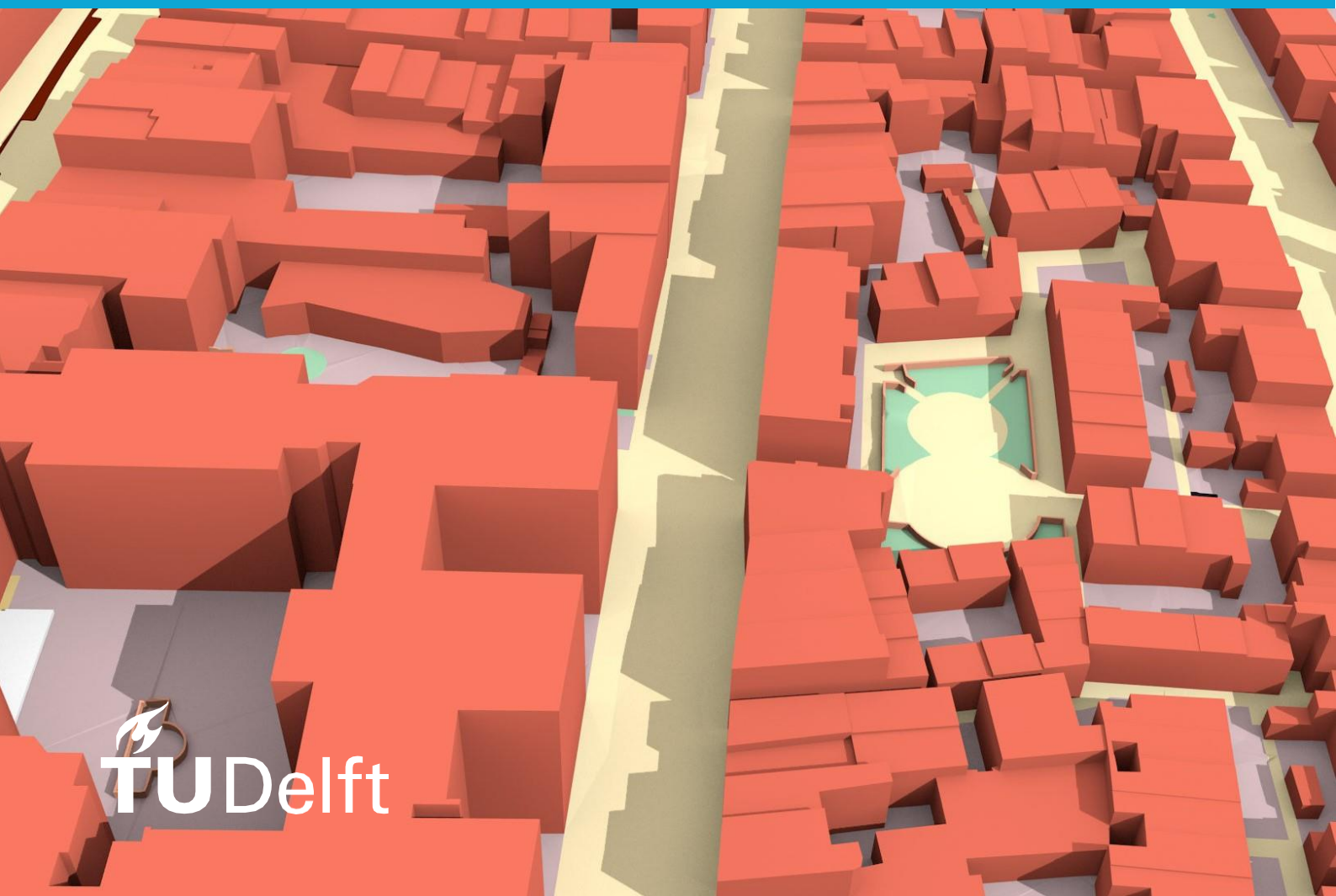
MSc thesis in Algorithmics

# Hierarchical Clustering Based State Abstraction In Reinforcement Learning

Yang Liu
2022



# HIERARCHICAL CLUSTERING BASED STATE ABSTRACTION IN REINFORCEMENT LEARNING

A thesis submitted to the Delft University of Technology in partial fulfillment of the requirements for the Degree of Master of Science in Embedded System

by

Yang Liu

in

August 2022

The work in this thesis was made in the:

Algorithmics group
Department of Software Technology
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology

EEMCS
TU Delft

Advisor: Dr. Matthijs Spaan
Supervisor: Dr. Yang Li

# ABSTRACT

Reinforcement learning (RL) has grown tremendously over one and a half decades and is increasingly emerging in many real-life applications. However, the application of RL is still limited due to its low training efficiencies and surplus training cost. The sampling and computation complexity normally depends on the size of the state space and splitting the state space can distribute computation and accelerate learning. State abstraction as a form of data-centric method shrinks the state space and reduces learning time, however, it is challenged by the fact that abstraction throws away information and might result in sub-optimal solutions. In this thesis, we propose the hierarchical clustering-based state grouping (HCSG) method to split the ground state space into clusters and train multiple agents for each cluster without changing the dimension of the state space. This approach allows us to distribute computation and improves training efficiency without losing the overall performance, and is also shown to outperform baseline and other state-of-art data-centric methods.

# ACKNOWLEDGEMENTS

Before you read my thesis that marks my Master of Science graduation. The motivation for proposal of this project is one of challenges in AI industry: low training efficiencies. In industries that generates products with short life cycle, the excessive training time in reinforcement learning could become a big disadvantage. Solving this problem could mark a drastic advance in AI industry. During the period of this project, I faced several challenges like limited research in state grouping and time-consuming training. Finally, I am happy that I have overcomed the obstacles and generated good results. The process of conducting the research widened my horizon and interests in reinforcement learning. I would like to thank my supervisor, Dr. Yang Li, for her assistance over the few months. She introduces me the topics I was not familiar with and helps me find the research direction.
I also want to thank my advisor Dr. Matthijs Spaan. He also helped me choose proposals and gave me many useful advice on my research.
I would also like to thank Casper for the Mediator environment he designed. It is a very fascinating environment and saved me much time in designing or implementing RL environments.
Also, I want to thank my fellow students in the Algorithm group who gave plenty of presentations on their researches which greatly widened my horizon in reinforcement learning.
I would thank my parents for their support on my studies and lives in the Netherlands. I made great memories and I would remember this time fondly.

I hope you enjoy reading my thesis.

Yang Liu
Delft, University of Technology
August 24, 2022

# CONTENTS


1 INTRODUCTION 1
- 1.1 Mediator project . . . 2
- 1.2 Research summary . . . 2
  - 1.2.1 Research questions . . . 3
  - 1.2.2 Methods and objectives . . . 3
  - 1.2.3 Contributions . . . 4
- 1.3 Outline . . . 4

2 RELATED WORK 5
- 2.1 Re-sampling and Re-weighting . . . 5
- 2.2 Transfer learning . . . 6
- 2.3 State abstraction in reinforcement learning . . . 6

3 PRELIMINARIES 7
- 3.1 Markov Decision Process . . . 7
- 3.2 Reinforcement learning . . . 7
- 3.3 Double Deep Q-Network . . . 9
- 3.4 Hierarchical clustering . . . 10
- 3.5 State abstraction . . . 11

4 SIMULATION ENVIRONMENTS 12
- 4.1 Mediator environment . . . 12
- 4.2 Markov Decision Process models . . . 13
- 4.3 Data imbalance in reinforcement learning . . . 15

5 METHODS 17
- 5.1 Hierarchical Clustering-based State Grouping . . . 19
  - 5.1.1 Motivation . . . 19
  - 5.1.2 Methods . . . 21
- 5.2 Reinforcement learning . . . 24
  - 5.2.1 Agent tester . . . 24
  - 5.2.2 Behaviour cloning . . . 24
  - 5.2.3 Exploration in critical scenarios . . . 25
  - 5.2.4 Overall algorithm . . . 25
- 5.3 Policy network . . . 26
  - 5.3.1 Master policy . . . 27
  - 5.3.2 Sub-policy networks . . . 27

6 EXPERIMENTS AND RESULTS 28
- 6.1 OpenAI GYM . . . 28
- 6.2 training setup . . . 28
  - 6.2.1 Training parameters . . . 28
  - 6.2.2 Algorithm implementation . . . 29
  - 6.2.3 Algorithm parameter selection . . . 29
- 6.3 Performance encoding . . . 30
- 6.4 Comparison method . . . 30
  - 6.4.1 Tree decision algorithm . . . 30
  - 6.4.2 DDQN Agent with mixed environment transfer . . . 30
- 6.5 Evaluation metrics . . . 31
- 6.6 Overview of experiment models . . . 32
- 6.7 Results and Discussion . . . 32
  - 6.7.1 Performance in environments . . . 33
  - 6.7.2 Performance in observation cases . . . 36
  - 6.7.3 Driver fitness . . . 36
- 6.8 Ablation study . . . 40
  - 6.8.1 Ablation statistics . . . 40

6.8.2 Behaviour cloning . . . . . 40
6.8.3 Exploration in critical scenarios . . . . . 41
6.8.4 Visualization of the neural network . . . . . 42
7 DISCUSSION AND CONCLUSION 43
7.1 Conclusion . . . . . 43
7.2 Discussion . . . . . 43
7.2.1 Algorithm advantages . . . . . 43
7.2.2 Drawbacks of the agent . . . . . 44
7.3 Future work . . . . . 44
A APPENDIX A. EXPERIMENT SET UP FOR THE BASE ENVIRONMENT 49
B APPENDIX B. DECISION TREE/BASELINE ALGORITHM 50
C APPENDIX C. ENVIRONMENT ENCODING 51
D APPENDIX D. STATE ENCODING 52
E APPENDIX E. EXPERIMENTS SETUP FOR BASE, L4 AND L3 ENVIRONMENTS 53
F APPENDIX H. EXPERIMENT SETUP TTDU CONFIGURATION 56
G APPENDIX I. CASE TEST RESULTS 57
H APPENDIX J. EXPERIMENT SETUP 5 61
I APPENDIX K. EXPERIMENT SETUP 6 62

# LIST OF FIGURES


Figure 1.1 Interaction of the mediator system with the environment. . . 2
Figure 1.2 Overall structure of the HCSG algorithm . . . . . 4
Figure 4.1 The transition of the Mediator environment. . . . . 14
Figure 4.2 Reward model of the Mediator environment. . . . . 14
Figure 4.3 Left: Count of each environment state for training 100k steps. Right: Distribution of state space for an experiment in Appendix.I.1. . . . . 15
Figure 4.4 Model's accuracies on head and tail scenarios for various head-tail split ratios. . . . . 16
Figure 5.1 HCSD reinforcement learning algorithm diagram . . . . . 18
Figure 5.2 Training average episode reward for the base, L3 and L2 environments . . . . . 20
Figure 5.3 Optimal actions for various state combinations. . . . . 21
Figure 5.4 Policy structure for the HCSG-RL algorithm . . . . . 26
Figure 5.5 HCSG-RL master policy . . . . . 27
Figure 6.1 Hierarchical clustering based state grouping . . . . . 29
Figure 6.2 State clusters are covered by the resultant sub-policies. . . . . 33
Figure 6.3 Model's accuracies on the head and tail scenarios for various head-tail split ratios from HCSG-RL algorithm with $\theta$ of 100%. Accura- cities calculated based on the per cent of actions with the maximum reward. . . . . 34
Figure 6.4 Left: Average test episode reward on the base environment for HCSG-RL models with dimensions of 1, 2 and 3. Right: Average test episode reward on-base environment on HCSG-RL model, MET model and baseline. . . . . 35
Figure 6.5 Cumulative average episode reward with an increasing number of environments involved. . . . . 35
Figure 6.6 Driver unfit count(left) and duration(right). . . . . 37
Figure 6.7 Rendering of one episode for training HCSG D3 model on the base environment. . . . . 38
Figure 6.8 Rendering of one episode for baseline algorithm on-base environment. . . . . 38
Figure 6.9 State clusters are covered by the resultant sub-policies. . . . . 39
Figure 6.10 State clusters are covered by the resultant sub-policies. . . . . 39
Figure 6.11 Ablation statistics . . . . . 40
Figure 6.12 Training results with(left) and without(right) behaviour cloning. 41
Figure 6.13 train loss(left) and rewards(right) for agents with various train-trajectory ratios($500_0 means no behaviour cloning$). 41
Figure 6.14 Average episode rewards . . . . . 42
Figure 6.15 Neural net visualization of HCSG sub-policies. . . . . 42
Figure B.1 Decision tree/baseline algorithm. . . . . 50
Figure G.1 Normalized case test reward of HCSG at D1 and MET algorithm . . . . . 57
Figure G.2 Normalized case test reward of HCSG at D1 and pre-trained model . . . . . 57
Figure G.3 Normalized case test reward of HCSG at D1 and baseline algorithm . . . . . 58
Figure G.4 Normalized case test reward of HCSG at D2 and MET algorithm . . . . . 58

Figure G.5 Normalized case test reward of HCSG at D2 and pre-trained model . . . 58
Figure G.6 Normalized case test reward of HCSG at D2 and baseline algorithm . . . 59
Figure G.7 Normalized case test reward of HCSG at D3 and MET algorithm . . . 59
Figure G.8 Normalized case test reward of HCSG at D3 and pre-trained model . . . 59
Figure G.9 Normalized case test reward of HCSG at D3 and baseline algorithm . . . 60

# LIST OF TABLES


Table 4.1 Configuration parameters for the Mediator environment . . . 12
Table 4.2 A description of the state space and the state variable domains 13
Table 4.3 A description of the available actions. . . . . 13
Table 5.1 Some concepts of the HCSG algorithm . . . . . 17
Table 5.2 Test rewards for the agents trained in base, L3 and L2 environments. . . . . 20
Table A.1 Experiment A . . . . . 49
Table E.1 Experiment Base Environment . . . . . 53
Table E.2 Experiment L4 environment . . . . . 54
Table E.3 Experiment L3 environment . . . . . 55
Table F.1 Experiment 5 . . . . . 56
Table H.1 Experiment 5 . . . . . 61
Table I.1 Experiment 5 . . . . . 62

# List of Algorithms

5.1 Hierarchical clustering based state grouping . . . . . 23
5.2 DDQN Agent with behaviour cloning . . . . . 26
6.1 DDQN Agent with mixed environment transfer . . . . . 31

# 1 INTRODUCTION

According to the latest research study, the global demand for semi-autonomous driving vehicles is predicted to reach a compound annual growth rate (CAGR) of 20.8%, during the period 2021 to 2028. Self-driving artificial intelligence(AI) technologies contribute largely to this growth, as they enhance the human driving experience and functionalities of the driver-assistant system.
The Society of Automotive Engineers (SAE) proposed 6 levels of vehicle automation ranging from level 0 fully driver control to level 5 fully automation control. The most advanced driver control system has reached level L3. It is arguable that more automation is better, which may not always be the case. That depends, for example, on driver preference and the complexity of road conditions. Thus, a smart level switching policy offers promising growth opportunities for the semi-autonomous cars industry.
Over the last decades, rule-based methods in autonomous driving have been deeply investigated for decision-making algorithms. Montemerlo et al. designed a finite state machine (FSM) Junior software under various scenarios to make the robot robust to unconsidered situations [19]. They indicated that the model is less reliable in practice, involving a diverse set of traffic participants. In the field of automation level switching, Van Wyk et al. solve the "out of the loop" (OOTL) problem by proposing a mathematical framework to determine the optimal automation level at any point along a trip [37]. There are imitations to the rule-based methods. Firstly, it relies heavily on driving data and is largely unstable. A small change in the data may result in a structural change in the algorithm [21]. Secondly, rule-based methods are susceptible to human cognitive biases and limited by human experience [35], and can result in sub-optimal policy in complicated scenarios.
Reinforcement learning (RL) is a self-learning algorithm based on a system of rewards and punishments. As the latest machine learning model besides supervised learning and unsupervised learning, it learns through trial and error aimed at maximizing environmental rewards. RL agents outperform rule-based methods in complex environments and learn without relying on labelled driving data [27, 6].
Despite its outstanding performance over well-designed baselines, RL agents suffer from low training efficiencies [45]. That is one of the biggest challenges in implementing machine learning models in the industry. Several reasons can result in this issue.
Firstly, the state space can be very huge and with limited time and memory resources, it can be extremely difficult to explore the entire state space extensively. Normally a huge network is required to train on a large state space which requires tremendous computation and memory budget. And the budget increase exponentially with the size of state space (curse of dimensionality).
Secondly, the training in one environment may become biased because some states occur more frequently than others, which tends to form a 'long-tail' distribution. This is due to the fact that an RL agent ultimately became fond of selecting actions that lead the environment to reach specific optimal states [13]. Models trained on these biased data tend to perform well on over-sampled subgroups while sacrificing the performance of under-sampled subgroups [2]. The RL agent might need a huge amount of time to accumulate enough learning experience for the minority data. Another cause of the biased training data is the low diversity of the training environment [44]. Training in a single environment may not cover all possible scenarios

and agents tended to avoid certain states with low returns. The ground state space may not be fully explored and the trained agents tend to perform badly in unseen scenarios. For example, in robot control, the agent trained in manually designed simulation environments generalizes poorly in realistic environments [28].
Current AI industries utilize big data plus huge computing power to explore the bottleneck of deep neural networks. With convenient software open source repository and highly efficient hardware realizations. More AI industries start to work on improving training data. The paradigm shift from model-centric to data-centric AI means, that instead of building models and configuring training parameters, AI developers endeavour to build balanced training experience to train well-performed models. In this paper, we would solve the low training efficiency issues using data-centric methods.

## 1.1 MEDIATOR PROJECT

The Mediator project develops a mediating system that manages to switch among current available autonomous levels, based on driver distractions. Fig 1.1 illustrates the interaction. The Mediator controller takes input the driver and vehicle-related states including driver distraction level and automation level, and outputs available actions for control of the environment. The purpose of the Mediator agent is to maximize safety and comfort, based on available driver states.

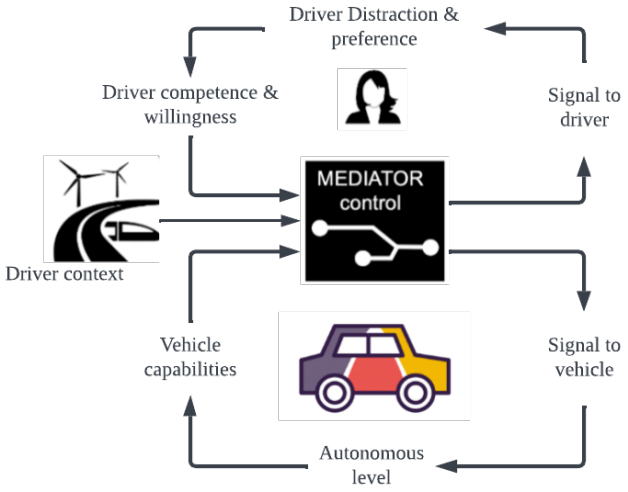



**Figure 1.1:** Interaction of the mediator system with the environment.

## 1.2 RESEARCH SUMMARY

In this thesis, we focus on improving RL training efficiency from two aspects:

- Decomposing the state space into smaller clusters with a low training budget
- Balancing the training dataset so that agents do not lose experience during the learning

### 1.2.1 Research questions

The purpose of the RL agent is to improve RL training efficiency while also maintaining/improving the model's overall performance. Based on this goal, we propose the following research questions:
The first question is related to our methods:

- How to improve RL agents' training efficiency while also ensuring that agents' overall performances are not degraded?

The following questions are about evaluation:

- How to evaluate the advantage of the HCSG algorithm in improving agents' training efficiencies and overall performance?
- How well does the method outperform other competitive methods?

### 1.2.2 Methods and objectives

Two of the most important components required for a well-trained RL agents are: a well-defined reward model that is concise, specific and effective, and an exploration mechanism that ensures various states are covered. The reward function basically guides the learning of agents during RL training. Designing a reward function in a complex environment is normally a challenging task and easily exposed to developer's bias. In this thesis, we focus on state exploration and use data-centric methods to generate balanced and easy-to-train dataset.
The main research questions are solved in two ways: decomposing the ground state space and distributing the training with less training budget, so as to accelerate training, and using hierarchical tree-based state decomposition to produce a balanced training experience so as to improve RL agents' overall performance. The training efficiencies are measured by evaluating agents' performance with limited training budget. The data imbalance is one of the main causes of the low training efficiencies. It is defined as the skewed distribution of data where some of the data lies in the majority while the rest lies in minority. The problem was challenged by using multiple environments to fully explore the whole state space. Training in multiple environments requires the agent to transfer learned experience across environments.
Training RL agents in multiple environments/MDPs propose big challenges. Mutti et al. proposed a transfer learning method for unsupervised learning in multiple environments [20]. The algorithm first pre-trains the model on the whole environment and fine-tuned on sub-environments. However, in case of surplus scenarios, the model tends to lose historical experience and training becomes increasingly difficult as more environments are involved.
In this thesis, instead of training the RL agent on the whole state space, we split the whole environment into smaller ones, each consists of specific distribution of states and train an agent for each subset. For smaller state space, we can save computation cost by using smaller policy networks and less training episodes, so as to accelerate learning.
Fig 1.2 illustrates the overall structure of our proposed method. It consists of three parts: Hierarchical clustering based state grouping, reinforcement learning and policy network. We adopted hierarchical clustering method for state space splitting and combination to minimize the number of agents. The resultant model is a hierarchical structure consisting of one master policy and multiple sub-policies.

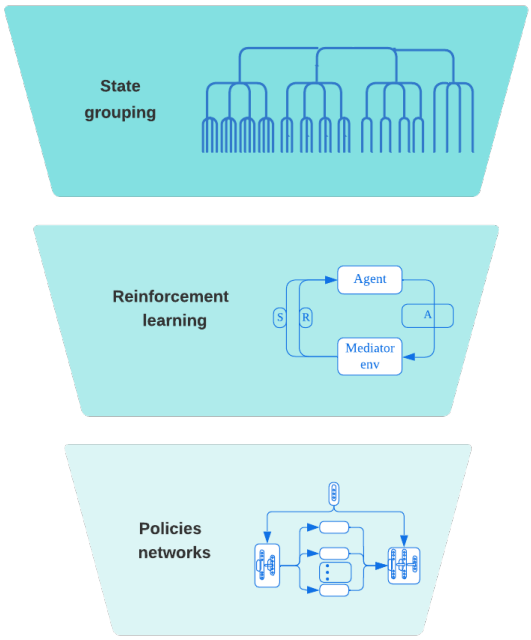



**Figure 1.2:** Overall structure of the HCSG algorithm

### 1.2.3 Contributions

Our main contributions are as follows:

- We proposed using hierarchical clustering method for state abstraction to accelerate RL training. To the best of our knowledge, there is no method at the moment to adopt hierarchical clustering for state abstraction in reinforcement learning.
- Our proposed method accelerates learning and achieve better performance compared with baseline methods.

## 1.3 OUTLINE

The thesis is structured as follows, Chapter 2 introduces some research studies and some knowledge background are explained in Chapter 3. Chapter 4 depicts the Mediator simulation environment and describes the long-tail problem for RL training in the environment. Chapter 5 presents the main algorithms. The experiment details and results are shown in Chapter 6. Chapter 7 concludes the work and provides future research directions.

# 2 RELATED WORK

Solutions for the low training efficiencies rely on environment and algorithm respectively. An environment that can systematically explores various scenarios helps learn a generalized agent. An example of this environment is called CoinRun, introduced by OpenAI to investigate insights for improving an agent's overall performance and training efficiencies [5]. This environment can produce various distributions of CoinRun levels and train agents for each of the levels.
There are plenty of research works trying to solve low training efficiency issues. most of which focus on the data level, including data-based methods like re-sampling/ re-weighting and state abstraction, and policy-based such as transfer learning.

## 2.1 RE-SAMPLING AND RE-WEIGHTING

Data-based methods focus on solving the data imbalance issues, including re-sampling and re-weighting. Re-weighting alleviates performance bias by adjusting the weights of dataset classes by means of class frequency [33], loss values [22] or prediction probabilities [16]. Existing sample re-weighting methods require integrated optimizations of models and weighting parameters, which requires expensive second-order computation [47]. Our algorithm is separated from the RL training and does not add computation during training.
The re-sampling method focuses on the data itself and solves the data imbalance problem by altering the data distribution. It can be divided into two types: over-sampling (ROS) which enlarges the population for the minority distribution [40, 3] or under-sampling (RUS) which discards data from the majority group [24, 41]. Some use a hybrid of the two methods [25, 15].
As for re-sampling, when the class is extremely skewed, over-sampling tends to overfit to tail scenarios while under-sampling tends to degrade performance on head scenarios [46]. To cope with this issue, class-balanced re-sampling and scheme-oriented sampling follow. SimCal [31] proposed a bi-level sampling strategy to handle long-tailed instance segmentation. Dynamic curriculum learning (DCL) [34] developed a curriculum strategy in which each class is assigned a sampling probability that is inversely proportional to instances of the class. Following that, Feature Augmentation and Sampling Adaption (FASA) [42] proposed to use training loss to adjust the feature sampling rate so that tail distribution can be sampled more often. Re-weighting techniques improved model performance on the tail distribution by adjusting loss weights based on label frequencies [23, 8], prediction hardness [17] and etc. Class re-balancing methods are proven to achieve comparable or even better performance than other long-tail learning paradigms.
The HCSG method is similar to re-sampling which balances training data to improve the model's training efficiency and overall performance. Re-sampling techniques are playing on a performance seesaw, they improve the performance of the tail-class at the cost of degraded head-class performance [46]. The HCSG algorithm, instead, benefits overall performance by splitting the training on head and tail data distribution.

## 2.2 TRANSFER LEARNING

Transfer learning seeks to transfer knowledge among datasets to improve agent learning during RL training. In deep long-tail learning, prominent methods include experience transfer, pre-training-based methods and policy distillation. Experience transfer mainly tries to transfer the experience learned from the majority group to the minority group, so as to improve the overall performance. Successful implementation is a major-to-minor translation (M2m) [12] transferring the experience via perturbation-based optimization, and MetaModelNet [32] trains head and tail samples separately and use a meta-network to map few-shot model parameters to many-shot parameters. In model pre-training, the agent can be initially pre-trained with tail samples for representation and then fine-tuned with a balanced dataset (DSTL [7]). Self-supervised pre-training (SSP) [39] proposed to pre-train the model on the whole dataset followed by standard training on long-tail data. Learning from multiple experts (LFME) [38] divides the whole long-tail dataset into subsets with smaller degrees of imbalance and trains multiple experts for each subset, it finally trained a unified agent using adaptive knowledge distillation.
Transfer learning is worth exploring for improving performance on both head and tail data. However, transfer learning only worked well if the current and target policy is similar. In the case of RL training, the agent might need to perform distinctly in head and tail scenarios. If the training data might be too far from the old training experience, the trained model might perform worse than expected (negative transfer [43]). To better conduct transfer learning for long-tail learning remains an open question.

## 2.3 STATE ABSTRACTION IN REINFORCEMENT LEARNING

In reinforcement learning, the agent learns from various state combinations. A well-trained model normally ought to perform well in possibly all-state scenarios. State abstraction is a technique that compresses the feature size of states and binds similar states together. Kim et al. proposed a state-grouping algorithm that utilizes a genetic optimizer to learn the optimal action for each sub-group. He combines states by calculating distances based on action distribution. The method helps avoid excessive exploration and contributes to a significant reduction in initial learning time [14]. Following that, Feiyun et al. defined a group-driven RL algorithm in the mHealth and proved it gains obvious advantages over the state-of-the-art RL methods [48]. These two types of research use k-means methods for clustering the states. It effectively eliminates unnecessary explorations so as to improve overall performance with less memory and less time. However, most state abstraction methods condense state space which might result in the loss of important information and result in sub-optimal solutions. The HCSG algorithm separate state space to smaller ones and would not cause information loss.

# 3 PRELIMINARIES

## 3.1 MARKOV DECISION PROCESS

The RL environments are modeled as an Markov decision process (MDP), specified by a five-tuple $< S, A, P_t, R, \gamma >$, where

- S is the space of states from the environment that begins with an initial state $S_0$.
- A is a set of actions.
- $P(s'|s,a) \in (0,1)$ is the transition probability from state s to state $s'$ by taking action $a$
- $R(s',s,a)$ is the reward function by taking action $a$ that transits state s to $s'$.
- $\gamma$ is the discount factor for future rewards.

In reinforcement learning, the agent learns policies by interacting with an environment through MDP. It explores policies that return actions to maximize long-term rewards. Through the MDP, the environment was able to update its states by executing actions predicted by the RL agent.
The agent interacts with the environment at discrete time steps. At each time step during RL training, the agent observes states from the environment and explores the best action for these states. The environment then executes the action and enters a new state. The MDP generates a reward according to $R(s',s,a)$ on this process that is fed to the agent for policy updates. Upon reaching a terminal state, a new episode is started with a new initial state of the environment.

## 3.2 REINFORCEMENT LEARNING

Reinforcement learning is a machine learning method, it enforces learning by rewarding desired behaviour outcomes and punishing undesired ones. It is akin to human learning which learns by trial and error. The appeal of RL is that it can be used in very complicated scenarios that humans may find hard to solve. The RL agent learns by interacting with the MDP environment which produces rewards or penalties for the RL agent. In each training episode, the RL agent stores training experience into the reply buffers and update its policy network regularly. The episode repeats until the behaviour of the RL agent is stabilized. Reinforcement learning can be categorized into two kinds:

- **Model-based learning**: the learning is based on a model of the environment by taking actions and observing the outcomes that include the next state and the immediate reward
- **Model-free learning**: the learning does not use the transition probability distribution (and the reward function) associated with the MDP

Model-based learning is sample-efficient, it does not require further sampling once the model and cost function are known. Besides, model-based methods are transferable to similar tasks and simulations as they already have a base model to work

with. However, the performance of the agent from model-based learning is limited by the historical experience they have learned, which limits the accuracy of the model. While model-free agents do not have this problem, as they learn directly from the observed data. But because of this, they normally required a large amount of data to learn enough experience.

**Exploration vs. exploitation**
Exploration and exploitation are two ways agents select action to feed the environment. In exploration, the agent selects actions by searching from the action space, while in exploitation, the agent section action is based on the agent policy itself. Exploration-exploitation trade-off plays an important role in reinforcement learning [10]. Surplus exploitation might cause the agent to stuck into local maxima while excessive exploration wastes time on solutions that are less likely to be optimal and lose information that has been gathered. A popular method to address this issue is the $\epsilon$-greedy policy [18]

$$\pi(s|a) = \begin{cases} max\ Q_t(a) & with probability 1-\epsilon \\ random\ action(a) & with probability \epsilon \end{cases} \tag{3.1}$$

where $\epsilon$ is a hyper-parameter tuned during RL training. High $\epsilon$ values indicate a large exploration since there is a high possibility that the agent explores random actions. Likewise, when $\epsilon$ is small, we focus more on exploitation as the agent is more likely to use the existing policy network to make predictions. During reinforcement learning, $\epsilon$ normally starts off large and decays after each episode to help reach convergence and realize an exploration-exploitation trade-off.

**State-action value function**
Given a reward model, the objective of RL is to maximize the long-term return $J(\pi)$ from the environment. The optimal policy $\pi^*$ should pick actions that result in the maximum average episode reward

$$J(\pi) = \frac{1}{N}\frac{1}{T}\sum_{t=0}^{T-1} r_t \tag{3.2}$$

$$\pi^* = arg\ max\ J(\pi) \tag{3.3}$$

where N is the number of episodes, and T represents the episode length.
Rewards are discounted with time to help RL agents learn actions that produce long-term rewards. The discount gives higher weight to the latest rewards, the reward at time step t can then be defined as

$$r_t = \sum_{k=t}^{T} \gamma^{k-t} r_k(s_k, a_k) \tag{3.4}$$

RL agents choose actions based on value function. Such kinds of RL agents are called value-based agents. They store the value function on which they make their decisions. Value-based agents use V-function to measure the goodness of each state according to the reward $J(\pi)$ following a policy $\pi$. In a formal way, the value of $V_{\pi(s)}$ is defined as:

$$V_\pi(s) = E_\pi[R_t|s = s_t] = E_\pi[\sum_{j=1}^{T} \gamma^j r_{t+j+1}|s = s_t] \tag{3.5}$$

where $\gamma$ is the discount factor.
The equation describes the expected value of the episode reward at time t and follows the policy $\pi$. The expectation was used since the environment transition might be stochastic.

## 3.3 DOUBLE DEEP Q-NETWORK

Q function measures the value of the action / state-action pairs which is the value of taking action an at state s the following policy $\pi$:

$$Q_\pi(s,a) = E_\pi[R_t|s = s_t, a = a_t] = E_\pi[\sum_{j=1}^{T} \gamma^j r_{t+j+1}|s = s_t, a = a_t] \tag{3.6}$$

The state-value function is equivalent to the sum of action-value functions multiplied by the policy probability for selecting the actions:

$$V_\pi(s) = \sum_a \pi(a|s)\ Q_\pi(s,a) \tag{3.7}$$

When we solve MDP we are actually trying to find the optimal state-value function as well as the action-value function:

$$V^*(s) = max_\pi\ V_\pi(s) \qquad Q^*(s,a) = max_\pi\ Q_\pi(s,a) \tag{3.8}$$

Bellman equation is one of the central elements required for calculating the value functions. It decomposes the value function into immediate rewards plus discounted future values. The equations for state-value function and action-value function are defined as:

$$V_\pi(s) = \sum_a \pi(a|s) \sum_{s'} P^a_{ss'}(r(s,a) + \gamma V_\pi(s')) \tag{3.9}$$

$$Q_\pi(s,a) = \sum_{s'} P^a_{ss'}(r(s,a) + \gamma \sum_{a'} \pi(a'|s')\ Q_\pi(s',a')) \tag{3.10}$$

In reinforcement learning, the RL agent improves its policies by updating its Q values

$$Q_{t+1} = E'_s[r + \gamma max'_a Q_i(s',a'|s,a)]\ \forall s,s' \in S\ and\ a,a' \in A \tag{3.11}$$

where S and A are state and action spaces. Since it is not possible to compute values for all state-action pairs, these values can be approximated using deep neural networks. A well-known approach to this policy function approximation method is the Deep Q-network (DQN), in which the parameters of the network $\theta$ are updated based on loss values calculated from the difference between the predicted and the optimal actions

$$L_i(\theta_i)^{DQN} = E_{(s,a,r,s')}[(r + max\ Q(s',a';\theta_i^-) - Q(s,a;\theta_i))^2] \tag{3.12}$$

where $max\ Q(s',a';\theta_i^-)$ is the target Q value for the next state, which is calculated by taking the maximum value of all possible actions. However, these values are overestimated. Double Deep Q Network (DDQN) proposes to use a separate target network to estimate the value of the chosen action by the target network instead. And now the loss function of the DDQN becomes

$$L_i(\theta_i)^{DDQN} = E_{(s,a,r,s')}[(r + Q_{target}(s', arg\ max(s',a';\theta_i^-);\theta_i^-) - Q(s,a;\theta_i))^2]$$

The new target Q values reduce overestimation and can result in more stable and reliable training performance [30].
With the loss values, the agents are able to update their policies. Various policy gradient methods are used to find the optimal policies. Policy gradient methods learn the policy network directly instead of value functions. It updates the policy parameters using gradient descent by the loss values

$$\theta_{i+1} = \theta_i + \alpha \nabla L_i(\theta_i)^{DDQN} \tag{3.13}$$

As the loss values are calculated based on the predictions of the Q networks which are related to the network parameters, the gradient updates of the loss values and the updated parameters can then be expressed as follows

$$\nabla L_i(\theta_t)^{DDQN} \propto E_{s_i,a_i} \sim \pi[R_i \frac{\nabla \pi(a_i|s_i,\theta)}{\pi(a_i|s_i,\theta)}] \tag{3.14}$$

$$\theta_{i+1} = \theta_i + \alpha \frac{\nabla \pi(a_i|s_i,\theta)}{\pi(a_i|s_i,\theta)} \tag{3.15}$$

Reply buffer is a buffer that stores the following information for each step during RL training:

$$< S_i, A_i, S_i^1, R_i, D_i >$$

where i is the position of the information in the reply buffer, S is the old state, A is the action, $S^{'}$ is the new state, R is the reward, D is the episode done information, with true representing the environment reaches its terminal state.
RL agents learn from the information in the reply buffer which is also called a learning experience. Every several episodes, the agent randomly selects a fixed amount of state-action pairs from the reply buffer and uses this information to update its policy parameters. Thus, a well-trained agent should have enough and balanced learning experience in reply buffer for adequate and useful policy updates. RL agents are trained in episodes. The initial state for each episode can affect the follow-up actions and states, and thus influence the state distribution of this episode. The episode termination prevents the agent from diverging too far from the target.

## 3.4 HIERARCHICAL CLUSTERING

Hierarchical clustering is an unsupervised learning algorithm that seeks to build a hierarchy of clusters where each cluster is distinct from other clusters, and objects in each cluster in broadly similar to each other. There are two ways to create a cluster hierarchy:

- Agglomerative: a bottom-up approach
- Divisive: a top-down approach

The agglomerative approach starts by treating each end node as a cluster and combining similar clusters for a new cluster. It repeats until all possible clusters are merged. The metrics used to calculate the similarity among the clusters can be the Manhattan or Euclidean distance.

$$D_m = \sum_i |a_i - b_i| \tag{3.16}$$

$$D_e = \sqrt{\sum_i (a_i - b_i)^2} \tag{3.17}$$

where $a_i$ and $b_i$ are position of node i, $D_m$ is the Manhattan distance and $D_e$ is the Euclidean distance
Divisive clustering is basically the opposite of agglomerative and was initially published as the DIANA (Divisive ANAlysis Clustering) algorithm [11]. It begins with one cluster that includes all the records and is split into unsimilar clusters. The process repeats until all possible clusters are split.

## 3.5 STATE ABSTRACTION

The purpose of state abstraction is to condense the state space of an environment by grouping together similar states in the sense that it does not change the underlying problem [1]. The state abstraction maps each environment to a finite abstract state. An obvious benefit of using state abstraction is it increases effective sample size so as to lower estimation error. For example, when we collect N samples each (s, a) pairs the abstraction maps $s^1$ and $s^2$ to the same abstract state. When we apply state abstraction, for the one state pairs (x, a), we got 2n samples, That means we double the sample size for estimating the transition and reward functions which results in lower estimation errors.
The advantage of state abstraction comes with a drawback. When state abstraction aggregates distinct states, it may cause a loss of important information and lead to sub-optimal solutions. In other words, it causes high approximation errors. Thus how to combine states remains a theme for the trade-off between approximation error and estimation error. The exact state abstraction aggregates states that are strictly similar. It is defined as:

**Definition 3.5.1** (Abstraction abstractions)**.** Given MDP M = (S, A, P, R, $\gamma$) and state abstraction $\theta$ that operates on S, define the following types of abstractions:

- $\theta$ is an $\pi^*$-irrelevant if there exists an optimal policy $\pi^*$, such that $\forall s^1,\ s^2 \in S$ where $\theta(s^1) = \theta(s^2), \pi^*_M(s^1) = \pi^*_M(s^2)$.
- $\theta$ is $Q^*$-irrelevant if $\forall s^1,\ s^2$ where $\theta(s^1) = \theta(s^2), \forall a \in A, Q^*_M(s^1, a) = Q^*_M(s^2, a)$.
- $\theta$ is model-irrelevant if $\forall s^1,\ s^2$ where $\theta(s^1) = \theta(s^2), \forall a \in A, x' \in \theta(S)$, The transition over S is transformed to the transition over $\theta(S)$.

$$R(s^1, a) = R(s^2, a), \sum_{s' \in \theta^{-1}(x')} P(s'|s^1, a) = \sum_{s' \in \theta^{-1}(x')} P(s'|s^2, a) \tag{3.18}$$

Exact abstraction is hard to realize and detect. Approximate state grouping can still preserve near-optimal behaviour at a high level under $\epsilon$:

**Definition 3.5.2** (Approximate abstractions)**.** Given MDP M = (S, A, P, R, $\gamma$) and state abstraction $\theta$ that operates on S, define the following types of abstractions:

- $\theta$ is an $\epsilon_{\pi^*}$-approximate $\pi^*$-irrelevant abstraction, if there exists an abstract policy $\pi : \theta(S) \to$ A, such that $|V^*_M - V^{[\pi M]}_M|_{\inf \leq \epsilon^*_Q}$
- $\theta$ is $\epsilon^*_Q$-approximate $Q^*$ - irrelevant abstraction if there exists an abstraction Q-value function $f : \theta(S)xA \to R$, such that $|[f]_M - Q^*_M|_{\inf \leq \epsilon^*_Q}$
- $\theta$ is $(\epsilon_R, \epsilon_P)$-approximate model-irrelevant abstraction if for any $s^1$ and $s^2$ where $\theta(s^1) = \theta(s^2), \forall a \in A$,

$$|R(s^1, a) - R(s^2, a)| \leq \epsilon_R,\ |\epsilon(P(s^1, a) - \epsilon(P(s^2, a)))| \leq \epsilon_P \tag{3.19}$$

# 4 SIMULATION ENVIRONMENTS

Developing and testing policy strategies in real-world driving scenarios is usually challenging and even not feasible. Creating a simulation environment is an alternative solution. In this paper, we train and test our algorithm in the Mediator simulation environment designed by one of our expert engineers. This environment simulates various real-world driving scenarios. It considers various cases and events that can be triggered during driving, which can change the fitness level of the drivers and cars. With this simulated environment, we can experiment our HCSG algorithm and evaluate its performance in the context of improving driver safety and comfort.

## 4.1 MEDIATOR ENVIRONMENT

**Environment parameters**
There are more than one hundred configuration parameters in the Mediator environment. These parameters define the initial states and properties of events, driver, car and road, which would eventually influence the state and action space, state distribution, transitions and rewards. Table 4.1details some of these parameters and uncertainties in the environment which are used to change environment state distributions.
The initial state distribution of the environment can be adjusted by some of the parameters in Table4.1. The *road_length* and *timestep* are important parameters as they determine the simulation speed which influences the state distributions. For example, TTDU decreases with time and for agents to learn from experience in critical scenarios where TTDU reaches low, we should set high values for *road_length* or *timestep*.

**Table 4.1:** Configuration parameters for the Mediator environment

| Parameter | Value |
|---|---|
| timestep | Adjustable |
| initial_level_probabilities | Adjustable |
| max_level_probabilities | Adjustable |
| allowed_driver_events | [”DISTRACTION”, ”NDRT”] |
| max_occurrences_of_driver_event | [300, 5] |
| init_distraction_probabilities | Adjustable |
| driver_event_probability | [0.99, 0] |
| distraction_increase_prob | 0.005 |
| distractions_ends_prob | 0.1 |
| ndrt_ends_prob | 0.001 |
| road_length | Adjustable |
| available_actions | [’DN’,’SSL4’, ’ESL4’, ’CD’, ’ES’] |
| suggested_shift_response_probability | 1 |
| suggested_shift_acceptance_probability | 0.5 |
| cd_success_probability | 0.8 |

## 4.2 MARKOV DECISION PROCESS MODELS

In this section, the MDP model for the Mediator environment is introduced. The model is formalized as a tuple list $< S, A, P, R, D >$ as below:

- S = Automation level (L), distraction level (D), time to driver unfit (TTDU), maximum automation level (M) (see table 4.2)
- A = Do nothing (DN), Correct distraction (CD), suggest shift $L4$ $(SSL4)$, enforcement shift $L4$ $(ESL4)$, emergency stop (ES) (see table 4.3)
- P = $P(s'|s, a)$, a set of conditional transitions with probabilities mentioned in table 4.1.
- R = S X A, the reward function
- D = True/False, indicate if the episode is terminated

**Table 4.2:** A description of the state space and the state variable domains

| Variable name | Meaning | Domain |
|---|---|---|
| Automation Level | This state variable stores the current level of automation that the vehicle is in. (N.B. value L1 is left out from the domain because it is considered 'standard' in every car nowadays and thus equivalent to L0) | L0 (no automation), L2 (partial automation), L3 (conditional automation) and L4 (full automation). |
| Distraction Level | This state variable represents the belief state on the driver's level of fatigue. It is encoded as a list of probabilities over the domain. | F0, F1, F2 and F3, where F0 = 'alert' and F3 means 'sleepy'. |
| Max automation level | The maximum available automation level. | L0, L2, L3 and L4. |
| TTDU | Time to driver unfitness. This describes the time until the driver becomes unavailable. | Continuous |
| Driver Response | A variable that saves the driver's latest response to a request. This variable becomes applicable after an action Shift_Request was executed. | -1 (Not initialized), 0 (Pending), 1 (Accept), and 2 (Decline). |

**Table 4.3:** A description of the available actions.

| Action name | Meaning |
|---|---|
| Do Nothing (DN) | Default action. |
| Suggest Shift L4 (SSL4) | Ask the user whether they want to shift to a different automation level L4. |
| Enforce shift L4 (SSL4) | Take over the control of the vehicle through an automatic shift to Level L4. |
| Correct Distraction (CD) | Warn the driver to be more alert when they are showing signs of fatigue. |
| Emergency Stop (ES) | Execute an emergency stop. This action is to be used (only) in critical situations w.r.t. safety. |

Transitions in a MDP model are generally defined as $T(s'|s,a)$ for a transition from state s to $s'$. Transitions with uncertainties are introduced by actions CD and $SSL4$. Action CD succeeds with probabilities since the action might not take effect in reality. $SSL4$ changes automation levels based on the driver response which is also probabilistic since driver preference differs. Fig 4.1 illustrates various transitions from the Mediator environment. For each training step, given action, the environment update states based on these transitions.

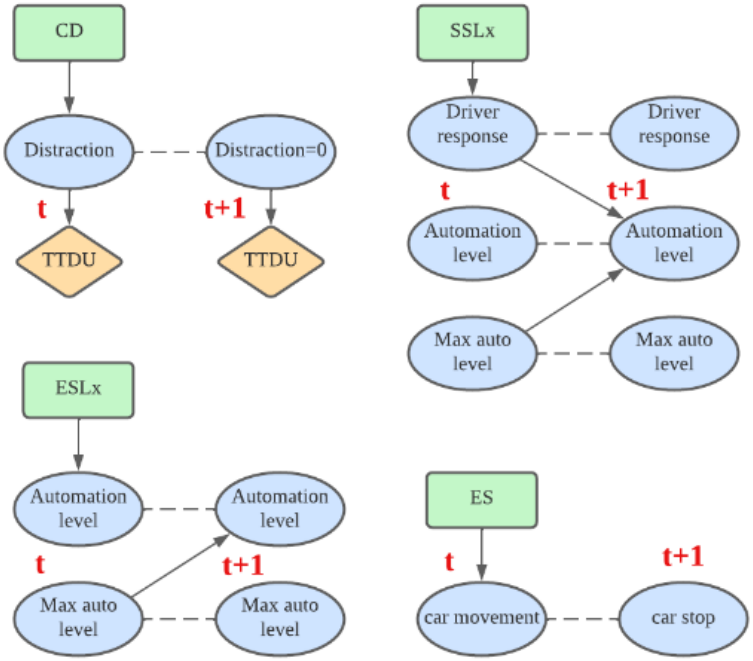



**Figure 4.1:** The transition of the Mediator environment.

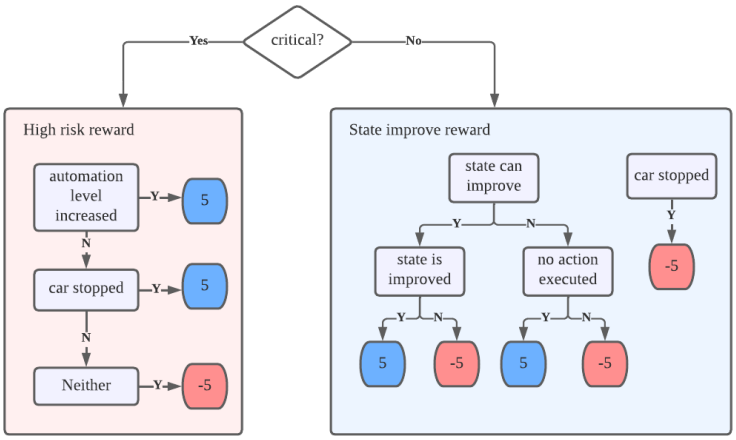



**Figure 4.2:** Reward model of the Mediator environment.

The reward in MDP reflects the rules that are expected for the agent to learn. It basically guides the RL agent learning and can be decisive in the performance of the RL models. As discussed in the section1.2.2, reward design in various complicated environments is normally very challenging and thus we focus on exploration in this paper. The reward design is rather simple, Fig 4.2illustrates the reward model designed for the Mediator environment. We added a gradient to the reward model

to accelerate the training convergence of the RL algorithm. The basic algorithm behind the reward model is that, under safe situations, the MDP model awards the action that can improve the current state. The environment is improved when a driver becomes less tired (higher TTDU) or the automation level is increased at high distraction levels. Car emergency stop is discouraged in this situation. In critical scenarios mostly caused by low TTDU values, emergency actions including $SSL4$ or ES are awarded based on the car and driver states.

## 4.3 DATA IMBALANCE IN REINFORCEMENT LEARNING

Data imbalance is one of the main reasons for low training efficiency issues. Data imbalance happens when the training dataset follows a skewed distribution. To illustrate this phenomenon, we trained an RL agent in the Mediator environment in which the initial states are evenly distributed so that all possible initial states are covered. The experiment setup is included in Appendix.I.1. Fig.4.3 shows the count of occurrences for each of the state combinations in the reply buffer during reinforcement learning for 1000 training episodes. It is obvious that the learning experience follows a 'long-tail' distribution where the training experience in the reply buffer is dominated by some state scenarios, while some other states are much less frequently explored. Besides, plenty of state scenarios are unexplored.

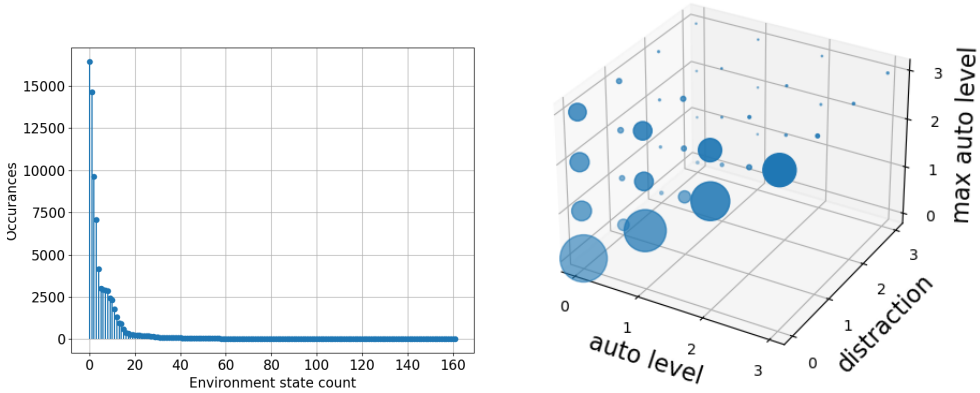



**Figure 4.3:** Left: Count of each environment state for training 100k steps. Right: Distribution of state space for an experiment in Appendix.I.1.

As mentioned in the section1, as the RL agent learns during RL training, it tends to select actions that lead the environment to reach specific optimal states. The exploration-exploitation trade-off together with the transition functions are the main reason for the imbalanced data distribution in Fig .4.3. Initially, during RL training, the RL agent does more explorations which leads to more variety of state space. However, as the RL training proceeds, it relies more on the trained model for action selection which leads to specific states that accumulate in the reply buffer. The result of this is that certain state combinations take increasing proportion in the reply buffer and finally form a long-tail distribution as in Fig .4.3. The policy network is updated periodically by randomly selecting experiences from the updated reply buffer and tends to be trained more on the majority and less on the minority in the reply buffer. The result of this is a biased model performance in which the agent performs well in the majority state combinations while badly in the minority state combinations.

Early stop seems a solution to the aforementioned issue. However, it is hard to determine the time point to stop training depending on the problem's complexity. Also, the training may not converge before it generates a 'long-tail' training experience in the reply buffer.

The imbalanced data distribution also contains a large proportion of unexplored state space which have zero occurrences in the figure. This occurs, for example, when the initial state contains high distraction with its configured TTDU values, before the TTDU reaches too low values in the main exploration phase, the agent tends to guide the environment to decrease distraction levels with high TTDU values, prohibiting the environment from reaching high distraction with low TTDU values.
A biased training dataset could result in a biased model performance. The long-tail problem may lead to the model's poor performance on the tail distribution. Fig.4.4 indicates the model's test performance on the head, tail and unexplored scenarios from Fig.4.3. It is obtained by setting various head-tail split thresholds, and for each threshold, calculate the per cent of scenarios with the maximum reward. The figures indicate that the model's performance on one scenario is positively related to the fraction of the scenario in the experience buffer. And the model achieves better results on the head distribution than the tail and unexplored scenarios.

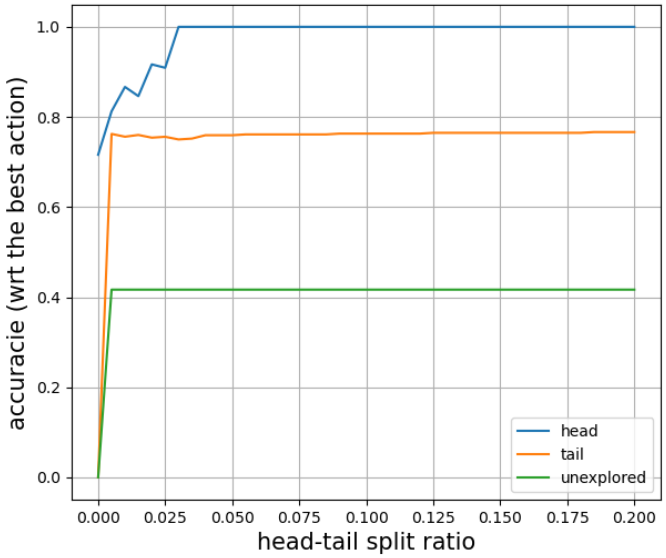



**Figure 4.4:** Model's accuracies on head and tail scenarios for various head-tail split ratios.

# 5 METHODS

This chapter describes the hierarchical clustering state decomposition method and its implementation in reinforcement learning. The partition of state space produces two benefits, firstly, it distributes the training cost of the whole state space so as to accelerate RL learning. Secondly, it improves the agents' overall performance by alleviating the 'long-tail' problem in the learning dataset as it exaggerates the 'tail' data distribution by exploring in smaller state space.

Before proceeding with a description of the algorithm structure, some concepts are explained in table 5.1.

**Table 5.1:** Some concepts of the HCSG algorithm

| **Name** | **Meaning** | **Example** |
|---|---|---|
| State dimension $d$ | The attribute of state equals the number of entities in the state. | state $< L = L0, D = L1, M = L4 >$ is one state with state dimension of 3. |
| state cluster / group | A group of states with specific state dimensions. | State cluster $< L = L0, D = L1 >$ with state dimension of two contains all states whose automation levels equal $L0$ and distraction levels equal $L1$. |
| state cluster/group environments | an environment with a manual configuration that maximizes the state distribution of the state cluster. | State cluster environment $< L = L0, D = L1 >$ generates states with automation level of $L0$ and distraction level of $L1$. |
| Group decomposer $S_d$ | A state entity used to decompose parent state clusters into child state clusters. | State cluster $< L = L0 >$ with group decomposer $S_d = D$ split $< L = L0 >$ into four state clusters (depending on the possible values of group decomposer) $< L = L0, D = L0 >, < L = L0, D = L1 >, < L = L0, D = L2 >, < L = L0, D = L3 >$. |

Fig 5.1 illustrates the overall structure of our proposed HCSD algorithm in reinforcement learning (HCSD-RL). It basically comprises three parts: state decomposition, reinforcement learning and a policy network system.

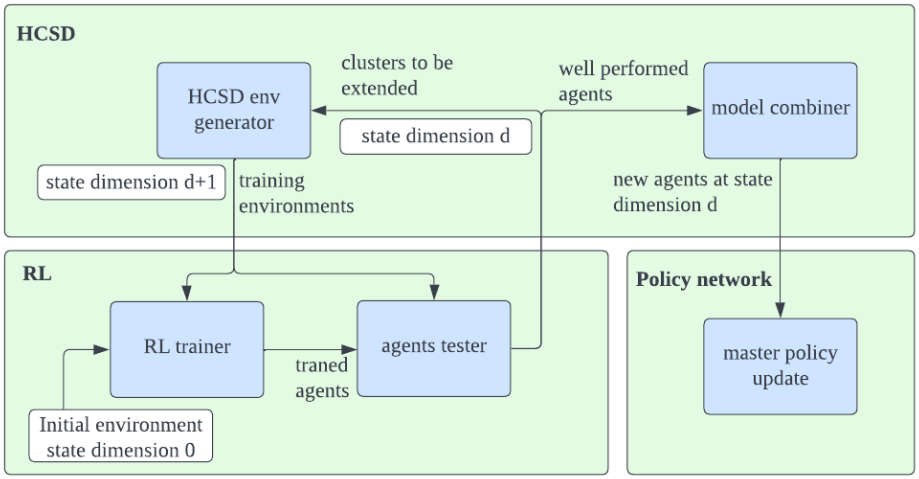



Figure 5.1: HCSD reinforcement learning algorithm diagram

Firstly, we introduce the HCSD method and its implementation in reinforcement learning. Motivations for this method are explained in section 5.1.1. Hierarchical clustering proceeds successively by either merging smaller clusters into larger ones or by splitting larger clusters, to form a tree of clusters. The HCSD method partitions the state cluster environments at lower state dimensions into environments each with smaller state space and sends them to RL to train multiple agents for each of the environments, based on the agents' performance on existing state clusters, the HCSD method determines the state clusters that the agents perform badly and need further to be decomposed in higher state dimension, and combines state clusters in which the trained agents have similar performance. For each state dimension, the well-performed agents and corresponding state clusters are used to update the policy network.

Regarding the selection of the training algorithm, compared with model-free algorithms, the model-based algorithm is more sample efficient and is preferable as they help minimize exploration cost and speed up learning. On-policy methods might suffer from the exploitation-exploration dilemma during implementation, which might result in convergence to local minimal. The off-policy method solves the problem by separating the policy into behaviour policy and target policy, and learns the target policy from the behaviour data generated from the off-optimal behaviour policy that explores probability. Since new data might be generated each iteration, off-policy methods are preferred as the probabilistic exploration ensures a widespread coverage of behaviour data for agents to learn. As a result, double DQN is adopted as our RL training algorithm.

Behaviour cloning is implemented to accelerate learning and improve agent performance. The off-policy methods require experience reply which makes it convenient to collect pre-trained experience for the Q-network. In circumstances where exploration fails to converge to the global optimal solution, behaviour cloning can provide agents with the foundation to learn successfully.

The last part of our algorithm is the policy network. We adopted a hierarchy of policies in which the master policy selects which sub-policy to use based on the input MDP state. Considering the difficulties in training master policy in the hierarchical policy network and the fact that the agent selection can be traced during training of the HCSG-RL, we can set a deterministic master policy that is updated during RL training of the HCSG method. The sub-policies consist of agents trained from the HCSG-RL method. For each successive state dimension, the master policy is updated and new agents are added to the sub-policy network. Each agent in the policy network is a feed-forward network with dense linear layers, each ends with

a RELU function. Feed-forward networks are the most popular network for RL. As for model-based algorithms, the tuning of network parameters might result in extra computation costs. We, therefore, set the neural network as small as possible to minimize the number of parameters so as to accelerate RL learning.
The following section will detail each method, first by state decomposition definitions and the HCSG method, then the RL algorithm and the resultant policy network.

## 5.1 HIERARCHICAL CLUSTERING-BASED STATE GROUPING

The purpose of the HCSG method is to produce the minimum number of agents and state clusters that meet the performance criteria in equation 5.1. As Fig 5.1 illustrates, given inputs on the existing state clusters and agents' performance, the HCSD method has the following tasks:

- state cluster partition for clusters that agents perform badly at
- state cluster combination for clusters that agents perform well and follow the $Q$-irrelevant clustering approximation

In this section, we would explain the motivations behind the method followed by the methods and algorithm.

### 5.1.1 Motivation

We proposed the state grouping algorithm based on the following observations during training in the Mediator environment:

- Training became easier if state space is decomposed.
- Model trained on specific state groups tends to perform well in other similar state groups.
- In different state sub-groups, the optimal agents may have distinct behaviour

#### *Training in subgroups*

Training tends to become easier if state space is reduced. A recent work [36] decomposed state space in reinforcement learning and trained agents for each state space. These works prove using smaller neural nets to train subgroups to distribute computation accelerated learning and dramatically reduced training time. We tested this observation in the Mediator environment. To compare the model's training performance in the whole and decomposed state spaces, we trained agents for the following environments:

- Base environment: with all automation and distraction levels
- L4 environment: with automation level L4 and all distraction levels
- L3 environment: with automation level L3 and all distraction levels

Each agent was trained for 500 episodes. The details of the experimental setup are included in appendix E. Fig 5.2 shows the result of the average episode rewards for training in the three environments, each environment was trained and recorded three times. It is obvious that the agents trained in the decomposed state space

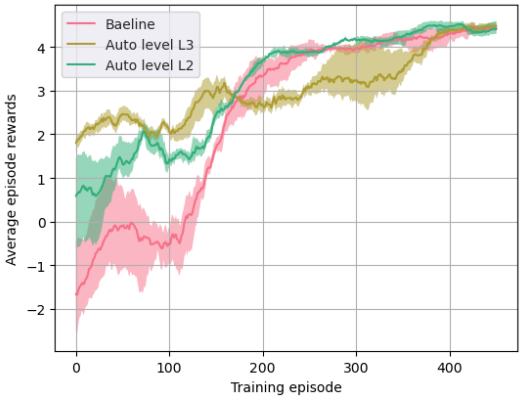



**Figure 5.2:** Training average episode reward for the base, L3 and L2 environments

converge faster than agents trained on the base environment. This indicates splitting the whole state space by automation levels makes training easier than without splitting.

Table 5.2 includes the test rewards for the agents trained in the three environments. To evaluate the training efficiencies, the agents were tested in the same environment where they are trained on. Agents trained in L2 and L3 environments achieve better performance than agents trained in the base environments. State decomposing can also help achieve better performance in a Mediator environment.

**Table 5.2:** Test rewards for the agents trained in base, L3 and L2 environments.

| Seed | Baseline tree | Base env | L3 env | L2 env |
|---|---|---|---|---|
| 1 | 4.88 | 4.76 | 4.91 | 4.89 |
| 2 | 4.88 | 4.74 | 4.90 | 4.92 |
| 3 | 4.88 | 4.72 | 4.87 | 4.89 |

#### *Testing in similar state groups*

Models trained in one state subgroup can perform well in other similar sub-groups. This is obvious in a continuous state where some works tried to discretize the states and group similar states together [9]. Also due to the transition dynamics in the environment, optimal models are expected to have similar behaviour. Fig 5.3 shows the optimal action distribution in the state space in the Mediator environment. As the driver becomes distracted, the optimal agent tends to do action CD when the driver is fit and when TTDU drops too low values, the agent tends to do ESL4 or ES. The trends are expected in most of the scenarios in automation levels L1 and L2. The model that is well trained in one of the small sub-groups can also perform well in other sub-groups in these scenarios.

#### *Model's behaviour in different state groups*

From Fig 5.3, it is obvious that the optimal actions for automation levels L0 and L3 are distinct from that of automation levels L2 and L3. Thus model networks for the two scenarios are expected to have distinct structure and parameter values. Visualization of the neural net in section 6.8.4 validates this observation. It indicates the difficulty of training one model for various scenarios and also proves the discussion in section 1.2.2that splitting training for head and tail distribution accelerates learning and helps learn more generalized models.

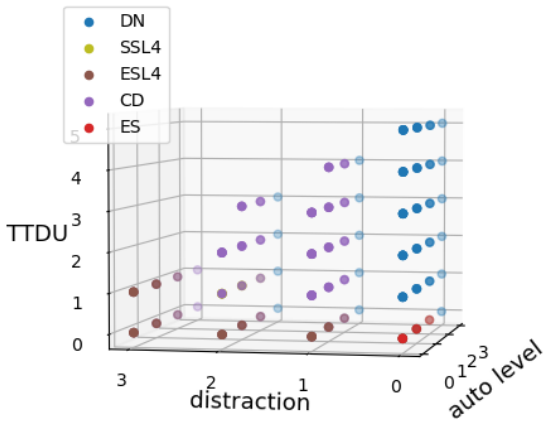



**Figure 5.3:** Optimal actions for various state combinations.

### 5.1.2 Methods

In the HCSD-RL algorithm, state cluster partition and combination take place after agent tester in RL algorithm in Fig 5.1 depending on the agent's performance. State abstraction changes the MDP state space to abstracted state space and trains agents from the abstracted states, which can lead to the loss of important information. Our proposed state clustering does not condense the state space, instead, we partition the ground state space and distribute the RL training, which makes the training easier and faster.

#### *State group partition*

In state partition, we decompose without overlap the parent state cluster $< S >$ given group decomposer $S_d$ at state dimension of $d$, into child sub-spaces $< S_\phi >$ at state dimension of $d+1$, and the combination of the child state clusters form the parent state space. The partition of state cluster at state dimension of $d$ outputs both state clusters and state cluster environment at state dimension of $d+1$. For example, the parent state cluster $< L = L0 >$ with group decomposer $S_d = D$ decompose the state cluster into state clusters $< L = L0, D = L0, M = L0 > ... < L = L0, D = L3, M = L4 >$ and state cluster environments $< L = L0, D = L0 >, < L = L0, D = L1 >, < L = L0, D = L2 >, < L = L0, D = L3 >$.
The partition is based on the RL agent performance on the state cluster. For the state clusters the agents were not trained well, we split them into smaller state groups to accelerate RL learning. To be exact, a state cluster can be partitioned if:

$$\frac{1}{N}\sum_{n=0}^{N}\sum_{t=0}^{T_n} r_d(t) \leq \epsilon_{Eenv} \quad or \quad \frac{1}{E}\sum_{m=0}^{E} r_d(m) \leq \epsilon_{Ecase} \tag{5.1}$$

where N is the number of test episodes, $T_n$ is the number of steps in each episode, E is the number of test cases, $r_d(t)$ is the test reward for each step of each episode and $r_d(m)$ is the reward for each test case, at state dimension d, $\epsilon_{Eenv}$ is the environment reward threshold and $\epsilon_{Ecase}$ is the case test reward threshold
As the equation indicates, the state cluster environments are used to train agents and test the environment rewards and the state clusters are used to test case rewards.c The reason for using case tests (detail in section ) is that agents achieving high rewards in a test environment may not necessarily predict the best actions in various test cases (detail in section 5.2.1) which is important to measure the agent's overall performance.
When choosing $\epsilon_{Eenv}$ and $\epsilon_{Ecase}$, we need to consider the performance-efficiency

trade-off. High values of the thresholds produce well-performed agents but increase the training cost and state cluster decomposition and re-training, resulting in surplus agents. While the low values increase training efficiency with fewer state cluster decomposition, at the expense of reduced model performance.

#### *State group combination*

Section 5.1.1discussed that agents trained on similar clusters could have the same performance on the clusters. The state group combination is used to minimize memory space for the trained agents. It removes duplicate agents that have the same performance but are trained by different clusters and groups the state clusters to form a new cluster.
Our state clustering combination method is similar to $Q^*$-irrelevant state abstraction and is defined as follows:

**Definition 5.1.1** (State approximate clustering)**.** Given MDP M = (S,A,P,R,$\gamma$), two states $s^1, s^2 \in S$ form a cluster under $\epsilon_Q$-approximate $Q$-irrelevant clustering $\phi$ if there exists a Q-value function f: $\phi(S)\ X\ A \rightarrow \Re$, such that $|f_{s^1} - f_{s^2}| \leq \epsilon_Q$

Based on this definition for states, two state clusters can be combined under the two agents trained from the state clusters if they follow the $Q$-irrelevant clustering approximation:

$$\sum_{n=0}^{N} |Q_{s_n^1} - Q_{s_n^2}| \leq \epsilon_{Qsum} \tag{5.2}$$

where N is the total number of states in the two-state groups, $Q_{s_n^1}$ is the Q value by the first agent on state $s_n$ and $Q_{s_n^2}$ by the second agent. $\epsilon_{Qsum}$ is the Q threshold value.
The equation is similar to the equation 3.16as we use Manhattan distance to calculate the similarity between two state groups. The Q values in equation 5.2 are outputs from the agents for each state in the state clusters. Like state decomposition, the parameter $\epsilon_{Qsum}$ also needs to consider the performance-efficiency trade-off for state cluster grouping. Surplus grouping causes degraded agent performance while ultra little grouping may not help minimize the train/test memory space.

#### *Overall algorithm*

Algorithm 5.1explains the HCSG algorithm. The following sections would explain each part of the algorithm.
The selection of the threshold values depends on our target of the algorithm. For high performance we use high values of $\epsilon_{Eenv}$ and $\epsilon_{Ecase}$ and low values of $\epsilon_{Qsum}$. While for improving training efficiencies, we use low values of $\epsilon_{Eenv}$ and $\epsilon_{Ecase}$ and high values of $\epsilon_{Qsum}$. Our method purposed on improving agent training efficiency without losing performance, A good trade-off would be started with focusing on high performance and gradually improving training efficiency.
The maximum state dimension $d_{max}$ is the maximum number of states we consider for state decomposition. For high state space, only the states are chosen such that decompositions of the state dramatically alleviate training difficulties and facilitate agent learning. $d_{max}$ determines the problem complexity and trade-offs that need to be considered when choosing this value. High values result in small state space and accelerate learning, but might cause surplus agent training and combination which is not efficient. Low values make the problem simpler but might add training difficulties. Again, a good start is to focus on model performance first by setting high values for $d_{max}$ and reducing it if problem simplification is required.
Considering the randomness in agent exploration during RL training, we train each agent multiple times for the best performance. The maximum train attempt $max_{att}$ limit the number of training attempt for each agent. To compare the training difficulties for each state cluster, we set the same training episodes $N_{train}$ for RL training

in each state cluster environment. This value should be set high enough to ensure the RL training will reach convergence.
The outputs of the HCSG algorithm are the policy agents and associated state groups at each state dimension. Before we enter the main loop of the algorithm, We started with evaluating the training performance on the base environment using equation 5.1. The base environment is an environment with a state dimension of zero and has equal probabilities for each initialized state. For this evaluation, we generate both state cluster and state cluster environment using methods in section 5.1.2. State clusters that meet the partition criteria are then used for state partition at higher state dimensions.

**Algorithm 5.1:** Hierarchical clustering based state grouping

**Data:** Environment reward threshold $\epsilon_{Eenv}$, case test reward threshold $\epsilon_{Ecase}$, Q value threshold $\epsilon_{Qsum}$, maximum state dimension $d_{max}$, maximum train attempt $max_{att}$, training episodes $N_{train}$
**Result:** Policy network $Q$ and state groups $G_Q$

Initialize master policy and state cluster $s_0$ = all;
Train agent $A_0$ on environment $env_0$ for $N_{train}$ episodes ;
Find all state clusters at state dimension 1 $s_1$ ;
Evaluate performance of agents $A_0$ on $s_1$ using equation 5.1 ;
Save state groups in $s_0^e$ that meet the partition criteria, save $A_0$ in $Q(0)$ and the rest state clusters in $G_Q(0)$;
**while** $d <= d_{max}$ **do**
    Generate state cluster $s_d$ and state cluster environment $env_d$ from $s_{d-1}^e$;
    Train agents $A_d$ on $env_d$ for $N_{train}$ episodes;
    Evaluate the performance of agents $A_d$ using equation 5.1 ;
    Save agents in $s_d^e$ that meet the partition criteria, save the rest agents in $Q(d)$ and state clusters in $G_Q(d)$;
    Initialize *attempt* = 0 ;
    **while** $s_d^e$ *not empty* **do**
        **for** i $= 1$, Length$(s_{ed})$ **do**
            **if** $attempt(i) >= max_{att}$ **then**
                remove $s_d^e(i)$ from $s_d^e$;
                save $s_d^e(i)$ in $s_{d+1}^e$;
            **else**
                generate environment $env_d(i)$ re-train agent $A_d(i)$ on $env_d(i)$ using *Behaviour Cloning DDQN algorithm* for $N_{train}$ episodes;
                **if** $A_d(i)$ *meet equation 5.1* **then**
                    $attempt(i) = attempt(i) + 1$ ;
                **else**
                    remove $s_d^e(i)$ from $s_d^e$;
                    save $A_d(i)$ in $Q(d)$ and $s_d^e(i)$ in $G_Q(d)$ ;
                **end**
            **end**
        **end**
    **end**
    Combine agents in $Q(d)$ and state groups in $G_Q(d)$ using equation 5.2 ;
    remove duplicate models in $Q(d)$ and combine the clusters in $G_Q(d)$;
    $d = d + 1$;
    Update master policy;
**end**

In the main loop, for each state dimension, new agents are trained based on the partitioned state groups and then evaluated. For agents that are not trained well, maximum $max_{att}$ attempts are tried with different seeds and apply behaviour

cloning to facilitate learning. The resultant agents that meet the partition criteria are inputs to the next loop. For well-performed agents, state clusters that meet equation 5.2 are combined and duplicate agents are removed. These agents are then included in the policy network and update the master policy (detail in section 5.3). The process ends until all end clusters are explored or no state clusters need to be extended.

## 5.2 REINFORCEMENT LEARNING

The main purpose of reinforcement learning in the HCSG algorithm (HCSG-RL) is to find the cluster environments that are hard to train and send them to the HCSG method to decompose. The RL method is composed of RL trainers and tester. The RL trainer trains multiple agents taking input from the state cluster environments and the agent tester tests the agents on the state cluster and cluster environments using equation 5.1.

### 5.2.1 Agent tester

The agent tester evaluates the performance of the trained agents and outputs the state clusters with high and low training difficulties to the HCSG method. The state cluster (environment) with high training difficulties is based on the agent's performance evaluated by equation 5.1. And the rest state clusters are classified as with low training difficulties. As equation 5.1indicates, both environment and case rewards are considered for evaluating an agent's performance. The environment rewards are calculated as the average episode reward and case rewards as the average case test reward. The case test helps understand agents' behaviour for each case scenario in the state cluster. As a result, only states in the state cluster are considered for the case tests. The reason for using case tests is that

agents achieving high rewards in a test environment may not necessarily output high rewards in various test cases. The reason is that the agent tends to choose actions that result in states with the highest rewards, avoiding reaching some states in which the agents may not predict the best actions.

### 5.2.2 Behaviour cloning

Behaviour cloning (BC) is one of the simplest forms of imitation learning. Besides training experience, it generates state-only demonstration information and helps increase agent learning speed [29]. Our DDQN as a model-based off-policy algorithm makes it convenient to generate a demonstration experience for agents to learn before RL training. There are mainly two reasons for using behaviour cloning. Firstly, the BC method accelerates training since the agent is initialized with a useful learning experience. Secondly, when it is extremely hard to reach certain states during training and for an agent to learn through state experience, the BC method augments this information that helps the agent to learn better in these cases.
In spite of its advantages, the demonstration experience might limit RL agents' exploration abilities and result in sub-optimal solutions. Thus we only adopt behaviour cloning when normal RL training cannot produce good agents.
The demonstration experience is cloned from the optimal policy, which is based on the reward model. The optimal policy runs as follows: given a state, the optimal action is determined by executing each of the actions under the state and comparing the rewards. The optimal action is one with the maximum reward. The behaviour cloning method finds the optimal action and rewards for each state of

the state cluster. The state-action-reward pair is then formulated as a dataset in the reply buffer. During RL training, the agent selection randomly certain data from the reply buffer to learn and these data in the reply buffer enables the agent to learn more efficiently. Depending on the training difficulties, we may replicate these data to increase demonstration experience in the reply buffer.

### 5.2.3 Exploration in critical scenarios

In the Mediator environment, when TTDU reaches too low values, the agent is expected to enforce a level shift to L4 or emergency stop if the L4 level is not available. However, this experience is hard to learn in reinforcement learning. The reason is that as the agent learns, it tends to avoid reaching low TTDU values which causes a tail distribution for the low TTDU states in the reply buffer. And the agent would perform badly for these states. One solution to this issue is to use behaviour cloning to generate extra experience as in section5.2.2. Another direction is to exploit existing learning experiences for the low TTDU values and try to reduce the percentage of negative experiences which is caused by exploring the wrong actions. Thus we changed the exploration rule in critical scenarios. As shown in the training algorithm 5.2, when TTDU reaches too low values, with some probability we explore actions ES and ESL4 instead of the whole action space.
Compared with state clustering, this method groups the action specific to certain scenarios. It makes agent exploration more efficient and facilitates learning by avoiding selecting actions that not likely to be best under existing states. It can be applied to other environments by condensing the action space under certain state scenarios during exploration.

### 5.2.4 Overall algorithm

Algorithm 5.2 demonstrates the DDQN RL training algorithm. This algorithm is a standard DDQN algorithm except for that methods in section 5.2.2 and 5.2.3 are included. The BC method is executed before the training loop starts to initialize the reply buffer. Exploration in critical scenarios is considered during agent action selection when the agent decides to explore an action.

**Algorithm 5.2:** DDQN Agent with behaviour cloning

**Data:** environment *env*, state cluster environment $G$, total episodes $t_{max}$, cloning duplicates $Clong_n$,

Initialize policies $Q$, $Q_target$ and replay buffer *Replay* ;
**for** each possible state $s_i$ in $G$ **do**
    Find the best action $a_i$ for $s_i$ ;
    reset *env* and execute the action in *env* ;
    store $(s_i, a_i, rew_i, s^{'_i})$ in *Replay* ;
**end**
**for** episode = 1, MaxEpisode¿ **do**
    **while** *episode not end* **do**
        **if** *Epsilon* **then**
            **if** *ttdu*<=*1* **then**
                **Set** $a_t =$
                $\begin{cases} \textbf{randomly choose from } [ES, ESL4] & \text{P} \\ \textbf{randomly choose from } [DN, CD, SSL4, ES, ESL4] & \text{1-p} \end{cases}$
            **else**
                **randomly choose from** $[DN, CD, SSL4, ES, ESL4]$
            **end**
        **else**
            $A_T = max_a Q^*(s_t, a; \theta)$
        **end**
    **end**
    Execute action and store transition $(s_i, a_i, rew_i, s^{'_i})$ to *Replay* ;
    sample random minibatch of transition from experience reply ;
    perform policy gradient ;
**end**

## 5.3 POLICY NETWORK

The whole architecture of the policy structure is shown in Fig 5.4. We basically built a hierarchy of policies consisting of the master policy and sub-policies. The sub-policies consist of agents trained during HCSG-RL training. The master policy determined which of the sub-policies to use given the input state, and sub-policies are responsible for action prediction on the state input.

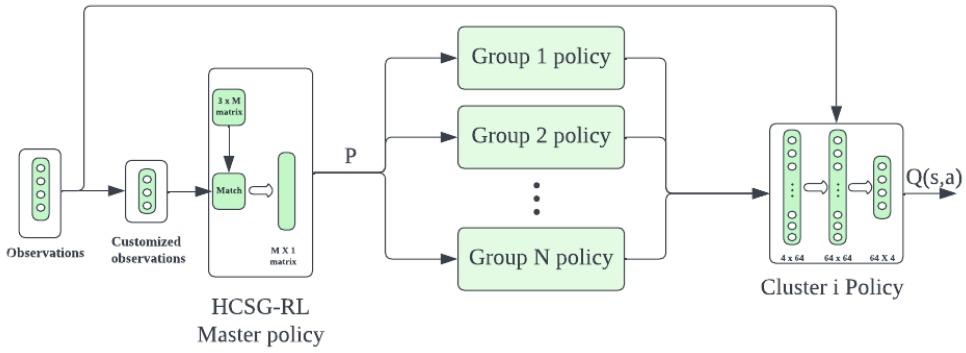



**Figure 5.4:** Policy structure for the HCSG-RL algorithm

### 5.3.1 Master policy

The master policy basically find the state clusters given input to the state. The state cluster number is also the policy number of the sub-policies. The policy structure in Fig 5.4 is similar to the hierarchical neural network. However, the differences are that, we do not train a neural network for master policy. The reason for this is that in HRL, training master policy is equivalent to training one agent for all the scenarios that entails integrating all the learning experience from various environments, which makes the training highly inefficient. Considering that the prediction rules can be traced during HCSG-RL training, we define a deterministic matrix whose parameters can be updated during HCSG-RL training. The deterministic master policy saves a huge amount of training costs for the master policy network. Another advantage of the deterministic master policy is that it cost little calculation memory compared with neural network computation.
Fig 5.5details the composition of the master neural policy. The input to the policy is a reduced observation $< L, D, M >$. The master policy network is composed of a state encoder and a policy matrix. The state encoder generates a $1xM$ stream of bits that has a value of one indicating the position of the current state observation. It consists of a $Mx3$ matrix that records all possible combinations of the states, and a finder which generates the bit stream based on the current input and the matrix. The bit stream is then multiplied with the $Mx1$ policy matrix to output the policy number, with each neuron corresponding to one state observation. The values of the policy matrix are updated at the end of each dimension during training based on the new agents and corresponded state clusters.

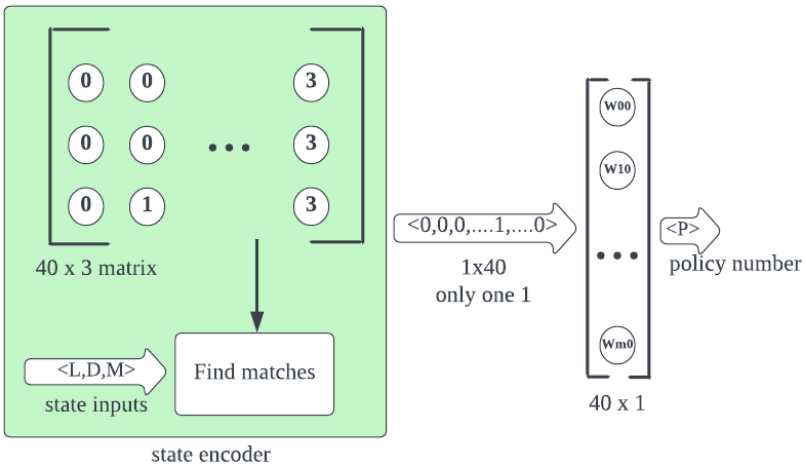



Figure 5.5: HCSG-RL master policy

### 5.3.2 Sub-policy networks

The sub-policies consist of multiple agents, each represented by a small neural network. The size of the neural network plays an important role in the results of the HCSG-RL method. Huge neural networks can achieve better performance and can be used for bigger state group, but they suffer from low training efficiency as it normally requires tremendous computation memory and time [4]. On the other hand, small neural networks accelerate training but can only accommodate small state space, which might require surplus agents for the resultant model. A trade-off needs to be made between the size of agent space and neural networks.

# 6 EXPERIMENTS AND RESULTS

This section describes general experiment settings including experiment tools, training setup and evaluation methods. It also covers the experimental results compared with different algorithm including a mixed environment transfer.
The neural networks were implemented using the open-source package from Pytorch. The source code for the project can be found online on Github. All the experiments were executed on Intel(R) Core(TM) i7-6700HQ CPU devices.

## 6.1 OPENAI GYM

The environment was built and trained using the OpenAI gym. The gym is an open-source toolkit to develop RL by providing a set of standard APIs to communicate between agents and the environments. The standard tasks of the APIs include: defining the environment, resetting the environment and executing the action in the environment and returning the outputs from the environment (mainly new observations and rewards). It also contains standard sets of environments compliant with the APIs. In this paper, we implement the Mediator environment in the OpenAI gym.

## 6.2 TRAINING SETUP

This section introduces the general settings of the HCSG-RL algorithm, including parameter selection and algorithm implementation in the Mediator environment.

### 6.2.1 Training parameters

The hyper-parameters for the DDQN RL algorithm are set based on an experiment on DDQN in [26]. The values are summarised in Table 6.2.1. These hyperparameters are the same for all experiments. The differences are that, firstly, we use a smaller neural net with one hidden layer of 64 neurons for low state and action spaces. As discussed in section 3.1, exploration helps agents to learn new experiences while exploitation accelerates agents' training convergence. Suitable values of $\epsilon$ are from 0.5 to 0.02. We chose rather high values of $\epsilon$ for the reason that, although the training converges faster with reduced exploration, the agent may not gain enough experience of correctly explored actions. The batch length was not set too high as the total training steps were set at low values to train small neural nets on state sub-spaces. And this value helps store enough experience for the agent to learn efficiently from historical information.

| Parameter | Value |
|---|---|
| Activation function | ReLU |
| learning rate | 0.0001 |
| Initial layer weight | Normal |
| mini batch size | 8 |
| batch length | $1 * 10^5$ |
| target update period | 4 |
| $[\epsilon_{max}, \epsilon_{min}]$ | [0.5, 0.02] |
| $\epsilon$ decay | 0.995 |
| $\tau$ | 0.01 |
| discount $\gamma$ | 0.99 |

### 6.2.2 Algorithm implementation

Fig 6.1 illustrates the full hierarchy of cluster environments generated by the HCSG method for the Mediator environment. Each node can be represented as a state cluster environment. For example, a node with 'Dimension = 1' and 'Auto level' of $L0$ represents an environment at a state dimension of 1 and is configured to contain states in which the automation level is constrained to $L0$. The HCSG-RL training starts with training an agent on node 'Dimension = 0' and testing its performance on the next four nodes at 'Dimension = 1'. The nodes with high training difficulties would then be extended to nodes at higher state dimensions.
The order of states for state partition at each dimension influences the structure of the tree in Fig 6.1. We focus on producing the least number of leaf nodes to minimize the training cost which results in the order of Automation level - Distraction - Max automation level. Agent combiner in the HCSG-RL algorithm groups nodes together and simplify the tree.

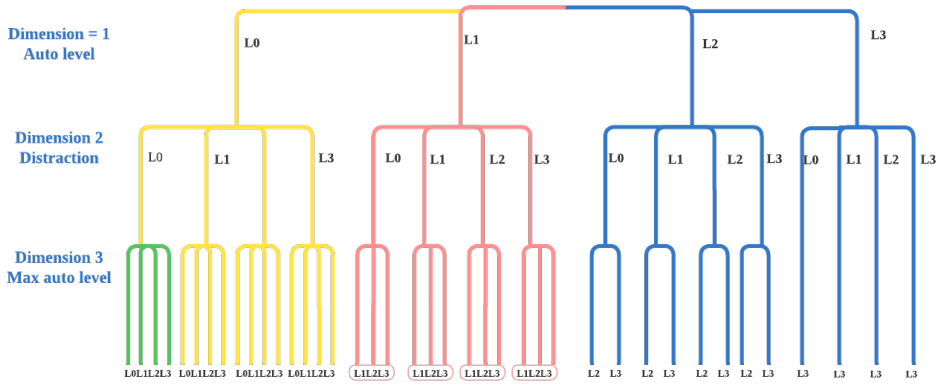



**Figure 6.1:** Hierarchical clustering based state grouping

### 6.2.3 Algorithm parameter selection

This section discusses the selection of values for parameters in the HCSG-RL algorithm5.1. The size of the neural network for each RL agent should be high enough to ensure it at the minimum can learn the experience from the leaf node environments in Fig 6.1. Experiments show that neural networks with one hidden layer of 60 neurons can meet this requirement.
Regarding the algorithm values, using the structure in Fig 6.1, the maximum state dimension is set to three. As section 5.1.2explains, we choose the environment and case test reward thresholds both as high as 4.9 (max reward equals 5) and a Q value threshold as low as 0.01 to maximize the performance of the trained agents.

## 6.3 PERFORMANCE ENCODING

Training in a single environment is hard to produce a well-performed model as most of the states may have not been explored. Knowing that beforehand, we define various MDPs/environments, each with different initial states. For evaluation of the experiment, we generate various state clusters and state cluster environments from Fig 6.1, and encode the environments as in appendix Cand the state clustersD. The model's rewards in the environment give a high-level indication of the model's performance, and case tests were designed to understand the model's behaviour at the micro-level. This case test is different from the case test in the HCSG-RL algorithm. In the experiments, we generate all possible cases and evaluate them by the resultant policy network. With the results of the case test, we are able to understand in which conditions the model outperforms models from other algorithms.

## 6.4 COMPARISON METHOD

We compare the resultant agents trained using the HCSG-RL algorithm with a baseline tree decision algorithm and a transfer learning method (detail in section 6.4.2). The purpose is to evaluate how agents perform in various environment scenarios/-cases. To achieve this, we adopted environment encoding and case encoding mentioned in section6.3. We used both environment tests and case tests to evaluate the model's overall performance. The end purpose of training the generalized model is to minimize driver unfit situations so as to maximize safety. Evaluations on drivers unfit were conducted at the end.

### 6.4.1 Tree decision algorithm

The decision tree algorithm is manually designed that aims at maximizing the trade-off between safety and comfort. Fig B.1 in appendix B illustrates the tree algorithm. The tree decision makes decisions based on the distraction level and TTDU values on different automation levels. For no distraction, it does not take any action. When distraction level increase, it chooses to correct distractions when TTDU reaches low or suggests shift automation levels when higher auto levels are available. In emergency situations where at high distraction levels the drivers refuse to shift automation level, emergency stop or enforce shift auto level is executed.

### 6.4.2 DDQN Agent with mixed environment transfer

In the case of agents RL learning among multiple environments, transfer learning is one of the most popular methods. In this paper, we compare the performance of the HCSG-RL algorithm with mixed environment transfer (MET). The MET algorithm is detailed in Algorithm 6.1 and works as follows: Initially, we generate all the environment configurations and assign selection probabilities to each environment. For every 100 episodes, the environment probabilities are updated based on the performance of the agent in these environments which is evaluated by case tests.

**Algorithm 6.1:** DDQN Agent with mixed environment transfer

**Data:** environment *env*, total episodes $t_{max}$, cloning times $Clong_n$,
Initialize policies $Q$, $Q_target$ and replay buffer *Replay* ;
Initialize environments *envs* ;
**for** episode = 1, MaxEpisode¿ **do**
  **if** *episode* % *100* == *0* **then**
    case test the model on *envs* ;
    Update probabilities for selecting each environment ;
    Update environments ;
  **else**
  **end**
  episode not end **if** *Epsilon* **then**
    **if** *ttdu*<=*1* **then**
      **Set** $a_t =$
      $\begin{cases} \textbf{randomly choose from } [ES, ESL4] & \text{with probability of P} \\ \textbf{randomly choose from } [DN, CD, SSL4, ES, ESL4] & \text{1-p} \end{cases}$
    **else**
      **randomly choose from** $[DN, CD, SSL4, ES, ESL4]$
    **end**
  **else**
    $A_T = max_a Q^*(s_t, a; \theta)$
  **end**
  Execute action and store transition $(s_i, a_i, rew_i, s^{i)})$ to *Replay* ;
  sample random minibatch of transition from experience reply ;
  perform policy gradient ;
**end**

## 6.5 EVALUATION METRICS

We define the following as the performance metrics:

1. Average episode reward and episode loss during the training procedure

$$R = \frac{1}{E} \sum_{e=0}^{E} \sum_{t=0}^{T_e} r(t) \tag{6.1}$$

   where E is the number of episodes and $T_e$ is the length of episode e

2. Average episode reward for test in one environment, using the same equation as 6.6.

3. Cumulative average episode reward for tests (CAER) in multiple environments

$$R = \frac{1}{NE} \sum_{n=0}^{N} \sum_{e=0}^{E_n} \sum_{t=0}^{T_e^n} r(t) \tag{6.2}$$

   where N is the total number of environments $E_n$ is the number of test episodes in environment n and $T_e^n$ is the length of episode e in environment n.

4. Cumulative case test rewards (CCTR)

$$R_c = \sum_{n=0}^{N} c(n) \quad wherec(n) = \begin{cases} 1, & \text{if } r(n) == r_{opt}. \\ 0, & \text{otherwise.} \end{cases} \tag{6.3}$$

   where N is the total number of test cases.

5. Normalized cumulative case rewards (NCCR)

$$R_c = \sum_{n=0}^{N} c(n)/c_{max} \tag{6.4}$$

where N is the total number of test cases.

6. Driver unfit count and duration

$$C_{unfit} = \sum_{n=0}^{N} c(n) \quad wherec(n) = \begin{cases} 1, & \text{if } \exists t \in T_n \;\; TTDU_t == 0 \; and \; autoLevel_t < L3 \; . \\ 0, & \text{otherwise.} \end{cases} \tag{6.5}$$

$$D_{unfit} = \frac{1}{N} \sum_{n=0}^{N} \sum_{t=0}^{T_n} c(t) \quad wherec(n) = \begin{cases} 1, & \text{if } TTDU_t == 0 \; and \; autoLevel_t < L3 \; . \\ 0, & \text{otherwise.} \end{cases} \tag{6.6}$$

where N is the total number of test episodes and $T_n$ is the length of episode n.

The training average episode reward and loss give insight into how well agents learn during RL training. An effective RL training should result in enough updates on policy networks which is related to high loss values. As the episode length may vary in each episode, we adopted an average episode reward to represent the agent's performances more accurately. The training reward and loss can represent the performance of the RL methods in RL training. For the test environment, the average episode reward reflects the model's performance in a specific test environment, while the CAER in multiple environments reflects the model's generalization ability and robustness in changing environments. Compare with CAER which reflects the high-level long-term performance of the agents, cumulative case test reward indicates the detailed behaviour and directly reflects agents' overall performance.

## 6.6 OVERVIEW OF EXPERIMENT MODELS

The following models were generated for evaluation of the algorithms:

- HCSG D1: 4 sub-policies (1 base and 3 clusters) and one master policy
- HCSG D2: 5 sub-policies (1 base and 4 clusters) and one master policy
- HCSG D3: 7 sub-policies (1 base and 6 clusters) and one master policy
- MET: one policy trained on a mixture of 40 MDPs
- Baseline: rule-based decision tree algorithm

## 6.7 RESULTS AND DISCUSSION

This chapter evaluates the HCSG-RL abilities to improve agents' training efficiencies. With limited training resources (training episode, neural network size, etc), the evaluation results can basically reflect the agent training efficiencies. The evaluation basically shows that the HCSH-RL algorithm improves both the agent's training efficiencies and overall performance. Firstly, in one environment, the agents' performance on the tail training dataset is improved without losing experience from head distribution. Next, the agent does not or hardly decrease performance as the test environments are changing, so as to prove the model's overall performance in various scenarios. The results from baseline and transfer learning methods are compared to prove our algorithm's advantage.

Fig 6.2 shows the resultant hierarchy of the state clusters, each colour represents a well-trained agent. For the Mediator environment, we trained out 7 sub-policies in total, with 3 agents for states in dimension 1, one agent in state dimension 2 and two extra agents in state dimension 3.

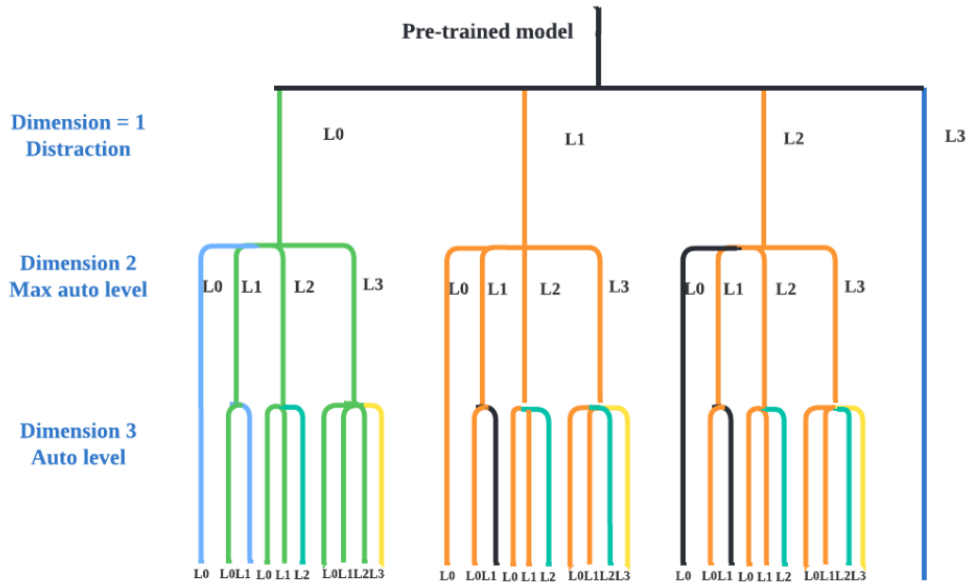



**Figure 6.2:** State clusters are covered by the resultant sub-policies.

We start by comparing the resultant model's performance in the base environment (environment in appendix I.1) with the baseline and MET algorithm by looking into tested average episode reward and reward in head and tail. Then we extend to multiple environments. Next, we use a case test to compare the model's exact behaviour and features in a multi-dimension state space, which directly reflects the model's generalization ability. After the environment and case test, we evaluate how a generalised model gives benefits by looking into the model's effectiveness in reducing drivers' unfit situations. And we investigate some ablation statistics to explain the reasons for this effect. At last, we will do some ablation study in which we investigate the effect of behaviour cloning and its necessity in accelerating agent learning and improving performance. And we then do some visualizations of neural net structures for the sub-policy agents to investigate the agent differences.

### 6.7.1 Performance in environments

#### *Rewards in the base environment*

In the section4.3, we trained agents in the base environment with settings in appendix I.1. And illustrated the detected long-tail problem and its consequence on the head-tail performance. This sectionshows the performance improvement of the trained model using the HCSG-RL algorithm in the base environment. Fig 6.3 shows the head-tail performance of the HCSG-RL model for various head-tail split criteria ratios. Compared with the result in Fig 4.4, the model trained from our algorithm gains significant improvement on the tail performance without sacrificing performance on the head class distribution.

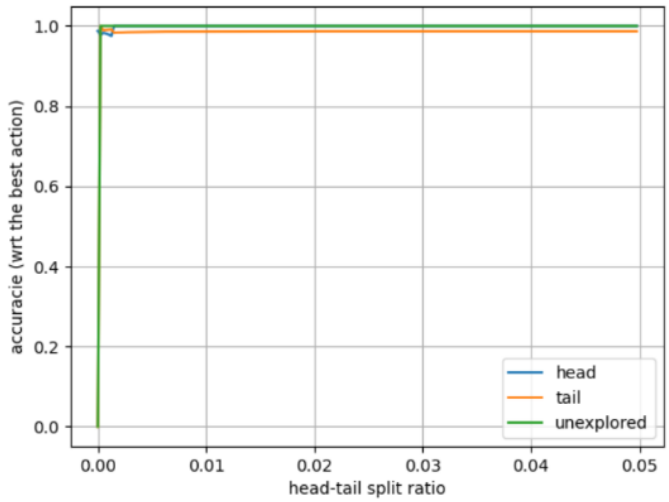



**Figure 6.3:** Model's accuracies on the head and tail scenarios for various head-tail split ratios from HCSG-RL algorithm with $\theta$ of 100%. Accura- cities calculated based on the per cent of actions with the maximum reward.

Section 4.1 discussed the influence of the environment parameter *initial TTDUs* and *road length* on a variety of scenarios. To test the model's overall performance, we define a different variety of scenarios by changing road length and configuring TTDU values. In the following experiments, we set initial TTDU values fixed and vary road lengths. Experiment details are included in appendix F. Fig 6.4 (left) includes the results of the HCSG-RL model trained on various state dimensions. The high threshold values in the HCSG algorithm produce agents with high performance and thus HCSG D3 achieve the best episode reward in various scenarios. HCSG D1 and D2 models are not generalized enough as their average episode reward decreases with increasing road length. This is because HCSG D1 and D2 models may still have state clusters not well-trained, and for high road length, more critical scenarios occur and HCSG D1 and D2 models result in sub-optimal solutions to these scenarios. The HCSG D3 model trained on smaller state clusters achieves dramatic performance improvement and is able to generalize to various scenarios since reduced state clusters add balance to the training data. And the HCSG D3 model achieves the highest episode reward regardless of the road length.
Next, we compare the average episode rewards of the HCSG D3 model with the baseline and the MET model. To ensure the MET model has gained enough experience, it was trained in a total of 1000k steps. Each agent from HCSG was trained 1k steps in D1, 500 steps in D2 and 300 steps in D3. Fig 6.4 (right) shows the result, compared with baseline, it seems that transfer learning in multiple environments does improve the agent performance in various scenarios, but is still hard to be as generalized as HCSG models that train in sub-groups.

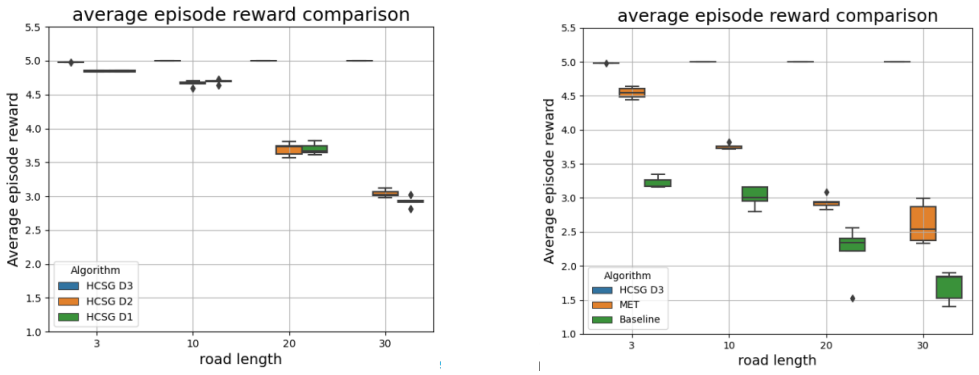



**Figure 6.4:** Left: Average test episode reward on the base environment for HCSG-RL models with dimensions of 1, 2 and 3. Right: Average test episode reward on-base environment on HCSG-RL model, MET model and baseline.

### *Performance in multiple environments*

The goal of the experiments in this section is to compare HCSG's ability in improving training efficiencies by evaluating agents' overall performance with limited training resources. CAER is a useful benchmark metric for this task. Considering the uncertainties in the Mediator environment, the experiments were repeated at least three times with different seeds for each algorithm. Each environment has different configuration settings resulting in various initial states, transitions and episode properties.
Fig 6.5shows the CAER results of various state dimensions of the HCSG algorithm compared with baseline and MET algorithms. The performance of the baseline algorithm drops as new environments are tested, showing that the baseline is not perfect in various scenarios. MET agent that was trained by a mixture of environments has relatively stable performance as new environments come in. HCSG D1 HCSG D2 models are only well-trained for parts of the state clusters, and are less generalized than the MET model. HCSG D3 model has well-trained agents for all the stat clusters and produces the best-generalized model and its CAER stabilizes at the peak related to the number of environments added in.

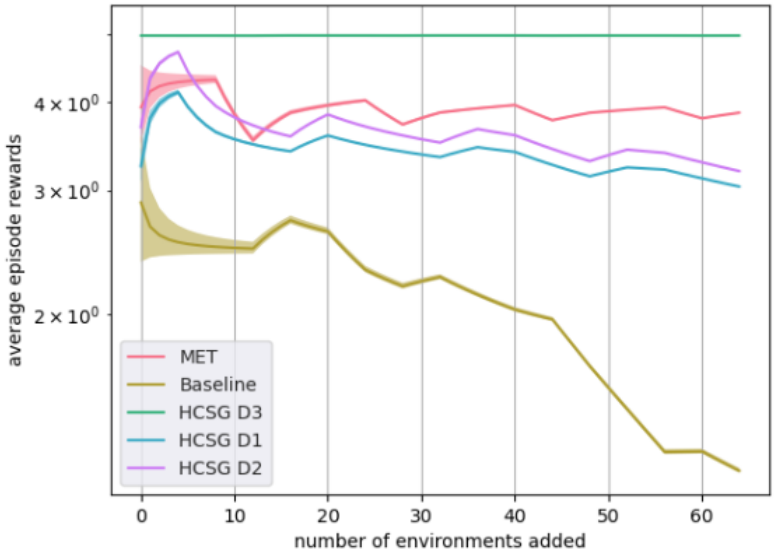



**Figure 6.5:** Cumulative average episode reward with an increasing number of environments involved.

### 6.7.2 Performance in observation cases

Besides environment tests, a case test is another way to evaluate a model's overall performance from a micro level. Fig **??** compares the normalized case rewards of the HCSG, MET and baseline algorithms, the rest of the similar results for other models are stored in appendix

Table 6.7.2 lists the calculated CCTR results of models on the four automation levels. It summarizes the information from Fig **??** and basically explains the difference in models' performance in environmental tests. Overall, HCSG $D_3$ achieves the highest NCCR values and thus has the best performance. The baseline algorithm is not generalized in most of the cases with very low CCTR values. Low CCTR and high NCCR in MET indicate that the MET algorithm results in many sub-optimal solutions. The results from HCSG indicate that optimal agents are hard to train at high automation levels. This conforms to the intuition that, at low auto levels critical scenarios are more likely to happen and thus agents were well trained in those scenarios. Whereas at high auto levels, agents easily learn action *correct distraction*s which avoid critical scenarios to happen.

| | CCTR in $L_0$ | CCTR in $L_1$ | CCTR in $L_2$ | CCTR in $L_3$ | NCCR |
|---|---|---|---|---|---|
| Max CCTR | 16 | 12 | 8 | 4 | 40 |
| HCSG $D_1$ | 14 | 9 | 3 | 1 | 35.3 |
| HCSG $D_2$ | 16 | 9 | 3 | 1 | 35.8 |
| HCSG $D_3$ | 16 | 12 | 6 | 4 | 39.55 |
| Baseline | 1 | 4 | 5 | 0 | 20.9 |
| MET | 5 | 5 | 3 | 1 | 30.8 |

### 6.7.3 Driver fitness

The section evaluates the algorithm for improving real-world scenarios. We use driver unfit to test models' abilities in optimizing driver safety. Some ablation studies were conducted on statistics to understand models' behaviours in critical scenarios.

For the evaluation of models' performance on driver unfit. We tested each model for 100 episodes on the base environment. Details on the experiments are stored in appendixH. Well-trained RL agents would choose the optimal actions and try to avoid situations that make drivers unfit. Fig 6.6illustrates the counts and duration of driver unfit scenarios for the experiments on HCSG $D_3$, MET and baseline algorithms.

These results reflect the models' performance in critical scenarios where TTDU reaches low and drivers become unfit. It is obvious from the figure that the MET algorithm outperforms the baseline algorithm in driver safety with a lower average driver unfit count and duration. The HCSG model trained on smaller state clusters could put care more on the driver unfit situations and could avoid driver unfit situations to occur, thus it performs the best of the three algorithms in driver unfit evaluation.

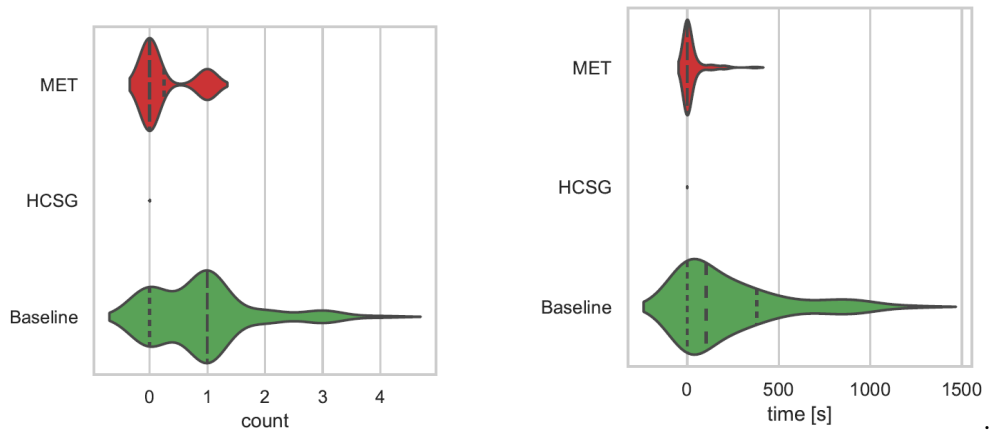



**Figure 6.6:** Driver unfit count(left) and duration(right).

Next, we look into the two episodes case by case to understand how the HCSG model avoids driver unfit and outperforms the baseline algorithm. Tree decision algorithms only makes decisions only based on the circumstances humans can think of, while RL agents trained on generalized state scenarios could outperform tree decisions. The first test episode illustrates this benefit. Fig 6.7 and 6.8 shows the states of the first episode. Near the end of this episode, the following scenario happened:

- Distraction level at Lo and TTDU reaches zero
- Auto level equal to the maximum auto-level and is not fully automated

Over a long road, the driver may become tired (TTDU decrease to zero) even without distraction. For the above-mentioned scenario, action CD, ESL4 and SLL4 are not possible under the distraction of Lo and the highest automation level. The partitioned state cluster enables the HCSG method to gain some experience in this scenario and can handle it by doing emergency stops, so as to avoid occurrences of driver unfit situations and maximize average episode reward, at the expense of reduced episode length. Whereas the tree decision algorithm did not take this scenario into account.

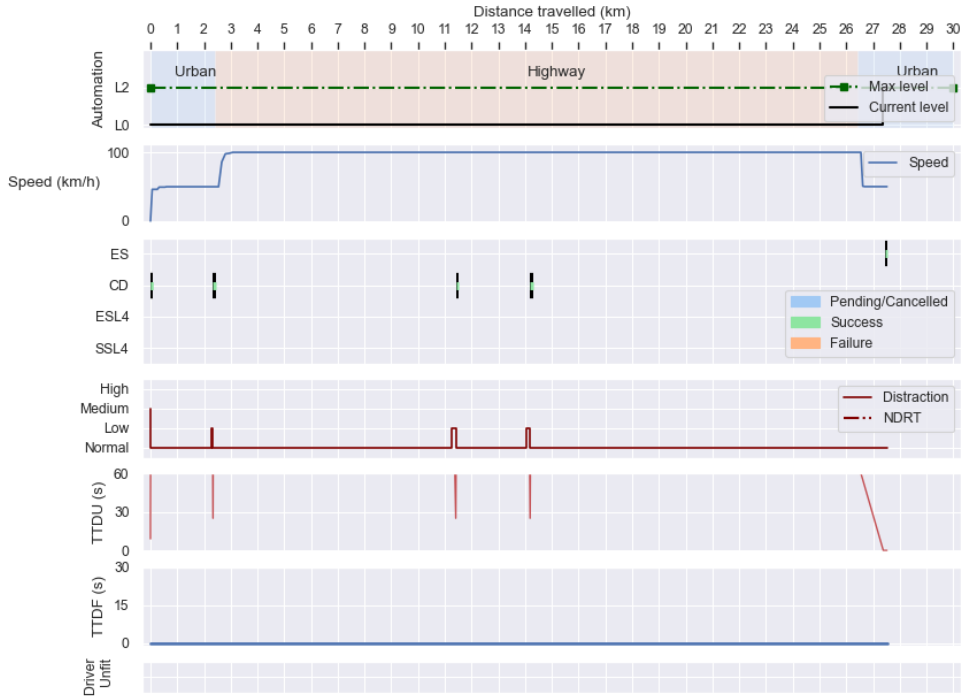



**Figure 6.7:** Rendering of one episode for training HCSG D3 model on the base environment.

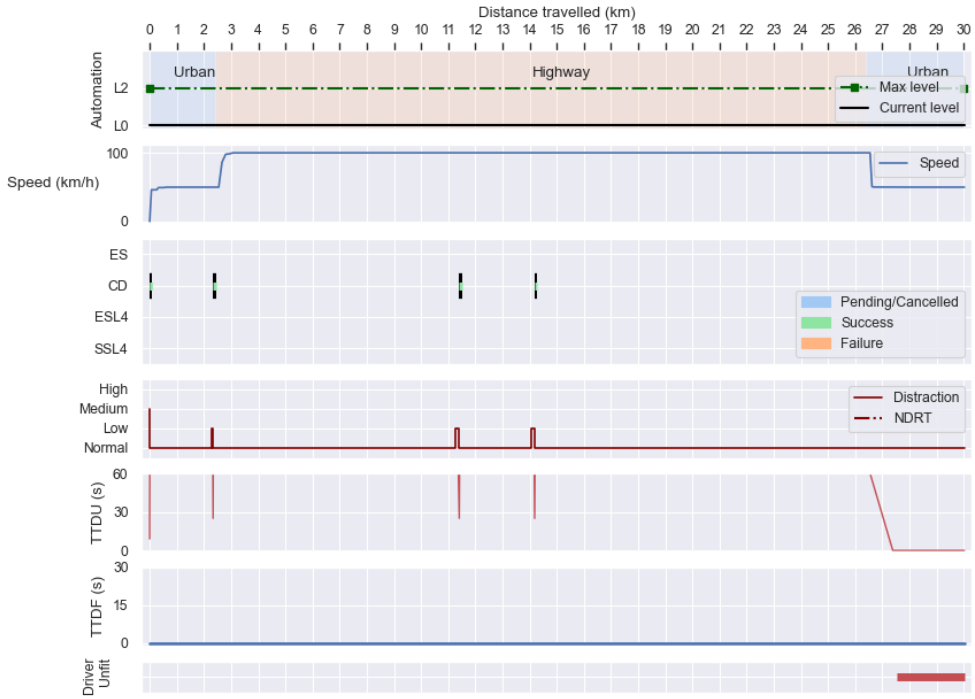



**Figure 6.8:** Rendering of one episode for baseline algorithm on-base environment.

Besides its advantage in generalization, RL agents also outperform the decision trees in maximizing the long-term reward. The decision tree algorithm takes actions based on the existing state, which might disregard some future circumstances that the tree algorithm may not consider, resulting in sub-optimal performance. While the DDQN RL agent is trained by trial and error on predictions and can achieve

better results in terms of long-term rewards. The following test episode illustrates this benefit. In this episode, the following critical scenario happens:

- Distraction level at Lo and TTDU reaches zero
- Auto level smaller than the maximum auto-level L3

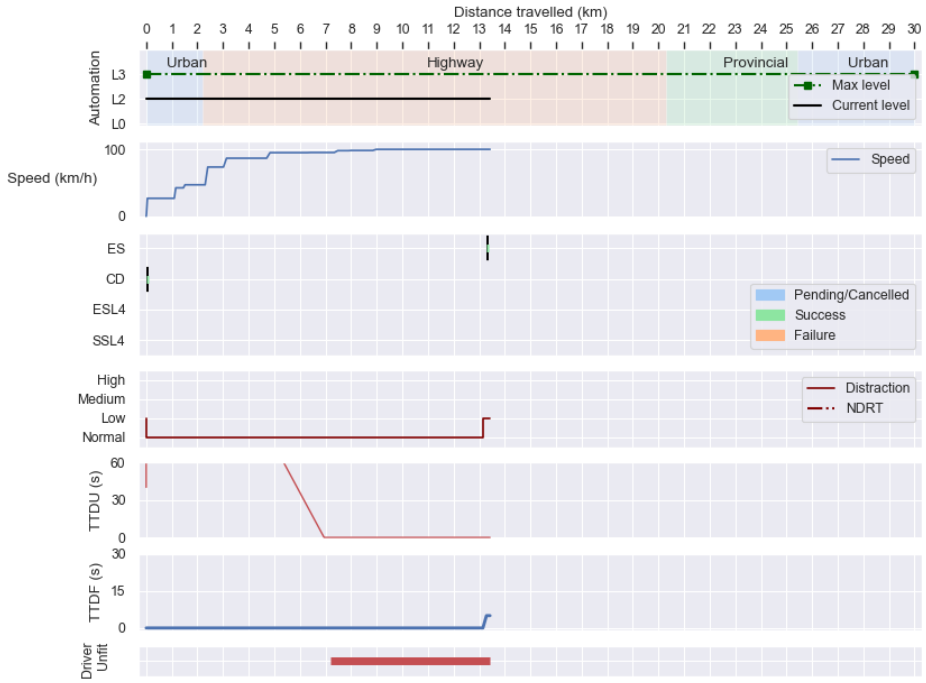



**Figure 6.9:** State clusters are covered by the resultant sub-policies.

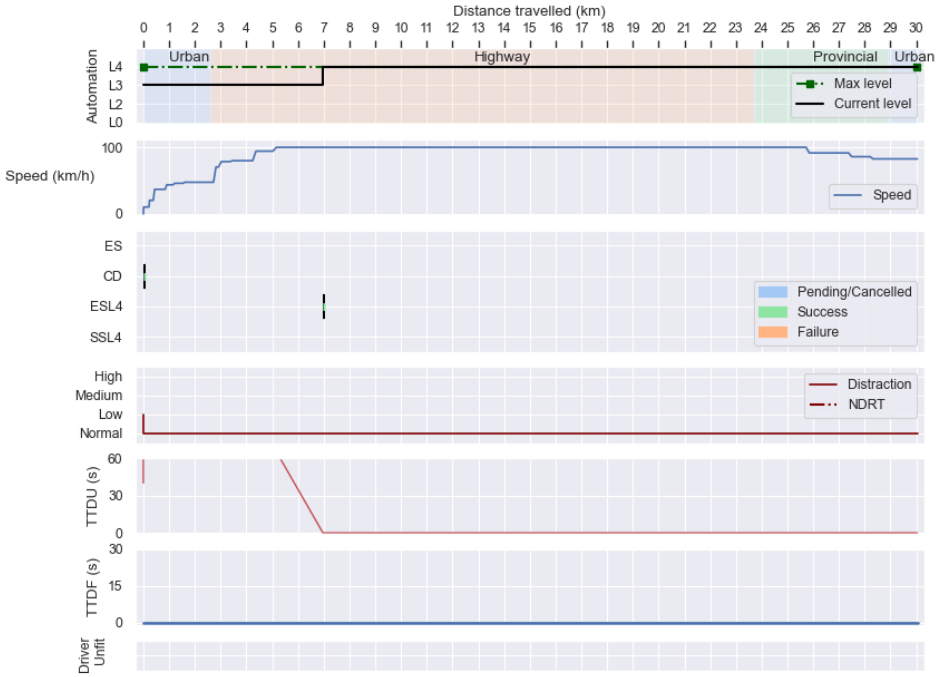



**Figure 6.10:** State clusters are covered by the resultant sub-policies.

Fig 6.9 and 6.10 includes the states of this scenario. For this scenario, a higher automation level is available, the HCSG D3 agent enforces auto-level to the highest level to avoid any delay or uncertainties from driver response, so as to avoid this critical scenario to continue. While the baseline algorithm makes decisions on the existing distraction level, with no distraction, the tree decision considers a safe situation and chooses to do nothing, which causes the driver to become unfit and eventually stopped the car, the HCSG D3 agent let the automation system take full control so that the car can drive to the end and the long-term reward of the environment is maximized.

## 6.8 ABLATION STUDY

In this section, we compare the algorithms by looking into action and state statistics. We do an ablation study by reviewing techniques and analysing their necessities for improvements in the results.

### 6.8.1 Ablation statistics

An ablation study was done on statistics from the base experiment with details in appendix H. Table 6.11 shows the results. The statistics were obtained by testing the baseline algorithm and models from MET and HCSG for 100 episodes. As for the HCSG D3 model, the average time between actions and the average number of actions are the lowest and thus the HCSG agents take action the most frequently. Besides, the HCSG model takes the most emergency stops. This explains why the HCSG model outperforms the other two algorithms in critical scenarios. Although the MET agent was trained in a variety of environments. It could lose information learned during knowledge transfer among environments. These statistics prove the benefit of distributing training over transfer learning in RL learning in multiple environments.

| | Num of ES | Num of action per 100km | Average time between actions | Percent of time driving at | | | |
|---|---|---|---|---|---|---|---|
| | | | | L0 | L1 | L2 | L3 |
| MET | 0 | 4 | 103 | 27.1 | 24.8 | 19 | 29.1 |
| HCSG D3 | **76** | **3** | **89** | 14.3 | 17.7 | 11.4 | **56.6** |
| Baseline | 54 | 2 | 102 | 15.4 | 11.4 | 44.7 | 28.5 |

**Figure 6.11:** Ablation statistics

### 6.8.2 Behaviour cloning

In this paper, we implemented behaviour cloning in RL training to accelerate agent training efficiencies. We would do experiments in a random environment to evaluate this advantage. Details of the experiments are stored in appendix I. We trained two agents each for 350 episodes for the experiment, one with behaviour cloning and initialize the replay buffer with 1000 times the formalized experience, and another agent without behaviour cloning.
Fig 6.12shows the training reward and losses of the two agents. Looking into the

training rewards, with some initial expert experience, the agent converges within 350 training episodes. While training without behaviour cloning does not converge. Loss values give a rough indication of agent learning. As equation 3.13 shows, high loss values mean huge policy updates. When the loss value stabilizes, the agent converges and stops to learn. Loss values of the agent without behaviour cloning increase at episode 60 indicating that the agent just starts to learn at this episode, and continues learning at the end of the episode. As for the agent trained with behaviour cloning, with some initial experience, the agent learns fast at the beginning with big gradient updates and stabilizes after around 200 episodes.

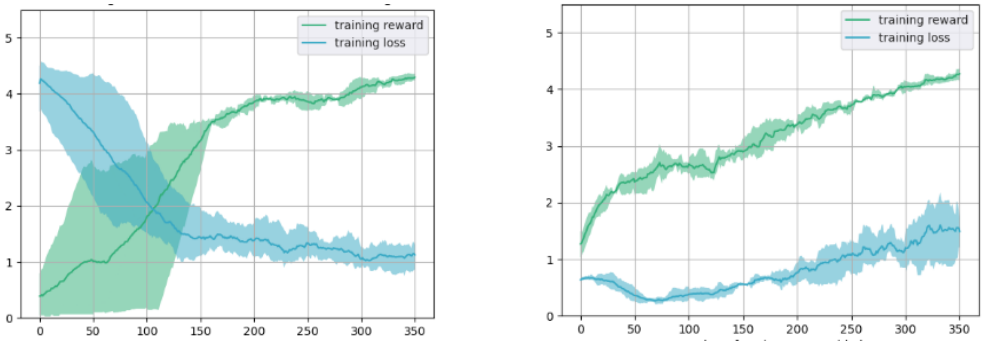



**Figure 6.12:** Training results with(left) and without(right) behaviour cloning.

The ratio of expert and training experience is worth exploring. With a high proportion of expert experience, the trained agent might be overfitting to the expert data resulting in reduced generalization ability to similar state clusters. And for a low proportion, the agent may not gain enough experience and become underfit. We set up several experiments, each trains the agent for 500 episodes but is initialized with various lengths of expert data in the reply buffer. Fig 6.13 shows the training rewards and losses for these experiments. It seems that policy gradient update increases with the proportion of the expert experience, the agents do not differ much from training rewards. These findings prove that the agents learn faster with a higher proportion of expert learning experience.

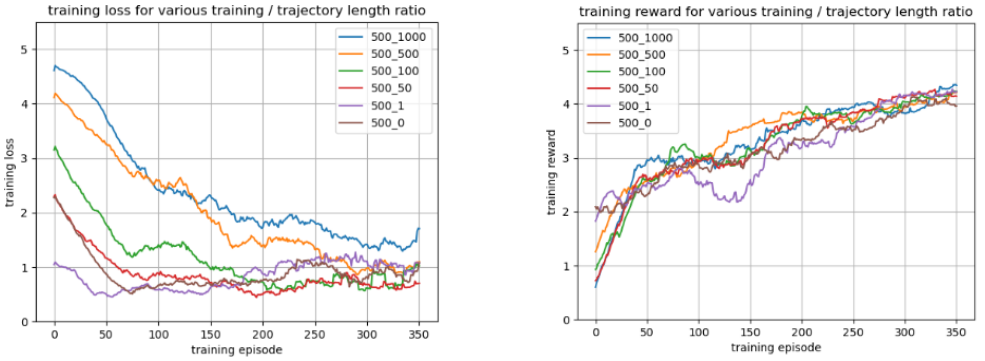



**Figure 6.13:** train loss(left) and rewards(right) for agents with various train-trajectory ratios($500_0 means no behaviour cloning$).

### 6.8.3 Exploration in critical scenarios

In algorithm 5.1, during exploration, we condense the action space in critical scenarios to accelerate learning in these scenarios. This section would prove the benefit of this method. We test by comparing the agent performance in critical scenarios trained with this method and the agent trained by the same exploration in all scenarios.

To learn how much the method accelerates learning, we train both agents for 100 episodes with setup details in appendixI. This experiment setup ensures enough occurrences of low TTDU values even at low distraction. We limit the training episodes to 100 to compare agents' performance with limited exploration steps to investigate the training efficiency of the two exploration methods. The agents are tested on the same setup for 100 episodes with various road lengths. Higher road length results in lower TTDU values and thus generates more critical scenarios.
Table 6.14shows the average episode rewards for the two agents. With higher road length, more critical scenarios occur and the agent trained with exploration in all action space does not learn well with limited training episodes and results in lower test rewards. While the agent trained with condensed action space exploration in critical scenario learns critical scenarios fast within 100 training episodes as the episode reward slightly increase as road length increases. These results prove condensing action space during the exploration can dramatically speed up learning.

| | Road length | | | | |
|---|---|---|---|---|---|
| | 10 | 20 | 30 | 40 | 50 |
| Varied exploration | 4.82 | 4.90 | 4,93 | 4.95 | 4.96 |
| Same exploration | 3.91 | 0.2 | -1.33 | -2.19 | -2.72 |

**Figure 6.14:** Average episode rewards

### 6.8.4 Visualization of the neural network

Visualization of neural nets gives unique insights into neural nets. Fig 6.15shows the plotted weights of the layers for each of the 7 sub-policies. The distinct weight distribution of the neurons indicates the difficulty in combining the agents and the difficulty in using the existing neural net structures to learn various scenarios. It is also interesting to note that some policies have similar weight distribution, this gives hint for neural net ensembling or the use of a bigger neural network to replace these policies networks.

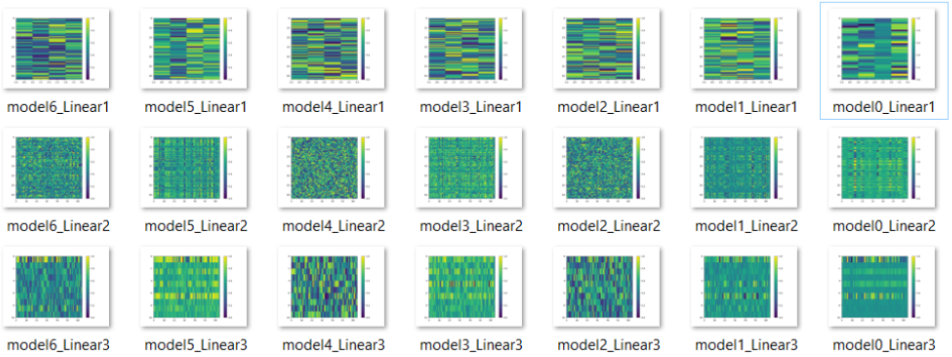



**Figure 6.15:** Neural net visualization of HCSG sub-policies.

# 7 DISCUSSION AND CONCLUSION

## 7.1 CONCLUSION

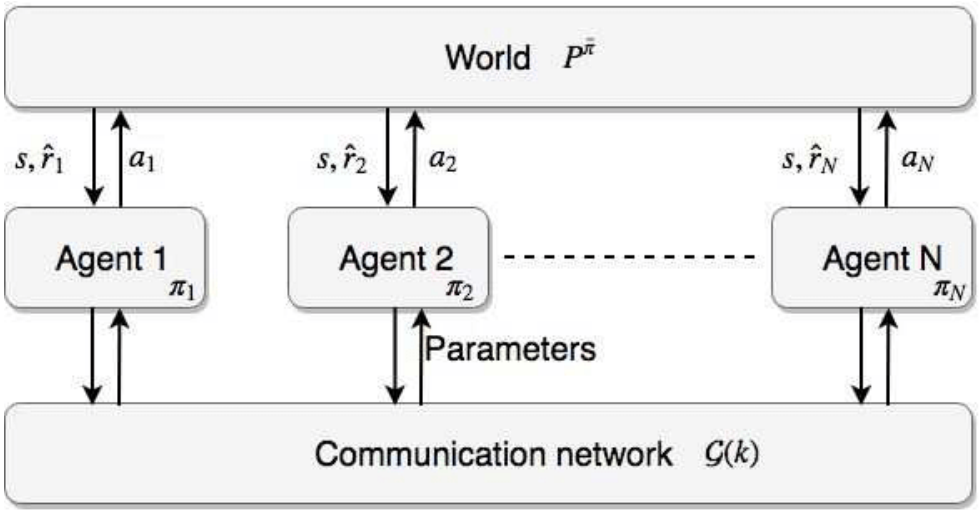
In this paper, we introduced a novel hierarchical clustering-based state grouping algorithm to improve the model's training efficiencies and overall performance in reinforcement learning. This algorithm preserves near-optimal behaviour makes efficient computation and lowers the time and data needed for decision-making.
Looking back to the research questions in section 1.2.1, as for the first question, the HCSG algorithm accelerates RL learning by partitioning the ground state space into smaller state clusters and distributing the training budget, this method also adds balance to the training data which helps maintain/improve model performance.
For the second research question, by limiting the training memory and episodes, the performance of the trained agents can reflect the training efficiencies. We use multiple evaluation metrics to evaluate the model performance, and compared the HCSG algorithm with mixed environment transfer and baseline algorithm. We use case tests and collected some ablation statistics to understand the different performances from macro and high levels. We compare models trained with different maximum state dimensions of the HCSG algorithm to evaluate the effects of state space splitting. As for the last question, the improved rewards in base environment regardless of the road length for the HCSG model indicates that the HCSG algorithm achieves better performance in improving agents' overall performance in various scenarios compared with the MET and baseline algorithm. The CAER results in multiple environments prove that the HCSG algorithm is more robust to changing environments. Besides, the HCSG algorithm avoids driver unfit situations and surpassed the other two algorithms in improving driving safety. At last, the state distribution for two test scenarios proved that our proposed algorithm transcends the decision tree algorithm by generalizing to more scenarios and maximizing the long-term rewards. The case test results help explain the performance differences by looking into each model's behaviour in each specific case, and the ablation statistics explain that the HCSG model outperforms MET and baseline algorithm by avoiding critical scenarios in which it takes more frequent actions and keeping high automation levels.

## 7.2 DISCUSSION

### 7.2.1 Algorithm advantages

Modern AI industries mostly use huge neural networks plus big data to solve complicated tasks. Our HCSG algorithm, instead, uses small neural networks to solve complicated tasks. This provides a promising future for reinforcement learning in the AI industry. The smaller policy networks require less computation budget. Thus, the HCSG algorithm weakens the reliance on hardware and reduces the cost of implementing the AI models in the industry.
the HCSG-RL algorithm is also transferable. For this paper, the HCSG-RL algorithm is trained and tested solely in a Mediator environment. It can also be implemented in other environments or RL algorithms. The algorithm is independent of environments. As Fig 1.2 shows, the HCSG-RL algorithm has separate algorithm structures,

each part can be easily adapted to different training algorithms and neural networks. HCSG generated and optimize multiple agents and distribute computation for complicated tasks. This provides promising future work for few-shot learning and Distributed Reinforcement Learning.




### 7.2.2 Drawbacks of the agent

This Mediator environment has discrete and limited action and state space. In the case of continuous or high dimensional state space, it may generate surplus sub-policies. Current methods for grouping the sub-polices use case test, for high state space, this might seem inefficient. But does not indicates that the HCSG algorithm is inferior in high dimension/complicated state space. Conversely, as for transfer learning becomes much harder as state-space grows, splitting state-space seems a more promising method in high-dimension state spaces.

## 7.3 FUTURE WORK

This section provides possible future research direction for our future work.

**Extending to more environments** The current algorithm was evaluated based on the Mediator environment and produced a fairly generalized model. As the algorithm is separated from the RL training environment, It might be worth investigating the adaptability of HCSG-RL to other environments and evaluating its effectiveness to improve the model's generalization abilities.

**Extending to high dimension state space** The basic idea behind the HCSG algorithm is to decompose the training environment into multiple MDPs and train multiple agents for groups of them and apply hierarchical RL that uses a master policy for choosing agents. As mentioned in the above section, the HCSG algorithm might suffer from surplus sub-policies in high dimensional state space. Current state and environment encoding might become inefficient, thus a smarter algorithm in state and agents grouping might give more efficient results. **RL algorithm** The model-based off-policy algorithm is selected for the current RL algorithm in HCSG. A better RL algorithm could train a better agent and simplify the training procedure. D3QN with prioritized experience replies might train agents with better performance on tail distribution. Compared with other RL algorithms, DQN can be unstable and gives poor convergence. SAC with a regularization entropy factor can be more efficient. Improving the RL algorithm in HCSG probably can give a more

optimized model.

**Network architecture** Current HCSG uses a rather small neural net for the sub-agents with one hidden layer. A deeper/wider neural net might be more generalized than small neural nets. And that can reduce the number of agents for the HCSG algorithm. A bigger neural net is whereas harder to train. Thus a trade-off is worth investigating.

**Reply buffer** For the existing HCSG algorithm, we utilize all the experience learned during training to update the policies. Some of the experience in the reply buffer might be useless and disturbs the agent from learning important information. The algorithm that produces a rather even distribution of states in the reply buffer might generate agents with better overall performance. It could be worth investigating methods to efficiently utilize replay buffers to accelerate agent learning.

**Master policy** In this paper, we use a deterministic master policy for selecting trained agents from state clusters. Its number of parameters is proportional to the size of the state space. Huge state space might require surplus parameters for the master policy which is inefficient in training and testing. Thus it might be worth looking into how to better design master policies for selecting the sub-policies.

# BIBLIOGRAPHY


[1] David Abel. "A theory of state abstraction for reinforcement learning". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33. 01. 2019, pp. 9876–9877.

[2] Jing An, Lexing Ying, and Yuhua Zhu. "Why resampling outperforms reweighting for correcting sampling bias with stochastic gradients". In: *arXiv preprint arXiv:2009.13447* (2020).

[3] Shin Ando. "Deep Over-sampling Framework for Classifying Imbalanced Data". In: *Lecture Notes in Computer Science book series* 10534 (2017), pp. 770–785. DOI: https://doi-org.tudelft.idm.oclc.org/10.1007/978-3-319-71249-9_46.

[4] Dan C Cireşan et al. "High-performance neural networks for visual object classification". In: *arXiv preprint arXiv:1102.0183* (2011).

[5] Karl Cobbe et al. "Quantifying generalization in reinforcement learning". In: *International Conference on Machine Learning*. PMLR. 2019, pp. 1282–1289.

[6] Jingliang Duan et al. "Hierarchical reinforcement learning for self-driving decision-making without reliance on labelled driving data". In: *IET Intelligent Transport Systems* 14.5 (2020), pp. 297–305.

[7] Dumitru Erhan et al. "Why does unsupervised pre-training help deep learning?" In: *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings. 2010, pp. 201–208.

[8] Youngkyu Hong et al. "Disentangling label distribution for long-tailed visual recognition". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 6626–6636.

[9] Kao-Shing Hwang, Yu-Jen Chen, and Chun-Ju Wu. "Fusion of multiple behaviors using layered reinforcement learning". In: *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans* 42.4 (2012), pp. 999–1004.

[10] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. "Reinforcement learning: A survey". In: *Journal of artificial intelligence research* 4 (1996), pp. 237–285.

[11] Leonard Kaufman and Peter J Rousseeuw. *Finding groups in data: an introduction to cluster analysis*. John Wiley & Sons, 2009.

[12] Jaehyung Kim, Jongheon Jeong, and Jinwoo Shin. "M2m: Imbalanced classification via major-to-minor translation". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 13896–13905.

[13] Man-Je Kim and Chang Wook Ahn. "Hybrid fighting game AI using a genetic algorithm and Monte Carlo tree search". In: *Proceedings of the genetic and evolutionary computation conference companion*. 2018, pp. 129–130.

[14] Man-Je Kim et al. "Genetic state-grouping algorithm for deep reinforcement learning". In: *Expert Systems with Applications* 161 (2020), p. 113695.

[15] Cian Lin. "Towards hybrid over- and under-sampling combination methods for class imbalanced datasets: an experimental study". In: *Artificial Intelligence Review* (2022). DOI: https://doi-org.tudelft.idm.oclc.org/10.1007/s10462-022-10186-5.

[16] Tsung-Yi Lin. "Focal Loss for Dense Object Detection". In: (2017). DOI: https://doi.org/10.48550/arXiv.1708.02002.

[17] Tsung-Yi Lin et al. "Focal loss for dense object detection". In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 2980–2988.

[18] Aakash Maroti. "Rbed: Reward based epsilon decay". In: *arXiv preprint arXiv:1910.13701* (2019).

[19] Michael Montemerlo et al. "Junior: The stanford entry in the urban challenge". In: *Journal of field Robotics* 25.9 (2008), pp. 569–597.

[20] Mirco Mutti, Mattia Mancassola, and Marcello Restelli. "Unsupervised Reinforcement Learning in Multiple Environments". In: *arXiv preprint arXiv:2112.08746* (2021).

[21] Anthony J Myles et al. "An introduction to decision tree modeling". In: *Journal of Chemometrics: A Journal of the Chemometrics Society* 18.6 (2004), pp. 275–285.

[22] Seulki Park. "Influence-Balanced Loss for Imbalanced Visual Classification". In: (2021). DOI: https://doi.org/10.48550/arXiv.2110.02444.

[23] Jiawei Ren et al. "Balanced meta-softmax for long-tailed visual recognition". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 4175–4186.

[24] Show-JaneYen. "Cluster-based under-sampling approaches for imbalanced data distributions". In: *Expert Systems with Applications* (2009). DOI: https://doi.org/10.1016/j.eswa.2008.06.108.

[25] Show-JaneYen. "Deep Long-Tailed Learning: A Survey". In: (2021). DOI: https://doi.org/10.48550/arXiv.2110.04596.

[26] Hang Song et al. "Prioritized Replay Dueling DDQN Based Grid-Edge Control of Community Energy Storage System". In: *IEEE Transactions on Smart Grid* 12.6 (2021), pp. 4950–4961. DOI: 10.1109/TSG.2021.3099133.

[27] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[28] Josh Tobin et al. "Domain randomization for transferring deep neural networks from simulation to the real world". In: *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE. 2017, pp. 23–30.

[29] Faraz Torabi, Garrett Warnell, and Peter Stone. "Behavioral cloning from observation". In: *arXiv preprint arXiv:1805.01954* (2018).

[30] Hado Van Hasselt, Arthur Guez, and David Silver. "Deep reinforcement learning with double q-learning". In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 30. 1. 2016.

[31] Tao Wang et al. "The devil is in classification: A simple framework for long-tail instance segmentation". In: *European conference on computer vision*. Springer. 2020, pp. 728–744.

[32] Yu-Xiong Wang, Deva Ramanan, and Martial Hebert. "Learning to model the tail". In: *Advances in Neural Information Processing Systems* 30 (2017).

[33] Xudong Wang. "Long-tailed Recognition by Routing Diverse Distribution-Aware Experts". In: (2021). DOI: https://doi.org/10.48550/arXiv.2010.01809.

[34] Yiru Wang et al. "Dynamic curriculum learning for imbalanced data classification". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, pp. 5017–5026.

[35] Allan P White and Wei Zhong Liu. "Bias in information-based measures in decision tree induction". In: *Machine Learning* 15.3 (1994), pp. 321–329.

[36] Esther Wong, Kin Leung, and Tony Field. "State-space decomposition for Reinforcement Learning". In: (2021).

[37] Franco van Wyk, Anahita Khojandi, and Neda Masoud. "Optimal switching policy between driving entities in semi-autonomous vehicles". In: *Transportation Research Part C: Emerging Technologies* 114 (2020), pp. 517–531.

[38] Liuyu Xiang, Guiguang Ding, and Jungong Han. "Learning from multiple experts: Self-paced knowledge distillation for long-tailed classification". In: *European Conference on Computer Vision*. Springer. 2020, pp. 247–263.

[39] Yuzhe Yang and Zhi Xu. "Rethinking the value of labels for improving class-imbalanced learning". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 19290–19301.

[40] Xiucai Ye. "An oversampling framework for imbalanced classification based on Laplacian eigenmaps". In: *Neurocomputing* (2020). DOI: https://doi.org/10.1016/j.neucom.2020.02.081.

[41] ShowJane Yen. "Under-Sampling Approaches for Improving Prediction of the Minority Class in an Imbalanced Dataset". In: *Lecture Notes in Control and Information Sciences* 344 (2006), pp. 731–740.

[42] Yuhang Zang, Chen Huang, and Chen Change Loy. "Fasa: Feature augmentation and sampling adaptation for long-tailed instance segmentation". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 3457–3466.

[43] Yusen Zhan, Haitham Bou Ammar, et al. "Theoretically-grounded policy advice from multiple teachers in reinforcement learning settings with applications to negative transfer". In: *arXiv preprint arXiv:1604.03986* (2016).

[44] Amy Zhang, Nicolas Ballas, and Joelle Pineau. "A dissection of overfitting and generalization in continuous reinforcement learning". In: *arXiv preprint arXiv:1806.07937* (2018).

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

[46] Yifan Zhang et al. "Deep long-tailed learning: A survey". In: *arXiv preprint arXiv:2110.04596* (2021).

[47] Zizhao Zhang and Tomas Pfister. "Learning fast sample re-weighting without reward data". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 725–734.

[48] Feiyun Zhu et al. "Group-driven reinforcement learning for personalized mhealth intervention". In: *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer. 2018, pp. 590–598.

# A APPENDIX A. EXPERIMENT SET UP FOR THE BASE ENVIRONMENT

| **Parameters** | **values** |
|---|---|
| toolkit | pytorch |
| training algorithm | DDQN |
| learning rate | 5e-5 |
| loss function | smooth L1 loss |
| optimizer | Adam |
| exploration $\epsilon$ | 0.5 to 0.02 |
| *eps_decay* | 0.995 |
| $\gamma$ | 0.99 |
| $\tau$ | 0.01 |
| total batch length | 100k |
| batch size | 8 |
| target update period | 4 |
| seed | 0 |
| *initial_level_probabilities* | [0.25,0.25,0.25,0.25] |
| *initial_distraction_probabilities* | [0.25,0.25,0.25,0.25] |
| *maximum_level_probabilities* | [0.25,0.25,0.25,0.25] |
| *allowed_driver_events* | ["DISTRACTION", "NDRT"] |
| $max_occurrences\_of\_driver\_event$ | [300, 5] |
| *driver_event_probability* | [0.99, 0] |
| *distraction_increase_prob* | 0.005 |
| *distraction_ends_midway_prob* | 0.02 |
| *distractions_ends_prob* | 0.1 |
| *ndrt_ends_prob* | 0.001 |
| *driver_request_cancel_prob* | 0.005 |
| *decline_threshold* | 120.0 |
| *road_length* | 30 |
| *road_types* | ["URBAN", "PROVINCIAL", "HIGHWAY"] |
| $max\_road_types$ | [1, 1, 3] |
| *urban_max_level* | L4 |
| *provincial_max_level* | L4 |
| *highway_max_level* | L4 |
| *allowed_static_events* | [] |
| *allowed_dynamic_events* | [] |
| *available_actions* | ['DN','SSL4', 'ESL4', 'CD', 'ES'] |
| *total_simultaneous_actions* | 1 |
| *ssl_max_response_time* | 10.0 |
| *suggested_shift_response_probability* | 1 |
| *suggested_shift_acceptance_probability* | 0.5 |
| *pd_success_probability* | 0.95 |
| *cd_success_probability* | 0.8 |
| *esl_time* | 2 |

**Table A.1:** Experiment A

# B APPENDIX B. DECISION TREE/BASELINE ALGORITHM

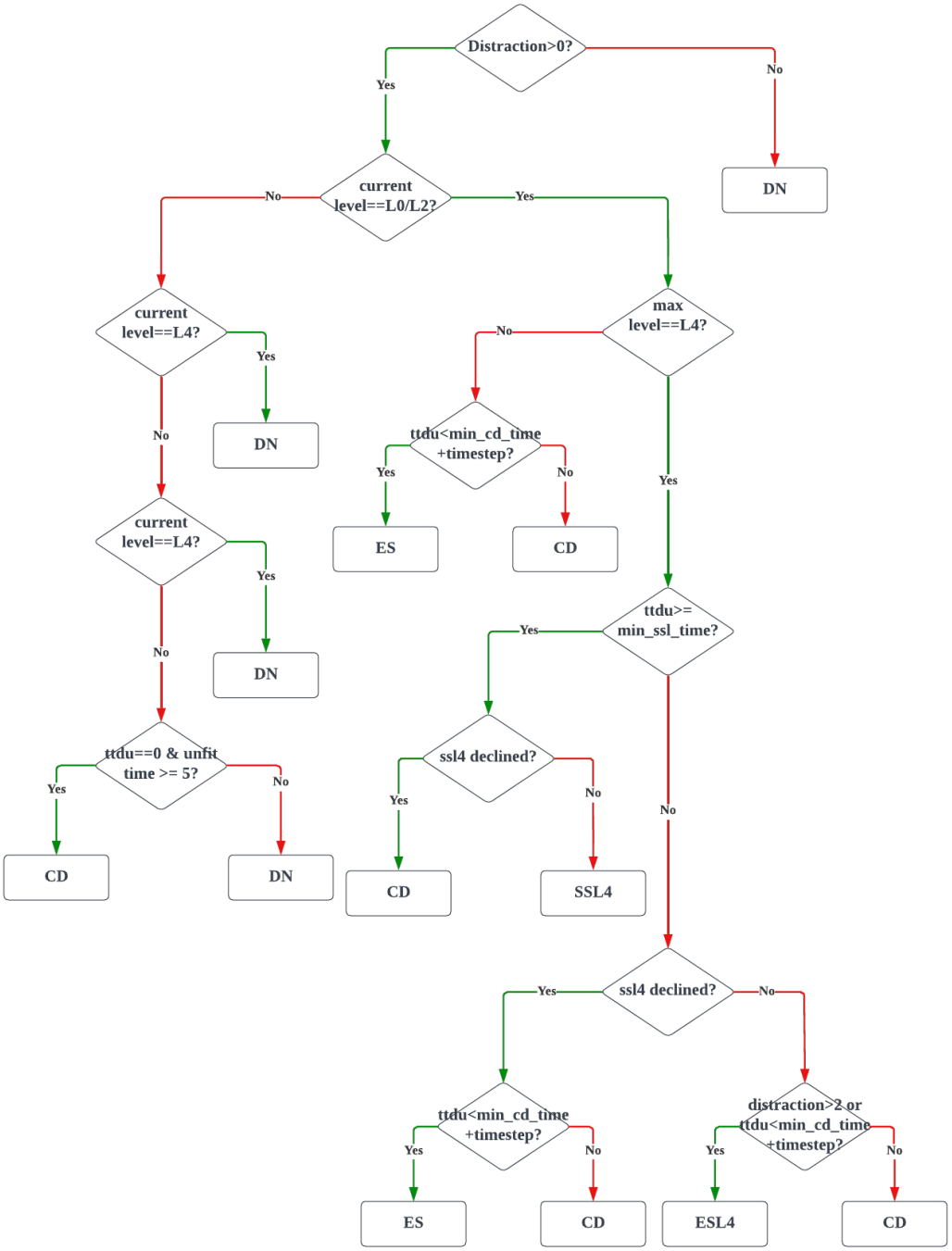



**Figure B.1:** Decision tree/baseline algorithm.

# C APPENDIX C. ENVIRONMENT ENCODING

| Environment Number | Initial Automation Level | Initial Distraction Level | Max Automation Level |
|---|---|---|---|
| 0 | [1,0,0,0] | [1,0,0,0] | [1,0,0,0] |
| 1 | [1,0,0,0] | [1,0,0,0] | [0,1,0,0] |
| 2 | [1,0,0,0] | [1,0,0,0] | [0,0,1,0] |
| 3 | [1,0,0,0] | [1,0,0,0] | [0,0,0,1] |
| 4 | [1,0,0,0] | [0,1,0,0] | [1,0,0,0] |
| 5 | [1,0,0,0] | [0,1,0,0] | [0,1,0,0] |
| 6 | [1,0,0,0] | [0,1,0,0] | [0,0,1,0] |
| 7 | [1,0,0,0] | [0,1,0,0] | [0,0,0,1] |
| 8 | [1,0,0,0] | [0,0,1,0] | [1,0,0,0] |
| 9 | [1,0,0,0] | [0,0,1,0] | [0,1,0,0] |
| 10 | [1,0,0,0] | [0,0,1,0] | [0,0,1,0] |
| 11 | [1,0,0,0] | [0,0,1,0] | [0,0,0,1] |
| 12 | [1,0,0,0] | [0,0,0,1] | [1,0,0,0] |
| 13 | [1,0,0,0] | [0,0,0,1] | [0,1,0,0] |
| 14 | [1,0,0,0] | [0,0,0,1] | [0,0,1,0] |
| 15 | [1,0,0,0] | [0,0,0,1] | [0,0,0,1] |
| 16 | [0,1,0,0] | [1,0,0,0] | [0,1,0,0] |
| 17 | [0,1,0,0] | [1,0,0,0] | [0,0,1,0] |
| 18 | [0,1,0,0] | [1,0,0,0] | [0,0,0,1] |
| 19 | [0,1,0,0] | [0,1,0,0] | [0,1,0,0] |
| 20 | [0,1,0,0] | [0,1,0,0] | [0,0,1,0] |
| 21 | [0,1,0,0] | [0,1,0,0] | [0,0,0,1] |
| 22 | [0,1,0,0] | [0,0,1,0] | [0,1,0,0] |
| 23 | [0,1,0,0] | [0,0,1,0] | [0,0,1,0] |
| 24 | [0,1,0,0] | [0,0,1,0] | [0,0,0,1] |
| 25 | [0,1,0,0] | [0,0,0,1] | [0,1,0,0] |
| 26 | [0,1,0,0] | [0,0,0,1] | [0,0,1,0] |
| 27 | [0,1,0,0] | [0,0,0,1] | [0,0,0,1] |
| 28 | [0,0,1,0] | [1,0,0,0] | [0,0,1,0] |
| 29 | [0,0,1,0] | [1,0,0,0] | [0,0,0,1] |
| 30 | [0,0,1,0] | [0,1,0,0] | [0,0,1,0] |
| 31 | [0,0,1,0] | [0,1,0,0] | [0,0,0,1] |
| 32 | [0,0,1,0] | [0,0,1,0] | [0,0,1,0] |
| 33 | [0,0,1,0] | [0,0,1,0] | [0,0,0,1] |
| 34 | [0,0,1,0] | [0,0,0,1] | [0,0,1,0] |
| 35 | [0,0,1,0] | [0,0,0,1] | [0,0,0,1] |
| 36 | [0,0,0,1] | [1,0,0,0] | [0,0,0,1] |
| 37 | [0,0,0,1] | [0,1,0,0] | [0,0,0,1] |
| 38 | [0,0,0,1] | [0,0,1,0] | [0,0,0,1] |
| 39 | [0,0,0,1] | [0,0,0,1] | [0,0,0,1] |

# D APPENDIX D. STATE ENCODING

| index | Auto level | distraction level | max auto level | TTDU | index | Auto level | distraction level | max auto level | TTDU |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 83 | 1 | 1 | 2 | 0 |
| 1 | 0 | 0 | 0 | 1 | 84 | 1 | 1 | 2 | 1 |
| 2 | 0 | 0 | 0 | 2 | 85 | 1 | 1 | 2 | 2 |
| 3 | 0 | 0 | 0 | 3 | 86 | 1 | 1 | 2 | 3 |
| 4 | 0 | 0 | 0 | 4 | 87 | 1 | 1 | 2 | 4 |
| 5 | 0 | 0 | 0 | 5 | 88 | 1 | 1 | 3 | 0 |
| 6 | 0 | 0 | 1 | 0 | 89 | 1 | 1 | 3 | 1 |
| 7 | 0 | 0 | 1 | 1 | 90 | 1 | 1 | 3 | 2 |
| 8 | 0 | 0 | 1 | 2 | 91 | 1 | 1 | 3 | 3 |
| 9 | 0 | 0 | 1 | 3 | 92 | 1 | 1 | 3 | 4 |
| 10 | 0 | 0 | 1 | 4 | 93 | 1 | 2 | 1 | 0 |
| 11 | 0 | 0 | 1 | 5 | 94 | 1 | 2 | 1 | 1 |
| 12 | 0 | 0 | 2 | 0 | 95 | 1 | 2 | 1 | 2 |
| 13 | 0 | 0 | 2 | 1 | 96 | 1 | 2 | 1 | 3 |
| 14 | 0 | 0 | 2 | 2 | 97 | 1 | 2 | 2 | 0 |
| 15 | 0 | 0 | 2 | 3 | 98 | 1 | 2 | 2 | 1 |
| 16 | 0 | 0 | 2 | 4 | 99 | 1 | 2 | 2 | 2 |
| 17 | 0 | 0 | 2 | 5 | 100 | 1 | 2 | 2 | 3 |
| 18 | 0 | 0 | 3 | 0 | 101 | 1 | 2 | 3 | 0 |
| 19 | 0 | 0 | 3 | 1 | 102 | 1 | 2 | 3 | 1 |
| 20 | 0 | 0 | 3 | 2 | 103 | 1 | 2 | 3 | 2 |
| 21 | 0 | 0 | 3 | 3 | 104 | 1 | 2 | 3 | 3 |
| 22 | 0 | 0 | 3 | 4 | 105 | 1 | 3 | 1 | 0 |
| 23 | 0 | 0 | 3 | 5 | 106 | 1 | 3 | 1 | 1 |
| 24 | 0 | 1 | 0 | 0 | 107 | 1 | 3 | 2 | 0 |
| 25 | 0 | 1 | 0 | 1 | 108 | 1 | 3 | 2 | 1 |
| 26 | 0 | 1 | 0 | 2 | 109 | 1 | 3 | 3 | 0 |
| 27 | 0 | 1 | 0 | 3 | 110 | 1 | 3 | 3 | 1 |
| 28 | 0 | 1 | 1 | 0 | 111 | 2 | 0 | 2 | 0 |
| 29 | 0 | 1 | 1 | 1 | 112 | 2 | 0 | 2 | 1 |
| 30 | 0 | 1 | 1 | 2 | 113 | 2 | 0 | 2 | 2 |
| 31 | 0 | 1 | 1 | 3 | 114 | 2 | 0 | 2 | 3 |
| 32 | 0 | 1 | 2 | 0 | 115 | 2 | 0 | 2 | 4 |
| 33 | 0 | 1 | 2 | 1 | 116 | 2 | 0 | 2 | 5 |
| 34 | 0 | 1 | 2 | 2 | 117 | 2 | 0 | 3 | 0 |
| 35 | 0 | 1 | 2 | 3 | 118 | 2 | 0 | 3 | 1 |
| 36 | 0 | 1 | 3 | 0 | 119 | 2 | 0 | 3 | 2 |
| 37 | 0 | 1 | 3 | 1 | 120 | 2 | 0 | 3 | 3 |

# E APPENDIX E. EXPERIMENTS SETUP FOR BASE, L4 AND L3 ENVIRONMENTS

| **Parameters** | **values** |
|---|---|
| toolkit | pytorch |
| training algorithm | DDQN |
| Episodes | 500 |
| learning rate | 5e-5 |
| loss function | smooth L1 loss |
| optimizer | Adam |
| exploration $\epsilon$ | 0.5 to 0.02 |
| *eps_decay* | 0.995 |
| $\gamma$ | 0.99 |
| $\tau$ | 0.01 |
| total batch length | 100k |
| batch size | 8 |
| target update period | 4 |
| seed | 0 |
| *initial_level_probabilities* | [0.25,0.25,0.25,0.25] |
| *initial_distraction_probabilities* | [0.25,0.25,0.25,0.25] |
| *maximum_level_probabilities* | [0.25,0.25,0.25,0.25] |
| *allowed_driver_events* | [”DISTRACTION”, ”NDRT”] |
| $max_0ccurrences\_of\_driver\_event$ | [300, 5] |
| *driver_event_probability* | [0.99, 0] |
| *distraction_increase_prob* | 0.005 |
| *distraction_ends_midway_prob* | 0.02 |
| *distractions_ends_prob* | 0.1 |
| *ndrt_ends_prob* | 0.001 |
| *driver_request_cancel_prob* | 0.005 |
| *decline_threshold* | 120.0 |
| *road_length* | 30 |
| *road_types* | [”URBAN”, ”PROVINCIAL”, ”HIGHWAY”] |
| $max\_road_types$ | [1, 1, 3] |
| *urban_max_level* | L4 |
| *provincial_max_level* | L4 |
| *highway_max_level* | L4 |
| *allowed_static_events* | [] |
| *allowed_dynamic_events* | [] |
| *available_actions* | [’DN’,’SSL4’, ’ESL4’, ’CD’, ’ES’] |
| *total_simultaneous_actions* | 1 |
| *ssl_max_response_time* | 10.0 |
| *suggested_shift_response_probability* | 1 |
| *suggested_shift_acceptance_probability* | 0.5 |
| *pd_success_probability* | 0.95 |
| *cd_success_probability* | 0.8 |
| *esl_time* | 2 |

**Table E.1:** Experiment Base Environment

| **Parameters** | **values** |
|---|---|
| toolkit | pytorch |
| training algorithm | DDQN |
| Episodes | 500 |
| learning rate | 5e-5 |
| loss function | smooth L1 loss |
| optimizer | Adam |
| exploration $\epsilon$ | 0.5 to 0.02 |
| *eps_decay* | 0.995 |
| $\gamma$ | 0.99 |
| $\tau$ | 0.01 |
| total batch length | 100k |
| batch size | 8 |
| target update period | 4 |
| seed | 0 |
| *initial_level_probabilities* | [0,0,0,1] |
| *initial_distraction_probabilities* | [0.25,0.25,0.25,0.25] |
| *maximum_level_probabilities* | [0,0,0,1] |
| *allowed_driver_events* | [”DISTRACTION”, ”NDRT”] |
| $max_0ccurrences\_of\_driver\_event$ | [300, 5] |
| *driver_event_probability* | [0.99, 0] |
| *distraction_increase_prob* | 0.005 |
| *distraction_ends_midway_prob* | 0.02 |
| *distractions_ends_prob* | 0.1 |
| *ndrt_ends_prob* | 0.001 |
| *driver_request_cancel_prob* | 0.005 |
| *decline_threshold* | 120.0 |
| *road_length* | 30 |
| *road_types* | [”URBAN”, ”PROVINCIAL”, ”HIGHWAY”] |
| $max\_road_types$ | [1, 1, 3] |
| *urban_max_level* | L4 |
| *provincial_max_level* | L4 |
| *highway_max_level* | L4 |
| *allowed_static_events* | [] |
| *allowed_dynamic_events* | [] |
| *available_actions* | [’DN’,’SSL4’, ’ESL4’, ’CD’, ’ES’] |
| *total_simultaneous_actions* | 1 |
| *ssl_max_response_time* | 10.0 |
| $suggested\_shift\_response\_probability$ | 1 |
| $suggested\_shift\_acceptance\_probability$ | 0.5 |
| *pd_success_probability* | 0.95 |
| *cd_success_probability* | 0.8 |
| *esl_time* | 2 |

**Table E.2:** Experiment L4 environment

| **Parameters** | **values** |
|---|---|
| toolkit | pytorch |
| training algorithm | DDQN |
| Episodes | 500 |
| learning rate | 5e-5 |
| loss function | smooth L1 loss |
| optimizer | Adam |
| exploration $\epsilon$ | 0.5 to 0.02 |
| *eps_decay* | 0.995 |
| $\gamma$ | 0.99 |
| $\tau$ | 0.01 |
| total batch length | 100k |
| batch size | 8 |
| target update period | 4 |
| seed | 0 |
| *initial_level_probabilities* | [0,0,1,0] |
| *initial_distraction_probabilities* | [0.25,0.25,0.25,0.25] |
| *maximum_level_probabilities* | [0,0,1,0] |
| *allowed_driver_events* | [”DISTRACTION”, ”NDRT”] |
| $max_0ccurrences\_of\_driver\_event$ | [300, 5] |
| *driver_event_probability* | [0.99, 0] |
| *distraction_increase_prob* | 0.005 |
| *distraction_ends_midway_prob* | 0.02 |
| *distractions_ends_prob* | 0.1 |
| *ndrt_ends_prob* | 0.001 |
| *driver_request_cancel_prob* | 0.005 |
| *decline_threshold* | 120.0 |
| *road_length* | 30 |
| *road_types* | [”URBAN”, ”PROVINCIAL”, ”HIGHWAY”] |
| $max\_road_types$ | [1, 1, 3] |
| *urban_max_level* | L4 |
| *provincial_max_level* | L4 |
| *highway_max_level* | L4 |
| *allowed_static_events* | [] |
| *allowed_dynamic_events* | [] |
| *available_actions* | [’DN’,’SSL4’, ’ESL4’, ’CD’, ’ES’] |
| *total_simultaneous_actions* | 1 |
| *ssl_max_response_time* | 10.0 |
| *suggested_shift_response_probability* | 1 |
| *suggested_shift_acceptance_probability* | 0.5 |
| *pd_success_probability* | 0.95 |
| *cd_success_probability* | 0.8 |
| *esl_time* | 2 |

**Table E.3:** Experiment L3 environment

# F APPENDIX H. EXPERIMENT SETUP TTDU CONFIGURATION

| Parameters | values |
|---|---|
| toolkit | pytorch |
| tTest model | HCSG D3 |
| Episodes | 100 |
| seed | 0 |
| *initial_level_probabilities* | [0.25,0.25,0.25,0.25] |
| *initial_distraction_probabilities* | [0.25,0.25,0.25,0.25] |
| *maximum_level_probabilities* | [0.25,0.25,0.25,0.25] |
| *allowed_driver_events* | [”DISTRACTION”, ”NDRT”] |
| $max_0ccurrences\_of\_driver\_event$ | [300, 5] |
| *driver_event_probability* | [0.99, 0] |
| *distraction_increase_prob* | 0.005 |
| *distraction_ends_midway_prob* | 0.02 |
| *distractions_ends_prob* | 0.1 |
| *ndrt_ends_prob* | 0.001 |
| *driver_request_cancel_prob* | 0.005 |
| *decline_threshold* | 120.0 |
| *road_length* | [3,10,20,30] |
| *road_types* | [”URBAN”, ”PROVINCIAL”, ”HIGHWAY”] |
| $max\_road_types$ | [1, 1, 3] |
| *urban_max_level* | L4 |
| *provincial_max_level* | L4 |
| *highway_max_level* | L4 |
| *allowed_static_events* | [] |
| *allowed_dynamic_events* | [] |
| *available_actions* | [’DN’,’SSL4’, ’ESL4’, ’CD’, ’ES’] |
| *total_simultaneous_actions* | 1 |
| *ssl_max_response_time* | 10.0 |
| *suggested_shift_response_probability* | 1 |
| *suggested_shift_acceptance_probability* | 0.5 |
| *pd_success_probability* | 0.95 |
| *cd_success_probability* | 0.8 |
| *esl_time* | 2 |
| initial TTDU at L0 | [D0:200,D1:25,D2:14,D3:3] |
| initial TTDU at L2 | [D0:500,D1:45,D2:30,D3:10] |
| initial TTDU at L3 | [D0:500,D1:45,D2:30,D3:10] |
| initial TTDU at L4 | [D0:500,D1:45,D2:30,D3:10] |

**Table F.1:** Experiment 5

# G APPENDIX I. CASE TEST RESULTS

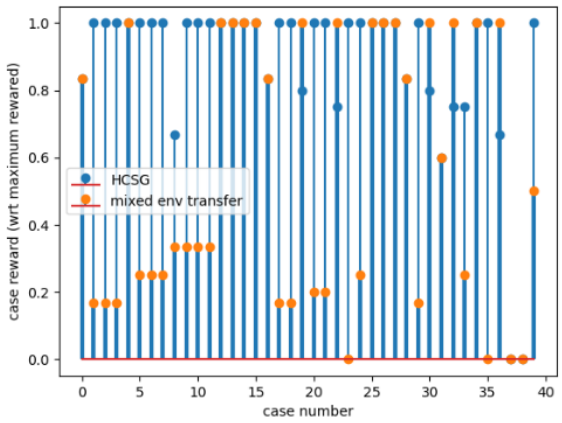



**Figure G.1:** Normalized case test reward of HCSG at D1 and MET algorithm

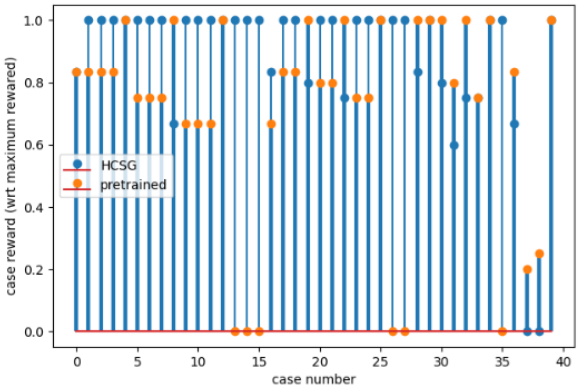



**Figure G.2:** Normalized case test reward of HCSG at D1 and pre-trained model

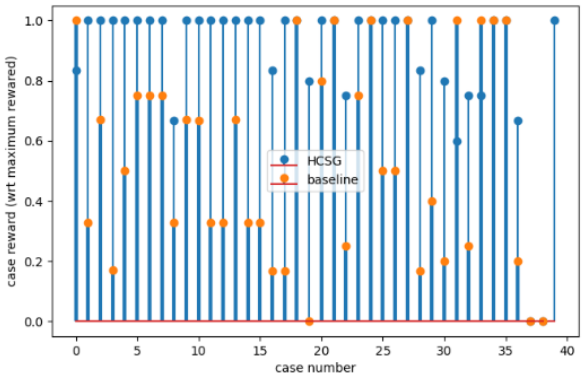



**Figure G.3:** Normalized case test reward of HCSG at D1 and baseline algorithm

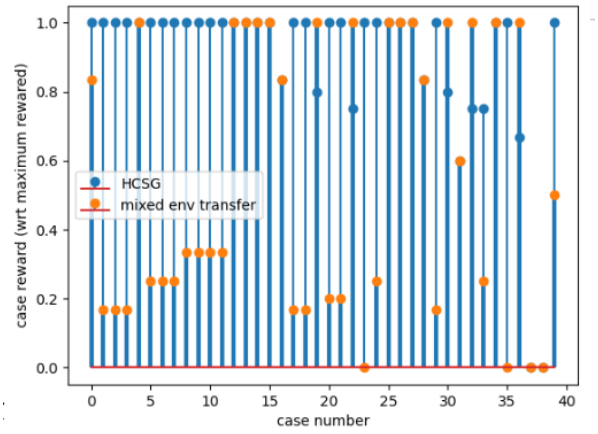



**Figure G.4:** Normalized case test reward of HCSG at D2 and MET algorithm

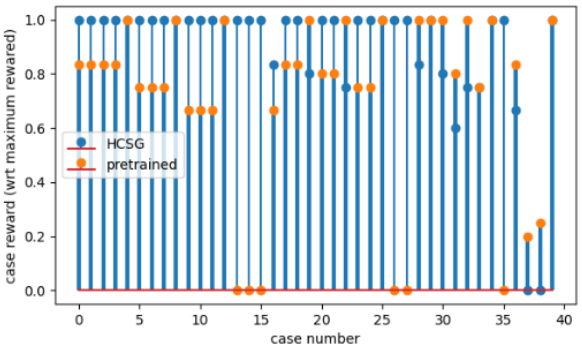



**Figure G.5:** Normalized case test reward of HCSG at D2 and pre-trained model

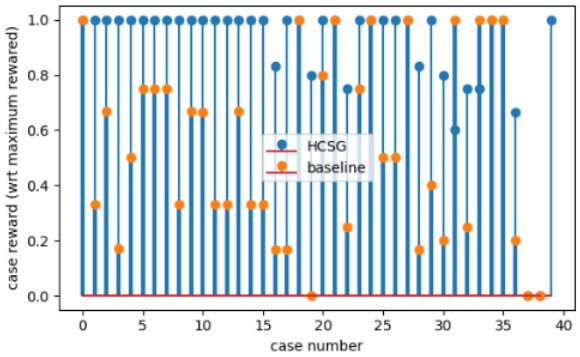



**Figure G.6:** Normalized case test reward of HCSG at D2 and baseline algorithm

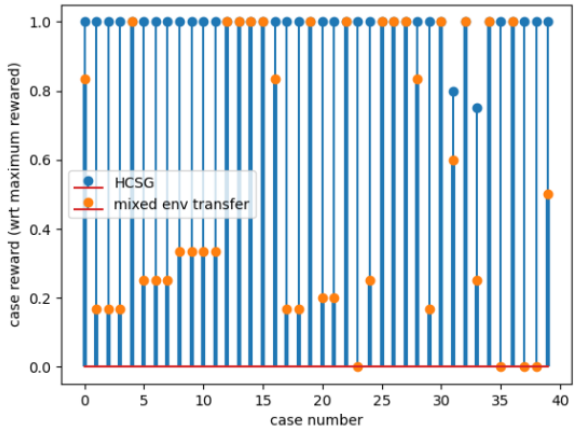



**Figure G.7:** Normalized case test reward of HCSG at D3 and MET algorithm

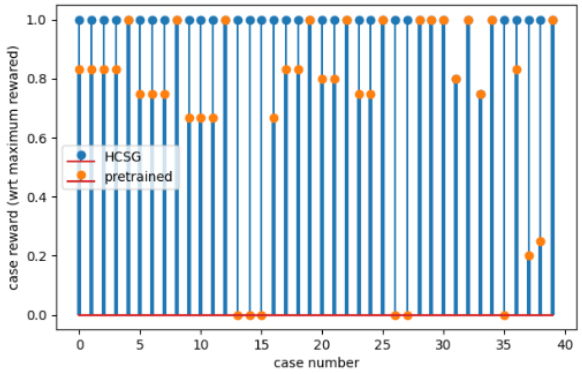



**Figure G.8:** Normalized case test reward of HCSG at D3 and pre-trained model

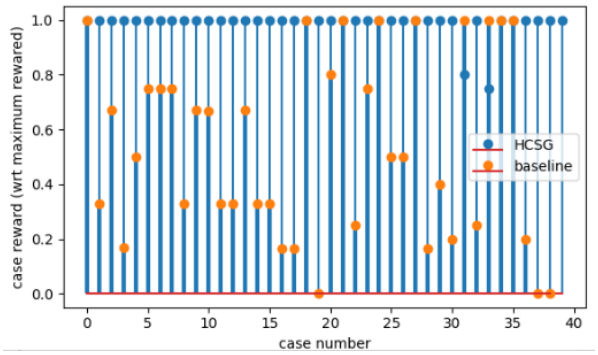



**Figure G.9:** Normalized case test reward of HCSG at D3 and baseline algorithm

# H APPENDIX J. EXPERIMENT SETUP 5

| Parameters | values |
|---|---|
| toolkit | pytorch |
| tTest model | [HCSG D3, HCSG, baseline] |
| Episodes | 100 |
| seed | 0 |
| *initial_level_probabilities* | [0.25,0.25,0.25,0.25] |
| *initial_distraction_probabilities* | [0.25,0.25,0.25,0.25] |
| *maximum_level_probabilities* | [0.25,0.25,0.25,0.25] |
| *allowed_driver_events* | [”DISTRACTION”, ”NDRT”] |
| $max_0ccurrences\_of\_driver\_event$ | [300, 5] |
| *driver_event_probability* | [0.99, 0] |
| *distraction_increase_prob* | 0.005 |
| *distraction_ends_midway_prob* | 0.02 |
| *distractions_ends_prob* | 0.1 |
| *ndrt_ends_prob* | 0.001 |
| *driver_request_cancel_prob* | 0.005 |
| *decline_threshold* | 120.0 |
| *road_length* | 30 |
| *road_types* | [”URBAN”, ”PROVINCIAL”, ”HIGHWAY”] |
| $max\_road_types$ | [1, 1, 3] |
| *urban_max_level* | L4 |
| *provincial_max_level* | L4 |
| *highway_max_level* | L4 |
| *allowed_static_events* | [] |
| *allowed_dynamic_events* | [] |
| *available_actions* | [’DN’,’SSL4’, ’ESL4’, ’CD’, ’ES’] |
| *total_simultaneous_actions* | 1 |
| *ssl_max_response_time* | 10.0 |
| *suggested_shift_response_probability* | 1 |
| *suggested_shift_acceptance_probability* | 0.5 |
| *pd_success_probability* | 0.95 |
| *cd_success_probability* | 0.8 |
| *esl_time* | 2 |
| initial TTDU at L0 | [D0:200,D1:25,D2:14,D3:3] |
| initial TTDU at L2 | [D0:500,D1:45,D2:30,D3:10] |
| initial TTDU at L3 | [D0:500,D1:45,D2:30,D3:10] |
| initial TTDU at L4 | [D0:500,D1:45,D2:30,D3:10] |

**Table H.1:** Experiment 5

# I APPENDIX K. EXPERIMENT SETUP 6

| **Parameters** | **values** |
|---|---|
| toolkit | pytorch |
| tTest model | [HCSG D3, HCSG, baseline] |
| Episodes | 100 |
| seed | 0 |
| *initial_level_probabilities* | [0,0,1,0] |
| *initial_distraction_probabilities* | [0,0,1,0] |
| *maximum_level_probabilities* | [0,0,0,1] |
| *allowed_driver_events* | [”DISTRACTION”, ”NDRT”] |
| $max_0ccurrences\_of\_driver\_event$ | [300, 5] |
| *driver_event_probability* | [0.99, 0] |
| *distraction_increase_prob* | 0.005 |
| *distraction_ends_midway_prob* | 0.02 |
| *distractions_ends_prob* | 0.1 |
| *ndrt_ends_prob* | 0.001 |
| *driver_request_cancel_prob* | 0.005 |
| *decline_threshold* | 120.0 |
| *road_length* | 30 |
| *road_types* | [”URBAN”, ”PROVINCIAL”, ”HIGHWAY”] |
| $max\_road_types$ | [1, 1, 3] |
| *urban_max_level* | L4 |
| *provincial_max_level* | L4 |
| *highway_max_level* | L4 |
| *allowed_static_events* | [] |
| *allowed_dynamic_events* | [] |
| *available_actions* | [’DN’,’SSL4’, ’ESL4’, ’CD’, ’ES’] |
| *total_simultaneous_actions* | 1 |
| *ssl_max_response_time* | 10.0 |
| $suggested\_shift\_response\_probability$ | 1 |
| $suggested\_shift\_acceptance\_probability$ | 0.5 |
| *pd_success_probability* | 0.95 |
| *cd_success_probability* | 0.8 |
| *esl_time* | 2 |
| initial TTDU at L0 | [D0:200,D1:25,D2:14,D3:3] |
| initial TTDU at L2 | [D0:500,D1:45,D2:30,D3:10] |
| initial TTDU at L3 | [D0:500,D1:45,D2:30,D3:10] |
| initial TTDU at L4 | [D0:500,D1:45,D2:30,D3:10] |

**Table I.1:** Experiment 5