Technische Hochschule Ingolstadt

Bachelor's Thesis

in the study program
B. Eng. in Autonomous Vehicle Engineering
Faculty of Electrical Engineering and Information Technology

Leveraging Criticality in Reinforcement Learning for Effective Transfer Learning in Autonomous Driving

First, Last name	:	**Daniil Navodey**
Matriculation number	:	00126504
First Examiner	:	Prof. Dr. Lenz Belzner
Second Examiner	:	Prof. Dr. Torsten Schön
Guide	:	Chidvilas Karpenahalli Ramakrishna
Institute	:	Almotion Bavaria, TH Ingolstadt

Affidavit

I declare that I have authored the present thesis independently and that I have not used any other sources/resources than the ones declared here. I have explicitly indicated all the material which has been quoted either literally or by consent from the sources used and I have marked verbatim and indirect quotations as such. I also declare that I have not presented the work elsewhere for examination purpose.

Ingolstadt, _____

Daniil Navodey

Acknowledgments

My deepest appreciation goes to Professor Lenz Belzner for his invaluable guidance, insightful feedback, and encouragement throughout the development of this work. His introduction to the topics of machine learning, decision-making algorithms and Python programming provided me with a strong foundation to build upon. Expertise and mentorship from his side have been crucial in shaping the direction and execution of my research.

I am also profoundly grateful to Research Assistant Chidvilas Karpenahalli Ramakrishna for his continuous support and guidance during the entire process of writing my thesis. His clear explanations of complex concepts, assistance in accessing essential resources, and support in setting up the necessary computational environment greatly enriched my understanding and facilitated my experiments.

I would like to express my heartfelt gratitude to Technische Hochschule Ingolstadt and all the professors of the Autonomous Vehicle Engineering study program for providing me with a solid foundation in this field. The knowledge, skills, and insights gained throughout my studies have been invaluable, not only for the successful completion of my thesis but for my entire academic journey.

Throughout the study program, I had the privilege of gaining both theoretical knowledge and practical experience in the field of autonomous vehicle engineering. The curriculum, combined with innovative research opportunities, allowed me to develop critical thinking and problem-solving skills that I will carry forward into my professional career.

Thank you to everyone who has contributed to this work, directly or indirectly. Your support, knowledge, and encouragement have significantly influenced the successful completion of my thesis and my academic growth throughout the study program.

Daniil Navodey

Ingolstadt, Germany

Abstract

While Reinforcement Learning is showing extensive potential in applications which include autonomous driving, its major drawback is the gigantic computing resources and time necessary to train a model from scratch. As a solution, transfer learning was introduced, reducing required data and time when adapting pre-trained models to new environments. However, huge sets of randomly sampled data are traditionally used in transfer learning, introducing great inefficiencies. This research proposes an approach to optimise the standard transfer learning process by leveraging criticality–a measure of the significance of one/more actions in a given state. The training time is reduced when the replay buffer of a model is fed with only critical states during the retraining process while the achieved model performance is maintained comparable to conventional transfer learning. The approach was validated by applying Deep Q-Network (DQN) in *Highway-Env* and *Merge-Env* environments for simulated autonomous driving. Results demonstrate that safety and other performance metrics are evaluated to be comparable for transfer learning based on criticality, although requiring only a fraction of training data. This suggests criticality approach is more feasible for applications in the real world. Insights gathered throughout the thesis will endow the research in the optimisation of machine learning and the development of autonomous driving.

List of figures

List of tables

Table of contents

1. Introduction

1.1. Motivation

Transportation nowadays has acquired a new round of development with the introduction of autonomous driving technologies based on machine learning, neural networks, and artificial intelligence. This has influenced safety, efficiency and accessibility of transportation. The key challenge of a self-driving vehicle is the number of decisions required for an algorithm to be taken in real-time, which may cause endangering consequences for drivers, passengers and other traffic participants. Reinforcement Learning is a promising approach to tackle this challenge because it allows systems to learn by trying different actions and improving over time. However, to train Reinforcement Learning sufficiently for real-world driving, it needs a lot of time and big data sets for the learning process.

The major weakness of Reinforcement Learning is the lack of adaptivity of models to new environments. This requires an additional learning process, which is time-consuming and inefficient. This is where Transfer Learning can help. It enables the ability of a system to take what it has learned in one environment and then apply this gathered knowledge to a different task, with less effort and time taken. Emphasizing the focus of Transfer Learning to be trained on critical states of the system–the moments where making the right decision matters for safety and performance–may enhance the learning speed and perform better under real-world conditions.

This thesis explores an approach to improving Reinforcement Learning's knowledge transfer in autonomous driving by applying critical situations to the models' training process. The aim is to reduce training time and computational costs while maintaining or even enhancing the performance of models, broadening the variety of challenges that could be handled, and ensuring the effectiveness of learning. By addressing these issues, this research could contribute to the further development of autonomous driving systems by making them more practical and reliable.

By experiments conducted in simulated autonomous driving environments, the thesis demonstrates the reduction of training time through applying the approach that leverages criticality-based Transfer Learning. The resultant model maintains performance, comparable to standard Transfer Learning.

1.2. Objective

The objective of this bachelor thesis is to research incorporating criticality-based training to enhance the efficiency of Transfer Learning. Traditional Reinforcement Learning methods require broad datasets and are computationally expensive to achieve optimal performance. Utilising Transfer Learning, pre-trained models can be leveraged in new environments, which can greatly reduce time. Despite that, standard Transfer Learning algorithms are still based on large sets of data, introducing inefficiencies.

One novel approach is introduced by this thesis: feed the pre-trained model with an exclusive set of critical states, optimising, by that, the Transfer Learning. Setting a focus on high-impact and highly informative states, the aim is to diminish training time, and costs, maintaining or even bettering the model's performance. The paper will validate the hypothesis through practical implementation, analysing performance results for both traditional Transfer Learning and criticality-aware Transfer Learning in simulated environments for autonomous driving.

1.3. Scope

This research is focused on the application of Reinforcement Learning in the field of autonomous driving. Specifically, the thesis is centred on evaluating the influence of Transfer Learning based on critical states inside of two environments for simulated driving:

1. **Highway-Env:** An environment, where an agent has to learn to handle lane-changing and collision-avoiding situations while navigating the simplified highway simulation autonomously.
2. **Merge-Env:** An environment, that realises more complex highway-merging scenarios, encouraging the agent to handle dynamic scenarios involving surrounding traffic.

Deep Q-Learning (DQN) will mainly be used as a base for the implementations of Reinforcement Learning models. Resultant models will further be accessed according to their performance under various methodologies of training. Crucial model types are:

1. **Basic Reinforcement Learning Model:** A model trained using DQN purely in a single environment.
2. **Traditional Transfer Learning Model:** A model trained with the use of a randomised set of states from a target environment based on the pre-trained model.
3. **Criticality-Based Transfer Learning Model:** A model trained with the use of preselected critical states of the target environment, which tends to optimise the efficiency of learning.

The testing in real-world scenarios of autonomous driving is not included in the scope. It is mainly centred on the implementation and effectiveness validation of the proposed algorithm.

2. Literature survey

2.1. Transfer Learning

Training a machine learning algorithm of optimal quality and performance requires big sets of training and evaluation data. Those sets are usually hard to gather or generate, as it costs time and money. The approach to be used as a better means of data collection is Transfer Learning (TL). The philosophy of this approach is to utilize models trained for tasks related to the target challenge so that the solution does not need to be developed from scratch [1]. TL tends to help Reinforcement Learning (RL) to become more universal and adaptive to new tasks as it creates models which own a collection of knowledge of other models.

The approach of TL can easily be found in real life. For instance, in linguistics, if a person knows one language, it becomes easier for them to learn a new language from the same language group. In the scope of this thesis, TL was performed between two environments: *Highway-Env* and *Merge-Env*.

The necessity and privilege of using TL can be seen when applying the approach to tasks related to one another. The autonomous driving task of the agent is to be equally applicable to both environments used throughout the thesis. Moreover, there are some feature sets that those environments share, such as:

- Operating between multiple lanes,
- Surrounding vehicles capable of accelerating/decelerating and changing lanes.

Looking into the scientific field, an analogy can represent the exact approach of TL: studying math and physics. If a person who studied math starts deepening their knowledge in physics, this can be called TL, as the math expertise makes it easier to acquire new knowledge in the field of physics. However, some themes in math will not be required while learning physics. Compared to the learning process without prior math knowledge, the time and effort investments are noticeably less when TL is applied.

2.2. Criticality

One small branch that can be applied to any type of learning which is not explored deep enough and not widely applied yet is criticality. The criticality has various ways to be computed when used in RL and autonomous driving. Despite that the definition of criticality is general for all the possible implementation fields: the higher value of criticality correspondence to the lower value of safety and higher threat level[2] in a chosen situation.

In the field of autonomous driving the knowledge of criticality, especially the ability of the machine to approximate this value, is crucial for the algorithms that are either already used or are still under development. The ability of models that control driving behaviour to process the environment, observations and disturbances to evaluate the measure for the threat level of the situation in real-time can boost the performance and increase the reliability of autonomous driving technologies.

2.2.1. Threshold method

The policy can be a base to calculate the necessary value. The criticality of a state can be accessed through the comparison of an entropy of the policy's output action distribution

captured at a state `s` with the threshold value. In this method the threshold value `t` is the one controlling the number of states to be considered "critical" [3]:

$$C_\pi = \{s \mid \mathcal{H}(\pi(\cdot \mid s)) < t\} \tag{1}$$

- C_π is the set of critical states under policy π.
- s a state in the environment.
- $H(\pi(\cdot \mid s))$ is the entropy of the distribution of the policy's output action at the state s.
- t is a threshold value determining how much the highest Q-value should differ from an average for the state to be graded as critical.

Instead of a policy, actor-critic methods can be useful in considering the action-value function (Q-function). Mainly, the formula decides if the state is critical according to the following condition: if the result produced by the random decision-making process is much worse than the result of acting optimally, then the state is critical.

$$C_\pi = \{s \mid \max_a Q^\pi(s,a) - \frac{1}{|A|}\sum_a Q^\pi(s,a) > t\} \tag{2}$$

where:

- C_π is the set of critical states under policy π.
- s a state in the environment.
- a is a possible action which can be taken in state s.
- $Q^\pi(s,a)$ is the action-value function, representing the expected return when taking action a in state s and following policy π.
- $\max_a Q^\pi(s,a)$ is the highest Q-value among all possible actions in the state s.
- $|A|$ is the number of all possible actions.
- t is a threshold value determining how much the highest Q-value should differ from an average for the state to be classified as critical.

2.2.2. Peak variance method

There are many other numerical methods to define the criticality value to be used in different algorithms and systems. However, for this thesis, the difference between the maximum and average Q-value was substituted by the variance value. The variance is calculated by averaging squares of deviations from the distribution mean:

$$\sigma^2 = \frac{\sum_a (Q^\pi(s,a))^2}{|A|} - \left(\frac{\sum_a Q^\pi(s,a)}{|A|}\right)^2 \tag{3}$$

The reliable indicator of the criticality is a variance of the Q-function [4]. When the variability of the Q-function is high among all available actions for a given state, it is more likely for the trained policy to favour one action over others. A state is to be called "critical" if one action has a distinctly higher probability than the rest.

Compared to the difference between the maximum and average of a Q-value, variance can determine criticality not only based on one peaking value but combining all existing values as a set. The variance is also more informative about the overall distribution of Q-values among all actions.

The approach to analyse the variance of Q-values was based on the peak values of the metric. The algorithm and tools used to determine peaking variances will be further explored in Section 3.5.

Considering the existing research into different methods of calculation, evaluation or approximation of criticality the target of this bachelor's work was chosen not to develop a more complex, optimised or effective technique to measure criticality, not at all. Current research tries to go deeper into applications of criticality and, more precisely critical states, in TL.

2.3. Background and Related Work

RL is a paradigm of machine learning in which learning is realised through interaction with an environment and receiving rewards for performed actions. The goal of the agent is to maximise the cumulative reward over time.

RL has been widely researched for applications in areas of decision-making and autonomous driving. The focus of many works is on the optimisation of RL efficiency through TL techniques, data sampling strategies and improving the experience replay technologies. The following section explores key contributions to this field, especially those relevant to leveraging criticality in RL to make TL more effective.

2.3.1. Prioritized Experience Replay

An implementation of the Experience Replay as a concept that stabilises and improves the training process in DQN introduced a crucial advancement in the efficiency of RL training. Experiences used, contain information about current status as well as action to be taken, a corresponding reward and the following state. Despite that some experiences are more valuable, in terms of information, than others, the traditional algorithm samples state from Replay Buffer randomly to break correlation between consecutive states.

Prioritised Experience Replay (PER) was introduced in 2016 with "Prioritised Experience Replay"[5] paper to replace the random sampling method with a more effective one. This research uses the Temporal-Difference (TD) error (calculated with Equation 5) to prioritise samples where the difference between estimated and real values was the highest. This gives the higher priority to samples with a higher learning potential, focusing on crucial transitions.

The benefit of PER was achieved due to the selective sampling used in RL, which depends on the importance of experiences. The following idea is related to the concept of criticality, which represents the significance of states for the overall performance. PER is focused on the prioritisation of experience based on TD error values, whereas the criticality concept extends this idea and identifies states which have the higher impact on the performance and safety of the model.

2.3.2. Criticality-Based Advice in Reinforcement Learning

In "Criticality-Based Advice in Reinforcement Learning"[4] paper, from 2022, authors explore the criticality application in RL, to be precise advising to the agent while training. The idea was to use criticality measures to identify the most significant states. Based on critical states advice was provided to the agent to develop a better decision-making algorithm.

The research went towards inequality of importance of different states, where if focused on critical states, the agent learned more efficiently avoiding costly mistakes. Applications where the consequences of wrong decisions can be severe, like autonomous driving, are the ones where this approach is highly relevant. Reducing the number of unsafe actions, criticality-based advice improves the performance of models significantly.

2.3.3. Other Related Research

Multiple research papers exist, exploring the exploitation of criticality in RL. For instance, the usage of criticality to establish trust in autonomous systems explored by Huang et al. (2018)[3]. They aimed to build more reliable RL models that can be achieved via the identification of critical states.

Related studies highlight the potential of criticality measures as an instrument in the improvement of safety and efficiency of RL. Whereas some works focus on the identification of criticality, the research in various application possibilities of this measure is still limited.

2.4. Literature Gap

RL has been successfully applied to solve complex decision-making challenges, it has proven to be a powerful tool, especially in the fields of robotics and autonomous driving. Despite that, other learning methods are being researched, due to the high time and power resources required for training. Efficiency, being the key criterion for this kind of algorithm, is the major drawback of RL. Because there are millions of steps in the training environment required to obtain a model, which will be able to perform well in complex tasks. This makes the standard method highly impractical in real-world scenarios.

To address this issue, TL has been introduced. The following technique trains the algorithm based on the pre-trained model, accelerating the training process for new environments. This learning approach utilises knowledge gained in one task to further apply it to another challenge while saving time and reducing the amount of training data for a new model. However, the corresponding method still relies on large and randomly sampled data sets to be fed into the pre-trained model imposing inefficiencies and considerable computational costs.

Based on existing insights into criticality and TL in general, this thesis aims to propose and verify an improved approach in TL: setting a focus exclusively on critical states while retraining the model in a new environment. This can, theoretically, decrease computational costs and the training time, while preserving or even boosting model performance.

3. Method

The methodology chapter outlines the approaches and tools used in this bachelor thesis. The primary objective of the study is to leverage the criticality in RL with the use of Deep Q-Networks (DQN) to obtain effective TL in conjunction with *Highway-Env* and *Merge-Env* environments. For development and implementation purposes of the RL agent, which will be capable of navigating and interacting with the given environment, the `Stable-Baselines3` framework was used. The visualisation was realised by the use of `matplotlib` and `seaborn` libraries.

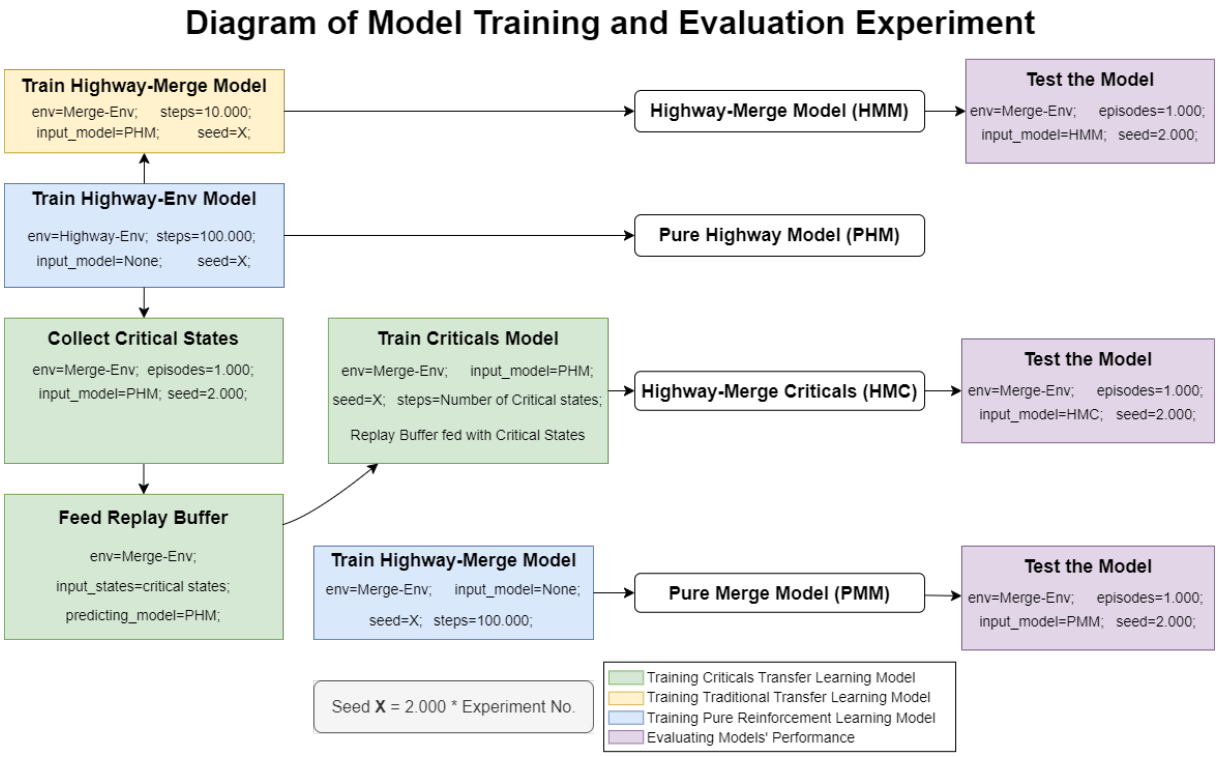

Figure 1: The diagram, representing the progress of every experiment including preparation, training and evaluation steps.

3.1. Environments: Highway-Env and Merge-Env

The operation of the RL agents in this thesis is realised within two environments: *Highway-Env* [6] and *Merge-Env*. Those are realistic yet controlled simulations of autonomous driving tasks, that provide a setting for evaluating RL algorithms in various driving scenarios. Both environments model traffic behaviour, including surrounding vehicles and multiple lanes, allowing the agent to navigate dynamically and interact with nearby traffic. The environment provides rewards depending on the RL agent's behaviour on a highway; the agent has to avoid crashes while optimizing speed and deciding about proper lane positioning.

The *Highway-Env* simulation environment is widely used for research in autonomous driving. Simulating a multi-lane highway encourages the agent vehicle to navigate in traffic, perform collision avoidance, and behave efficiently throughout driving. The following features are included in the environment:

- **Dynamic Traffic:** Realistic behaviour of neighbouring traffic, simulating lane changing behaviour and velocity adjustments;
- **Reward Structure:** Actions that maintain safe and efficient driving are rewarded, while dangerous and unsafe situations are penalized. The agent can receive a reward for the correct choice of lane, avoiding collisions and retaining a higher speed;
- **Observation:** One perceives the surrounding world through observations of the velocities, positions, and relative distances of surrounding cars.

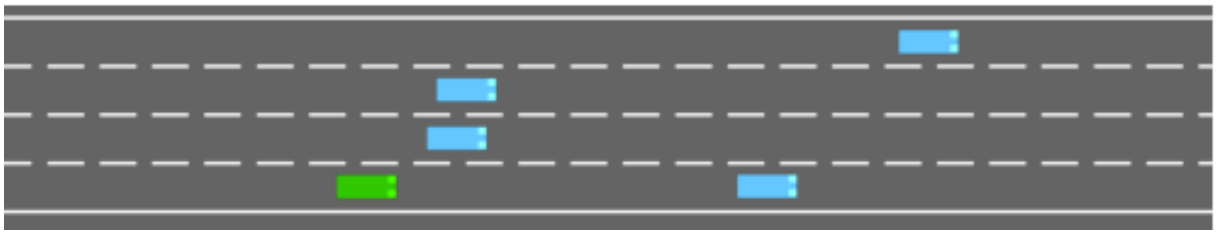

Figure 2: *Highway-Env* render.

To perform the TL and to challenge the agent in handling mode complex situations, the second environment was introduced – *Merge-Env*. This is a modified version of *Highway-Env*, which specializes in merging scenarios. In *Merge-Env* agent is targeted to navigate in merging situations, maintaining a smooth join of vehicles from on-ramps into main highway traffic. Performing timely decisions, velocity control and judging traffic gaps is what the agent desired to do in this environment to ensure effective, safe and smooth merging. Following additional complexity is added to this environment:

Merging Dynamics: On-ramp vehicles, that are trying to fuse into the highway, create an unpredictable traffic nature, forcing the agent to adapt quickly; Challenging Scenarios: Merging raises the complexity level of decision-making, as safety and efficiency have to be maintained while coordinating its actions with merging traffic; Enhanced Observations: Observations additionally include the velocity, position and relative distances of merging lane vehicles.

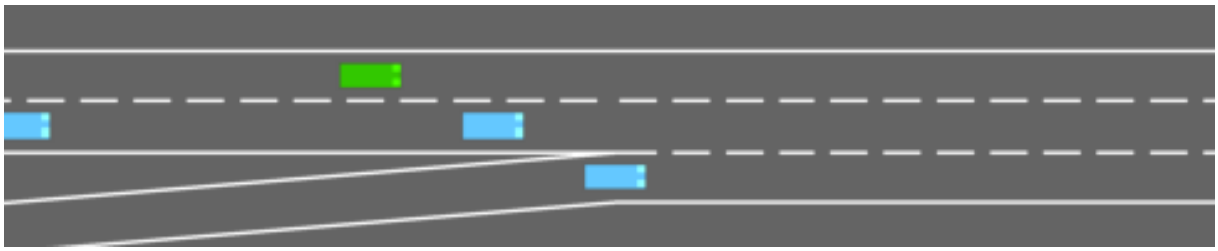

Figure 3: *Merge-Env* render.

Gymnasium API [7] is a standardised RL interface used for the implementation of both environments. It has a modular and flexible framework which allows compatibility with popular RL libraries, such as StableBaselines3. Thus, the seamless integration of ready-to-use algorithms is enabled, allowing high training and evaluation efficiency in the chosen environments.

A combination of two environments creates a testbed for the performance evaluation of RL agents, as for autonomous driving scenarios. By a combination of those environments, this thesis

aims to assess the effectiveness of TL concerning handling complex tasks, which contributes to the improvement of safety and efficiency in autonomous driving.

3.1.1. Action Space

Action Space defines possible actions to be considered in training, decision-making and evaluating processes. The *Highway-Env* and *Merge-Env* are implementations where the action space can be chosen to be either Discrete, Continuous, or Discrete Meta-Actions. The continuous action space in these environments means mainly that the velocity and steering angle of the agent can take arbitrary values in a predefined range. By controlling velocity and steering angle values, lane change, braking, overtaking, and other actions can be realized throughout the simulated traffic. However, the variety of actions makes the training and decision-making process more complex as the agent can leave the borders of the simulated highway.

The discrete action space is a quantized continuous space, which is represented as a uniformly distributed grid of possible values for throttle and steering angle. Despite the size of the action space being gradually decreased compared to the continuous space, there is a third, simplest controlling action space to be used for this thesis: the Discrete Meta-Actions space.

The Discrete Meta-Action space consists of 5 predefined actions which can replace the continuous action space. These actions are a tolerable replacement of continuous and discrete actions, which make the action space smaller and cover the most demanding actions of the agent. The available actions are [8]:

- `IDLE` – the velocity and steering angle are not adapted compared to the previous state;
- `LANE_LEFT` – the velocity value is saved unchanged, while the agent steers the vehicle to the left, changing the lane completely. In case the agent occupies the far-left lane, this action is not included in the possible action space; if the action is taken despite being unavailable, the `IDLE` action is performed;
- `LANE_RIGHT` – the velocity value is saved unchanged, while the agent steers the vehicle to the right, changing the lane completely. In case the agent occupies the far-right lane, this action is not included in the possible action space; if the action is taken despite being unavailable, the `IDLE` action is performed;
- `FASTER` – the steering angle remains unchanged, while velocity is increased;
- `SLOWER` – the steering angle remains unchanged, while velocity is decreased.

3.2. Stable-Baselines3

The `Stable-Baselines3` is a library that includes some of the most popular, efficient and reliable RL algorithms, that can be easily applied and used for different environment types like *Highway-Env* and *Merge-Env*. The list of implemented algorithms consists of Proximal Policy Optimization (PPO), Soft-Actor-Critic (SAC), and the algorithm that was used in this paper, Deep Q-Network (DQN). The privilege of the `Stable-Baselines3` is the simplicity of the Application Programming Interface (API), built on top of PyTorch, providing the ease of use of implemented models, requiring only a few lines of code to initialise, train and utilise the RL, which will be able to control the agent based on the learned policy [9]. Moreover, the developed API is not a black-box structure, but has open code, clear user guides and a huge

number of active users that published struggles and solutions online, which was crucial during the implication of criticality throughout this thesis paper.

3.3. Deep Q-Network (DQN)

The RL algorithm used is Deep Q-Network (DQN). This algorithm combines the properties of "reinforcement learning with a class of artificial neural networks" [10]. Following the combination of different RL approaches makes it possible for the DQN to handle environments with large state spaces, where the autonomous driving task is one example of such. The interaction with the environment is the main source of experience for the agent to learn an optimal policy. This algorithm was originally developed by Mnih et al. [10], where they introduced it as a method to address the RL challenges with high-dimensional state spaces through the use of deep learning, especially to estimate the Q-function throughout the learning process.

The core idea behind DQN is the process of the approximation of the Q-function based on the outputs of the deep neural network, giving an estimated reward based on the given current state and the action performed by the agent in this state. This algorithm performs self-education based on the gained experience and tries to maximize the cumulative rewards by making decisions and optimizing its actions. The Bellman equation is the one used to compute the value for state-action pair recursively through expected furture rewards:

$$Q(s,a) = \mathbb{E}\left[r + \gamma \max_{a'} Q(s',a') \mid s,a\right] \tag{4}$$

where:

- $Q(s,a)$ is the optimal action-value function,
- r is the immediate reward received after taking action a in state s,
- $\gamma \in [0,1]$ is the discount factor, controlling the importance of future rewards,
- s' is the next state,
- a' is the action taken in the next state,
- The expectation $\mathbb{E}$ represents averaging over possible next states and rewards.

The Q-values update is realised by Bellman equation with the help of deep neural network. The loss fulction used by DQN referes to TD error $\mathcal{L}(\theta)$:

$$\mathcal{L}(\theta) = \mathbb{E}\left[(y_t - Q(s,a;\theta))^2\right] \tag{5}$$

where the target value y_t is computed as:

$$y_t = r + \gamma \max_{a'} Q(s',a';\theta^-) \tag{6}$$

Here, θ represents current Q-network parameters, and θ^- corresponds to periodically updated target network. The periodical update tends to stabilize the training. The approximation of optimal Q-values is improved by the iterative minimisation of the loss function. This results into better decision-making process over time.

Additionally, there is a feature introduced with the DQN—the Replay Buffer, which played a crucial role in the implementation of the criticality in my thesis. The Replay Buffer offers the opportunity to store past experiences in the following form: firstly, the observations of the current state are saved; then the action that was taken by the policy based on the observation of the state; and finally, rewards that were obtained as a result of the action taken in the current state, followed by the observations of the resultant state. With the use of this feature, the experience can be not only saved but also replayed. This innovation assists during the process of approximation of the Q-values, mitigating the issues of overestimation of those, and helping agents to maintain effective and efficient learning in complex environments.

In this study, the DQN algorithm was used to train the agent in *Highway-Env* [6] and *Merge-Env* environments for it to be able to navigate the highway autonomously, maximizing rewards and minimizing penalties, for example, due to collisions. To optimize the training process and to reach better performance, some parameters of the DQN policy were tuned. For instance, hyperparameters, including the learning rate and exploration rate. The exploration rate shows the fraction of the actions where the exploration epsilon-greedy policy will be applied [11]. This means if the exploration rate is p (taking the values from 0 to 1), during $100 \cdot p$ per cent of the training steps (N), the following behaviour of decision-making will occur: in step 1, actions will be taken randomly in 100% of steps; further, up until step $N \cdot p$, the proportion of actions taken randomly, not depending on the learned policy, will decrease proportionally, and at the point of step $N \cdot p$ it will reach 0%. The efficacy of exploration (random actions) compared to exploitation (following the learned policy) is extremely high at the beginning of the training, where the policy does not act effectively, and exploration helps in gaining various experiences.

The learning rate is another key hyperparameter that determines the training performance of a Deep Q-learning (DQN) model. It defines how much the model parameters are changed during the update after each training step. In the DQN algorithm, the learning rate controls the magnitude of adjustments of the neural network weight, based on gradient descent. During training, the weight update is performed according to the following rule:

$$\theta \leftarrow \theta - \alpha \nabla_\theta \mathcal{L}(\theta) \tag{7}$$

where:

- α is the learning rate,
- $\mathcal{L}(\theta)$ is the loss function (e.g., mean squared error).

Training becomes unstable if the learning rate is too high (α is large) because the model weights change too much, which can lead to oscillations or divergence. If the learning rate is too low (α is small), the learning process becomes slow and the model can get stuck in local minima or take excessive time to reach an acceptable level of performance. In the context of the thesis, using a correct learning rate in the DQN algorithm allows the agent to effectively learn complex scenarios such as highway driving or merging manoeuvres, minimizing errors and providing high-quality results.

To train the model, the following mandatory programming steps should be completed:

1. **Initialise the environment:** Use the *Highway-Env* or *Merge-Env* for training purposes with the `gymnasium.make()` function from the Gymnasium library [7]. The `make()` function takes as input the name of the environment: `"highway-v0"` for *Highway-Env* and `"merge-v0"` for *Merge-Env*. To reduce training time, the faster variant `"highway-fast-v0"`

is used, which is a modified version of the original implementation with degraded simulation accuracy for better performance [6].

2. **Initialise the learning model:** Create the learning model to be used for training and assign it to the variable `model`. This thesis is based on the Deep Q-Learning (DQN) policy, implemented using the StableBaselines3 library [9]. The `DQN()` policy class has multiple adjustable parameters, but only three are considered relevant for this setup:
   - **Type of DQN policy:** Three options are available – `"MlpPolicy"`, `"CnnPolicy"`, and `"MultiInputPolicy"`. Use `"CnnPolicy"` for image inputs, `"MultiInputPolicy"` for dictionary inputs, and `"MlpPolicy"` (used here) for simple observations.
   - **Environment:** Use the previously initialised environment.
   - **Seed value:** Ensures reproducibility. Using the same seed value results in identical randomization sets.
3. **Start the training process:** Use the `model.learn()` function of the DQN class to begin training. This function requires the number of training steps as input. To track progress, use the `progress_bar` argument and set it to `True` to display the training progress.
4. **Save the trained model:** Use the `model.save()` function to save the trained model to a file. This allows the model to be saved once and later loaded with `model.load()` for further use, such as visualization or continued training (e.g., TL).

3.4. Transfer Learning

The base model for knowledge transfer was trained on the *Highway-Env*. While training the model, the agent was taught to:

- Maintain a safe distance from the vehicle in front,
- Assess the possibility of performing a lane change,
- Execute lane change maneuvers,
- Accelerate or decelerate to manoeuvre in traffic and receive corresponding rewards,
- Behave in traffic to maximize rewards under particular conditions.

The obtained model had "experience" on the highway but none in merge scenarios. Despite this, the model could make decisions based on previous knowledge, handle operations in pre- and post-merge scenarios at a good level, and avoid collisions during merging, whether by chance or through learned behaviour.

The target was, however, to obtain a policy that would operate the agent inside *Merge-Env* efficiently enough for the policy to be called safe. This could have been achieved via direct learning, following the same path as with the highway. While this approach would not require additional implementations, it would be as expensive and time-consuming as training the first model. Since this thesis uses environments that roughly simulate real-life scenarios, the scale of research into RL allowed experiments like training the model purely on *Merge-Env* for comparison purposes. In real-life cases, however, such experiments are impossible on a larger scale, which highlights the effectiveness of TL.

One approach is to use the pre-trained model for straightforward but less intensive learning. Instead of starting with an empty policy, the policy trained for the highway task is used as the

initial policy state for the training process in merging. Because RL recursively uses the policy's experience and knowledge, inputting the trained model into the DQN accelerates the model's learning process. Therefore, the second model was trained on *Merge-Env* using 10 times fewer steps than the initial highway model. This resulted in a model to be validated in future work.

Reducing the number of steps due to the "experienced" input model provides cost and time advantages. However, the training data used during DQN learning is still random. The approach explored throughout this thesis is to organize the training data more effectively. For this purpose, criticality principles will be used.

3.5. Criticality

The decision-making process is highly power-consuming for people because different actions taken in a particular situation could lead to drastically diverging results. An identical principle is to be considered for the decision-making performed by the machine. Exceptional meaning to wrongly taken actions should be taken into account during autonomous driving. Decisions taken by the autonomous vehicle are in the end to be brought to real road scenarios, where the lives of drivers, passengers and other traffic participants are of exceptional value. Thus, special attention is to be paid to possible dangerous situations that might occur in various driving scenarios.

Unfortunately, it is impossible to create a list of all potential situations that can be considered to be unsafe. However, there is one approach that can help the machine to identify if the state is dangerous, more precisely if any of the available actions in the given state can lead to a critical situation. The states where one or some of the possible actions lead to such situations are to be called critical state [12].

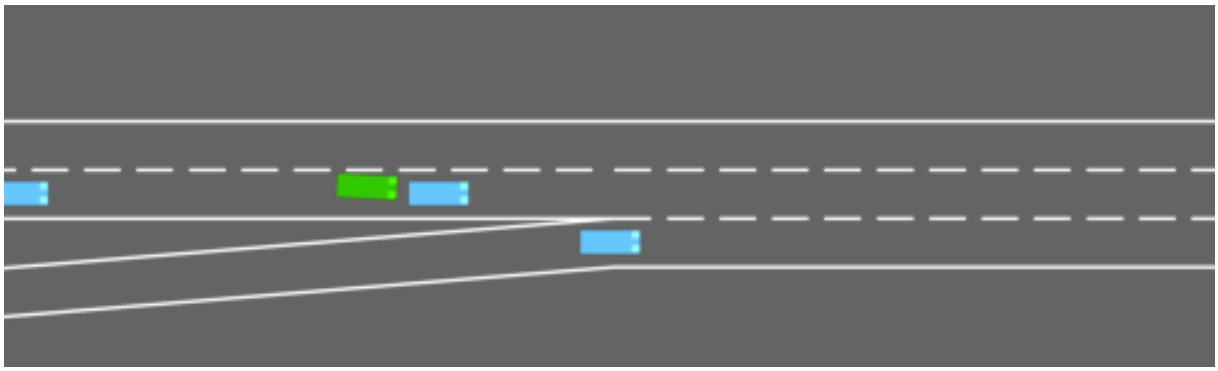

Figure 4: The state of a *Merge-Env*, which was identified as critical.

Figure 5 demonstrates the state that was identified as critical. According to the position of the agent vehicle (green), the safest action among all available, described in Section 3.1.1, will be `SLOWER`. The current velocity of the vehicle is unknown, therefore, the most meaningful action to take is decreasing velocity. Due to that one action favouring the decision-making, the value of criticality rose and the state was saved to be further used for leveraging criticality in TL.

The crucial role in decision-making is played by action values (Q-values). For a given state of the environment, each of the possible actions has its Q-value that was calculated by accumulating the possible prospected reward to be received if the action is taken. The calculations are performed according to the following formula [13]:

$$Q^{\pi}(s,a) = \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^{k} R_{t+k+1} \mid S_t = s, A_t = a, \pi\right] \tag{8}$$

where:

- $Q_{\pi}(s,a)$ is the action-value function, which represents the expected cumulative reward when starting in state s, taking action a, and following policy π.
- $\mathbb{E}_{\pi}[\cdot]$ denotes the expectation over all possible future trajectories, assuming actions are chosen according to policy π.
- $\sum_{k=0}^{\infty} \gamma^{k} R_{t+k+1}$ is the discounted sum of future rewards.
- $\gamma \in [0,1]$ is the discount factor, which determines the importance of future rewards.
- R_{t+k+1} is the reward received at time step $t+k+1$.
- S_t is the state at time step t.
- A_t is the action taken at time step t.
- π represents the policy used to select actions.

In DQN the approximation of the action-value for every action is performed by the neural network. Then the policy performs the action which has the highest Q-value, according to:

$$Q^{*}(s,a) = \max_{\pi} Q^{\pi}(s,a) \tag{9}$$

To leverage the criticality in TL there was a complex procedure performed. The base model, trained purely on the *Highway-Env*, was applied to perform "test drives" in the *Merge-Env* to collect necessary data for the future applications of TL. The most important data to be collected for criticality purposes was: observations of all the completed steps throughout episodes. The model can give the Q-net of the observation out, which represents the Q-values for each of the 5 possible actions(described in Section 3.1.1). Based on the maximum action-value among those in Q-net, as described earlier, the algorithm was making decisions about which action to perform for a given observation. However, to find critical states the variance of Q-values was taken into account(calculated via 3).

The purpose of calculating the variance is to find the deviation of values inside of a Q-net, mainly to identify if one of the action values was reasonably higher than all others. Higher variance represents that one of the actions was much more likely to be taken by the policy than others. The Q-net variances of all the observations recorded should have been further analysed. Obtained values were analysed to find and save every local maximum throughout episodes. Finding the local maxima can be realised in multiple ways, for example via the derivative value of a function, however, the simpler method is to compare two neighbouring values. This algorithm was implemented into the scipy.signal.find_peaks() function.

Scipy library [14] is an implementation of various scientific approaches with Python programming language. The subpackage `signal` focuses on various signal processing techniques and linear systems theory with a huge functionality including peak finding. The scipy.signal.find_peaks() function does the following: it traverses the input 1D array while comparing the current value with two neighbouring values one from the right and one from the left, in case the value is greater than both neighbours, the value represents a local peak, otherwise, an algorithm continues. As

an output, the function returns an array of peaks. To identify the observation that produced the following peak, the array of observations was traversed in parallel.

When observations of critical states are collected, the next goal is to use them for TL of the initial Highway policy. The ordinary DQN policy is trained in the way, that when the desired number of training epochs the algorithm uses the initial states of the system. Those are fed into the algorithm, and based on them the policy is trained. It either explores or exploits the initial state and the following states triggered by the action taken by the existing policy. Based on this experience the agent is trained to perform the decision-making in favour of one action in every given state, thus maximising rewards and, by that, the driving performance. However, the crucial point of initial state sampling for the training is that this is done randomly. Meaning every initial state of the system is randomly generated: the lane of the agent, the positions of surrounding vehicles, its velocities and the velocity of the agent car, everything is initialised randomly in every step.

The theory of this bachelor paper is, instead of a set of randomised states, to feed the DQN algorithm with a set of critical states of a smaller length and evaluate if an acceptable performance can be achieved via this method. The set of input states is generated by the DQN internally. This is performed by the Replay Buffer.

3.6. Replay Buffer

As critical states are collected, the next processing step is feeding this set into the algorithm for purposes of TL. The target is to control the input of the DQN by replacing random state samples with a list of critical states. This was realised with the use of an in-built component of DQN called Replay Buffer(or Experience Replay).

The whole idea behind the principle of experience replay is grounded in neuroscience. Observing and exploring rodents, scientists suggest that the brain replays sequences of prior experience during its sleeping and awake resting phases [15]. Based on this knowledge DQN was extended by the Replay Buffer, where the design is following: experience, completed once will not be erased after one step, but on the contrary will be stored inside of the Replay Buffer to be reapplied later [5]. Moreover, it becomes more probable for the rare-appearing experience samples to be learned multiple times. Utilising a mix of recent and older experience stabilises the training process.

Google DeepMind's "Prioritized Experience Replay" [5] research towards the optimisation of the usage of Replay Buffer in the DQN served as an inspiration for implementation within this thesis. The paper explores one possible improvement of a general Replay Buffer realisation, where the DQN will utilise the set of random states in a manner, where some states will be prioritized over others. The purpose of the research is brilliantly described in one sentence: "This paper addresses only the latter: making the most effective use of the replay memory for learning, assuming that its contents are outside of our control." After digging into the prioritised replay approach, the concept of overwriting the Replay Buffer data came to mind. Generally, the intention, that was pursued during the development of the Replay Buffer, was to reduce the amount of experience required for effective learning.

A list of observations is not all information, which is required to create a working buffer. Additionally, actions taken under the conditions of the state, the received reward and the resultant observations are required. To obtain the following data, the pre-trained highway model was used, which is the same model that collected critical states. Before the start of the transfer

model training process, the experience buffer is filled with all necessary values and connected to a system to be employed by the DQN algorithm.

4. Experiments, Results and Discussion

The practical part of the bachelor diploma consisted of a technique development to find the set of critical states experimentally and then apply the criticality knowledge to the DQN policy via the Replay Buffer. Besides that, there are verification and validation parts left. To ensure that the research and implementation can give valuable results and that criticality can assist in the TL process, the assessment of the effectiveness and performance of the obtained models should be conducted. The evaluation process and results will be discussed throughout this chapter.

Generalising results, the criticality approach in TL achieves an average episode duration (Figure 8) close to the one of traditional TL, outperforming in half of the experiments but showing lower stability. Comparison of general rewards (Figure 9) demonstrates a similar behaviour of the proposed TL method: the average value is close to standard TL, showing a higher value in some experiments, while, in general, having a higher range of rewards. The indicator of collision rate (Figure 7) reflects that the approach requires further development. All in all, the reduction in the number of training steps (Figure 6) and, thus, in training time will become much more noticeable on a larger scale and in more complex environments, which indicates the importance of exploration into the criticality in TL.

4.1. Models

4.1.1. Model Type 1 `"Pure Highway Model"`

The first model type is the base model for TL and the search engine for critical states. This was trained by applying the algorithm described in Section 3.3 to the *Highway-Env* environment. The number of steps was 100000, a value determined experimentally. Initially, a model trained for 1000 steps exhibited poor performance.

To find the number of steps required for a model to achieve a stable, better performance without requiring excessive computational power and training time, several experiments were conducted:

1. **Training with Different Steps:** Multiple models were trained using varying numbers of steps: 1000, 10000, 50000, and 100000. These ranges cover various stages of training and illustrate the impact of increasing the number of steps.
2. **Training Setup:** The training environment was set up, and each model was trained according to the steps defined in Section 3.3 of this thesis. During training, the time elapsed for each model was recorded, which became a key factor in determining the optimal number of steps.
3. **Performance Evaluation:** The trained models were evaluated by running each through 100 episodes and recording key metrics, including the average episode time, cumulative reward, and collision rate.
4. **Results Analysis:** The results were analyzed by plotting the recorded metrics against the number of steps, providing a clear visualization of performance improvements as the number of steps increased.

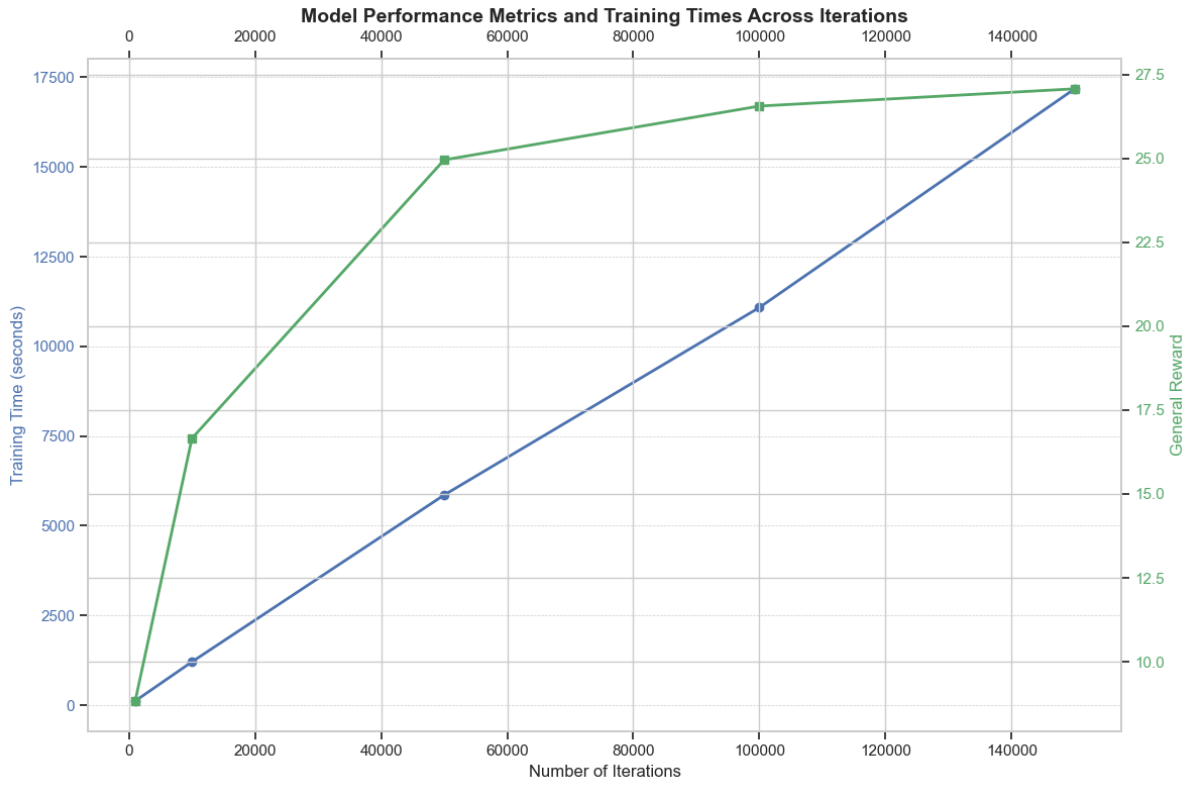

Figure 5: Performance metrics and training times for models trained with different step counts.

Steps	Avg Episode Duration (s)	Collision Rate (%)	Training Time (s)
1,000	0.395	99.00	116
10,000	0.729	80.00	1,207
50,000	1.070	20.00	5,857
100,000	2.254	11.00	11,083
150,000	2.257	11.00	17,178

Table 1: Performance metrics after 1000 testing runs across models with various steps count.

Gathering results while performing Step 4 of this experiment, the data represented in Figure 5 and Table 1 was obtained. Training time, as expected, follows the proportional rise. An increase of step count by 1.5 from value 100000 to 150000, results in the training time being 1.55 times longer from 11083s to 17178s respectively, which satisfies proportionality manner under reasonable uncertainty due to the inconsistency of a computational unit.

Observing the collision rate, it does not show the behaviour of any simple function, however, its general actions can be described as an unevenly decaying function. While the steps count grows tenfold from thousand to ten thousand, the collision rate goes slightly down by 19 per cent, whereas a latter fivefold jump triggers a rapid fall of the dependent variable by 60%.

The last two measurements do not show the gradual change of collision rate, meaning the most important drop of this value is to be observed on the steps ranging from 10 to 50 thousand.

Regarding average time and general reward values, they show a similar dependency on step number, the considerable rise of results up to 50000 for general reward and up to 100000 for average episode time. Afterwards, both values tend to slow down the growth significantly.

Considering all captured performance measures, the step count of 100000 appears optimal. It is a checkpoint of gradual performance improvements of the model, while higher values have a much weaker impact.

4.1.2. Model Type 2 "Highway-Merge Model"

The second model is used for performance comparison between TL models. The goal of training this model was to assess the effectiveness of the criticality method application. The learning process of this model implied principles of TL.

Firstly, the *Merge-Env* environment was initialized, as the further training procedure would be conducted there. Afterwards, the pre-trained `Pure Highway Model` was loaded using the `model.load()` function. In a standard DQN initialization, random values are assigned to all weights inside the DQN. However, this time, the model was not initialized from scratch, but the weight values from `Pure Highway Model` were used as input to the training process of the DQN algorithm. Non-random values represent the knowledge base of the input model, which was acquired from the *Highway-Env*.

Further, the model was trained in the *Merge-Env* with 10000 steps. The purpose of this was to feed the experience of the merge environment into a model that already possessed a certain performance level in highway scenarios. This realized the simplest TL method, and thus the model will be later assessed together with the model obtained using the criticality approach.

4.1.3. Model Type 3 "Highway-Merge Criticals"

The third model type is the key algorithm to be explored in this thesis — the Criticality TL Model. This is the main experimental model to be further evaluated and assessed to determine if the criticality approach in TL is an effective strategy to be applied. The latter part will describe the steps by which this model was obtained:

Implementation of training has one major difference, in comparison to Model Type 2, the manual population of the Replay Buffer with critical states. All steps before the training part are identical to the model described above, whereas there is an additional step to be performed afterwards. Before the learning process is started, we need to fill the Replay Buffer with the critical states that were found, to ensure that exceptionally this set of states will be used throughout the training process.

From the criticality analysis procedure described in Section 3.5, there was a set of observations of states found, which corresponded to global and local peak values of criticality measurement. For the Replay Buffer, it is required to have not only a starting observation of a state, but also an action to be performed based on this observation, the reward received when the action is performed, the consequent observation obtained and the boolean value if the state, following critical, contains collision of the agent vehicle, or not. To determine the action, next observation, reward and collision the loaded pre-trained highway model was used. When inputting the critical observation into the model, it performs a decision-making step, based on the learned Q-Net.

Following the described procedure, the model was obtained by applying TL based on an isolated set of critical states to the pre-trained highway model. The next step is to assess the performance quality of the resultant model and prove that the received results are stable and reproducible.

4.1.4. Model Type 4 "Pure Merge Model"

Compared with the fourth model, the general effectiveness of the TL applied in Model Types 2 and 3 can be evaluated. This model is trained similarly to highway model number 1, going through 100000 steps in a single environment, however instead of *Highway-Env*, the *Merge-Env* was taken. The purpose of training the corresponding model is to have reference measurements of the non-TL model to identify the level achieved by the Model of Type 3.

The training algorithm was borrowed from the one used in `Pure Highway Model`(described in Section 3.3), with a minor change: the *Merge-Env* environment was applied. The value for steps count was also chosen to be 100000, taking results obtained in Section 4.1.1 into account.

4.2. Performance metrics

The key to performance assessment is the choice of metrics to be collected during the evaluation of models. The right set of resultant data will enable a deeper analysis of performance. Moreover, the combination of various outputs can show unexpected dependencies and valuable trends.

Three identical sets of 1000 episodes were offered to Model Types 2, 3 and 4 as an environment to perform a "test drive" and to measure the effectiveness of algorithms. Every model ran through 1000 episodes of merge environment, and the following data was captured and saved for later analysis:

4.2.1. Rewards

While driving, the *Merge-Env* accesses the driving behaviour of the agent vehicle according to multiple criteria and, based on this, rewards the vehicle. Reward values are combined into a dictionary with keys set to reward categories and values being lists of reward points received. There are 6 categories to be accessed in the *Merge-Env*:

- **Collision reward** (`"collision_reward"`) – a penalty to be received by the vehicle in case of a collision. The purpose of the collision reward is to reduce points, encouraging an algorithm to prioritize safety by avoiding hazardous situations.
- **Right lane reward** (`"right_lane_reward"`) – the reward is received when the agent is continuously driving in the far-right driving lane.
- **High-speed reward** (`"high_speed_reward"`) – when the agent drives while maintaining a high speed, the high-speed reward is granted.
- **Merging speed reward** (`"merging_speed_reward"`) – this is an altruistic reward granted to the ego vehicle depending on the velocities of all vehicles except for the merging vehicle. The formula is [6]:

$$R_{\text{merge}} = \sum_{i \in \mathcal{V}_{\text{merge}}} \frac{v_i^{\text{target}} - v_i}{v_i^{\text{target}}}, \tag{10}$$

Where $\mathcal{V}_{\text{merge}}$ is all vehicles except for the one driving in the merging lane, v_i^{target} is set to 30 in the *Merge-Env* implementation, and v_i is the velocity of the chosen vehicle. The reward is named altruistic because it examines the velocities of all vehicles, encouraging cooperative behaviour, where the agent synchronizes its speed with neighbouring traffic to ensure the smoothness of a merging scenario.

- **Lane change reward** (`"lane_change_reward"`) – a reward to be granted when a lane change action is performed, either `LANE_RIGHT` or `LANE_LEFT` action.
- **General reward** (`"general_reward"`) – a combined value of all rewards and penalties received during one action step of the agent. The value is received as an output of the function `env.step(action)`. The value is summed throughout all action steps inside a single episode.

4.2.2. Collisions

The boolean value represents if the episode was terminated due to a collision. Keeping track of these metrics will help in identifying the collision rate of the ego vehicle. Calculations of the collision rate are performed with the use of the following formula:

$$\text{Collision Rate} = \frac{N_{\text{collision}}}{N_{\text{total}}}, \tag{11}$$

where $N_{\text{collision}}$ is the number of episodes which ended with a collision of agent vehicle, is divided by N_{total} total number of episodes(1000).

4.2.3. Episode duration

The duration of an episode is the measure to collect information about the driving time of the agent. Throughout the implementation, episode duration is collected in the following way: at the beginning of an episode, the current timestamp is saved to a variable using `time.time()` function, later when the episode is finished, the starting time is subtracted from the current timestamp at the end of an episode, resulting into the episode duration.

4.2.4. Criticality values

Additionally, the array of criticality values(variance of Q-values) is collected during the examination of models. Sets are obtained via the algorithm described in Section 3.5 of this thesis. The purpose of this metric is to observe not only the performance of models but also the distribution of state criticality throughout episodes. The hypothesis is that the model trained with the Replay Buffer of critical states should have fewer states when the criticality value is high. This will be explored via the criticality distribution plot further.

4.3. Reproducibility test

Approaching the measurement of the reliability of the criticality method, along with its performance, it was decided to conduct 10 experiments. The number of tests was taken to be 10 due to the duration of every experiment. All in all, each experiment consists of multiple steps, with the final point being to train the model based on the critical observations. The steps preceding the last one were performed to prepare the criticality data for the target model;

and obtain models that will be included in the final assessment procedure of the experimental algorithm. As for the steps of a single experiment:

1. Train the model, based purely on *Highway-Env* with the steps count equal to 100000. This model was obtained following steps explained in Section 3.3 of this thesis;
2. Apply the TL to the pre-trained *Highway-Env* model. The initial model from Step 1 is fed with 10000 steps of *Merge-Env* to gain experience of a new environment with additional complexity in the face of merging lanes and vehicles. The TL process used during this step will later be upgraded by isolating the training data sampling process to only critical states;
3. Test the model obtained in Step 2 through 1000 episodes of *Merge-Env*, collecting the metrics listed in Section 4.2. Gathered performance data will further be used in comparison purposes while estimating the resultant effectiveness of `Highway-Merge Criticals` model;
4. Prepare the set of critical state observations by running the Step 1 *Highway-Env* model for 1000 episodes of *Merge-Env*. The collected testing data is similar for every evaluation process;
5. Train the experimental TL model based on the adjusted Replay Buffer. The used Experience Replay Buffer was populated purely with critical state observations gathered in Step 4, and additional information about the action, reward, and following state observations, was obtained via the decision-making process of the *Highway-Env* model from Step 1. The result of this step is the goal model of this paper;
6. Rate the `Highway-Merge Criticals` model through 1000 episodes of *Merge-Env*, saving all obtained results;
7. Train a reference *Merge-Env* model without applying TL—going through 100000 steps learning process inside *Merge-Env*;
8. Compile the assessment results for the reference model in Step 7 with the evaluating process analogous to the ones described in Steps 3, 4, and 6.

After all necessary measurements for every obtained model are gathered and saved, it is time to visualise the data and, based on this, draw conclusions. There are various ways to represent the data in a clear and accessible way, but I tried to find the most crucial formats and describe them further.

4.4. Discussion of results

This part focuses on explaining results gathered throughout conducted experiments and discussing observations and conclusions to be derived from visualised results.

4.4.1. Number of critical observations

Each of the 10 experiments had a different seed number to control obtained models. The randomness value was kept fixed throughout the experiment and applied only while training. However, the evaluation of all models was conducted on the set of 1000 environment states where seed numbers were spread between numbers 2000 and 2999. The seed number enables the reproducibility of experiments and creates different starting conditions while performing every test. Due to the variability of initial conditions during training of *Highway-Env* model, a different

number of critical observations was produced by Step 4 of the algorithm, described in Section 4.3.

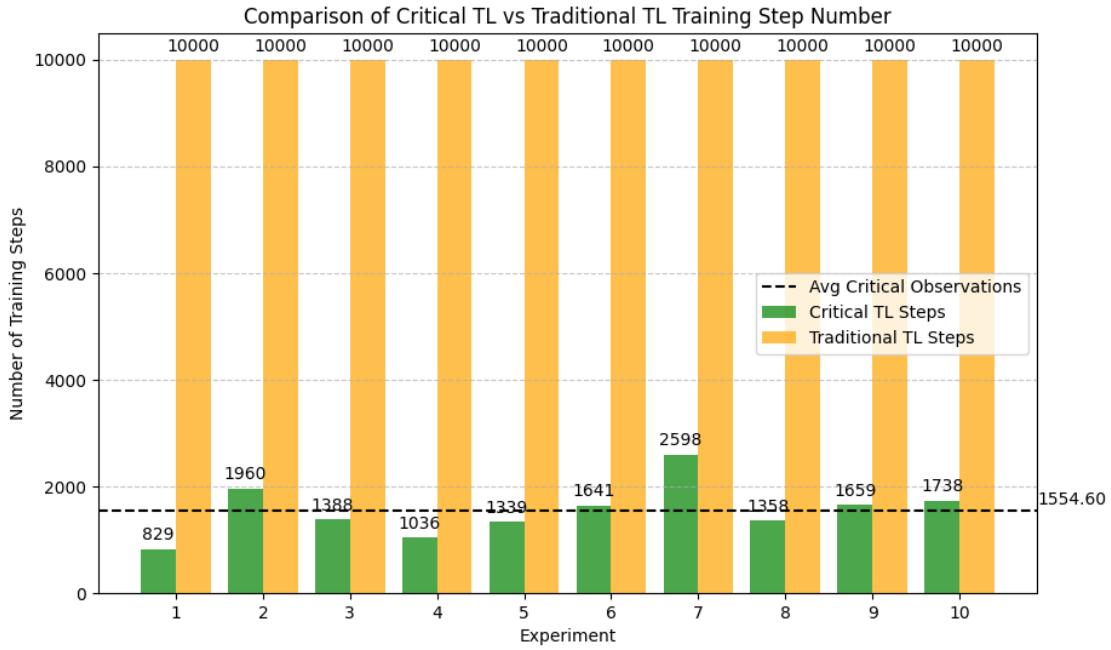

Figure 6: The distribution of the number of critical states collected and used for training in each experiment

Figure 6 visualises the training steps count for two TL models during each experiment. Moreover, blue bars represent the number of critical states(identification method is explained in Section 3.5) that were fed into the Replay Buffer(Step 5 in Section 4.3) while training the `Highway-Merge Criticals` model in every test step. It can be seen that by changing the seed number a variety of training data for the target model was produced. This made it possible to search for interconnections between the number of critical states and the performance of the obtained model.

As the step number for traditional TL was fixed to 10000 and the number of critical states found differs from test to test, the time gain is also variable. However, taking the mean value of critical states, the training of TL with a criticality approach was 6.4 times faster than that of the traditional TL algorithm.

However, the comparison of the models' training time was not fully accurate. This limitation arises from the fixed number of training steps for traditional TL. There the optimal performance could have been achieved in fewer steps. Due to that, the time reduction between the two TL approaches cannot be accessed objectively. The examination of learning curves of trained models can be helpful in analysing the training time of different methods. By evaluation of the model evolving throughout the training, deeper insights can be achieved.

4.4.2. Collision rate

When addressing the performance and reliability of dynamic environments, such as autonomous driving, models the collision rate tends to be a crucial metric. Given the proportion of episodes where collisions occur, it provides a fair measure of safety. Through the analysis of collision rates across various model types and different experiment conditions, different insights into model behaviour can be revealed. For instance, the effectiveness of TL, or the effect of the application of critical states on the decision-making process.

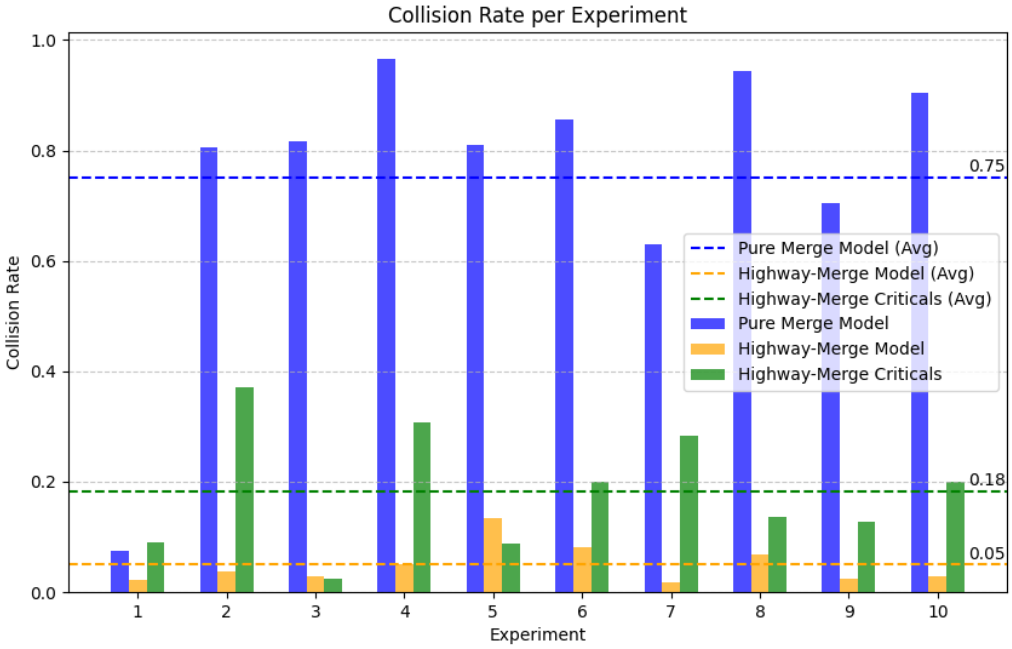

Figure 7: Agent's collision rate per experiment for each model.

Throughout 10 experiments, the collision boolean value was tracked. It takes a True value if the episode finished due to an accident, and a False value otherwise. Taking into account booleans from all 1000 episodes of each experiment, the plot in Figure 7 was created. A single point on a graph represents a collision rate value, calculated using a formula described in Section 4.2.2. The x-axis of a plot defines the experiment step number and the y-axis represents collision rate values, which occupy a continuous interval from 0.0 to 1.0.

In Figure 7, there are relatively low collision rate values to be observed for all models, including some fluctuations around the average value and outliers. In general, the concentration of values is to be found between 0.0 and 0.35, as for average values, an even smaller range is to be occupied.

`Pure Merge Model` model shows not only the most unstable behaviour among all but also has an extremely high level of unsafety. Owning an average collision rate of 0.75, it shows the worst performance. On average approximately 750 episodes out of a total number of 1000 ended up with the crash. Moreover, among all tests, there were 3 where the accident rate rose to the level of 95%, meaning that in almost 1/3 of conducted experiments, the probability of a car

finishing the episode safe was only 5%. If, theoretically, this model was implemented inside of a real autonomous vehicle, it would have been extremely dangerous to use following cars.

Despite generally terrible performance metrics, there is an outlier to be noticed in the graph of `Pure Merge Model`. In the first evaluation task, the model shows a collision rate of only 0.1 as an average value of 1000 runs. Such a huge drop is to be explored further with other plots.

Generalising the behaviour of a pure RL algorithm, the higher average collision rate of the model, if compared to others, reflects its limitations while being trained solely on the *Merge-Env*. It lacks the variability of experience and abilities to deal with complex merging scenarios, making it highly prone to collisions.

The lowest average collision rate of 0.05 is achieved by the `Highway-Merge Model`. Based only on the collision rate, its performance is stable and safe. Comparing the number of input states, used while performing TL, it can be seen how 10000 random states, used for this model, outperform the lower number of total states used in `Highway-Merge Criticals`. The orange graph remains stable in the scope of most test drives, although having two moderate peaks during experiments 5 and 7. Nevertheless, the rise of jumps concerning an overall average of 0.05, is only 0.1 and 0.03 respectively.

Focusing on the `Highway-Merge Criticals` model, it shows middle values among other trendlines. Compared with `Highway-Merge Model`, it has bigger deviations from the mean value, behaving astatically. During experiments 3 and 5, it achieved a collision rate lower than `Highway-Merge Model`, while experiments number 4, 8 and 10 had more collisions than others. There can be 3 peaks of collision rate observed in the 2nd, 4th and 7th tests, which indicates a lower safety level of driving as approximately every 3rd driving episode ended with an accident. Referring to the Figure 6 data, the mean number of collected critical states can be calculated, it is 1554.6. While utilising, on average, a 6.4 times smaller set of TL input states, `Highway-Merge Criticals` model has a mean collision rate of only 3.6 times higher than the one of `Highway-Merge Model`. This comparison indicates potential improvements that can be achieved through the use of the criticality method within the scope of TL.

4.4.3. Average episode duration

An additional metric that represents the reliability of the autonomous driving agent is the average time of an episode. The duration of an episode has a direct connection to the safety of an algorithm.

Observing Figure 8 and taking into account the values range of the average episode time, it is worth saying that the time measure is relative. The number of seconds to be captured from the beginning to the end of the episode takes values of within 1 second due to the rendering speed of the *Highway-Env* environment, this time cannot be referenced as a real-time measure.

Analysis of the general behaviour of three trendlines, presented in Figure 8, shows the pure superiority of the TL approach over the basic RL, applied in the `Pure Merge Model`. Here, average values for TL models differ only by 0.03 seconds, whereas the third model owns a mean episode duration value of 0.52s after 10 experiments. `Pure Merge Model` shows an incredible performance instability with the interval of values of 0.85 seconds and 0.35 seconds.

The first test drive is to be considered exceptional. This is because the data gathered for experiment 1 does not follow general trendlines in either of the three metrics that have already

been explored. This test shows an overall average episode time taking the highest values among all experiments for all three models, without exceptions. Additionally, the accident rate for every model is relatively low, especially the `Pure Merge Model` value demanding attention. The collision rate of an unbelievable 10% is a suspiciously tiny number if compared with a standard range of values from 65% at the lowest point and 95% at the highest. Notwithstanding, the critical observations count is the smallest among 10 experiments, only of 829 samples. Thus episode 1 is to be excluded from the further discussion of general metrics' behavior.

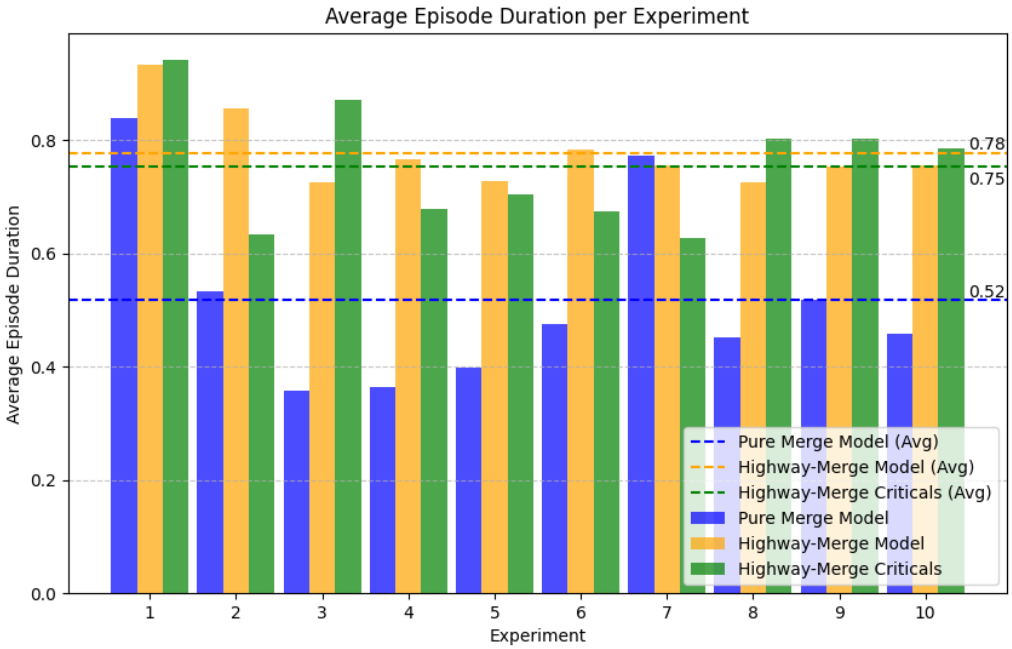

Figure 8: The plot of an average episode duration per experiment for every model.

Through analysis of the abnormal behaviour of a model `Pure Merge Model` within the first experiment, one theory was proposed. The following results could only be obtained if the set of randomized initial conditions for the experiment was created in such a way that the model was able to perform much better decision-making, and thus result in a higher value of episode time and a lower collision rate. Based on those two metrics the resultant number of critical observations was reasonably smaller.

The best result of a blue graph(if not taking episode 1 into account) was reached during episode number 7. Taking the data from Figures 6 and 7 into account, in the 7th episode `Pure Merge Model` shows a local peak value of average experiment time of 0.77 seconds while having the best collision rate and collecting the most number of critical observations of 2598.

There is a strong logical and quantitative correlation between those 3 metrics to be explored via the 7th task. As the agent was able to avoid accidents in approximately 37% of episodes, the mean runtime value increased up to 0.77s. The combination of longer episode time and lower collision rate boosted the number of critical states that were found during this run. The increase of critical states number from a medium value of approximately 1500 observations rocketed up to almost 2600 observations, which is a total of 173,3% from a mean value. Similar behaviour is also visible in episodes 2 and 9, albeit on a smaller scale.

As of `Highway-Merge Model`, average episode times follow a similar pattern if compared to the collision rate graph. It remains relatively stable around an average value of 0.78, with some fluctuations occurring in a range of 0.05 seconds more or less.

Exploring results of experiments where `Highway-Merge Criticals` was controlling the agent vehicle interesting observations were found. Even though the average value is not of big different from the second TL model, the graph is not as smooth as in the other model. There is one global peak to be seen in experiment 3 and two experiments where the value hit the lowest point(experiments number 2 and 7). Both dips of a graph in tests 2 and 7 are accompanied by peaks of a collision rate. Whereas, during the peak value of 0.97s run in experiment 3, the accident rate value drops to the lowest point of the graph of only 3%, which corresponds to the generalised interconnection of those metrics described above.

Based on the driven conclusion to exclude episode number 1 from the overall statistics, new average values were calculated(analogically to dashed lines, denoted on Figures 8 and 7). The Table 2 defines newly calculated values:

Table 2: Overall Averages for every model (Excluding Experiment 1)

Model	**Avg Collision Rate**	**Avg Episode Time**
Merge 100000 steps	0.8264	0.48
Highway-Merge Model	0.0524	0.76
Highway-Merge Criticals	0.1931	0.73

Average values for accident rates of models which employed TL, if excluding the first experiment, changed by a mean of 5%, whereas for `Pure Merge Model` this value rose by 10% up to 0.83. Analysing the consequences of the experiment number diminishing for the average episode time graph, the following results appeared: all values went down, however, the variation of the pure RL model was twice as big as other models, and time dropped down by 0.04s, with others only experiencing a decline of 0.02 seconds. In total, the statistics have not changed much, when removing the outlier.

4.4.4. General rewards

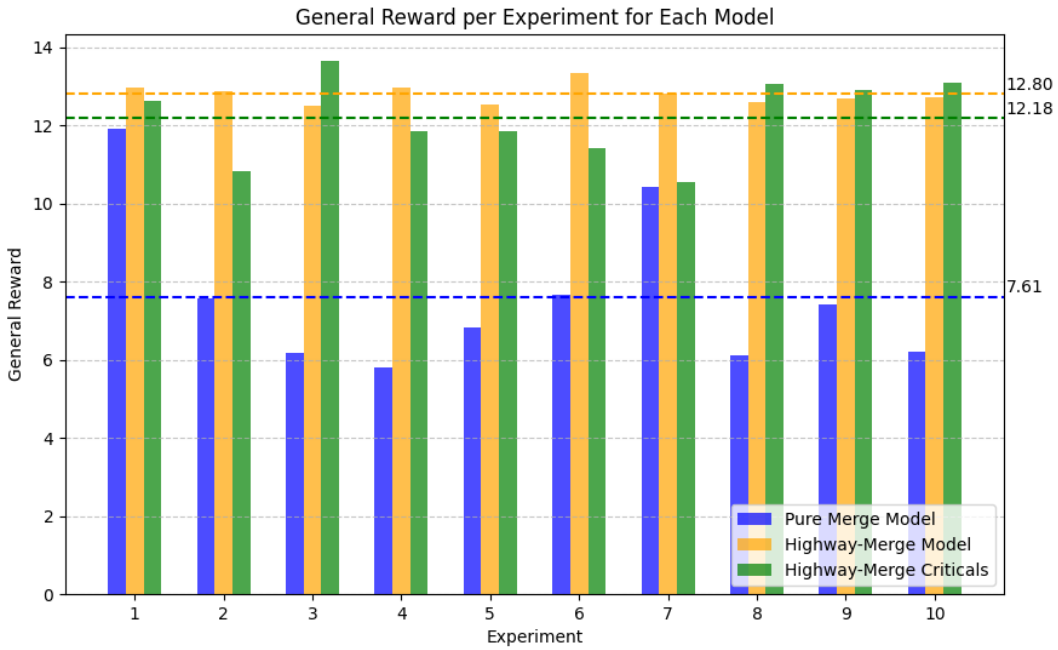

Figure 9: The plot of an average general reward per experiment for every model.

The system of rewards is a crucial component of RL's idea. In unsupervised learning, in complex and dynamic environments where the set of all possible outcomes is not possible to simulate and even more so to calculate the most optimal route in advance, the Q-learning strategy is to be applied. There, Q-values are calculated with the use of possible prospective reward to be received by the agent if a particular action was taken, as written in Equation 8.

Figure 9 represents one of the multiple types of collected rewards (as described in Section 4.2.1) the general reward. If comparing Figures 8 and 9, a striking resemblance of those graphs can be seen. The behaviour of graph lines for every model shows similar patterns with small deviations and scaling differences. As explained above, the reward value has a direct influence on the actions taken throughout the decision-making process, and thus on the results of time measurements for 1000 episodes of every test. Peaking values of average time correspond to the peaking result of a general reward, and vice versa.

4.4.5. Overall Rewards Comparison by Reward Types

For statistics purposes and as a chance to understand the behaviour of different types of models there were data about all possible types of rewards collected. Measurements of all 6 types of received rewards, described in Section 4.2.1, were separately connected. Figure 10 represents average values of rewards among 10 experiments by model and reward type.

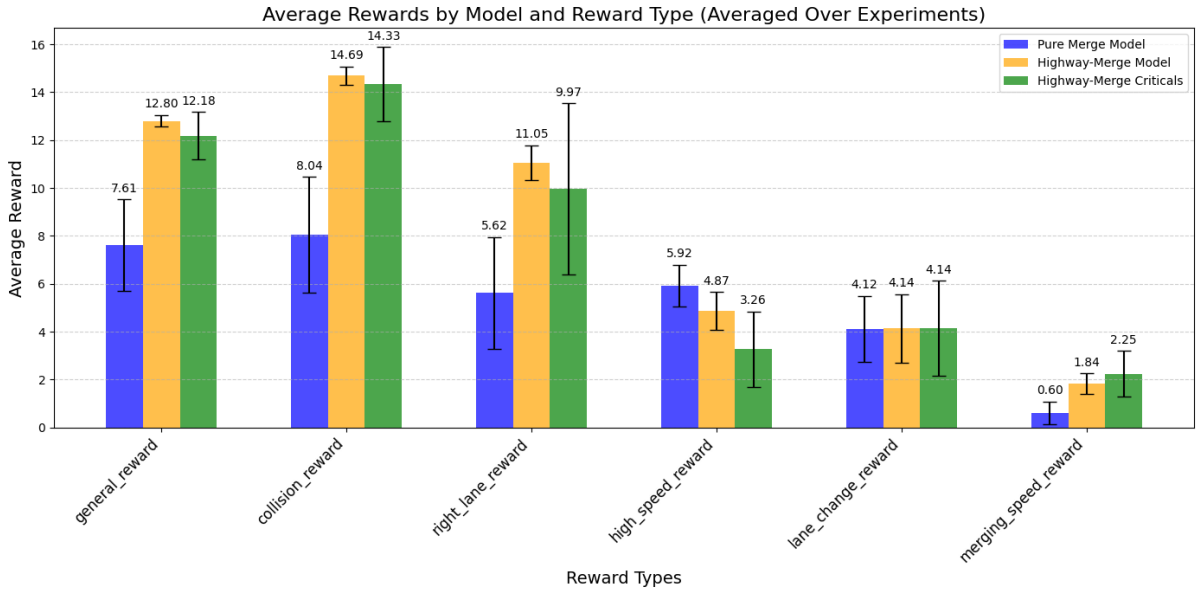

Figure 10: Error bar plot of rewards, averaged over 10 experiments, by model and reward type.

The general reward value was already explored deeper in Section 4.4.4, however, this reward type defines only the total performance of the model. Whereas other types help find out what is the direction of decision-making that was applied throughout the tests.

Analysis of the collision reward should be conducted relying on the collision rate graph, too. If compared with the collision rate, collision reward gives reasonable results. The highest average collision reward is owned by the `Highway-Merge Model`, due to the lowest mean value of a collision rate depicted on Figure 7 of only 0.05. The second highest value of collision reward, with a difference of only 0.34, has `Highway-Merge Criticals` model. This matches with the data obtained for the accident rate, where TL models showed better performance than the `Pure Merge Model`. The reward value obtained by `Pure Merge Model` due to accidents managing abilities of a model is almost twice as low as ones of other models. A similar tendency is also plotted on the collision rate graph, which demonstrates the inability of the pure RL model to drive safely, omitting collisions.

Going further to the right lane reward does not change the overall situation entirely, the ranking of models remains unchanged. The blue bar has the worst result of 5.62, showing that this model was prone to take the middle of the leftmost lane during the episode. At the same time, the other two models perform a lot better at right-lane keeping throughout experiments. For better performance, the results of 9.97 and 11.05 were granted to models `Highway-Merge Criticals` and `Highway-Merge Model` respectively.

Points, with which the model is rewarded when driving with a high speed, show an exceptional distribution among models. The first and the only time `Pure Merge Model` showing the best result among all. 5.92 is the high-speed reward owned by this model, whereas other models received a lower reward by driving slower.

Direct connection between collision and higher speed can be observed if taking into account the lane change reward. Showing almost identical values for this reward, with a small difference of only 0.02, models decided to change the lane on average an equal number of times. As a lane change reward is received if lane change action was taken(additionally described in Section 4.2.1),

it does not consider the success of this action, thus this can be observed by observing collision and high-speed rewards. Among an equal number of lane change attempts, considering driving with the highest speed of all models, `Pure Merge Model` succeeded almost twice as rare as others. This means that a trained policy of `Pure Merge Model` prioritised driving faster than avoiding collisions with surrounding vehicles. If digging into the policy of `Highway-Merge Criticals` model, it can be seen, that being trained on a smaller set of critical states than `Highway-Merge Model`, it learned to retain a lower speed to avoid accidents effectively. By following this strategy it almost succeeded in receiving a collision reward as high as one of the second models.

The last reward metric to be explored is the one for merging speed(the formula used to calculate the value of this reward is described in Section 4.2.1). The best value for this reward was achieved by `Highway-Merge Criticals` model, indicating the best cooperative behaviour of agent vehicles among all. Second and third places are taken by `Highway-Merge Model` and `Pure Merge Model` respectively.

If looking at the standard deviation of values, the following observations can be seen. Throughout almost all values the upper boundary of the confidence level of `Highway-Merge Criticals`, the average value plus standard deviation of the value, reaching higher values than ones of `Highway-Merge Model`. This behaviour can be seen for all metrics except for high-speed reward. The higher overall standard deviation of rewards of `Highway-Merge Criticals` defines the lower stability of this model. For instance, the deviation of results for a general reward of the research model is noticeably higher than ones of `Highway-Merge Model`; this shows that despite that on average the reward is lower, the performance of the critical model is better in some experiments, which can be observed on Figure 9. Similar behaviour can be observed with collision reward when looking at values of this metric through single experiments on Figure 7.

5. Conclusion

In the present chapter, the results, accomplishments and some shortcomings of the present thesis are summarized in Section 5.1. Some ideas for future research work are highlighted in Section 5.2.

5.1. Summary

This bachelor's thesis explores leveraging criticality in RL to improve TL in the field of autonomous driving. The research aimed to identify whether a transfer to a new environment can be efficiently realised through training a model on critical states of a target environment. There were four model types developed and evaluated in the *Highway-Env* and *Merge-Env* to assess the impact of TL, the choice of a set of critical states, and the training methodology.

In the beginning, global metrics underwent the examination. Metrics included: collision rate, average episode time and general reward. Through that, the basic knowledge about the model's overall performance level and functionality was obtained, including the ability to handle complex scenarios. The study found that those models, which were obtained through leveraging the TL approach, outperformed models, trained in a single environment, consistently. This concerns both stability and safety. The final discovery is, that the efficienty of proposed hypothesis–leveraging criticality for effective TL–was confirmed experimentally. The consistency of results in global metrics among conducted experiments confirms the idea that the use of critical states for transfer training purposes is effective.

A new approach was introduced, which consists of performing TL purely on critical states to improve the efficiency of TL. In doing so, the focus of the training model was on challenging scenarios. This approach reduced the training step number significantly, maintaining the performance on a level compatible with conventional TL.

The model `Highway-Merge Model` that was trained with traditional TL on an array of states, sampled randomly, outperformed the `Pure Merge Model` trained on the *Merge-Env* from scratch. The latter, in turn, demonstrates a higher accident rate, inconsistency of decision-making, and overall unreliability. The `Highway-Merge Criticals` model, which used a list of previously collected critical states, was showing promising results maintaining a comparable performance while diminishing training time.

Through a deep exploration of data acquired for various metrics, significant insights into the performance and behaviour of `Highway-Merge Criticals` model were obtained. Understanding the strengths and focus areas of the model was achieved through the detailed analysis of those metrics. This assisted later in the evaluation of the effectiveness.

`Highway-Merge Criticals` model was able to achieve such performance results by making behaviour adjustments. Unlike the second TL model, the target model decreased the average driving speed(based on Section 4.4.5) to achieve better results in accident rate and merging speed reward. This capability of criticality TL models to re-evaluate the importance of different types of rewards can further be explored.

After the analysis of results, it is clear that `Highway-Merge Criticals` model can successfully navigate merging scenarios, being trained on a noticeably smaller set of input states. At the same time, overall results depict minor fluctuations, where the future optimisation of this approach can take place.

5.2. Future prospects

Future research can be focused on adapting an algorithm to improve the output of TL with the use of critical states. This can include the following points:

Exploring Different Ways of Criticality Identification: This thesis is based on critical states that were found when observing the behaviour of variance of Q-values for possible actions(Section 3.5). Evaluate the performance that could have been obtained if using improved or completely different methods to find critical states. For instance, the threshold selection method can be used to identify critical states. This, if refined in future research, may lead to a scharp rise of the effectiveness of criticality approach in TL.

Applying Different RL Algorithms: DQN algorithm was used throughout the paper. However, multiple RL methods exist, besides DQN, which can achieve higher performance results. For example Soft Actor-Critic(SAC) or Proximal Policy Optimisation(PPO).

Exploring Different Replay Buffer Setups: During the training process the pre-filled Replay Buffer was used, where critical experiences were sampled in random batches. To ensure the sampling of all critical states while training the model, it is worth attempting the application of non-random sampling, which will ensure that each critical state is sampled at least once. Moreover, while learning, the DQN did not update the Experience Replay with newly collected samples, leaving the Buffer static. Continuous appending or gradually replacing used critical states with freshly experienced states may avoid overfitting and profit training effectiveness. This leads to the deeper idea of Dynamic Criticality Sampling.

Exploring the Dynamic Criticality Sampling: The additional set of critical states can also be collected while training the model. Appending the Replay Buffer with newly found critical states can further improve the robustness and performance of the model.

Scaling to More Complex Environments: The implementation of a novel TL method was completed in between *Highway-Env* and *Merge-Env*. However, to validate the approach to the level where it can be applied in real-world scenarios, further investigation is required, when applying, explored above, mechanisms in more complex scenarios in the field of autonomous driving.

This paper demonstrates the approach of leveraging criticality for TL being an effective tool to boost RL in autonomous driving. The results of a validation defined a model, trained entirely on critical states, reduces training time while maintaining good effectiveness and securing the safety of the decision-making process. These results contribute to the wider RL field and autonomous vehicles in general, offering insights to be applied in future real-world realisations.

5.3. Outcome

The leveraging of criticality applied in TL demonstrates performance comparable with standard TL while requiring fewer training steps, thus less time and computation power. However, there are still some imperfections in the results and algorithm in general that need to be improved in future research.

When utilising critical states by sampling them from Replay Buffer to improve predictions of Neural Network for Q-values, the model can reach higher levels of safety and efficiency than traditional TL. Current implementation achieved those levels while, at this moment, still being less stable than ordinary TL, this was represented by wider standard deviation values in Figure 10. This work shows that the model trained using critical states chooses to sacrifice the reward

for higher speed to earn higher rewards for other driving aspects, like collision avoidance, merging speed or right lane occupation. Following prioritisation of rewards in a simulated environment will be reflected in a much safer and more effective behaviour of autonomous vehicles in real-world scenarios. It extracts valuable information from critical scenarios, which is then applied pointwise to train a policy to make more "thoughtful" decisions. Further research into the application of criticality can level up the TL in future, making it more accessible and widely used in various fields of everyday life.

References

[1] Asmaul Hosna et al. "Transfer learning: a friendly introduction". In: *Journal of Big Data* 9.1 (2022), p. 102. DOI: 10.1186/s40537-022-00652-w. URL: https://doi.org/10.1186/s40537-022-00652-w.

[2] Yuanfei Lin and Matthias Althoff. "CommonRoad-CriMe: A Toolbox for Criticality Measures of Autonomous Vehicles". In: *2023 IEEE Intelligent Vehicles Symposium (IV)*. 2023, pp. 1–8. DOI: 10.1109/IV55152.2023.10186673.

[3] Sandy H. Huang et al. *Establishing Appropriate Trust via Critical States*. 2018. arXiv: 1810.08174 [cs.RO]. URL: https://arxiv.org/abs/1810.08174.

[4] Yitzhak Spielberg and Amos Azaria. "Criticality-Based Advice in Reinforcement Learning (Student Abstract)". In: *Proceedings of the AAAI Conference on Artificial Intelligence* 36.11 (2022), pp. 13057–13058. DOI: 10.1609/aaai.v36i11.21665. URL: https://ojs.aaai.org/index.php/AAAI/article/view/21665.

[5] Tom Schaul et al. *Prioritized Experience Replay*. 2016. arXiv: 1511.05952 [cs.LG]. URL: https://arxiv.org/abs/1511.05952.

[6] Edouard Leurent. *An Environment for Autonomous Driving Decision-Making*. https://github.com/eleurent/highway-env. 2018.

[7] Mark Towers et al. *Gymnasium: A Standard Interface for Reinforcement Learning Environments*. 2024. arXiv: 2407.17032 [cs.LG]. URL: https://arxiv.org/abs/2407.17032.

[8] Farama Foundation. *Highway-Env Documentation: Actions*. https://highway-env.farama.org/actions/. Accessed: 2025-01-11. 2025.

[9] Antonin Raffin et al. "Stable-Baselines3: Reliable Reinforcement Learning Implementations". In: *Journal of Machine Learning Research* 22.268 (2021), pp. 1–8. URL: http://jmlr.org/papers/v22/20-1364.html.

[10] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, et al. "Human-level control through deep reinforcement learning". In: *Nature* 518.7540 (2015), pp. 529–533. DOI: 10.1038/nature14236.

[11] Michael Tokic. "Adaptive ϵ-greedy exploration in reinforcement learning based on value differences". In: *Proceedings of the 33rd Annual German Conference on Advances in Artificial Intelligence* (2011), pp. 203–210.

[12] Yitzhak Spielberg and Amos Azaria. "The Concept of Criticality in Reinforcement Learning". In: *2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI)*. 2019, pp. 251–258. DOI: 10.1109/ICTAI.2019.00043.

[13] Yuan Xue, Megha Khosla, and Daniel Kudenko. "Regulating Action Value Estimation in Deep Reinforcement Learning". In: *Proceedings of the Adaptive and Learning Agents Workshop (ALA 2023)*. London, UK, 2023. URL: https://alaworkshop2023.github.io/papers/ALA2023_paper_62.pdf.

[14] Pauli Virtanen et al. "SciPy 1.0: fundamental algorithms for scientific computing in Python". In: *Nature Methods* 17.3 (Feb. 2020), 261–272. ISSN: 1548-7105. DOI: 10.1038/s41592-019-0686-2. URL: http://dx.doi.org/10.1038/s41592-019-0686-2.

[15] David J. Foster and Matthew A. Wilson. "Reverse replay of behavioural sequences in hippocampal place cells during the awake state". In: *Nature* 440.7084 (2006), pp. 680–683. DOI: 10.1038/nature04587. URL: https://doi.org/10.1038/nature04587.

[16] Stanislaw Weglarczyk. "Kernel density estimation and its application". In: *ITM Web of Conferences* 23 (Nov. 2018), p. 00037. DOI: 10.1051/itmconf/20182300037.

[17] Yen-Chi Chen. *A Tutorial on Kernel Density Estimation and Recent Advances*. 2017. arXiv: 1704.03924 [stat.ME]. URL: https://arxiv.org/abs/1704.03924.

[18] Michael Waskom. "seaborn: statistical data visualization". In: *Journal of Open Source Software* 6 (Apr. 2021), p. 3021. DOI: 10.21105/joss.03021.

A. Appendix: Criticality Distribution

The approach to calculating the measure of criticality was explained in Section 3.5 of this paper. In more detail, Equations (8), (9), and (3) are used to obtain a single value that represents the criticality measure for a single observation. While performing 10 experiments of 1000 episodes each, criticality measures were collected. Based on the number of appearances of particular criticality values, the distribution shown in Figure 11 was generated.

Density-Function Approach for Criticality Visualisation For the analysis and visualisation of criticality metrics recorded throughout the experiments, the density-function approach was employed. The value of criticality, being a variance of Q-values, is a continuous variable that spans a wide range of values. To observe the behaviour of models via criticality distribution, the `Seaborn` library, an extension of `Matplotlib`, was utilized. This made the visualisation smooth and interpretable. Specifically, for estimating the probability density function of criticality, `Kernel Density Estimation (KDE)` was employed.

A common and widely used method to visualise the distribution of a continuous variable is the histogram [16]. The principle of the histogram involves splitting the range of continuous data into a discrete number of bins and then counting the number of samples within each bin. When applied to criticality values, the histogram approach divides the data into discrete intervals and displays the occurrence counts for each interval. However, this approach has several limitations:

1. **Sensitivity to Bin Size:** Choosing a larger bin size can lead to excessive generalisation and oversimplification of the graph. Conversely, smaller bin intervals can introduce noise and complicate result interpretation.
2. **Discontinuity:** Producing a stepped output, histograms cannot accurately convey patterns of continuous values such as criticality.
3. **Information Loss:** The discretisation of continuous values inevitably leads to the loss of valuable information about fluctuations in the explored variable.

Given these limitations, histograms are not an optimal method for representing criticality as a continuous value. Instead, the `Kernel Density Estimation (KDE)` method was employed.

Kernel Density Estimation for Criticality Visualisation Unlike histograms, KDE does not rely on discrete intervals; rather, it produces a smooth curve by approximating the distribution of the variable. This method helps to generate graphs of continuous distributions while preserving the original complex patterns [16].

By applying a kernel function (usually a Gaussian function) to every data point and summing the resultant functions, KDE produces a smooth estimate of the density function. KDE is regulated by two key parameters: *bandwidth* and *kernel function type*.

1. **Bandwidth:** The bandwidth parameter characterises the width of the kernel function application interval and thus determines the smoothness of the final graph. A wider bandwidth results in a smoother function with fewer details, while a smaller bandwidth preserves details but can also introduce noise to the estimated graph [17].
2. **Kernel Function:** The type of kernel function defines the influence of each data point on the final curve. For its symmetric and smooth properties, the Gaussian distribution is commonly used as the kernel function.

Implementation of KDE for Criticality Visualisation To utilise KDE for criticality visualisation, the `Seaborn` library implementation was used. `Seaborn` is a Python library designed as an extension of `Matplotlib` [18]. The `sns.kdeplot()` function was employed for visualising the criticality variable. This function created smooth density curves for criticality values, as represented in Figure 11. The criticality values were recorded over a total of 10,000 episodes of model evaluation. To enhance the visualisation, a logarithmic scale was applied to the y-axis, representing the density of particular criticality values.

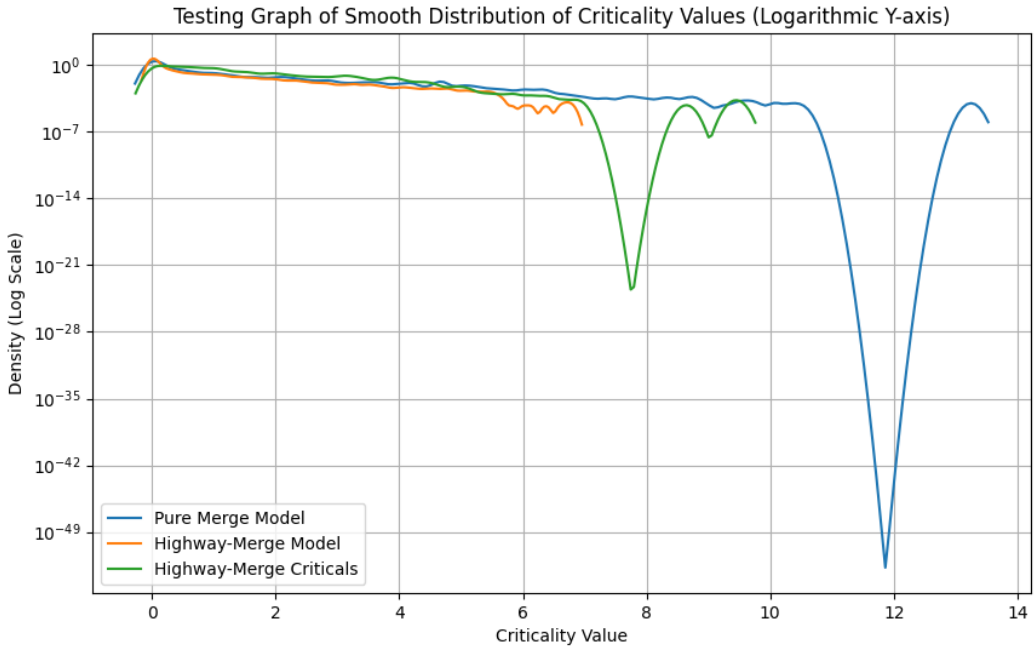

Figure 11: Combined distribution of criticality values throughout episodes for every model.

Observing Figure 11, differences in the criticality range of explored models can be noticed. The widest range of values is owned by `Pure Merge Model`. The biggest values observed in the criticality set of this model are approximately 13-13.5. If comparing this with the highest values of different models (`Highway-Merge Criticals`), numbers are getting down rapidly: 8.5-9.5. Analysing a wider range of criticality in the pure RL model it can be deduced that it tends to observe and/or initiate states where one of its actions plays the role of a single safe solution. Referring to TL models, the `Highway-Merge Model` performed with the statistics where the highest criticality values were in an interval of 6.5-7.

Although graphs show density function lines ending at particular points, there is also a form of graphs to be explored. As described previously, KDE approximates the probability density function of a variable by summing the chosen kernel function at each point of appearance. If observing the pattern of a blue graph on the interval 12-14 and the green graph on the interval 7.5-10 we can see multiple original Gaussian functions overlapping. However, those kinds of peaks represent single values and outliers of an overall set. To prove that, Figure 12 was constructed, where an additional single value was appended to the criticality values array for the `Highway-Merge Criticals` model. Thus the additional value 11 appeared as a Gaussian function centered at 11.

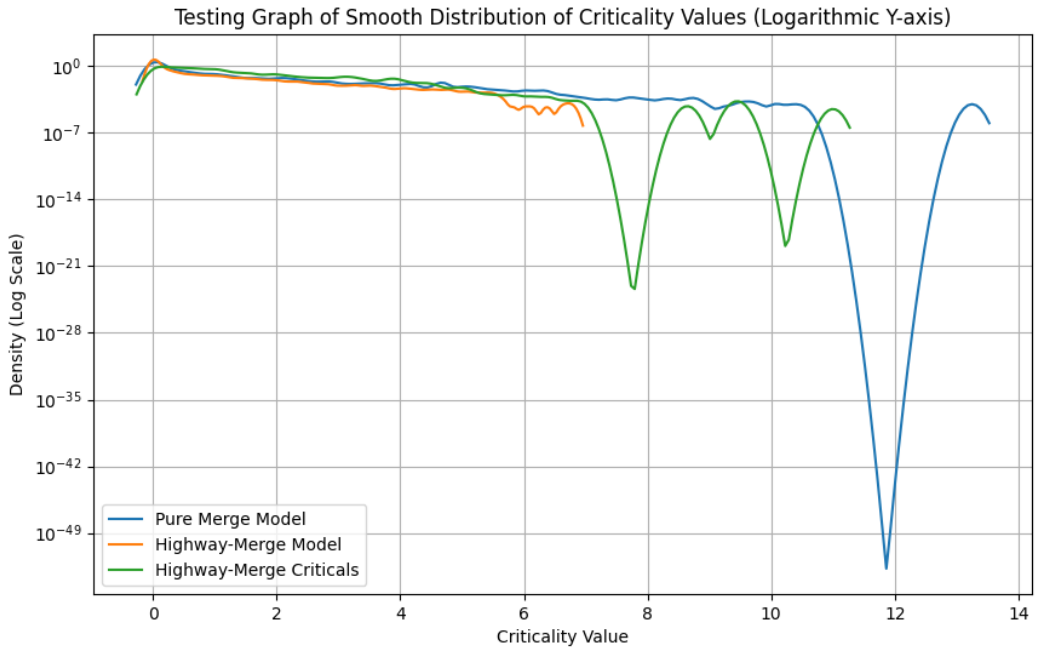

Figure 12: Test criticality distribution graph, created to show the principles of KDE

Based on this KDE principle exploration, values higher than 10.5 for `Pure Merge Model` and higher than 7.0 for `Highway-Merge Model` are single-value outliers.

In general, it can be noticed that TL models are better at handling criticality, and are prone to get into less critical scenarios rather than pure RL models.