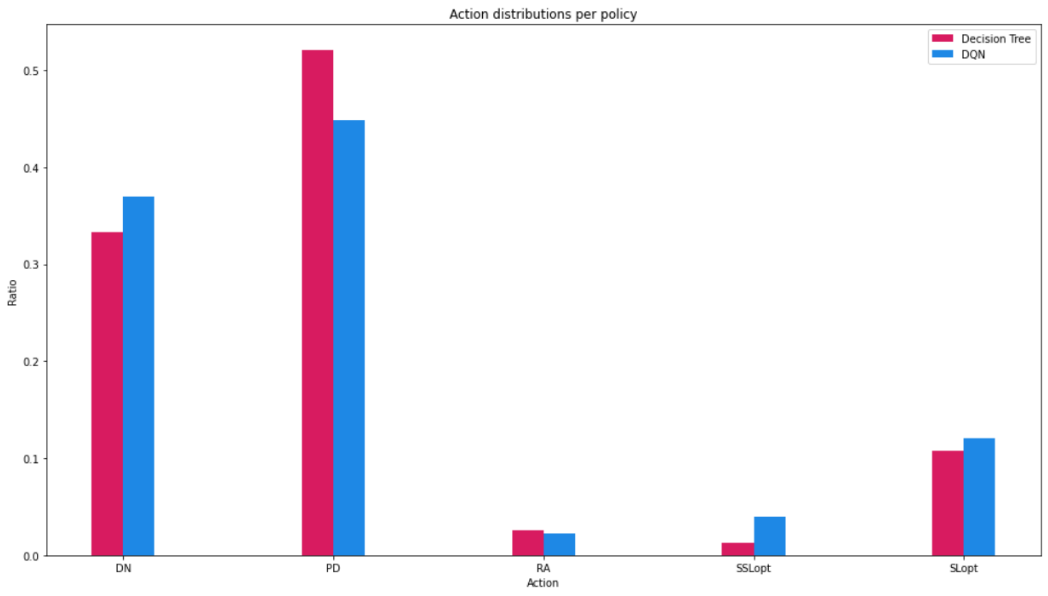


Figure F.1: Policy action distribution comparison. The similarity in the distributions likely indicate that the two policies are close to optimal